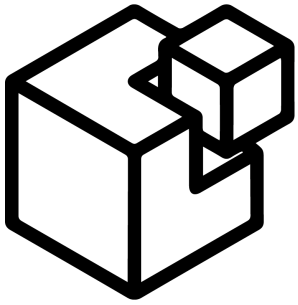


SALTSTACK

Salt Documentation

Release 2015.5.0

SaltStack, Inc.

May 11, 2015

Contents

1 Introduction to Salt **1**
- 1.1 The 30 second summary . . . 1
- 1.2 Simplicity . . . 1
- 1.3 Parallel execution . . . 1
- 1.4 Building on proven technology . . . 2
- 1.5 Python client interface . . . 2
- 1.6 Fast, flexible, scalable . . . 2
- 1.7 Open . . . 2
- 1.8 Salt Community . . . 2
- 1.9 Mailing List . . . 2
- 1.10 IRC . . . 3
- 1.11 Follow on Github . . . 3
- 1.12 Blogs . . . 3
- 1.13 Example Salt States . . . 3
- 1.14 Follow on ohloh . . . 3
- 1.15 Other community links . . . 4
- 1.16 Hack the Source . . . 4

2 Installation **5**
- 2.1 Quick Install . . . 5
- 2.2 Platform-specific Installation Instructions . . . 5
- 2.3 Dependencies . . . 26
- 2.4 Optional Dependencies . . . 27
- 2.5 Upgrading Salt . . . 27

3 Tutorials **29**
- 3.1 Introduction . . . 29
- 3.2 Basics . . . 31
- 3.3 States . . . 40
- 3.4 Advanced Topics . . . 69
- 3.5 Salt Virt . . . 109
- 3.6 Halite . . . 113
- 3.7 Using Salt at scale . . . 116

4 Targeting Minions **121**
- 4.1 Matching the `minion id` . . . 121
- 4.2 Grains . . . 122
- 4.3 Subnet/IP Address Matching . . . 125

4.4 Compound matchers . . . 126
4.5 Node groups . . . 127
4.6 Batch Size . . . 127
4.7 SECO Range . . . 128

5 Storing Static Data in the Pillar 131
5.1 Declaring the Master Pillar . . . 131
5.2 Pillar namespace flattened . . . 132
5.3 Pillar Namespace Merges . . . 133
5.4 Including Other Pillars . . . 134
5.5 Viewing Minion Pillar . . . 134
5.6 Pillar ``get'' Function . . . 134
5.7 Refreshing Pillar Data . . . 135
5.8 Targeting with Pillar . . . 135
5.9 Set Pillar Data at the Command Line . . . 135
5.10 Master Config In Pillar . . . 135
5.11 Master Provided Pillar Error . . . 136

6 Reactor System 137
6.1 Event System . . . 137
6.2 Mapping Events to Reactor SLS Files . . . 137
6.3 Fire an event . . . 138
6.4 Knowing what event is being fired . . . 138
6.5 Debugging the Reactor . . . 139
6.6 Understanding the Structure of Reactor Formulas . . . 139
6.7 A Complete Example . . . 142
6.8 Syncing Custom Types on Minion Start . . . 143

7 The Salt Mine 145
7.1 Mine Functions . . . 145
7.2 Mine Interval . . . 146
7.3 Mine in Salt-SSH . . . 146
7.4 Example . . . 146

8 External Authentication System 149
8.1 Access Control System . . . 149
8.2 Tokens . . . 150
8.3 LDAP and Active Directory . . . 150

9 Job Management 153
9.1 The Minion proc System . . . 153
9.2 Functions in the saltutil Module . . . 153
9.3 The jobs Runner . . . 153
9.4 Scheduling Jobs . . . 154
9.5 States . . . 156
9.6 Highstates . . . 156
9.7 Runners . . . 157
9.8 Scheduler With Returner . . . 157

10 Salt Event System 161
10.1 Event types . . . 161
10.2 Listening for Events . . . 164
10.3 Firing Events . . . 166
10.4 Firing Events from Python . . . 167

11 Salt Syndic **169**
11.1 Configuring the Syndic . . . 169
11.2 Running the Syndic . . . 169
11.3 Topology . . . 170
11.4 Syndic and the CLI . . . 170

12 Salt Proxy Minion Documentation **171**
12.1 Getting Started . . . 171
12.2 The __proxyenabled__ directive . . . 175

13 Windows Software Repository **177**
13.1 Operation . . . 177
13.2 Usage . . . 178
13.3 Generate Repo Cache File . . . 179
13.4 Install Windows Software . . . 180
13.5 Uninstall Windows Software . . . 180
13.6 Standalone Minion Salt Windows Repo Module . . . 180
13.7 Git Hosted Repo . . . 180
13.8 Troubleshooting . . . 181

14 Windows-specific Behaviour **183**
14.1 Group parameter for files . . . 183
14.2 Dealing with case-insensitive but case-preserving names . . . 183
14.3 Dealing with various username forms . . . 184
14.4 Specifying the None group . . . 184
14.5 Symbolic link loops . . . 184
14.6 Modifying security properties (ACLs) on files . . . 184

15 Salt Cloud **185**
15.1 Getting Started . . . 185
15.2 Using Salt Cloud . . . 185
15.3 Core Configuration . . . 190
15.4 Windows Configuration . . . 200
15.5 Cloud Provider Specifics . . . 202
15.6 Miscellaneous Options . . . 259
15.7 Troubleshooting Steps . . . 264
15.8 Extending Salt Cloud . . . 266
15.9 Using Salt Cloud from Salt . . . 275
15.10 Feature Comparison . . . 279
15.11 Tutorials . . . 282

16 `netapi` modules **287**
16.1 Writing netapi modules . . . 287
16.2 Introduction to netapi modules . . . 288
16.3 Client interfaces . . . 288

17 Salt Virt **291**
17.1 Salt Virt Tutorial . . . 291
17.2 The Salt Virt Runner . . . 291
17.3 Based on Live State Data . . . 292
17.4 Deploy from Network or Disk . . . 292

18 Understanding YAML **295**
18.1 Rule One: Indentation . . . 295
18.2 Rule Two: Colons . . . 295

18.3 Rule Three: Dashes . . . 296
18.4 Learning More . . . 296

19 Master Tops System **297**

20 Salt SSH **299**
20.1 Salt SSH Roster . . . 299
20.2 Calling Salt SSH . . . 300
20.3 States Via Salt SSH . . . 300
20.4 Targeting with Salt SSH . . . 300
20.5 Configuring Salt SSH . . . 300
20.6 Running Salt SSH as non-root user . . . 301
20.7 Define CLI Options with Saltfile . . . 301

21 Salt Rosters **303**
21.1 How Rosters Work . . . 303

22 Reference **305**
22.1 Full list of builtin auth modules . . . 305
22.2 Command Line Reference . . . 310
22.3 Client ACL system . . . 334
22.4 Python client API . . . 335
22.5 Full list of Salt Cloud modules . . . 345
22.6 Configuration file examples . . . 387
22.7 Configuring Salt . . . 411
22.8 Configuring the Salt Master . . . 413
22.9 Configuring the Salt Minion . . . 442
22.10 Running the Salt Master/Minion as an Unprivileged User . . . 456
22.11 Logging . . . 456
22.12 External Logging Handlers . . . 458
22.13 Salt File Server . . . 461
22.14 Full list of builtin fileserver modules . . . 466
22.15 Salt code and internals . . . 472
22.16 Full list of builtin execution modules . . . 479
22.17 Full list of netapi modules . . . 1040
22.18 Full list of builtin output modules . . . 1067
22.19 Peer Communication . . . 1073
22.20 Pillars . . . 1075
22.21 Full list of builtin pillar modules . . . 1075
22.22 Renderers . . . 1097
22.23 Returners . . . 1122
22.24 Full list of builtin roster modules . . . 1152
22.25 Salt Runners . . . 1156
22.26 State Enforcement . . . 1182
22.27 Full list of builtin state modules . . . 1232
22.28 Execution Modules . . . 1435
22.29 Master Tops . . . 1440
22.30 Full list of builtin master tops modules . . . 1440
22.31 Full list of builtin wheel modules . . . 1443
22.32 Full list of builtin beacon modules . . . 1446
22.33 Full list of builtin sdb modules . . . 1450

23 Salt Best Practices **1455**
23.1 General rules . . . 1455
23.2 Structuring States and Formulas . . . 1455

23.3 Structuring Pillar Files . . . 1456
23.4 Variable Flexibility . . . 1457
23.5 Modularity Within States . . . 1458
23.6 Storing Secure Data . . . 1461

24 Troubleshooting **1463**
24.1 Troubleshooting the Salt Master . . . 1463
24.2 Troubleshooting the Salt Minion . . . 1466
24.3 Running in the Foreground . . . 1468
24.4 What Ports do the Master and Minion Need Open? . . . 1469
24.5 Using salt-call . . . 1469
24.6 Too many open files . . . 1469
24.7 Salt Master Stops Responding . . . 1470
24.8 Salt and SELinux . . . 1470
24.9 Red Hat Enterprise Linux 5 . . . 1471
24.10 Common YAML Gotchas . . . 1471
24.11 Live Python Debug Output . . . 1476
24.12 Salt 0.16.x minions cannot communicate with a 0.17.x master . . . 1476
24.13 Debugging the Master and Minion . . . 1477

25 Developing Salt **1479**
25.1 Overview . . . 1479
25.2 Salt Client . . . 1479
25.3 Salt Master . . . 1479
25.4 Salt Minion . . . 1481
25.5 A Note on ClearFuncs vs. AESFuncs . . . 1482
25.6 Contributing . . . 1483
25.7 Deprecating Code . . . 1486
25.8 Dunder Dictionaries . . . 1487
25.9 External Pillars . . . 1489
25.10 Installing Salt for development . . . 1492
25.11 Logging Internals . . . 1496
25.12 Modular Systems . . . 1496
25.13 Package Providers . . . 1499
25.14 Community Projects That Use Salt . . . 1503
25.15 Salt Topology . . . 1504
25.16 Translating Documentation . . . 1504
25.17 Running The Tests . . . 1505
25.18 Automated Test Runs . . . 1508
25.19 Writing Tests . . . 1508
25.20 raet . . . 1521
25.21 SaltStack Git Policy . . . 1525
25.22 Salt Conventions . . . 1526

26 Release notes **1561**
26.1 Latest Stable Release . . . 1561
26.2 Previous Releases . . . 1569

27 Salt Based Projects **1667**
27.1 Salt Sandbox . . . 1667

28 Security disclosure policy **1669**
28.1 Security response procedure . . . 1670
28.2 Receiving security announcemnts . . . 1670

29 Frequently Asked Questions **1671**
29.1 Is Salt open-core? . . . 1671
29.2 What ports should I open on my firewall? . . . 1671
29.3 I'm seeing weird behavior (including but not limited to packages not installing their users properly) 1672
29.4 My script runs every time I run a *state.highstate*. Why? . . . 1672
29.5 When I run *test.ping*, why don't the Minions that aren't responding return anything? Returning `False` would be helpful. . . . 1672
29.6 How does Salt determine the Minion's id? . . . 1673
29.7 I'm trying to manage packages/services but I get an error saying that the state is not available. Why? 1673
29.8 I'm using gitfs and my custom modules/states/etc are not syncing. Why? . . . 1673
29.9 Why aren't my custom modules/states/etc. available on my Minions? . . . 1673
29.10 Module `X` isn't available, even though the shell command it uses is installed. Why? . . . 1673
29.11 Can I run different versions of Salt on my Master and Minion? . . . 1674
29.12 Does Salt support backing up managed files? . . . 1674
29.13 What is the best way to restart a Salt daemon using Salt? . . . 1674
29.14 Salting the Salt Master . . . 1675

30 Glossary **1677**

Salt Module Index **1681**

Index **1689**

CHAPTER 1

Introduction to Salt

We're not just talking about NaCl.

1.1 The 30 second summary

Salt is:

- a configuration management system, capable of maintaining remote nodes in defined states (for example, ensuring that specific packages are installed and specific services are running)
- a distributed remote execution system used to execute commands and query data on remote nodes, either individually or by arbitrary selection criteria

It was developed in order to bring the best solutions found in the world of remote execution together and make them better, faster, and more malleable. Salt accomplishes this through its ability to handle large loads of information, and not just dozens but hundreds and even thousands of individual servers quickly through a simple and manageable interface.

1.2 Simplicity

Providing versatility between massive scale deployments and smaller systems may seem daunting, but Salt is very simple to set up and maintain, regardless of the size of the project. The architecture of Salt is designed to work with any number of servers, from a handful of local network systems to international deployments across different data centers. The topology is a simple server/client model with the needed functionality built into a single set of daemons. While the default configuration will work with little to no modification, Salt can be fine tuned to meet specific needs.

1.3 Parallel execution

The core functions of Salt:

- enable commands to remote systems to be called in parallel rather than serially
- use a secure and encrypted protocol
- use the smallest and fastest network payloads possible
- provide a simple programming interface

Salt also introduces more granular controls to the realm of remote execution, allowing systems to be targeted not just by hostname, but also by system properties.

1.4 Building on proven technology

Salt takes advantage of a number of technologies and techniques. The networking layer is built with the excellent ZeroMQ networking library, so the Salt daemon includes a viable and transparent AMQ broker. Salt uses public keys for authentication with the master daemon, then uses faster AES encryption for payload communication; authentication and encryption are integral to Salt. Salt takes advantage of communication via msgpack, enabling fast and light network traffic.

1.5 Python client interface

In order to allow for simple expansion, Salt execution routines can be written as plain Python modules. The data collected from Salt executions can be sent back to the master server, or to any arbitrary program. Salt can be called from a simple Python API, or from the command line, so that Salt can be used to execute one-off commands as well as operate as an integral part of a larger application.

1.6 Fast, flexible, scalable

The result is a system that can execute commands at high speed on target server groups ranging from one to very many servers. Salt is very fast, easy to set up, amazingly malleable and provides a single remote execution architecture that can manage the diverse requirements of any number of servers. The Salt infrastructure brings together the best of the remote execution world, amplifies its capabilities and expands its range, resulting in a system that is as versatile as it is practical, suitable for any network.

1.7 Open

Salt is developed under the Apache 2.0 license, and can be used for open and proprietary projects. Please submit your expansions back to the Salt project so that we can all benefit together as Salt grows. Please feel free to sprinkle Salt around your systems and let the deliciousness come forth.

1.8 Salt Community

Join the Salt!

There are many ways to participate in and communicate with the Salt community.

Salt has an active IRC channel and a mailing list.

1.9 Mailing List

Join the salt-users mailing list. It is the best place to ask questions about Salt and see whats going on with Salt development! The Salt mailing list is hosted by Google Groups. It is open to new members.

https://groups.google.com/forum/#!forum/salt-users

1.10 IRC

The `#salt` IRC channel is hosted on the popular Freenode network. You can use the Freenode webchat client right from your browser.

Logs of the IRC channel activity are being collected courtesy of Moritz Lenz.

If you wish to discuss the development of Salt itself join us in `#salt-devel`.

1.11 Follow on Github

The Salt code is developed via Github. Follow Salt for constant updates on what is happening in Salt development:

https://github.com/saltstack/salt

1.12 Blogs

SaltStack Inc. keeps a blog with recent news and advancements:

http://www.saltstack.com/blog/

Thomas Hatch also shares news and thoughts on Salt and related projects in his personal blog The Red45:

http://red45.wordpress.com/

1.13 Example Salt States

The official `salt-states` repository is: https://github.com/saltstack/salt-states

A few examples of salt states from the community:

- https://github.com/blast-hardcheese/blast-salt-states
- https://github.com/kevingranade/kevingranade-salt-state
- https://github.com/uggedal/states
- https://github.com/mattmcclean/salt-openstack/tree/master/salt
- https://github.com/rentalita/ubuntu-setup/
- https://github.com/brutasse/states
- https://github.com/bclermont/states
- https://github.com/pcrews/salt-data

1.14 Follow on ohloh

https://www.ohloh.net/p/salt

1.15 Other community links

- Salt Stack Inc.
- Subreddit
- Google+
- YouTube
- Facebook
- Twitter
- Wikipedia page

1.16 Hack the Source

If you want to get involved with the development of source code or the documentation efforts, please review the *hacking section*!

CHAPTER 2

Installation

See also:

Installing Salt for development and contributing to the project.

2.1 Quick Install

On most distributions, you can set up a **Salt Minion** with the Salt Bootstrap.

2.2 Platform-specific Installation Instructions

These guides go into detail how to install Salt on a given platform.

2.2.1 Arch Linux

Installation

Salt (stable) is currently available via the Arch Linux Official repositories. There are currently -git packages available in the Arch User repositories (AUR) as well.

Stable Release

Install Salt stable releases from the Arch Linux Official repositories as follows:

```
pacman -S salt-zmq
```

To install Salt stable releases using the `RAET protocol`, use the following:

```
pacman -S salt-raet
```

Tracking develop

To install the bleeding edge version of Salt (**may include bugs!**), use the -git package. Installing the -git package as follows:

```
wget https://aur.archlinux.org/packages/sa/salt-git/salt-git.tar.gz
tar xf salt-git.tar.gz
cd salt-git/
makepkg -is
```

Note: yaourt

If a tool such as Yaourt is used, the dependencies will be gathered and built automatically.

The command to install salt using the yaourt tool is:

```
yaourt salt-git
```

Post-installation tasks

systemd

Activate the Salt Master and/or Minion via `systemctl` as follows:

```
systemctl enable salt-master.service
systemctl enable salt-minion.service
```

Start the Master

Once you've completed all of these steps you're ready to start your Salt Master. You should be able to start your Salt Master now using the command seen here:

```
systemctl start salt-master
```

Now go to the *Configuring Salt* page.

2.2.2 Debian Installation

Currently the latest packages for Debian Old Stable, Stable, and Unstable (Squeeze, Wheezy, and Sid) are published in our (saltstack.com) Debian repository.

Configure Apt

Squeeze (Old Stable)

For squeeze, you will need to enable the Debian backports repository as well as the debian.saltstack.com repository. To do so, add the following to `/etc/apt/sources.list` or a file in `/etc/apt/sources.list.d`:

```
deb http://debian.saltstack.com/debian squeeze-saltstack main
deb http://backports.debian.org/debian-backports squeeze-backports main contrib non-free
```

Wheezy (Stable)

For wheezy, the following line is needed in either `/etc/apt/sources.list` or a file in `/etc/apt/sources.list.d`:

```
deb http://debian.saltstack.com/debian wheezy-saltstack main
```

Jessie (Testing)

For jessie, the following line is needed in either `/etc/apt/sources.list` or a file in `/etc/apt/sources.list.d`:

```
deb http://debian.saltstack.com/debian jessie-saltstack main
```

Sid (Unstable)

For sid, the following line is needed in either `/etc/apt/sources.list` or a file in `/etc/apt/sources.list.d`:

```
deb http://debian.saltstack.com/debian unstable main
```

Import the repository key.

You will need to import the key used for signing.

```
wget -q -O- "http://debian.saltstack.com/debian-salt-team-joehealy.gpg.key" | apt-key add -
```

Note: You can optionally verify the key integrity with `sha512sum` using the public key signature shown here. E.g:

```
echo "b702969447140d5553e31e9701be13ca11cc0a7ed5fe2b30acb8491567560ee62f834772b5095d735dfcecb2384a5c
```

Update the package database

```
apt-get update
```

Install packages

Install the Salt master, minion, or syndic from the repository with the apt-get command. These examples each install one daemon, but more than one package name may be given at a time:

```
apt-get install salt-master
```

```
apt-get install salt-minion
```

```
apt-get install salt-syndic
```

Post-installation tasks

Now, go to the *Configuring Salt* page.

Notes

1. These packages will be backported from the packages intended to be uploaded into Debian unstable. This means that the packages will be built for unstable first and then backported over the next day or so.

2. These packages will be tracking the released versions of salt rather than maintaining a stable fixed feature set. If a fixed version is what you desire, then either pinning or manual installation may be more appropriate for you.

3. The version numbering and backporting process should provide clean upgrade paths between Debian versions.

If you have any questions regarding these, please email the mailing list or look for joehh on IRC.

2.2.3 Fedora

Beginning with version 0.9.4, Salt has been available in the primary Fedora repositories and EPEL. It is installable using yum. Fedora will have more up to date versions of Salt than other members of the Red Hat family, which makes it a great place to help improve Salt!

WARNING: Fedora 19 comes with systemd 204. Systemd has known bugs fixed in later revisions that prevent the salt-master from starting reliably or opening the network connections that it needs to. It's not likely that a salt-master will start or run reliably on any distribution that uses systemd version 204 or earlier. Running salt-minions should be OK.

Installation

Salt can be installed using `yum` and is available in the standard Fedora repositories.

Stable Release

Salt is packaged separately for the minion and the master. It is necessary only to install the appropriate package for the role the machine will play. Typically, there will be one master and multiple minions.

```
yum install salt-master
yum install salt-minion
```

Installing from `updates-testing`

When a new Salt release is packaged, it is first admitted into the `updates-testing` repository, before being moved to the stable repo.

To install from `updates-testing`, use the `enablerepo` argument for yum:

```
yum --enablerepo=updates-testing install salt-master
yum --enablerepo=updates-testing install salt-minion
```

Post-installation tasks

Master

To have the Master start automatically at boot time:

```
systemctl enable salt-master.service
```

To start the Master:

```
systemctl start salt-master.service
```

Minion

To have the Minion start automatically at boot time:

```
systemctl enable salt-minion.service
```

To start the Minion:

```
systemctl start salt-minion.service
```

Now go to the *Configuring Salt* page.

2.2.4 FreeBSD

Salt was added to the FreeBSD ports tree Dec 26th, 2011 by Christer Edwards <christer.edwards@gmail.com>. It has been tested on FreeBSD 7.4, 8.2, 9.0, and 9.1 releases.

Salt is dependent on the following additional ports. These will be installed as dependencies of the `sysutils/py-salt` port:

```
/devel/py-yaml
/devel/py-pyzmq
/devel/py-Jinja2
/devel/py-msgpack
/security/py-pycrypto
/security/py-m2crypto
```

Installation

On FreeBSD 10 and later, to install Salt from the FreeBSD pkgng repo, use the command:

```
pkg install py27-salt
```

On older versions of FreeBSD, to install Salt from the FreeBSD ports tree, use the command:

```
make -C /usr/ports/sysutils/py-salt install clean
```

Post-installation tasks

Master

Copy the sample configuration file:

```
cp /usr/local/etc/salt/master.sample /usr/local/etc/salt/master
```

rc.conf

Activate the Salt Master in `/etc/rc.conf` or `/etc/rc.conf.local` and add:

```
+ salt_master_enable="YES"
```

Start the Master

Start the Salt Master as follows:

```
service salt_master start
```

Minion

Copy the sample configuration file:

```
cp /usr/local/etc/salt/minion.sample /usr/local/etc/salt/minion
```

rc.conf

Activate the Salt Minion in `/etc/rc.conf` or `/etc/rc.conf.local` and add:

```
+ salt_minion_enable="YES"
+ salt_minion_paths="/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin"
```

Start the Minion

Start the Salt Minion as follows:

```
service salt_minion start
```

Now go to the *Configuring Salt* page.

2.2.5 Gentoo

Salt can be easily installed on Gentoo via Portage:

```
emerge app-admin/salt
```

Post-installation tasks

Now go to the *Configuring Salt* page.

2.2.6 OS X

Dependency Installation

It should be noted that Homebrew explicitly discourages the use of sudo:

> Homebrew is designed to work without using sudo. You can decide to use it but we strongly recommend not to do so. If you have used sudo and run into a bug then it is likely to be the cause. Please don't file a bug report unless you can reproduce it after reinstalling Homebrew from scratch without using sudo

So when using Homebrew, if you want support from the Homebrew community, install this way:

```
brew install saltstack
```

When using MacPorts, install this way:

```
sudo port install salt
```

When only using the OS X system's pip, install this way:

```
sudo pip install salt
```

Salt-Master Customizations

To run salt-master on OS X, the root user maxfiles limit must be increased:

```
sudo launchctl limit maxfiles 4096 8192
```

And sudo add this configuration option to the /etc/salt/master file:

```
max_open_files: 8192
```

Now the salt-master should run without errors:

```
sudo salt-master --log-level=all
```

Post-installation tasks

Now go to the *Configuring Salt* page.

2.2.7 RHEL / CentOS / Scientific Linux / Amazon Linux / Oracle Linux

Installation Using pip

Since Salt is on PyPI, it can be installed using pip, though most users prefer to install using RPMs (which can be installed from EPEL). Installation from pip is easy:

```
pip install salt
```

Warning: If installing from pip (or from source using `setup.py install`), be advised that the `yum-utils` package is needed for Salt to manage packages. Also, if the Python dependencies are not already installed, then you will need additional libraries/tools installed to build some of them. More information on this can be found *here*.

Installation from Repository

RHEL/CentOS 5

Due to the removal of some of Salt's dependencies from EPEL5, we have created a repository on Fedora COPR. Moving forward, this will be the official means of installing Salt on RHEL5-based systems. Information on how to enable this repository can be found here.

RHEL/CentOS 6 and 7, Scientific Linux, etc.

Beginning with version 0.9.4, Salt has been available in EPEL. It is installable using yum. Salt should work properly with all mainstream derivatives of RHEL, including CentOS, Scientific Linux, Oracle Linux and Amazon Linux. Report any bugs or issues on the issue tracker.

On RHEL6, the proper Jinja package `python-jinja2' was moved from EPEL to the ``RHEL Server Optional Channel''. Verify this repository is enabled before installing salt on RHEL6.

Enabling EPEL If the EPEL repository is not installed on your system, you can download the RPM from here for RHEL/CentOS 6 (or here for RHEL/CentOS 7) and install it using the following command:

```
rpm -Uvh epel-release-X-Y.rpm
```

Replace `epel-release-X-Y.rpm` with the appropriate filename.

Installing Stable Release Salt is packaged separately for the minion and the master. It is necessary only to install the appropriate package for the role the machine will play. Typically, there will be one master and multiple minions.

On the salt-master, run this:

```
yum install salt-master
```

On each salt-minion, run this:

```
yum install salt-minion
```

Installing from `epel-testing` When a new Salt release is packaged, it is first admitted into the `epel-testing` repository, before being moved to the stable repo.

To install from `epel-testing`, use the `enablerepo` argument for yum:

```
yum --enablerepo=epel-testing install salt-minion
```

ZeroMQ 4

We recommend using ZeroMQ 4 where available. SaltStack provides ZeroMQ 4.0.4 and pyzmq 14.3.1 in a COPR repository. Instructions for adding this repository (as well as for upgrading ZeroMQ and pyzmq on existing minions) can be found here.

If this repo is added *before* Salt is installed, then installing either `salt-master` or `salt-minion` will automatically pull in ZeroMQ 4.0.4, and additional states to upgrade ZeroMQ and pyzmq are unnecessary.

> **Warning**: RHEL/CentOS 5 Users Using COPR repos on RHEL/CentOS 5 requires that the `python-hashlib` package be installed. Not having it present will result in checksum errors because YUM will not be able to process the SHA256 checksums used by COPR.

Note: For RHEL/CentOS 5 installations, if using the new repository to install Salt (as detailed *above*), then it is not necessary to enable the zeromq4 COPR, as the new EL5 repository includes ZeroMQ 4.

Package Management

Salt's interface to `yum` makes heavy use of the **repoquery** utility, from the yum-utils package. This package will be installed as a dependency if salt is installed via EPEL. However, if salt has been installed using pip, or a host is being managed using salt-ssh, then as of version 2014.7.0 yum-utils will be installed automatically to satisfy this dependency.

Post-installation tasks

Master

To have the Master start automatically at boot time:

```
chkconfig salt-master on
```

To start the Master:

```
service salt-master start
```

Minion

To have the Minion start automatically at boot time:

```
chkconfig salt-minion on
```

To start the Minion:

```
service salt-minion start
```

Now go to the *Configuring Salt* page.

2.2.8 Solaris

Salt was added to the OpenCSW package repository in September of 2012 by Romeo Theriault <romeot@hawaii.edu> at version 0.10.2 of Salt. It has mainly been tested on Solaris 10 (sparc), though it is built for and has been tested minimally on Solaris 10 (x86), Solaris 9 (sparc/x86) and 11 (sparc/x86). (Please let me know if you're using it on these platforms!) Most of the testing has also just focused on the minion, though it has verified that the master starts up successfully on Solaris 10.

Comments and patches for better support on these platforms is very welcome.

As of version 0.10.4, Solaris is well supported under salt, with all of the following working well:

1. remote execution
2. grain detection
3. service control with SMF
4. `pkg' states with `pkgadd' and `pkgutil' modules
5. cron modules/states
6. user and group modules/states
7. shadow password management modules/states

Salt is dependent on the following additional packages. These will automatically be installed as dependencies of the `py_salt` package:

- py_yaml
- py_pyzmq
- py_jinja2
- py_msgpack_python
- py_m2crypto
- py_crypto

- python

Installation

To install Salt from the OpenCSW package repository you first need to install pkgutil assuming you don't already have it installed:

On Solaris 10:

```
pkgadd -d http://get.opencsw.org/now
```

On Solaris 9:

```
wget http://mirror.opencsw.org/opencsw/pkgutil.pkg
pkgadd -d pkgutil.pkg all
```

Once pkgutil is installed you'll need to edit it's config file `/etc/opt/csw/pkgutil.conf` to point it at the unstable catalog:

```
- #mirror=http://mirror.opencsw.org/opencsw/testing
+ mirror=http://mirror.opencsw.org/opencsw/unstable
```

OK, time to install salt.

```
# Update the catalog
root> /opt/csw/bin/pkgutil -U
# Install salt
root> /opt/csw/bin/pkgutil -i -y py_salt
```

Minion Configuration

Now that salt is installed you can find it's configuration files in `/etc/opt/csw/salt/`.

You'll want to edit the minion config file to set the name of your salt master server:

```
- #master: salt
+ master: your-salt-server
```

If you would like to use pkgutil as the default package provider for your Solaris minions, you can do so using the `providers` option in the minion config file.

You can now start the salt minion like so:

On Solaris 10:

```
svcadm enable salt-minion
```

On Solaris 9:

```
/etc/init.d/salt-minion start
```

You should now be able to log onto the salt master and check to see if the salt-minion key is awaiting acceptance:

```
salt-key -l un
```

Accept the key:

```
salt-key -a <your-salt-minion>
```

Run a simple test against the minion:

```
salt '<your-salt-minion>' test.ping
```

Troubleshooting

Logs are in `/var/log/salt`

2.2.9 Ubuntu Installation

Add repository

The latest packages for Ubuntu are published in the saltstack PPA. If you have the `add-apt-repository` utility, you can add the repository and import the key in one step:

```
sudo add-apt-repository ppa:saltstack/salt
```

---

add-apt-repository: command not found?

The `add-apt-repository` command is not always present on Ubuntu systems. This can be fixed by installing *python-software-properties*:

```
sudo apt-get install python-software-properties
```

The following may be required as well:

```
sudo apt-get install software-properties-common
```

Note that since Ubuntu 12.10 (Raring Ringtail), `add-apt-repository` is found in the *software-properties-common* package, and is part of the base install. Thus, `add-apt-repository` should be able to be used out-of-the-box to add the PPA.

Alternately, manually add the repository and import the PPA key with these commands:

```
echo deb http://ppa.launchpad.net/saltstack/salt/ubuntu `lsb_release -sc` main | sudo tee /etc/apt/so
wget -q -O- "http://keyserver.ubuntu.com:11371/pks/lookup?op=get&search=0x4759FA960E27C0A6" | sudo ap
```

After adding the repository, update the package management database:

```
sudo apt-get update
```

Install packages

Install the Salt master, minion, or syndic from the repository with the apt-get command. These examples each install one daemon, but more than one package name may be given at a time:

```
sudo apt-get install salt-master
```

```
sudo apt-get install salt-minion
```

```
sudo apt-get install salt-syndic
```

ZeroMQ 4

ZeroMQ 4 is available by default for Ubuntu 14.04 and newer. However, for Ubuntu 12.04 LTS, starting with Salt version `2014.7.5`, ZeroMQ 4 is included with the Salt installation package and nothing additional needs to be done.

Post-installation tasks

Now go to the *Configuring Salt* page.

2.2.10 Windows

Salt has full support for running the Salt Minion on Windows.

There are no plans for the foreseeable future to develop a Salt Master on Windows. For now you must run your Salt Master on a supported operating system to control your Salt Minions on Windows.

Many of the standard Salt modules have been ported to work on Windows and many of the Salt States currently work on Windows, as well.

Windows Installer

Salt Minion Windows installers can be found here. The output of *md5sum <salt minion exe>* should match the contents of the corresponding md5 file.

Download here

- 2014.7.5
- Salt-Minion-2014.7.5-x86-Setup.exe | md5
- Salt-Minion-2014.7.5-AMD64-Setup.exe | md5
- 2014.7.4
- Salt-Minion-2014.7.4-x86-Setup.exe | md5
- Salt-Minion-2014.7.4-AMD64-Setup.exe | md5
- 2014.7.2
- Salt-Minion-2014.7.2-x86-Setup.exe | md5
- Salt-Minion-2014.7.2-AMD64-Setup.exe | md5
- 2014.7.1
- Salt-Minion-2014.7.1-x86-Setup.exe | md5
- Salt-Minion-2014.7.1-AMD64-Setup.exe | md5
- 2014.7.0
- Salt-Minion-2014.7.0-1-win32-Setup.exe | md5
- Salt-Minion-2014.7.0-AMD64-Setup.exe | md5

Note: The 2014.7.0 installers have been removed because of a regression. Please use the 2014.7.1 release instead.

- 2014.1.13

- Salt-Minion-2014.1.13-x86-Setup.exe | md5
- Salt-Minion-2014.1.13-AMD64-Setup.exe | md5
- 2014.1.11
- Salt-Minion-2014.1.11-win32-Setup.exe | md5
- Salt-Minion-2014.1.11-AMD64-Setup.exe | md5
- 2014.1.10
- Salt-Minion-2014.1.10-win32-Setup.exe | md5
- Salt-Minion-2014.1.10-AMD64-Setup.exe | md5
- 2014.1.7
- Salt-Minion-2014.1.7-win32-Setup.exe | md5
- Salt-Minion-2014.1.7-AMD64-Setup.exe | md5
- 2014.1.5
- Salt-Minion-2014.1.5-win32-Setup.exe | md5
- Salt-Minion-2014.1.5-AMD64-Setup.exe | md5
- 2014.1.4
- Salt-Minion-2014.1.4-win32-Setup.exe | md5
- Salt-Minion-2014.1.4-AMD64-Setup.exe | md5
- 2014.1.3-1 (packaging bugfix)
- Salt-Minion-2014.1.3-1-win32-Setup.exe | md5
- Salt-Minion-2014.1.3-1-AMD64-Setup.exe | md5
- 2014.1.3
- Salt-Minion-2014.1.3-win32-Setup.exe | md5
- Salt-Minion-2014.1.3-AMD64-Setup.exe | md5
- 2014.1.1
- Salt-Minion-2014.1.1-win32-Setup.exe | md5
- Salt-Minion-2014.1.1-AMD64-Setup.exe | md5
- 2014.1.0
- Salt-Minion-2014.1.0-win32-Setup.exe | md5
- Salt-Minion-2014.1.0-AMD64-Setup.exe | md5
- 0.17.5-2 (bugfix release)
- https://docs.saltstack.com/downloads/Salt-Minion-0.17.5-2-win32-Setup.exe
- https://docs.saltstack.com/downloads/Salt-Minion-0.17.5-2-AMD64-Setup.exe
- 0.17.5
- https://docs.saltstack.com/downloads/Salt-Minion-0.17.5-win32-Setup.exe
- https://docs.saltstack.com/downloads/Salt-Minion-0.17.5-AMD64-Setup.exe
- 0.17.4

- https://docs.saltstack.com/downloads/Salt-Minion-0.17.4-win32-Setup.exe
- https://docs.saltstack.com/downloads/Salt-Minion-0.17.4-AMD64-Setup.exe
- 0.17.2
- https://docs.saltstack.com/downloads/Salt-Minion-0.17.2-win32-Setup.exe
- https://docs.saltstack.com/downloads/Salt-Minion-0.17.2-AMD64-Setup.exe
- 0.17.1.1 - Windows Installer bugfix release
- https://docs.saltstack.com/downloads/Salt-Minion-0.17.1.1-win32-Setup.exe
- https://docs.saltstack.com/downloads/Salt-Minion-0.17.1.1-AMD64-Setup.exe
- 0.17.1
- https://docs.saltstack.com/downloads/Salt-Minion-0.17.1-win32-Setup.exe
- https://docs.saltstack.com/downloads/Salt-Minion-0.17.1-AMD64-Setup.exe
- 0.17.0
- https://docs.saltstack.com/downloads/Salt-Minion-0.17.0-win32-Setup.exe
- https://docs.saltstack.com/downloads/Salt-Minion-0.17.0-AMD64-Setup.exe
- 0.16.3
- https://docs.saltstack.com/downloads/Salt-Minion-0.16.3-win32-Setup.exe
- https://docs.saltstack.com/downloads/Salt-Minion-0.16.3-AMD64-Setup.exe
- 0.16.2
- https://docs.saltstack.com/downloads/Salt-Minion-0.16.2-win32-Setup.exe
- https://docs.saltstack.com/downloads/Salt-Minion-0.16.2-AMD64-Setup.exe
- 0.16.0
- https://docs.saltstack.com/downloads/Salt-Minion-0.16.0-win32-Setup.exe
- https://docs.saltstack.com/downloads/Salt-Minion-0.16.0-AMD64-Setup.exe
- 0.15.3
- https://docs.saltstack.com/downloads/Salt-Minion-0.15.3-win32-Setup.exe
- https://docs.saltstack.com/downloads/Salt-Minion-0.15.3-AMD64-Setup.exe
- 0.14.1
- https://docs.saltstack.com/downloads/Salt-Minion-0.14.1-win32-Setup.exe
- https://docs.saltstack.com/downloads/Salt-Minion-0.14.1-AMD64-Setup.exe
- 0.14.0
- https://docs.saltstack.com/downloads/Salt-Minion-0.14.0-win32-Setup.exe
- https://docs.saltstack.com/downloads/Salt-Minion-0.14.0-AMD64-Setup.exe

Note: The executables above will install dependencies that the Salt minion requires.

The 64bit installer has been tested on Windows 7 64bit and Windows Server 2008R2 64bit. The 32bit installer has been tested on Windows 2003 Server 32bit. Please file a bug report on our GitHub repo if issues for other platforms are found.

The installer asks for 2 bits of information; the master hostname and the minion name. The installer will update the minion config with these options and then start the minion.

The *salt-minion* service will appear in the Windows Service Manager and can be started and stopped there or with the command line program *sc* like any other Windows service.

If the minion won't start, try installing the Microsoft Visual C++ 2008 x64 SP1 redistributable. Allow all Windows updates to run salt-minion smoothly.

Silent Installer option

The installer can be run silently by providing the */S* option at the command line. The options */master* and */minion-name* allow for configuring the master hostname and minion name, respectively. Here's an example of using the silent installer:

```
Salt-Minion-0.17.0-Setup-amd64.exe /S /master=yoursaltmaster /minion-name=yourminionname
```

Setting up a Windows build environment

This document will explain how to set up a development environment for salt on Windows. The development environment allows you to work with the source code to customize or fix bugs. It will also allow you to build your own installation.

The Easy Way

Prerequisite Software To do this the easy way you only need to install Git for Windows.

Create the Build Environment

1. Clone the Salt-Windows-Dev repo from github.

   Open a command line and type:

   ```
   git clone https://github.com/saltstack/salt-windows-dev
   ```

2. Build the Python Environment

   Go into the salt-windows-dev directory. Right-click the file named **dev_env.ps1** and select **Run with PowerShell**

   If you get an error, you may need to change the execution policy.

   Open a powershell window and type the following:

   ```
   Set-ExecutionPolicy RemoteSigned
   ```

   This will download and install Python with all the dependencies needed to develop and build salt.

3. Build the Salt Environment

   Right-click on the file named **dev_env_salt.ps1** and select **Run with Powershell**

   This will clone salt into `C:\Salt-Dev\salt` and set it to the 2015.5 branch. You could optionally run the command from a powershell window with a `-Version` switch to pull a different version. For example:

```
dev_env_salt.ps1 -Version '2014.7'
```

To view a list of available branches and tags, open a command prompt in your *C:Salt-Devsalt* directory and type:

```
git branch -a
git tag -n
```

The Hard Way

Prerequisite Software Install the following software:

1. Git for Windows
2. Nullsoft Installer

Download the Prerequisite zip file for your CPU architecture from the SaltStack download site:

- Salt32.zip
- Salt64.zip

These files contain all sofware required to build and develop salt. Unzip the contents of the file to `C:\Salt-Dev\temp`.

Create the Build Environment

1. Build the Python Environment

   - Install Python:

     Browse to the `C:\Salt-Dev\temp` directory and find the Python installation file for your CPU Architecture under the corresponding subfolder. Double-click the file to install python.

     Make sure the following are in your **PATH** environment variable:

     ```
     C:\Python27
     C:\Python27\Scripts
     ```

   - Install Pip

     Open a command prompt and navigate to `C:\Salt-Dev\temp` Run the following command:

     ```
     python get-pip.py
     ```

   - Easy Install compiled binaries.

     M2Crypto, PyCrypto, and PyWin32 need to be installed using Easy Install. Open a command prompt and navigate to `C:\Salt-Dev\temp\<cpuarch>`. Run the following commands:

     ```
     easy_install -Z <M2Crypto file name>
     easy_install -Z <PyCrypto file name>
     easy_install -Z <PyWin32 file name>
     ```

     Note: You can type the first part of the file name and then press the tab key to auto-complete the name of the file.

   - Pip Install Additional Prerequisites

     All remaining prerequisites need to be pip installed. These prerequisites are as follow:

- MarkupSafe
- Jinja
- MsgPack
- PSUtil
- PyYAML
- PyZMQ
- WMI
- Requests
- Certifi

Open a command prompt and navigate to `C:\Salt-Dev\temp`. Run the following commands:

```
pip install <cpuarch>\<MarkupSafe file name>
pip install <Jinja file name>
pip install <cpuarch>\<MsgPack file name>
pip install <cpuarch>\<psutil file name>
pip install <cpuarch>\<PyYAML file name>
pip install <cpuarch>\<pyzmq file name>
pip install <WMI file name>
pip install <requests file name>
pip install <certifi file name>
```

2. Build the Salt Environment

- Clone Salt

  Open a command prompt and navigate to `C:\Salt-Dev`. Run the following command to clone salt:

```
git clone https://github.com/saltstack/salt
```

- Checkout Branch

  Checkout the branch or tag of salt you want to work on or build. Open a command prompt and navigate to `C:\Salt-Dev\salt`. Get a list of available tags and branches by running the following commands:

```
git fetch --all

To view a list of available branches:
git branch -a

To view a list of availabel tags:
git tag -n
```

  Checkout the branch or tag by typing the following command:

```
git checkout <branch/tag name>
```

- Clean the Environment

  When switching between branches residual files can be left behind that will interfere with the functionality of salt. Therefore, after you check out the branch you want to work on, type the following commands to clean the salt environment:

Developing with Salt

There are two ways to develop with salt. You can run salt's setup.py each time you make a change to source code or you can use the setup tools develop mode.

Configure the Minion

Both methods require that the minion configuration be in the `C:\salt` directory. Copy the conf and var directories from `C:\Salt-Dev\salt\pkg\ windows\buildenv` to `C:\salt`. Now go into the `C:\salt\conf` directory and edit the file name `minion` (no extension). You need to configure the master and id parameters in this file. Edit the following lines:

```
master: <ip or name of your master>
id: <name of your minion>
```

Setup.py Method

Go into the `C:\Salt-Dev\salt` directory from a cmd prompt and type:

```
python setup.py install --force
```

This will install python into your python installation at `C:\Python27`. Everytime you make an edit to your source code, you'll have to stop the minion, run the setup, and start the minion.

To start the salt-minion go into `C:\Python27\Scripts` from a cmd prompt and type:

```
salt-minion
```

For debug mode type:

```
salt-minion -l debug
```

To stop the minion press Ctrl+C.

Setup Tools Develop Mode (Preferred Method)

To use the Setup Tools Develop Mode go into `C:\Salt-Dev\salt` from a cmd prompt and type:

```
pip install -e .
```

This will install pointers to your source code that resides at `C:\Salt-Dev\salt`. When you edit your source code you only have to restart the minion.

Build the windows installer

This is the method of building the installer as of version 2014.7.4.

Clean the Environment

Make sure you don't have any leftover salt files from previous versions of salt in your Python directory.

1. Remove all files that start with salt in the `C:\Python27\Scripts` directory
2. Remove all files and directorys that start with salt in the `C:\Python27\Lib\site-packages` directory

Install Salt

Install salt using salt's setup.py. From the `C:\Salt-Dev\salt` directory type the following command:

```
python setup.py install --force
```

Build the Installer

From cmd prompt go into the `C:\Salt-Dev\salt\pkg\windows` directory. Type the following command for the branch or tag of salt you're building:

```
BuildSalt.bat <branch or tag>
```

This will copy python with salt installed to the `buildenv\bin` directory, make it portable, and then create the windows installer . The .exe for the windows installer will be placed in the `installer` directory.

Testing the Salt minion

1. Create the directory C:\salt (if it doesn't exist already)
2. Copy the example `conf` and `var` directories from `pkg/windows/buildenv/` into C:\salt
3. Edit C:\salt\conf\minion

```
   master: ipaddress or hostname of your salt-master
```

4. Start the salt-minion

```
   cd C:\Python27\Scripts
   python salt-minion
```

5. On the salt-master accept the new minion's key

```
   sudo salt-key -A
```

   This accepts all unaccepted keys. If you're concerned about security just accept the key for this specific minion.

6. Test that your minion is responding

   On the salt-master run:

```
   sudo salt '*' test.ping
```

You should get the following response: `{'your minion hostname': True}`

Single command bootstrap script

On a 64 bit Windows host the following script makes an unattended install of salt, including all dependencies:

Not up to date.

This script is not up to date. Please use the installer found above

---

```
# (All in one line.)

"PowerShell (New-Object System.Net.WebClient).DownloadFile('http://csa-net.dk/salt/bootstrap64.bat',
```

You can execute the above command remotely from a Linux host using winexe:

```
winexe -U "administrator" //fqdn "PowerShell (New-Object ......);"
```

For more info check http://csa-net.dk/salt

Packages management under Windows 2003

On windows Server 2003, you need to install optional component ``wmi windows installer provider'' to have full list of installed packages. If you don't have this, salt-minion can't report some installed softwares.

2.2.11 SUSE Installation

With openSUSE 13.1, Salt 0.16.4 has been available in the primary repositories. The devel:language:python repo will have more up to date versions of salt, all package development will be done there.

Installation

Salt can be installed using `zypper` and is available in the standard openSUSE 13.1 repositories.

Stable Release

Salt is packaged separately for the minion and the master. It is necessary only to install the appropriate package for the role the machine will play. Typically, there will be one master and multiple minions.

```
zypper install salt-master
zypper install salt-minion
```

Post-installation tasks openSUSE

Master

To have the Master start automatically at boot time:

```
systemctl enable salt-master.service
```

To start the Master:

```
systemctl start salt-master.service
```

Minion

To have the Minion start automatically at boot time:

```
systemctl enable salt-minion.service
```

To start the Minion:

```
systemctl start salt-minion.service
```

Post-installation tasks SLES

Master

To have the Master start automatically at boot time:

```
chkconfig salt-master on
```

To start the Master:

```
rcsalt-master start
```

Minion

To have the Minion start automatically at boot time:

```
chkconfig salt-minion on
```

To start the Minion:

```
rcsalt-minion start
```

Unstable Release

openSUSE

For openSUSE Factory run the following as root:

```
zypper addrepo http://download.opensuse.org/repositories/devel:languages:python/openSUSE_Factory/deve
zypper refresh
zypper install salt salt-minion salt-master
```

For openSUSE 13.1 run the following as root:

```
zypper addrepo http://download.opensuse.org/repositories/devel:languages:python/openSUSE_13.1/devel:
zypper refresh
zypper install salt salt-minion salt-master
```

For openSUSE 12.3 run the following as root:

```
zypper addrepo http://download.opensuse.org/repositories/devel:languages:python/openSUSE_12.3/devel:
zypper refresh
zypper install salt salt-minion salt-master
```

For openSUSE 12.2 run the following as root:

```
zypper addrepo http://download.opensuse.org/repositories/devel:languages:python/openSUSE_12.2/devel:
zypper refresh
zypper install salt salt-minion salt-master
```

For openSUSE 12.1 run the following as root:

```
zypper addrepo http://download.opensuse.org/repositories/devel:languages:python/openSUSE_12.1/devel:
zypper refresh
zypper install salt salt-minion salt-master
```

For bleeding edge python Factory run the following as root:

```
zypper addrepo http://download.opensuse.org/repositories/devel:languages:python/bleeding_edge_python
zypper refresh
zypper install salt salt-minion salt-master
```

Suse Linux Enterprise

For SLE 11 SP3 run the following as root:

```
zypper addrepo http://download.opensuse.org/repositories/devel:languages:python/SLE_11_SP3/devel:lang
zypper refresh
zypper install salt salt-minion salt-master
```

For SLE 11 SP2 run the following as root:

```
zypper addrepo http://download.opensuse.org/repositories/devel:languages:python/SLE_11_SP2/devel:lang
zypper refresh
zypper install salt salt-minion salt-master
```

Now go to the *Configuring Salt* page.

2.3 Dependencies

Salt should run on any Unix-like platform so long as the dependencies are met.

- Python 2.6 >= 2.6 <3.0
- msgpack-python - High-performance message interchange format
- YAML - Python YAML bindings
- Jinja2 - parsing Salt States (configurable in the master settings)
- MarkupSafe - Implements a XML/HTML/XHTML Markup safe string for Python
- apache-libcloud - Python lib for interacting with many of the popular cloud service providers using a unified API
- Requests - HTTP library

Depending on the chosen Salt transport, ZeroMQ or RAET, dependencies vary:

- ZeroMQ:
  - ZeroMQ >= 3.2.0
  - pyzmq >= 2.2.0 - ZeroMQ Python bindings
  - PyCrypto - The Python cryptography toolkit
  - M2Crypto - ``Me Too Crypto'' - Python OpenSSL wrapper
- RAET:
  - libnacl - Python bindings to libsodium
  - ioflo - The flo programming interface raet and salt-raet is built on
  - RAET - The worlds most awesome UDP protocol

Salt defaults to the ZeroMQ transport, and the choice can be made at install time, for example:

```
python setup.py install --salt-transport=raet
```

This way, only the required dependencies are pulled by the setup script if need be.

If installing using pip, the `--salt-transport` install option can be provided like:

```
pip install --install-option="--salt-transport=raet" salt
```

2.4 Optional Dependencies

- mako - an optional parser for Salt States (configurable in the master settings)
- gcc - dynamic Cython module compiling

2.5 Upgrading Salt

When upgrading Salt, the master(s) should always be upgraded first. Backward compatibility for minions running newer versions of salt than their masters is not guaranteed.

Whenever possible, backward compatibility between new masters and old minions will be preserved. Generally, the only exception to this policy is in case of a security vulnerability.

CHAPTER 3

Tutorials

3.1 Introduction

3.1.1 Salt Masterless Quickstart

Running a masterless salt-minion lets you use Salt's configuration management for a single machine without calling out to a Salt master on another machine.

Since the Salt minion contains such extensive functionality it can be useful to run it standalone. A standalone minion can be used to do a number of things:

- Stand up a master server via States (Salting a Salt Master)
- Use salt-call commands on a system without connectivity to a master
- Masterless States, run states entirely from files local to the minion

It is also useful for testing out state trees before deploying to a production setup.

Bootstrap Salt Minion

The salt-bootstrap script makes bootstrapping a server with Salt simple for any OS with a Bourne shell:

```
wget -O - https://bootstrap.saltstack.com | sudo sh
```

See the salt-bootstrap documentation for other one liners. When using Vagrant to test out salt, the salty-vagrant tool will provision the VM for you.

Telling Salt to Run Masterless

To instruct the minion to not look for a master when running the `file_client` configuration option needs to be set. By default the `file_client` is set to `remote` so that the minion knows that file server and pillar data are to be gathered from the master. When setting the `file_client` option to `local` the minion is configured to not gather this data from the master.

```
file_client: local
```

Now the salt minion will not look for a master and will assume that the local system has all of the file and pillar resources.

Note: When running Salt in masterless mode, do not run the salt-minion daemon. Otherwise, it will attempt to

connect to a master and fail. The salt-call command stands on its own and does not need the salt-minion daemon.

Create State Tree

Following the successful installation of a salt-minion, the next step is to create a state tree, which is where the SLS files that comprise the possible states of the minion are stored.

The following example walks through the steps necessary to create a state tree that ensures that the server has the Apache webserver installed.

Note: For a complete explanation on Salt States, see the tutorial.

1. Create the `top.sls` file:

`/srv/salt/top.sls:`

```
base:
  '*':
    - webserver
```

2. Create the webserver state tree:

`/srv/salt/webserver.sls:`

```
apache:               # ID declaration
  pkg:                # state declaration
    - installed       # function declaration
```

Note: The apache package has different names on different platforms, for instance on Debian/Ubuntu it is apache2, on Fedora/RHEL it is httpd and on Arch it is apache

The only thing left is to provision our minion using salt-call and the highstate command.

Salt-call

The salt-call command is used to run module functions locally on a minion instead of executing them from the master. Normally the salt-call command checks into the master to retrieve file server and pillar data, but when running standalone salt-call needs to be instructed to not check the master for this data:

```
salt-call --local state.highstate
```

The `--local` flag tells the salt-minion to look for the state tree in the local file system and not to contact a Salt Master for instructions.

To provide verbose output, use `-l debug`:

```
salt-call --local state.highstate -l debug
```

The minion first examines the `top.sls` file and determines that it is a part of the group matched by `*` glob and that the `webserver` SLS should be applied.

It then examines the `webserver.sls` file and finds the `apache` state, which installs the Apache package.

The minion should now have Apache installed, and the next step is to begin learning how to write *more complex states*.

3.2 Basics

3.2.1 Standalone Minion

Since the Salt minion contains such extensive functionality it can be useful to run it standalone. A standalone minion can be used to do a number of things:

- Use salt-call commands on a system without connectivity to a master
- Masterless States, run states entirely from files local to the minion

---

Note: When running Salt in masterless mode, do not run the salt-minion daemon. Otherwise, it will attempt to connect to a master and fail. The salt-call command stands on its own and does not need the salt-minion daemon.

Telling Salt Call to Run Masterless

The salt-call command is used to run module functions locally on a minion instead of executing them from the master. Normally the salt-call command checks into the master to retrieve file server and pillar data, but when running standalone salt-call needs to be instructed to not check the master for this data. To instruct the minion to not look for a master when running salt-call the `file_client` configuration option needs to be set. By default the `file_client` is set to `remote` so that the minion knows that file server and pillar data are to be gathered from the master. When setting the `file_client` option to `local` the minion is configured to not gather this data from the master.

```
file_client: local
```

Now the salt-call command will not look for a master and will assume that the local system has all of the file and pillar resources.

Running States Masterless

The state system can be easily run without a Salt master, with all needed files local to the minion. To do this the minion configuration file needs to be set up to know how to return file_roots information like the master. The file_roots setting defaults to /srv/salt for the base environment just like on the master:

```
file_roots:
  base:
    - /srv/salt
```

Now set up the Salt State Tree, top file, and SLS modules in the same way that they would be set up on a master. Now, with the `file_client` option set to `local` and an available state tree then calls to functions in the state module will use the information in the file_roots on the minion instead of checking in with the master.

Remember that when creating a state tree on a minion there are no syntax or path changes needed, SLS modules written to be used from a master do not need to be modified in any way to work with a minion.

This makes it easy to ``script'' deployments with Salt states without having to set up a master, and allows for these SLS modules to be easily moved into a Salt master as the deployment grows.

The declared state can now be executed with:

```
salt-call state.highstate
```

Or the salt-call command can be executed with the `--local` flag, this makes it unnecessary to change the configuration file:

```
salt-call state.highstate --local
```

3.2.2 Opening the Firewall up for Salt

The Salt master communicates with the minions using an AES-encrypted ZeroMQ connection. These communications are done over TCP ports **4505** and **4506**, which need to be accessible on the master only. This document outlines suggested firewall rules for allowing these incoming connections to the master.

Note: No firewall configuration needs to be done on Salt minions. These changes refer to the master only.

Fedora 18 and beyond / RHEL 7 / CentOS 7

Starting with Fedora 18 FirewallD is the tool that is used to dynamically manage the firewall rules on a host. It has support for IPv4/6 settings and the separation of runtime and permanent configurations. To interact with FirewallD use the command line client `firewall-cmd`.

firewall-cmd example:

```
firewall-cmd --permanent --zone=<zone> --add-port=4505-4506/tcp
```

Please choose the desired zone according to your setup. Don't forget to reload after you made your changes.

```
firewall-cmd --reload
```

RHEL 6 / CentOS 6

The `lokkit` command packaged with some Linux distributions makes opening iptables firewall ports very simple via the command line. Just be careful to not lock out access to the server by neglecting to open the ssh port.

lokkit example:

```
lokkit -p 22:tcp -p 4505:tcp -p 4506:tcp
```

The `system-config-firewall-tui` command provides a text-based interface to modifying the firewall.

system-config-firewall-tui:

```
system-config-firewall-tui
```

openSUSE

Salt installs firewall rules in /etc/sysconfig/SuSEfirewall2.d/services/salt. Enable with:

```
SuSEfirewall2 open
SuSEfirewall2 start
```

If you have an older package of Salt where the above configuration file is not included, the `SuSEfirewall2` command makes opening iptables firewall ports very simple via the command line.

SuSEfirewall example:

```
SuSEfirewall2 open EXT TCP 4505
SuSEfirewall2 open EXT TCP 4506
```

The firewall module in YaST2 provides a text-based interface to modifying the firewall.

YaST2:

```
yast2 firewall
```

iptables

Different Linux distributions store their *iptables* (also known as netfilter) rules in different places, which makes it difficult to standardize firewall documentation. Included are some of the more common locations, but your mileage may vary.

Fedora / RHEL / CentOS:

```
/etc/sysconfig/iptables
```

Arch Linux:

```
/etc/iptables/iptables.rules
```

Debian

Follow these instructions: https://wiki.debian.org/iptables

Once you've found your firewall rules, you'll need to add the two lines below to allow traffic on `tcp/4505` and `tcp/4506`:

```
-A INPUT -m state --state new -m tcp -p tcp --dport 4505 -j ACCEPT
-A INPUT -m state --state new -m tcp -p tcp --dport 4506 -j ACCEPT
```

Ubuntu

Salt installs firewall rules in /etc/ufw/applications.d/salt.ufw. Enable with:

```
ufw allow salt
```

pf.conf

The BSD-family of operating systems uses packet filter (pf). The following example describes the additions to `pf.conf` needed to access the Salt master.

```
pass in on $int_if proto tcp from any to $int_if port 4505
pass in on $int_if proto tcp from any to $int_if port 4506
```

Once these additions have been made to the `pf.conf` the rules will need to be reloaded. This can be done using the `pfctl` command.

```
pfctl -vf /etc/pf.conf
```

3.2.3 Whitelist communication to Master

There are situations where you want to selectively allow Minion traffic from specific hosts or networks into your Salt Master. The first scenario which comes to mind is to prevent unwanted traffic to your Master out of security concerns, but another scenario is to handle Minion upgrades when there are backwards incompatible changes between the installed Salt versions in your environment.

Here is an example *Linux iptables* ruleset to be set on the Master:

```
# Allow Minions from these networks
-I INPUT -s 10.1.2.0/24 -p tcp -m multiport --dports 4505,4506 -j ACCEPT
-I INPUT -s 10.1.3.0/24 -p tcp -m multiport --dports 4505,4506 -j ACCEPT
# Allow Salt to communicate with Master on the loopback interface
-A INPUT -i lo -p tcp -m multiport --dports 4505,4506 -j ACCEPT
# Reject everything else
-A INPUT -p tcp -m multiport --dports 4505,4506 -j REJECT
```

Note: The important thing to note here is that the `salt` command needs to communicate with the listening network socket of `salt-master` on the *loopback* interface. Without this you will see no outgoing Salt traffic from the master, even for a simple `salt '*' test.ping`, because the `salt` client never reached the `salt-master` to tell it to carry out the execution.

3.2.4 Using cron with Salt

The Salt Minion can initiate its own highstate using the `salt-call` command.

```
$ salt-call state.highstate
```

This will cause the minion to check in with the master and ensure it is in the correct `state'.

Use cron to initiate a highstate

If you would like the Salt Minion to regularly check in with the master you can use the venerable cron to run the `salt-call` command.

```
# PATH=/bin:/sbin:/usr/bin:/usr/sbin

00 00 * * * salt-call state.highstate
```

The above cron entry will run a highstate every day at midnight.

Note: Be aware that you may need to ensure the PATH for cron includes any scripts or commands that need to be executed.

3.2.5 Remote execution tutorial

Before continuing make sure you have a working Salt installation by following the *installation* and the *configuration* instructions.

Stuck?

There are many ways to *get help from the Salt community* including our mailing list and our IRC channel #salt.

Order your minions around

Now that you have a *master* and at least one *minion* communicating with each other you can perform commands on the minion via the **`salt`** command. Salt calls are comprised of three main components:

```
salt '<target>' <function> [arguments]
```

See also:

salt manpage

target

The target component allows you to filter which minions should run the following function. The default filter is a glob on the minion id. For example:

```
salt '*' test.ping
salt '*.example.org' test.ping
```

Targets can be based on minion system information using the Grains system:

```
salt -G 'os:Ubuntu' test.ping
```

See also:

Grains system

Targets can be filtered by regular expression:

```
salt -E 'virtmach[0-9]' test.ping
```

Targets can be explicitly specified in a list:

```
salt -L 'foo,bar,baz,quo' test.ping
```

Or Multiple target types can be combined in one command:

```
salt -C 'G@os:Ubuntu and webser* or E@database.*' test.ping
```

function

A function is some functionality provided by a module. Salt ships with a large collection of available functions. List all available functions on your minions:

```
salt '*' sys.doc
```

Here are some examples:

Show all currently available minions:

```
salt '*' test.ping
```

Run an arbitrary shell command:

```
salt '*' cmd.run 'uname -a'
```

See also:

the full list of modules

arguments

Space-delimited arguments to the function:

```
salt '*' cmd.exec_code python 'import sys; print sys.version'
```

Optional, keyword arguments are also supported:

```
salt '*' pip.install salt timeout=5 upgrade=True
```

They are always in the form of `kwarg=argument`.

3.2.6 Pillar Walkthrough

Note: This walkthrough assumes that the reader has already completed the initial Salt *walkthrough*.

Pillars are tree-like structures of data defined on the Salt Master and passed through to minions. They allow confidential, targeted data to be securely sent only to the relevant minion.

Note: Grains and Pillar are sometimes confused, just remember that Grains are data about a minion which is stored or generated from the minion. This is why information like the OS and CPU type are found in Grains. Pillar is information about a minion or many minions stored or generated on the Salt Master.

Pillar data is useful for:

Highly Sensitive Data: Information transferred via pillar is guaranteed to only be presented to the minions that are targeted, making Pillar suitable for managing security information, such as cryptographic keys and passwords.

Minion Configuration: Minion modules such as the execution modules, states, and returners can often be configured via data stored in pillar.

Variables: Variables which need to be assigned to specific minions or groups of minions can be defined in pillar and then accessed inside sls formulas and template files.

Arbitrary Data: Pillar can contain any basic data structure, so a list of values, or a key/value store can be defined making it easy to iterate over a group of values in sls formulas

Pillar is therefore one of the most important systems when using Salt. This walkthrough is designed to get a simple Pillar up and running in a few minutes and then to dive into the capabilities of Pillar and where the data is available.

Setting Up Pillar

The pillar is already running in Salt by default. To see the minion's pillar data:

```
salt '*' pillar.items
```

Note: Prior to version 0.16.2, this function is named `pillar.data`. This function name is still supported for backwards compatibility.

By default the contents of the master configuration file are loaded into pillar for all minions. This enables the master configuration file to be used for global configuration of minions.

Similar to the state tree, the pillar is comprised of sls files and has a top file. The default location for the pillar is in /srv/pillar.

Note: The pillar location can be configured via the *pillar_roots* option inside the master configuration file. It must not be in a subdirectory of the state tree.

To start setting up the pillar, the /srv/pillar directory needs to be present:

```
mkdir /srv/pillar
```

Now create a simple top file, following the same format as the top file used for states:

`/srv/pillar/top.sls`:

```
base:
  '*':
    - data
```

This top file associates the data.sls file to all minions. Now the `/srv/pillar/data.sls` file needs to be populated:

`/srv/pillar/data.sls`:

```
info: some data
```

To ensure that the minions have the new pillar data, issue a command to them asking that they fetch their pillars from the master:

```
salt '*' saltutil.refresh_pillar
```

Now that the minions have the new pillar, it can be retrieved:

```
salt '*' pillar.items
```

The key `info` should now appear in the returned pillar data.

More Complex Data

Unlike states, pillar files do not need to define **formulas**. This example sets up user data with a UID:

`/srv/pillar/users/init.sls`:

```
users:
  thatch: 1000
  shouse: 1001
  utahdave: 1002
  redbeard: 1003
```

Note: The same directory lookups that exist in states exist in pillar, so the file `users/init.sls` can be referenced with `users` in the *top file*.

The top file will need to be updated to include this sls file:

`/srv/pillar/top.sls`:

```
base:
  '*':
    - data
    - users
```

Now the data will be available to the minions. To use the pillar data in a state, you can use Jinja:

`/srv/salt/users/init.sls`

```
{% for user, uid in pillar.get('users', {}).items() %}
{{user}}:
  user.present:
    - uid: {{uid}}
{% endfor %}
```

This approach allows for users to be safely defined in a pillar and then the user data is applied in an sls file.

Parameterizing States With Pillar

Pillar data can be accessed in state files to customise behavior for each minion. All pillar (and grain) data applicable to each minion is substituted into the state files through templating before being run. Typical uses include setting directories appropriate for the minion and skipping states that don't apply.

A simple example is to set up a mapping of package names in pillar for separate Linux distributions:

`/srv/pillar/pkg/init.sls`:

```
pkgs:
  {% if grains['os_family'] == 'RedHat' %}
  apache: httpd
  vim: vim-enhanced
  {% elif grains['os_family'] == 'Debian' %}
  apache: apache2
  vim: vim
  {% elif grains['os'] == 'Arch' %}
  apache: apache
  vim: vim
  {% endif %}
```

The new `pkg` sls needs to be added to the top file:

`/srv/pillar/top.sls`:

```
base:
  '*':
    - data
    - users
    - pkg
```

Now the minions will auto map values based on respective operating systems inside of the pillar, so sls files can be safely parameterized:

`/srv/salt/apache/init.sls`:

```
apache:
  pkg.installed:
    - name: {{ pillar['pkgs']['apache'] }}
```

Or, if no pillar is available a default can be set as well:

Note: The function `pillar.get` used in this example was added to Salt in version 0.14.0

`/srv/salt/apache/init.sls`:

```
apache:
  pkg.installed:
    - name: {{ salt['pillar.get']('pkgs:apache', 'httpd') }}
```

In the above example, if the pillar value `pillar['pkgs']['apache']` is not set in the minion's pillar, then the default of `httpd` will be used.

Note: Under the hood, pillar is just a Python dict, so Python dict methods such as *get* and *items* can be used.

Pillar Makes Simple States Grow Easily

One of the design goals of pillar is to make simple sls formulas easily grow into more flexible formulas without refactoring or complicating the states.

A simple formula:

`/srv/salt/edit/vim.sls`:

```
vim:
  pkg.installed: []

/etc/vimrc:
  file.managed:
    - source: salt://edit/vimrc
    - mode: 644
    - user: root
    - group: root
    - require:
      - pkg: vim
```

Can be easily transformed into a powerful, parameterized formula:

`/srv/salt/edit/vim.sls`:

```
vim:
  pkg.installed:
    - name: {{ pillar['pkgs']['vim'] }}

/etc/vimrc:
  file.managed:
    - source: {{ pillar['vimrc'] }}
    - mode: 644
    - user: root
    - group: root
    - require:
      - pkg: vim
```

Where the vimrc source location can now be changed via pillar:

`/srv/pillar/edit/vim.sls`:

```
{% if grains['id'].startswith('dev') %}
vimrc: salt://edit/dev_vimrc
{% elif grains['id'].startswith('qa') %}
vimrc: salt://edit/qa_vimrc
{% else %}
vimrc: salt://edit/vimrc
{% endif %}
```

Ensuring that the right vimrc is sent out to the correct minions.

Setting Pillar Data on the Command Line

Pillar data can be set on the command line like so:

```
salt '*' state.highstate pillar='{"foo": "bar"}'
```

The `state.sls` command can also be used to set pillar values via the command line:

```
salt '*' state.sls my_sls_file pillar='{"hello": "world"}'
```

Lists can be passed in pillar as well:

```
salt '*' state.highstate pillar='["foo", "bar", "baz"]'
```

Note: If a key is passed on the command line that already exists on the minion, the key that is passed in will overwrite the entire value of that key, rather than merging only the specified value set via the command line.

More On Pillar

Pillar data is generated on the Salt master and securely distributed to minions. Salt is not restricted to the pillar sls files when defining the pillar but can retrieve data from external sources. This can be useful when information about an infrastructure is stored in a separate location.

Reference information on pillar and the external pillar interface can be found in the Salt documentation:

Pillar

3.3 States

3.3.1 How Do I Use Salt States?

Simplicity, Simplicity, Simplicity

Many of the most powerful and useful engineering solutions are founded on simple principles. Salt States strive to do just that: K.I.S.S. (Keep It Stupidly Simple)

The core of the Salt State system is the SLS, or **S**a**L**t **S**tate file. The SLS is a representation of the state in which a system should be in, and is set up to contain this data in a simple format. This is often called configuration management.

Note: This is just the beginning of using states, make sure to read up on pillar *Pillar* next.

It is All Just Data

Before delving into the particulars, it will help to understand that the SLS file is just a data structure under the hood. While understanding that the SLS is just a data structure isn't critical for understanding and making use of Salt States, it should help bolster knowledge of where the real power is.

SLS files are therefore, in reality, just *dictionaries*, *lists*, *strings*, and *numbers*. By using this approach Salt can be much more flexible. As one writes more state files, it becomes clearer exactly what is being written. The result is a system that is easy to understand, yet grows with the needs of the admin or developer.

The Top File

The example SLS files in the below sections can be assigned to hosts using a file called **top.sls**. This file is described in-depth *here*.

Default Data - YAML

By default Salt represents the SLS data in what is one of the simplest serialization formats available - YAML.

A typical SLS file will often look like this in YAML:

Note: These demos use some generic service and package names, different distributions often use different names for packages and services. For instance *apache* should be replaced with *httpd* on a Red Hat system. Salt uses the name of the init script, systemd name, upstart name etc. based on what the underlying service management for the platform. To get a list of the available service names on a platform execute the service.get_all salt function.

Information on how to make states work with multiple distributions is later in the tutorial.

```
apache:
  pkg.installed: []
  service.running:
    - require:
      - pkg: apache
```

This SLS data will ensure that the package named apache is installed, and that the apache service is running. The components can be explained in a simple way.

The first line is the ID for a set of data, and it is called the ID Declaration. This ID sets the name of the thing that needs to be manipulated.

The second and third lines contain the state module function to be run, in the format `<state_module>.<function>`. The `pkg.installed` state module function ensures that a software package is installed via the system's native package manager. The `service.running` state module function ensures that a given system daemon is running.

Finally, on line five, is the word `require`. This is called a Requisite Statement, and it makes sure that the Apache service is only started after a successful installation of the apache package.

Adding Configs and Users

When setting up a service like an Apache web server, many more components may need to be added. The Apache configuration file will most likely be managed, and a user and group may need to be set up.

```
apache:
  pkg.installed: []
  service.running:
    - watch:
      - pkg: apache
      - file: /etc/httpd/conf/httpd.conf
      - user: apache
  user.present:
    - uid: 87
    - gid: 87
    - home: /var/www/html
    - shell: /bin/nologin
    - require:
```

```
      - group: apache
  group.present:
    - gid: 87
    - require:
      - pkg: apache

/etc/httpd/conf/httpd.conf:
  file.managed:
    - source: salt://apache/httpd.conf
    - user: root
    - group: root
    - mode: 644
```

This SLS data greatly extends the first example, and includes a config file, a user, a group and new requisite statement: `watch`.

Adding more states is easy, since the new user and group states are under the Apache ID, the user and group will be the Apache user and group. The `require` statements will make sure that the user will only be made after the group, and that the group will be made only after the Apache package is installed.

Next, the `require` statement under service was changed to watch, and is now watching 3 states instead of just one. The watch statement does the same thing as require, making sure that the other states run before running the state with a watch, but it adds an extra component. The `watch` statement will run the state's watcher function for any changes to the watched states. So if the package was updated, the config file changed, or the user uid modified, then the service state's watcher will be run. The service state's watcher just restarts the service, so in this case, a change in the config file will also trigger a restart of the respective service.

Moving Beyond a Single SLS

When setting up Salt States in a scalable manner, more than one SLS will need to be used. The above examples were in a single SLS file, but two or more SLS files can be combined to build out a State Tree. The above example also references a file with a strange source - `salt://apache/httpd.conf`. That file will need to be available as well.

The SLS files are laid out in a directory structure on the Salt master; an SLS is just a file and files to download are just files.

The Apache example would be laid out in the root of the Salt file server like this:

```
apache/init.sls
apache/httpd.conf
```

So the httpd.conf is just a file in the apache directory, and is referenced directly.

But when using more than one single SLS file, more components can be added to the toolkit. Consider this SSH example:

`ssh/init.sls:`

```
openssh-client:
  pkg.installed

/etc/ssh/ssh_config:
  file.managed:
    - user: root
    - group: root
    - mode: 644
    - source: salt://ssh/ssh_config
```

```
    - require:
      - pkg: openssh-client
```

ssh/server.sls:

```
include:
  - ssh

openssh-server:
  pkg.installed

sshd:
  service.running:
    - require:
      - pkg: openssh-client
      - pkg: openssh-server
      - file: /etc/ssh/banner
      - file: /etc/ssh/sshd_config

/etc/ssh/sshd_config:
  file.managed:
    - user: root
    - group: root
    - mode: 644
    - source: salt://ssh/sshd_config
    - require:
      - pkg: openssh-server

/etc/ssh/banner:
  file:
    - managed
    - user: root
    - group: root
    - mode: 644
    - source: salt://ssh/banner
    - require:
      - pkg: openssh-server
```

Note: Notice that we use two similar ways of denoting that a file is managed by Salt. In the */etc/ssh/sshd_config* state section above, we use the *file.managed* state declaration whereas with the */etc/ssh/banner* state section, we use the *file* state declaration and add a *managed* attribute to that state declaration. Both ways produce an identical result; the first way -- using *file.managed* -- is merely a shortcut.

Now our State Tree looks like this:

```
apache/init.sls
apache/httpd.conf
ssh/init.sls
ssh/server.sls
ssh/banner
ssh/ssh_config
ssh/sshd_config
```

This example now introduces the `include` statement. The include statement includes another SLS file so that components found in it can be required, watched or as will soon be demonstrated - extended.

The include statement allows for states to be cross linked. When an SLS has an include statement it is literally extended to include the contents of the included SLS files.

Note that some of the SLS files are called init.sls, while others are not. More info on what this means can be found in the *States Tutorial*.

Extending Included SLS Data

Sometimes SLS data needs to be extended. Perhaps the apache service needs to watch additional resources, or under certain circumstances a different file needs to be placed.

In these examples, the first will add a custom banner to ssh and the second will add more watchers to apache to include mod_python.

`ssh/custom-server.sls:`

```
include:
  - ssh.server

extend:
  /etc/ssh/banner:
    file:
      - source: salt://ssh/custom-banner
```

`python/mod_python.sls:`

```
include:
  - apache

extend:
  apache:
    service:
      - watch:
        - pkg: mod_python

mod_python:
  pkg.installed
```

The `custom-server.sls` file uses the extend statement to overwrite where the banner is being downloaded from, and therefore changing what file is being used to configure the banner.

In the new mod_python SLS the mod_python package is added, but more importantly the apache service was extended to also watch the mod_python package.

Using extend with require or watch

The `extend` statement works differently for `require` or `watch`. It appends to, rather than replacing the requisite component.

Understanding the Render System

Since SLS data is simply that (data), it does not need to be represented with YAML. Salt defaults to YAML because it is very straightforward and easy to learn and use. But the SLS files can be rendered from almost any imaginable medium, so long as a renderer module is provided.

The default rendering system is the `yaml_jinja` renderer. The `yaml_jinja` renderer will first pass the template through the Jinja2 templating system, and then through the YAML parser. The benefit here is that full programming constructs are available when creating SLS files.

Other renderers available are `yaml_mako` and `yaml_wempy` which each use the Mako or Wempy templating system respectively rather than the jinja templating system, and more notably, the pure Python or `py`, `pydsl` & `pyobjects` renderers. The `py` renderer allows for SLS files to be written in pure Python, allowing for the utmost level of flexibility and power when preparing SLS data; while the *pydsl* renderer provides a flexible, domain-specific language for authoring SLS data in Python; and the *pyobjects* renderer gives you a ``Pythonic'' interface to building state data.

Note: The templating engines described above aren't just available in SLS files. They can also be used in `file.managed` states, making file management much more dynamic and flexible. Some examples for using templates in managed files can be found in the documentation for the *file states*, as well as the *MooseFS example* below.

Getting to Know the Default - yaml_jinja

The default renderer - `yaml_jinja`, allows for use of the jinja templating system. A guide to the Jinja templating system can be found here: http://jinja.pocoo.org/docs

When working with renderers a few very useful bits of data are passed in. In the case of templating engine based renderers, three critical components are available, `salt`, `grains`, and `pillar`. The `salt` object allows for any Salt function to be called from within the template, and `grains` allows for the Grains to be accessed from within the template. A few examples:

`apache/init.sls:`

```
apache:
  pkg.installed:
    {% if grains['os'] == 'RedHat'%}
    - name: httpd
    {% endif %}
  service.running:
    {% if grains['os'] == 'RedHat'%}
    - name: httpd
    {% endif %}
    - watch:
      - pkg: apache
      - file: /etc/httpd/conf/httpd.conf
      - user: apache
  user.present:
    - uid: 87
    - gid: 87
    - home: /var/www/html
    - shell: /bin/nologin
    - require:
      - group: apache
  group.present:
    - gid: 87
    - require:
      - pkg: apache

/etc/httpd/conf/httpd.conf:
  file.managed:
    - source: salt://apache/httpd.conf
    - user: root
    - group: root
    - mode: 644
```

This example is simple. If the `os` grain states that the operating system is Red Hat, then the name of the Apache package and service needs to be httpd. A more aggressive way to use Jinja can be found here, in a module to set up

a MooseFS distributed filesystem chunkserver:

moosefs/chunk.sls:

```
include:
  - moosefs

{% for mnt in salt['cmd.run']('ls /dev/data/moose*').split() %}
/mnt/moose{{ mnt[-1] }}:
  mount.mounted:
    - device: {{ mnt }}
    - fstype: xfs
    - mkmnt: True
  file.directory:
    - user: mfs
    - group: mfs
    - require:
      - user: mfs
      - group: mfs
{% endfor %}

/etc/mfshdd.cfg:
  file.managed:
    - source: salt://moosefs/mfshdd.cfg
    - user: root
    - group: root
    - mode: 644
    - template: jinja
    - require:
      - pkg: mfs-chunkserver

/etc/mfschunkserver.cfg:
  file.managed:
    - source: salt://moosefs/mfschunkserver.cfg
    - user: root
    - group: root
    - mode: 644
    - template: jinja
    - require:
      - pkg: mfs-chunkserver

mfs-chunkserver:
  pkg.installed: []
mfschunkserver:
  service.running:
    - require:
{% for mnt in salt['cmd.run']('ls /dev/data/moose*') %}
      - mount: /mnt/moose{{ mnt[-1] }}
      - file: /mnt/moose{{ mnt[-1] }}
{% endfor %}
      - file: /etc/mfschunkserver.cfg
      - file: /etc/mfshdd.cfg
      - file: /var/lib/mfs
```

This example shows much more of the available power of Jinja. Multiple for loops are used to dynamically detect available hard drives and set them up to be mounted, and the `salt` object is used multiple times to call shell commands to gather data.

Introducing the Python, PyDSL, and the Pyobjects Renderers

Sometimes the chosen default renderer might not have enough logical power to accomplish the needed task. When this happens, the Python renderer can be used. Normally a YAML renderer should be used for the majority of SLS files, but an SLS file set to use another renderer can be easily added to the tree.

This example shows a very basic Python SLS file:

`python/django.sls:`

```
#!py

def run():
    '''
    Install the django package
    '''
    return {'include': ['python'],
            'django': {'pkg': ['installed']}}
```

This is a very simple example; the first line has an SLS shebang that tells Salt to not use the default renderer, but to use the `py` renderer. Then the run function is defined, the return value from the run function must be a Salt friendly data structure, or better known as a Salt *HighState data structure*.

Alternatively, using the *pydsl* renderer, the above example can be written more succinctly as:

```
#!pydsl

include('python', delayed=True)
state('django').pkg.installed()
```

The *pyobjects* renderer provides an ``Pythonic'' object based approach for building the state data. The above example could be written as:

```
#!pyobjects

include('python')
Pkg.installed("django")
```

This Python examples would look like this if they were written in YAML:

```
include:
  - python

django:
  pkg.installed
```

This example clearly illustrates that; one, using the YAML renderer by default is a wise decision and two, unbridled power can be obtained where needed by using a pure Python SLS.

Running and debugging salt states.

Once the rules in an SLS are ready, they should be tested to ensure they work properly. To invoke these rules, simply execute `salt '*' state.highstate` on the command line. If you get back only hostnames with a `:` after, but no return, chances are there is a problem with one or more of the sls files. On the minion, use the `salt-call` command: `salt-call state.highstate -l debug` to examine the output for errors. This should help troubleshoot the issue. The minions can also be started in the foreground in debug mode: `salt-minion -l debug`.

Next Reading

With an understanding of states, the next recommendation is to become familiar with Salt's pillar interface:

Pillar Walkthrough

3.3.2 States tutorial, part 1 - Basic Usage

The purpose of this tutorial is to demonstrate how quickly you can configure a system to be managed by Salt States. For detailed information about the state system please refer to the full *states reference*.

This tutorial will walk you through using Salt to configure a minion to run the Apache HTTP server and to ensure the server is running.

Before continuing make sure you have a working Salt installation by following the *installation* and the *configuration* instructions.

Stuck?

There are many ways to *get help from the Salt community* including our mailing list and our IRC channel #salt.

Setting up the Salt State Tree

States are stored in text files on the master and transferred to the minions on demand via the master's File Server. The collection of state files make up the `State Tree`.

To start using a central state system in Salt, the Salt File Server must first be set up. Edit the master config file (`file_roots`) and uncomment the following lines:

```
file_roots:
  base:
    - /srv/salt
```

Note: If you are deploying on FreeBSD via ports, the `file_roots` path defaults to `/usr/local/etc/salt/states`.

Restart the Salt master in order to pick up this change:

```
pkill salt-master
salt-master -d
```

Preparing the Top File

On the master, in the directory uncommented in the previous step, (`/srv/salt` by default), create a new file called `top.sls` and add the following:

```
base:
  '*':
    - webserver
```

The *top file* is separated into environments (discussed later). The default environment is `base`. Under the `base` environment a collection of minion matches is defined; for now simply specify all hosts (`*`).

Targeting minions

The expressions can use any of the targeting mechanisms used by Salt — minions can be matched by glob, PCRE regular expression, or by *grains*. For example:

```
base:
  'os:Fedora':
    - match: grain
    - webserver
```

Create an `sls` file

In the same directory as the *top file*, create a file named `webserver.sls`, containing the following:

```
apache:                 # ID declaration
  pkg:                  # state declaration
    - installed         # function declaration
```

The first line, called the *ID declaration*, is an arbitrary identifier. In this case it defines the name of the package to be installed.

Note: The package name for the Apache httpd web server may differ depending on OS or distro — for example, on Fedora it is `httpd` but on Debian/Ubuntu it is `apache2`.

The second line, called the *State declaration*, defines which of the Salt States we are using. In this example, we are using the `pkg state` to ensure that a given package is installed.

The third line, called the *Function declaration*, defines which function in the `pkg state` module to call.

Renderers

States `sls` files can be written in many formats. Salt requires only a simple data structure and is not concerned with how that data structure is built. Templating languages and DSLs are a dime-a-dozen and everyone has a favorite.

Building the expected data structure is the job of Salt *renderers* and they are dead-simple to write.

In this tutorial we will be using YAML in Jinja2 templates, which is the default format. The default can be changed by editing `renderer` in the master configuration file.

Install the package

Next, let's run the state we created. Open a terminal on the master and run:

```
% salt '*' state.highstate
```

Our master is instructing all targeted minions to run `state.highstate`. When a minion executes a highstate call it will download the *top file* and attempt to match the expressions. When it does match an expression the modules listed for it will be downloaded, compiled, and executed.

Once completed, the minion will report back with a summary of all actions taken and all changes made.

Warning: If you have created *custom grain modules*, they will not be available in the top file until after the first *highstate*. To make custom grains available on a minion's first highstate, it is recommended to use *this example* to ensure that the custom grains are synced when the minion starts.

SLS File Namespace

Note that in the *example* above, the SLS file `webserver.sls` was referred to simply as `webserver`. The namespace for SLS files follows a few simple rules:

1. The `.sls` is discarded (i.e. `webserver.sls` becomes `webserver`).
2. **Subdirectories can be used for better organization**.
    (a) Each subdirectory is represented by a dot.
    (b) `webserver/dev.sls` is referred to as `webserver.dev`.
3. A file called `init.sls` in a subdirectory is referred to by the path of the directory. So, `webserver/init.sls` is referred to as `webserver`.
4. If both `webserver.sls` and `webserver/init.sls` happen to exist, `webserver/init.sls` will be ignored and `webserver.sls` will be the file referred to as `webserver`.

Troubleshooting Salt

If the expected output isn't seen, the following tips can help to narrow down the problem.

Turn up logging Salt can be quite chatty when you change the logging setting to `debug`:

```
salt-minion -l debug
```

Run the minion in the foreground By not starting the minion in daemon mode (`-d`) one can view any output from the minion as it works:

```
salt-minion &
```

Increase the default timeout value when running **`salt`**. For example, to change the default timeout to 60 seconds:

```
salt -t 60
```

For best results, combine all three:

```
salt-minion -l debug &          # On the minion
salt '*' state.highstate -t 60  # On the master
```

Next steps

This tutorial focused on getting a simple Salt States configuration working. *Part 2* will build on this example to cover more advanced `sls` syntax and will explore more of the states that ship with Salt.

3.3.3 States tutorial, part 2 - More Complex States, Requisites

---

Note: This tutorial builds on topics covered in *part 1*. It is recommended that you begin there.

In the *last part* of the Salt States tutorial we covered the basics of installing a package. We will now modify our `webserver.sls` file to have requirements, and use even more Salt States.

Call multiple States

You can specify multiple *State declaration* under an *ID declaration*. For example, a quick modification to our `webserver.sls` to also start Apache if it is not running:

```
apache:
  pkg.installed: []
  service.running:
    - require:
      - pkg: apache
```

Try stopping Apache before running `state.highstate` once again and observe the output.

Require other states

We now have a working installation of Apache so let's add an HTML file to customize our website. It isn't exactly useful to have a website without a webserver so we don't want Salt to install our HTML file until Apache is installed and running. Include the following at the bottom of your `webserver/init.sls` file:

```
1  apache:
2    pkg.installed: []
3    service.running:
4      - require:
5        - pkg: apache
6
7  /var/www/index.html:                       # ID declaration
8    file:                                    # state declaration
9      - managed                              # function
10     - source: salt://webserver/index.html  # function arg
11     - require:                             # requisite declaration
12       - pkg: apache                        # requisite reference
```

line 9 is the *ID declaration*. In this example it is the location we want to install our custom HTML file. (**Note**: the default location that Apache serves may differ from the above on your OS or distro. `/srv/www` could also be a likely place to look.)

Line 10 the *State declaration*. This example uses the Salt `file state`.

Line 11 is the *Function declaration*. The `managed function` will download a file from the master and install it in the location specified.

Line 12 is a *Function arg declaration* which, in this example, passes the `source` argument to the `managed function`.

Line 13 is a *Requisite declaration*.

Line 14 is a *Requisite reference* which refers to a state and an ID. In this example, it is referring to the `ID declaration` from our example in *part 1*. This declaration tells Salt not to install the HTML file until Apache is installed.

Next, create the `index.html` file and save it in the `webserver` directory:

```
<html>
    <head><title>Salt rocks</title></head>
    <body>
        <h1>This file brought to you by Salt</h1>
    </body>
</html>
```

Last, call `state.highstate` again and the minion will fetch and execute the highstate as well as our HTML file from the master using Salt's File Server:

```
salt '*' state.highstate
```

Verify that Apache is now serving your custom HTML.

`require` vs. `watch`

There are two *Requisite declaration*, "require", and "watch". Not every state supports "watch". The `service state` does support "watch" and will restart a service based on the watch condition.

For example, if you use Salt to install an Apache virtual host configuration file and want to restart Apache whenever that file is changed you could modify our Apache example from earlier as follows:

```
/etc/httpd/extra/httpd-vhosts.conf:
  file.managed:
    - source: salt://webserver/httpd-vhosts.conf

apache:
  pkg.installed: []
  service.running:
    - watch:
      - file: /etc/httpd/extra/httpd-vhosts.conf
    - require:
      - pkg: apache
```

If the pkg and service names differ on your OS or distro of choice you can specify each one separately using a *Name declaration* which explained in *Part 3*.

Next steps

In *part 3* we will discuss how to use includes, extends, and templating to make a more complete State Tree configuration.

3.3.4 States tutorial, part 3 - Templating, Includes, Extends

Note: This tutorial builds on topics covered in *part 1* and *part 2*. It is recommended that you begin there.

This part of the tutorial will cover more advanced templating and configuration techniques for `sls` files.

Templating SLS modules

SLS modules may require programming logic or inline execution. This is accomplished with module templating. The default module templating system used is Jinja2 and may be configured by changing the `renderer` value in the master config.

All states are passed through a templating system when they are initially read. To make use of the templating system, simply add some templating markup. An example of an sls module with templating markup may look like this:

```
{% for usr in ['moe','larry','curly'] %}
{{ usr }}:
  user.present
{% endfor %}
```

This templated sls file once generated will look like this:

```
moe:
  user.present
larry:
  user.present
```

```
curly:
  user.present
```

Here's a more complex example:

```
{% for usr in 'moe','larry','curly' %}
{{ usr }}:
  group:
    - present
  user:
    - present
    - gid_from_name: True
    - require:
      - group: {{ usr }}
{% endfor %}
```

Using Grains in SLS modules

Often times a state will need to behave differently on different systems. *Salt grains* objects are made available in the template context. The *grains* can be used from within sls modules:

```
apache:
  pkg.installed:
    {% if grains['os'] == 'RedHat' %}
    - name: httpd
    {% elif grains['os'] == 'Ubuntu' %}
    - name: apache2
    {% endif %}
```

Calling Salt modules from templates

All of the Salt modules loaded by the minion are available within the templating system. This allows data to be gathered in real time on the target system. It also allows for shell commands to be run easily from within the sls modules.

The Salt module functions are also made available in the template context as `salt:`

```
moe:
  user.present:
    - gid: {{ salt['file.group_to_gid']('some_group_that_exists') }}
```

Note that for the above example to work, `some_group_that_exists` must exist before the state file is processed by the templating engine.

Below is an example that uses the `network.hw_addr` function to retrieve the MAC address for eth0:

```
salt['network.hw_addr']('eth0')
```

Advanced SLS module syntax

Lastly, we will cover some incredibly useful techniques for more complex State trees.

Include declaration

A previous example showed how to spread a Salt tree across several files. Similarly, *requisites* span multiple files by using an *Include declaration*. For example:

`python/python-libs.sls:`

```
python-dateutil:
  pkg.installed
```

`python/django.sls:`

```
include:
  - python.python-libs

django:
  pkg.installed:
    - require:
      - pkg: python-dateutil
```

Extend declaration

You can modify previous declarations by using an *Extend declaration*. For example the following modifies the Apache tree to also restart Apache when the vhosts file is changed:

`apache/apache.sls:`

```
apache:
  pkg.installed
```

`apache/mywebsite.sls:`

```
include:
  - apache.apache

extend:
  apache:
    service:
      - running
      - watch:
        - file: /etc/httpd/extra/httpd-vhosts.conf

/etc/httpd/extra/httpd-vhosts.conf:
  file.managed:
    - source: salt://apache/httpd-vhosts.conf
```

Using extend with require or watch

The `extend` statement works differently for `require` or `watch`. It appends to, rather than replacing the requisite component.

Name declaration

You can override the *ID declaration* by using a *Name declaration*. For example, the previous example is a bit more maintainable if rewritten as follows:

apache/mywebsite.sls:

```
include:
  - apache.apache

extend:
  apache:
    service:
      - running
      - watch:
        - file: mywebsite

mywebsite:
  file.managed:
    - name: /etc/httpd/extra/httpd-vhosts.conf
    - source: salt://apache/httpd-vhosts.conf
```

Names declaration

Even more powerful is using a *Names declaration* to override the *ID declaration* for multiple states at once. This often can remove the need for looping in a template. For example, the first example in this tutorial can be rewritten without the loop:

```
stooges:
  user.present:
    - names:
      - moe
      - larry
      - curly
```

Next steps

In *part 4* we will discuss how to use salt's `file_roots` to set up a workflow in which states can be ``promoted'' from dev, to QA, to production.

3.3.5 States tutorial, part 4

Note: This tutorial builds on topics covered in *part 1*, *part 2* and *part 3*. It is recommended that you begin there.

This part of the tutorial will show how to use salt's `file_roots` to set up a workflow in which states can be ``promoted'' from dev, to QA, to production.

Salt fileserver path inheritance

Salt's fileserver allows for more than one root directory per environment, like in the below example, which uses both a local directory and a secondary location shared to the salt master via NFS:

```
# In the master config file (/etc/salt/master)
file_roots:
  base:
    - /srv/salt
    - /mnt/salt-nfs/base
```

Salt's fileserver collapses the list of root directories into a single virtual environment containing all files from each root. If the same file exists at the same relative path in more than one root, then the top-most match ``wins''. For example, if `/srv/salt/foo.txt` and `/mnt/salt-nfs/base/foo.txt` both exist, then `salt://foo.txt` will point to `/srv/salt/foo.txt`.

Note: When using multiple fileserver backends, the order in which they are listed in the `fileserver_backend` parameter also matters. If both `roots` and `git` backends contain a file with the same relative path, and `roots` appears before `git` in the `fileserver_backend` list, then the file in `roots` will ``win'', and the file in gitfs will be ignored.

A more thorough explanation of how Salt's modular fileserver works can be found *here*. We recommend reading this.

Environment configuration

Configure a multiple-environment setup like so:

```
file_roots:
  base:
    - /srv/salt/prod
  qa:
    - /srv/salt/qa
    - /srv/salt/prod
  dev:
    - /srv/salt/dev
    - /srv/salt/qa
    - /srv/salt/prod
```

Given the path inheritance described above, files within `/srv/salt/prod` would be available in all environments. Files within `/srv/salt/qa` would be available in both `qa`, and `dev`. Finally, the files within `/srv/salt/dev` would only be available within the `dev` environment.

Based on the order in which the roots are defined, new files/states can be placed within `/srv/salt/dev`, and pushed out to the dev hosts for testing.

Those files/states can then be moved to the same relative path within `/srv/salt/qa`, and they are now available only in the `dev` and `qa` environments, allowing them to be pushed to QA hosts and tested.

Finally, if moved to the same relative path within `/srv/salt/prod`, the files are now available in all three environments.

Practical Example

As an example, consider a simple website, installed to `/var/www/foobarcom`. Below is a top.sls that can be used to deploy the website:

`/srv/salt/prod/top.sls:`

```
base:
  'web*prod*':
    - webserver.foobarcom
qa:
  'web*qa*':
    - webserver.foobarcom
dev:
  'web*dev*':
    - webserver.foobarcom
```

Using pillar, roles can be assigned to the hosts:

`/srv/pillar/top.sls:`

```
base:
  'web*prod*':
    - webserver.prod
  'web*qa*':
    - webserver.qa
  'web*dev*':
    - webserver.dev
```

`/srv/pillar/webserver/prod.sls:`

```
webserver_role: prod
```

`/srv/pillar/webserver/qa.sls:`

```
webserver_role: qa
```

`/srv/pillar/webserver/dev.sls:`

```
webserver_role: dev
```

And finally, the SLS to deploy the website:

`/srv/salt/prod/webserver/foobarcom.sls:`

```
{% if pillar.get('webserver_role', '') %}
/var/www/foobarcom:
  file.recurse:
    - source: salt://webserver/src/foobarcom
    - env: {{ pillar['webserver_role'] }}
    - user: www
    - group: www
    - dir_mode: 755
    - file_mode: 644
{% endif %}
```

Given the above SLS, the source for the website should initially be placed in `/srv/salt/dev/webserver/src/foobarcom`.

First, let's deploy to dev. Given the configuration in the top file, this can be done using `state.highstate`:

```
salt --pillar 'webserver_role:dev' state.highstate
```

However, in the event that it is not desirable to apply all states configured in the top file (which could be likely in more complex setups), it is possible to apply just the states for the `foobarcom` website, using `state.sls`:

```
salt --pillar 'webserver_role:dev' state.sls webserver.foobarcom
```

Once the site has been tested in dev, then the files can be moved from `/srv/salt/dev/webserver/src/foobarcom` to `/srv/salt/qa/webserver/src/foobarcom`, and deployed using the following:

```
salt --pillar 'webserver_role:qa' state.sls webserver.foobarcom
```

Finally, once the site has been tested in qa, then the files can be moved from `/srv/salt/qa/webserver/src/foobarcom` to `/srv/salt/prod/webserver/src/foobarcom`, and deployed using the following:

```
salt --pillar 'webserver_role:prod' state.sls webserver.foobarcom
```

Thanks to Salt's fileserver inheritance, even though the files have been moved to within `/srv/salt/prod`, they are still available from the same `salt://` URI in both the qa and dev environments.

Continue Learning

The best way to continue learning about Salt States is to read through the *reference documentation* and to look through examples of existing state trees. Many pre-configured state trees can be found on Github in the saltstack-formulas collection of repositories.

If you have any questions, suggestions, or just want to chat with other people who are using Salt, we have a very *active community* and we'd love to hear from you.

In addition, by continuing to *part 5*, you can learn about the powerful orchestration of which Salt is capable.

3.3.6 States Tutorial, Part 5 - Orchestration with Salt

Note: This tutorial builds on some of the topics covered in the earlier *States Walkthrough* pages. It is recommended to start with *Part 1* if you are not familiar with how to use states.

Orchestration is accomplished in salt primarily through the *Orchestrate Runner*. Added in version 0.17.0, this Salt *Runner* can use the full suite of *requisites* available in states, and can also execute states/functions using salt-ssh. This runner replaces the *OverState*.

The Orchestrate Runner

New in version 0.17.0.

As noted above in the introduction, the Orchestrate Runner (originally called the state.sls runner) offers all the functionality of the OverState, but with a couple advantages:

- All *requisites* available in states can be used.
- The states/functions can be executed using salt-ssh.

The Orchestrate Runner was added with the intent to eventually deprecate the OverState system, however the OverState will still be maintained for the foreseeable future.

Configuration Syntax

The configuration differs slightly from that of the OverState, and more closely resembles the configuration schema used for states.

To execute a state, use `salt.state`:

```
install_nginx:
  salt.state:
    - tgt: 'web*'
    - sls:
      - nginx
```

To execute a function, use `salt.function`:

```
cmd.run:
  salt.function:
    - tgt: '*'
    - arg:
      - rm -rf /tmp/foo
```

Triggering a Highstate

Whereas with the OverState, a Highstate is run by simply omitting an `sls` or `function` argument, with the Orchestrate Runner the Highstate must explicitly be requested by using `highstate: True`:

```
webserver_setup:
  salt.state:
    - tgt: 'web*'
    - highstate: True
```

Executing the Orchestrate Runner

The Orchestrate Runner can be executed using the `state.orchestrate` runner function. `state.orch` also works, for those that would like to type less.

Assuming that your `base` environment is located at `/srv/salt`, and you have placed a configuration file in `/srv/salt/orchestration/webserver.sls`, then the following could both be used:

```
salt-run state.orchestrate orchestration.webserver
salt-run state.orch orchestration.webserver
```

Changed in version 2014.1.1: The runner function was renamed to `state.orchestrate`. In versions 0.17.0 through 2014.1.0, `state.sls` must be used. This was renamed to avoid confusion with the `state.sls` execution function.

```
salt-run state.sls orchestration.webserver
```

More Complex Orchestration

Many states/functions can be configured in a single file, which when combined with the full suite of *requisites*, can be used to easily configure complex orchestration tasks. Additionally, the states/functions will be executed in the order in which they are defined, unless prevented from doing so by any *requisites*, as is the default in SLS files since 0.17.0.

```
cmd.run:
  salt.function:
    - tgt: 10.0.0.0/24
    - tgt_type: ipcidr
    - arg:
      - bootstrap

storage_setup:
  salt.state:
    - tgt: 'role:storage'
    - tgt_type: grain
    - sls: ceph
    - require:
      - salt: webserver_setup
```

```
webserver_setup:
  salt.state:
    - tgt: 'web*'
    - highstate: True
```

Given the above setup, the orchestration will be carried out as follows:

1. The shell command `bootstrap` will be executed on all minions in the 10.0.0.0/24 subnet.
2. A Highstate will be run on all minions whose ID starts with ``web'', since the `storage_setup` state requires it.
3. Finally, the `ceph` SLS target will be executed on all minions which have a grain called `role` with a value of `storage`.

The OverState System

Warning: The OverState runner is deprecated, and will be removed in the feature release of Salt codenamed Boron. (Three feature releases after 2014.7.0, which is codenamed Helium)

Often, servers need to be set up and configured in a specific order, and systems should only be set up if systems earlier in the sequence have been set up without any issues.

The OverState system can be used to orchestrate deployment in a smooth and reliable way across multiple systems in small to large environments.

The OverState SLS

The OverState system is managed by an SLS file named `overstate.sls`, located in the root of a Salt fileserver environment.

The overstate.sls configures an unordered list of stages, each stage defines the minions on which to execute the state, and can define what sls files to run, execute a `state.highstate`, or execute a function. Here's a sample `overstate.sls`:

```
mysql:
  match: 'db*'
  sls:
    - mysql.server
    - drbd
webservers:
  match: 'web*'
  require:
    - mysql
all:
  match: '*'
  require:
    - mysql
    - webservers
```

Note: The `match` argument uses *compound matching*

Given the above setup, the OverState will be carried out as follows:

1. The `mysql` stage will be executed first because it is required by the `webservers` and `all` stages. It will execute `state.sls` once for each of the two listed SLS targets (`mysql.server` and `drbd`). These states will be executed on all minions whose minion ID starts with ``db''.
2. The `webservers` stage will then be executed, but only if the `mysql` stage executes without any failures. The `webservers` stage will execute a `state.highstate` on all minions whose minion IDs start with ``web''.
3. Finally, the `all` stage will execute, running `state.highstate` on all systems, if, and only if the `mysql` and `webservers` stages completed without any failures.

Any failure in the above steps would cause the requires to fail, preventing the dependent stages from executing.

Using Functions with OverState

In the above example, you'll notice that the stages lacking an `sls` entry run a `state.highstate`. As mentioned earlier, it is also possible to execute other functions in a stage. This functionality was added in version 0.15.0.

Running a function is easy:

```
http:
  function:
    pkg.install:
      - httpd
```

The list of function arguments are defined after the declared function. So, the above stage would run `pkg.install http`. Requisites only function properly if the given function supports returning a custom return code.

Executing an OverState

Since the OverState is a *Runner*, it is executed using the `salt-run` command. The runner function for the OverState is `state.over`.

```
salt-run state.over
```

The function will by default look in the root of the `base` environment (as defined in `file_roots`) for a file called `overstate.sls`, and then execute the stages defined within that file.

Different environments and paths can be used as well, by adding them as positional arguments:

```
salt-run state.over dev /root/other-overstate.sls
```

The above would run an OverState using the `dev` fileserver environment, with the stages defined in `/root/other-overstate.sls`.

Warning: Since these are positional arguments, when defining the path to the overstate file the environment must also be specified, even if it is the `base` environment.

Note: Remember, salt-run is always executed on the master.

3.3.7 Syslog-ng usage

Overview

Syslog_ng state module is for generating syslog-ng configurations. You can do the following things:

- generate syslog-ng configuration from YAML,
- use non-YAML configuration,
- start, stop or reload syslog-ng.

There is also an execution module, which can check the syntax of the configuration, get the version and other information about syslog-ng.

Configuration

Users can create syslog-ng configuration statements with the `syslog_ng.config` function. It requires a *name* and a *config* parameter. The *name* parameter determines the name of the generated statement and the *config* parameter holds a parsed YAML structure.

A statement can be declared in the following forms (both are equivalent):

```
source.s_localhost:
  syslog_ng.config:
    - config:
        - tcp:
          - ip: "127.0.0.1"
          - port: 1233
```

```
s_localhost:
  syslog_ng.config:
    - config:
        source:
          - tcp:
            - ip: "127.0.0.1"
            - port: 1233
```

The first one is called short form, because it needs less typing. Users can use lists and dictionaries to specify their configuration. The format is quite self describing and there are more examples [at the end](#examples) of this document.

Quotation

The quotation can be tricky sometimes but here are some rules to follow:

- when a string meant to be `"string"` in the generated configuration, it should be like `'"string"'` in the YAML document
- similarly, users should write `"'string'"` to get `'string'` in the generated configuration

Full example

The following configuration is an example, how a complete syslog-ng configuration looks like:

```
# Set the location of the configuration file
set_location:
  module.run:
    - name: syslog_ng.set_config_file
    - m_name: "/home/tibi/install/syslog-ng/etc/syslog-ng.conf"

# The syslog-ng and syslog-ng-ctl binaries are here. You needn't use
# this method if these binaries can be found in a directory in your PATH.
```

```
set_bin_path:
  module.run:
    - name: syslog_ng.set_binary_path
    - m_name: "/home/tibi/install/syslog-ng/sbin"

# Writes the first lines into the config file, also erases its previous
# content
write_version:
  module.run:
    - name: syslog_ng.write_version
    - m_name: "3.6"

# There is a shorter form to set the above variables
set_variables:
  module.run:
    - name: syslog_ng.set_parameters
    - version: "3.6"
    - binary_path: "/home/tibi/install/syslog-ng/sbin"
    - config_file: "/home/tibi/install/syslog-ng/etc/syslog-ng.conf"


# Some global options
options.global_options:
  syslog_ng.config:
    - config:
        - time_reap: 30
        - mark_freq: 10
        - keep_hostname: "yes"

source.s_localhost:
  syslog_ng.config:
    - config:
        - tcp:
          - ip: "127.0.0.1"
          - port: 1233

destination.d_log_server:
  syslog_ng.config:
    - config:
        - tcp:
          - "127.0.0.1"
          - port: 1234

log.l_log_to_central_server:
  syslog_ng.config:
    - config:
        - source: s_localhost
        - destination: d_log_server

some_comment:
  module.run:
    - name: syslog_ng.write_config
    - config: |
        # Multi line
        # comment

# An other mode to use comments or existing configuration snippets
config.other_comment_form:
```

```
  syslog_ng.config:
    - config: |
        # Multi line
        # comment
```

The `syslog_ng.reloaded` function can generate syslog-ng configuration from YAML. If the statement (source, destination, parser, etc.) has a name, this function uses the id as the name, otherwise (log statement) it's purpose is like a mandatory comment.

After execution this example the syslog_ng state will generate this file:

```
#Generated by Salt on 2014-08-18 00:11:11
@version: 3.6

options {
    time_reap(
        30
    );
    mark_freq(
        10
    );
    keep_hostname(
        yes
    );
};


source s_localhost {
    tcp(
        ip(
            127.0.0.1
        ),
        port(
            1233
        )
    );
};


destination d_log_server {
    tcp(
        127.0.0.1,
        port(
            1234
        )
    );
};


log {
    source(
        s_localhost
    );
    destination(
        d_log_server
    );
};
```

```
# Multi line
# comment


# Multi line
# comment
```

Users can include arbitrary texts in the generated configuration with using the `config` statement (see the example above).

Syslog_ng module functions

You can use `syslog_ng.set_binary_path` to set the directory which contains the syslog-ng and syslog-ng-ctl binaries. If this directory is in your PATH, you don't need to use this function. There is also a `syslog_ng.set_config_file` function to set the location of the configuration file.

Examples

Simple source

```
source s_tail {
 file(
   "/var/log/apache/access.log",
   follow_freq(1),
   flags(no-parse, validate-utf8)
 );
};
```

```
s_tail:
  # Salt will call the source function of syslog_ng module
  syslog_ng.config:
    - config:
        source:
          - file:
            - file: '"/var/log/apache/access.log"'
            - follow_freq : 1
            - flags:
              - no-parse
              - validate-utf8
```

OR

```
s_tail:
  syslog_ng.config:
    - config:
        source:
            - file:
              - '"/var/log/apache/access.log"'
              - follow_freq : 1
              - flags:
                - no-parse
                - validate-utf8
```

OR

```
source.s_tail:
  syslog_ng.config:
    - config:
        - file:
          - '"/var/log/apache/access.log"'
          - follow_freq : 1
          - flags:
            - no-parse
            - validate-utf8
```

Complex source

```
source s_gsoc2014 {
 tcp(
   ip("0.0.0.0"),
   port(1234),
   flags(no-parse)
 );
};
```

```
s_gsoc2014:
  syslog_ng.config:
    - config:
        source:
          - tcp:
            - ip: 0.0.0.0
            - port: 1234
            - flags: no-parse
```

Filter

```
filter f_json {
 match(
   "@json:"
 );
};
```

```
f_json:
  syslog_ng.config:
    - config:
        filter:
          - match:
            - '"@json:"'
```

Template

```
template t_demo_filetemplate {
 template(
   "$ISODATE $HOST $MSG "
 );
 template_escape(
   no
```

```
 );
};
```

```
t_demo_filetemplate:
  syslog_ng.config:
    -config:
        template:
          - template:
            - '"$ISODATE $HOST $MSG\n"'
          - template_escape:
            - "no"
```

Rewrite

```
rewrite r_set_message_to_MESSAGE {
 set(
   "${.json.message}",
   value("$MESSAGE")
 );
};
```

```
r_set_message_to_MESSAGE:
  syslog_ng.config:
    - config:
        rewrite:
          - set:
            - '"${.json.message}"'
            - value : '"$MESSAGE"'
```

Global options

```
options {
   time_reap(30);
   mark_freq(10);
   keep_hostname(yes);
};
```

```
global_options:
  syslog_ng.config:
    - config:
        options:
          - time_reap: 30
          - mark_freq: 10
          - keep_hostname: "yes"
```

Log

```
log {
 source(s_gsoc2014);
 junction {
  channel {
   filter(f_json);
```

```
   parser(p_json);
   rewrite(r_set_json_tag);
   rewrite(r_set_message_to_MESSAGE);
   destination {
    file(
      "/tmp/json-input.log",
      template(t_gsoc2014)
    );
   };
   flags(final);
  };
  channel {
   filter(f_not_json);
   parser {
    syslog-parser(

    );
   };
   rewrite(r_set_syslog_tag);
   flags(final);
  };
 };
 destination {
  file(
    "/tmp/all.log",
    template(t_gsoc2014)
  );
 };
};
```

```
l_gsoc2014:
  syslog_ng.config:
    - config:
        log:
          - source: s_gsoc2014
          - junction:
            - channel:
              - filter: f_json
              - parser: p_json
              - rewrite: r_set_json_tag
              - rewrite: r_set_message_to_MESSAGE
              - destination:
                - file:
                  - '"/tmp/json-input.log"'
                  - template: t_gsoc2014
              - flags: final
            - channel:
              - filter: f_not_json
              - parser:
                - syslog-parser: []
              - rewrite: r_set_syslog_tag
              - flags: final
          - destination:
            - file:
              - "/tmp/all.log"
              - template: t_gsoc2014
```

3.4 Advanced Topics

3.4.1 SaltStack Walk-through

Note: Welcome to SaltStack! I am excited that you are interested in Salt and starting down the path to better infrastructure management. I developed (and am continuing to develop) Salt with the goal of making the best software available to manage computers of almost any kind. I hope you enjoy working with Salt and that the software can solve your real world needs!

- Thomas S Hatch
- Salt creator and Chief Developer
- CTO of SaltStack, Inc.

Getting Started

What is Salt?

Salt is a different approach to infrastructure management, founded on the idea that high-speed communication with large numbers of systems can open up new capabilities. This approach makes Salt a powerful multitasking system that can solve many specific problems in an infrastructure.

The backbone of Salt is the remote execution engine, which creates a high-speed, secure and bi-directional communication net for groups of systems. On top of this communication system, Salt provides an extremely fast, flexible, and easy-to-use configuration management system called `Salt States`.

Installing Salt

SaltStack has been made to be very easy to install and get started. Setting up Salt should be as easy as installing Salt via distribution packages on Linux or via the Windows installer. The *installation documents* cover platform-specific installation in depth.

Starting Salt

Salt functions on a master/minion topology. A master server acts as a central control bus for the clients, which are called `minions`. The minions connect back to the master.

Setting Up the Salt Master Turning on the Salt Master is easy -- just turn it on! The default configuration is suitable for the vast majority of installations. The Salt Master can be controlled by the local Linux/Unix service manager:

On Systemd based platforms (OpenSuse, Fedora):

```
systemctl start salt-master
```

On Upstart based systems (Ubuntu, older Fedora/RHEL):

```
service salt-master start
```

On SysV Init systems (Debian, Gentoo etc.):

```
/etc/init.d/salt-master start
```

Alternatively, the Master can be started directly on the command-line:

```
salt-master -d
```

The Salt Master can also be started in the foreground in debug mode, thus greatly increasing the command output:

```
salt-master -l debug
```

The Salt Master needs to bind to two TCP network ports on the system. These ports are `4505` and `4506`. For more in depth information on firewalling these ports, the firewall tutorial is available *here*.

Setting up a Salt Minion

Note: The Salt Minion can operate with or without a Salt Master. This walk-through assumes that the minion will be connected to the master, for information on how to run a master-less minion please see the master-less quick-start guide:

Masterless Minion Quickstart

The Salt Minion only needs to be aware of one piece of information to run, the network location of the master.

By default the minion will look for the DNS name `salt` for the master, making the easiest approach to set internal DNS to resolve the name `salt` back to the Salt Master IP.

Otherwise, the minion configuration file will need to be edited so that the configuration option `master` points to the DNS name or the IP of the Salt Master:

Note: The default location of the configuration files is `/etc/salt`. Most platforms adhere to this convention, but platforms such as FreeBSD and Microsoft Windows place this file in different locations.

`/etc/salt/minion:`

```
master: saltmaster.example.com
```

Now that the master can be found, start the minion in the same way as the master; with the platform init system or via the command line directly:

As a daemon:

```
salt-minion -d
```

In the foreground in debug mode:

```
salt-minion -l debug
```

When the minion is started, it will generate an `id` value, unless it has been generated on a previous run and cached in the configuration directory, which is `/etc/salt` by default. This is the name by which the minion will attempt to authenticate to the master. The following steps are attempted, in order to try to find a value that is not `localhost`:

1. The Python function `socket.getfqdn()` is run
2. `/etc/hostname` is checked (non-Windows only)
3. `/etc/hosts` (`%WINDIR%\system32\drivers\etc\hosts` on Windows hosts) is checked for hostnames that map to anything within **127.0.0.0/8**.

If none of the above are able to produce an id which is not `localhost`, then a sorted list of IP addresses on the minion (excluding any within **127.0.0.0/8**) is inspected. The first publicly-routable IP address is used, if there is one. Otherwise, the first privately-routable IP address is used.

If all else fails, then `localhost` is used as a fallback.

Note: Overriding the `id`

The minion id can be manually specified using the `id` parameter in the minion config file. If this configuration value is specified, it will override all other sources for the `id`.

Now that the minion is started, it will generate cryptographic keys and attempt to connect to the master. The next step is to venture back to the master server and accept the new minion's public key.

Using salt-key Salt authenticates minions using public-key encryption and authentication. For a minion to start accepting commands from the master, the minion keys need to be accepted by the master.

The `salt-key` command is used to manage all of the keys on the master. To list the keys that are on the master:

```
salt-key -L
```

The keys that have been rejected, accepted, and pending acceptance are listed. The easiest way to accept the minion key is to accept all pending keys:

```
salt-key -A
```

Note: Keys should be verified! The secure thing to do before accepting a key is to run `salt-key -f minion-id` to print the fingerprint of the minion's public key. This fingerprint can then be compared against the fingerprint generated on the minion.

On the master:

```
# salt-key -f foo.domain.com
Unaccepted Keys:
foo.domain.com:  39:f9:e4:8a:aa:74:8d:52:1a:ec:92:03:82:09:c8:f9
```

On the minion:

```
# salt-call key.finger --local
local:
    39:f9:e4:8a:aa:74:8d:52:1a:ec:92:03:82:09:c8:f9
```

If they match, approve the key with `salt-key -a foo.domain.com`.

Sending the First Commands Now that the minion is connected to the master and authenticated, the master can start to command the minion.

Salt commands allow for a vast set of functions to be executed and for specific minions and groups of minions to be targeted for execution.

The `salt` command is comprised of command options, target specification, the function to execute, and arguments to the function.

A simple command to start with looks like this:

```
salt '*' test.ping
```

The `*` is the target, which specifies all minions.

`test.ping` tells the minion to run the `test.ping` function.

In the case of `test.ping`, `test` refers to a *execution module*. `ping` refers to the `ping` function contained in the aforementioned `test` module.

Note: Execution modules are the workhorses of Salt. They do the work on the system to perform various tasks, such as manipulating files and restarting services.

The result of running this command will be the master instructing all of the minions to execute `test.ping` in parallel and return the result.

This is not an actual ICMP ping, but rather a simple function which returns `True`. Using `test.ping` is a good way of confirming that a minion is connected.

Note: Each minion registers itself with a unique minion ID. This ID defaults to the minion's hostname, but can be explicitly defined in the minion config as well by using the `id` parameter.

Of course, there are hundreds of other modules that can be called just as `test.ping` can. For example, the following would return disk usage on all targeted minions:

```
salt '*' disk.usage
```

Getting to Know the Functions Salt comes with a vast library of functions available for execution, and Salt functions are self-documenting. To see what functions are available on the minions execute the `sys.doc` function:

```
salt '*' sys.doc
```

This will display a very large list of available functions and documentation on them.

Note: Module documentation is also available *on the web*.

These functions cover everything from shelling out to package management to manipulating database servers. They comprise a powerful system management API which is the backbone to Salt configuration management and many other aspects of Salt.

Note: Salt comes with many plugin systems. The functions that are available via the `salt` command are called *Execution Modules*.

Helpful Functions to Know The *cmd* module contains functions to shell out on minions, such as `cmd.run` and `cmd.run_all`:

```
salt '*' cmd.run 'ls -l /etc'
```

The `pkg` functions automatically map local system package managers to the same salt functions. This means that `pkg.install` will install packages via `yum` on Red Hat based systems, `apt` on Debian systems, etc.:

```
salt '*' pkg.install vim
```

Note: Some custom Linux spins and derivatives of other distributions are not properly detected by Salt. If the above command returns an error message saying that `pkg.install` is not available, then you may need to override the pkg provider. This process is explained *here*.

The `network.interfaces` function will list all interfaces on a minion, along with their IP addresses, netmasks, MAC addresses, etc:

```
salt '*' network.interfaces
```

Changing the Output Format The default output format used for most Salt commands is called the `nested` outputter, but there are several other outputters that can be used to change the way the output is displayed. For instance, the `pprint` outputter can be used to display the return data using Python's `pprint` module:

```
root@saltmaster:~# salt myminion grains.item pythonpath --out=pprint
{'myminion': {'pythonpath': ['/usr/lib64/python2.7',
                             '/usr/lib/python2.7/plat-linux2',
                             '/usr/lib64/python2.7/lib-tk',
                             '/usr/lib/python2.7/lib-tk',
                             '/usr/lib/python2.7/site-packages',
                             '/usr/lib/python2.7/site-packages/gst-0.10',
                             '/usr/lib/python2.7/site-packages/gtk-2.0']}}
```

The full list of Salt outputters, as well as example output, can be found *here*.

`salt-call` The examples so far have described running commands from the Master using the `salt` command, but when troubleshooting it can be more beneficial to login to the minion directly and use `salt-call`.

Doing so allows you to see the minion log messages specific to the command you are running (which are *not* part of the return data you see when running the command from the Master using `salt`), making it unnecessary to tail the minion log. More information on `salt-call` and how to use it can be found *here*.

Grains Salt uses a system called *Grains* to build up static data about minions. This data includes information about the operating system that is running, CPU architecture and much more. The grains system is used throughout Salt to deliver platform data to many components and to users.

Grains can also be statically set, this makes it easy to assign values to minions for grouping and managing.

A common practice is to assign grains to minions to specify what the role or roles a minion might be. These static grains can be set in the minion configuration file or via the `grains.setval` function.

Targeting Salt allows for minions to be targeted based on a wide range of criteria. The default targeting system uses globular expressions to match minions, hence if there are minions named `larry1`, `larry2`, `curly1`, and `curly2`, a glob of `larry*` will match `larry1` and `larry2`, and a glob of `*1` will match `larry1` and `curly1`.

Many other targeting systems can be used other than globs, these systems include:

Regular Expressions Target using PCRE-compliant regular expressions

Grains Target based on grains data: *Targeting with Grains*

Pillar Target based on pillar data: *Targeting with Pillar*

IP Target based on IP address/subnet/range

Compound Create logic to target based on multiple targets: *Targeting with Compound*

Nodegroup Target with nodegroups: *Targeting with Nodegroup*

The concepts of targets are used on the command line with Salt, but also function in many other areas as well, including the state system and the systems used for ACLs and user permissions.

Passing in Arguments Many of the functions available accept arguments which can be passed in on the command line:

```
salt '*' pkg.install vim
```

This example passes the argument `vim` to the pkg.install function. Since many functions can accept more complex input then just a string, the arguments are parsed through YAML, allowing for more complex data to be sent on the command line:

```
salt '*' test.echo 'foo: bar'
```

In this case Salt translates the string `foo: bar' into the dictionary ``{`foo': `bar'}''

Note: Any line that contains a newline will not be parsed by YAML.

Salt States

Now that the basics are covered the time has come to evaluate `States`. Salt `States`, or the `State System` is the component of Salt made for configuration management.

The state system is already available with a basic Salt setup, no additional configuration is required. States can be set up immediately.

Note: Before diving into the state system, a brief overview of how states are constructed will make many of the concepts clearer. Salt states are based on data modeling and build on a low level data structure that is used to execute each state function. Then more logical layers are built on top of each other.

The high layers of the state system which this tutorial will cover consists of everything that needs to be known to use states, the two high layers covered here are the *sls* layer and the highest layer *highstate*.

Understanding the layers of data management in the State System will help with understanding states, but they never need to be used. Just as understanding how a compiler functions assists when learning a programming language, understanding what is going on under the hood of a configuration management system will also prove to be a valuable asset.

The First SLS Formula

The state system is built on SLS formulas. These formulas are built out in files on Salt's file server. To make a very basic SLS formula open up a file under /srv/salt named vim.sls. The following state ensures that vim is installed on a system to which that state has been applied.

`/srv/salt/vim.sls:`

```
vim:
  pkg.installed
```

Now install vim on the minions by calling the SLS directly:

```
salt '*' state.sls vim
```

This command will invoke the state system and run the `vim` SLS.

Now, to beef up the vim SLS formula, a `vimrc` can be added:

`/srv/salt/vim.sls:`

```
vim:
  pkg.installed: []

/etc/vimrc:
  file.managed:
```

```
    - source: salt://vimrc
    - mode: 644
    - user: root
    - group: root
```

Now the desired `vimrc` needs to be copied into the Salt file server to `/srv/salt/vimrc`. In Salt, everything is a file, so no path redirection needs to be accounted for. The `vimrc` file is placed right next to the `vim.sls` file. The same command as above can be executed to all the vim SLS formulas and now include managing the file.

Note: Salt does not need to be restarted/reloaded or have the master manipulated in any way when changing SLS formulas. They are instantly available.

Adding Some Depth

Obviously maintaining SLS formulas right in a single directory at the root of the file server will not scale out to reasonably sized deployments. This is why more depth is required. Start by making an nginx formula a better way, make an nginx subdirectory and add an init.sls file:

`/srv/salt/nginx/init.sls:`

```
nginx:
  pkg.installed: []
  service.running:
    - require:
      - pkg: nginx
```

A few concepts are introduced in this SLS formula.

First is the service statement which ensures that the `nginx` service is running.

Of course, the nginx service can't be started unless the package is installed -- hence the `require` statement which sets up a dependency between the two.

The `require` statement makes sure that the required component is executed before and that it results in success.

Note: The *require* option belongs to a family of options called *requisites*. Requisites are a powerful component of Salt States, for more information on how requisites work and what is available see: *Requisites*

Also evaluation ordering is available in Salt as well: *Ordering States*

This new sls formula has a special name -- `init.sls`. When an SLS formula is named `init.sls` it inherits the name of the directory path that contains it. This formula can be referenced via the following command:

```
salt '*' state.sls nginx
```

Note: Reminder!

Just as one could call the `test.ping` or `disk.usage` execution modules, `state.sls` is simply another execution module. It simply takes the name of an SLS file as an argument.

Now that subdirectories can be used, the `vim.sls` formula can be cleaned up. To make things more flexible, move the `vim.sls` and vimrc into a new subdirectory called `edit` and change the `vim.sls` file to reflect the change:

`/srv/salt/edit/vim.sls:`

```
vim:
  pkg.installed

/etc/vimrc:
  file.managed:
    - source: salt://edit/vimrc
    - mode: 644
    - user: root
    - group: root
```

Only the source path to the vimrc file has changed. Now the formula is referenced as `edit.vim` because it resides in the edit subdirectory. Now the edit subdirectory can contain formulas for emacs, nano, joe or any other editor that may need to be deployed.

Next Reading

Two walk-throughs are specifically recommended at this point. First, a deeper run through States, followed by an explanation of Pillar.

1. *Starting States*
2. *Pillar Walkthrough*

An understanding of Pillar is extremely helpful in using States.

Getting Deeper Into States

Two more in-depth States tutorials exist, which delve much more deeply into States functionality.

1. Thomas' original states tutorial, *How Do I Use Salt States?*, covers much more to get off the ground with States.
2. The *States Tutorial* also provides a fantastic introduction.

These tutorials include much more in-depth information including templating SLS formulas etc.

So Much More!

This concludes the initial Salt walk-through, but there are many more things still to learn! These documents will cover important core aspects of Salt:

- *Pillar*
- *Job Management*

A few more tutorials are also available:

- *Remote Execution Tutorial*
- *Standalone Minion*

This still is only scratching the surface, many components such as the reactor and event systems, extending Salt, modular components and more are not covered here. For an overview of all Salt features and documentation, look at the *Table of Contents*.

3.4.2 MinionFS Backend Walkthrough

New in version 2014.1.0.

Sometimes, you might need to propagate files that are generated on a minion. Salt already has a feature to send files from a minion to the master:

```
salt 'minion-id' cp.push /path/to/the/file
```

This command will store the file, including its full path, under `cachedir` `/master/minions/minion-id/files`. With the default `cachedir` the example file above would be stored as */var/cache/salt/master/minions/minion-id/files/path/to/the/file*.

Note: This walkthrough assumes basic knowledge of Salt and `cp.push`. To get up to speed, check out the *walkthrough*.

Since it is not a good idea to expose the whole `cachedir`, MinionFS should be used to send these files to other minions.

Simple Configuration

To use the minionfs backend only two configuration changes are required on the master. The `fileserver_backend` option needs to contain a value of `minion` and `file_recv` needs to be set to true:

```
fileserver_backend:
  - roots
  - minion

file_recv: True
```

These changes require a restart of the master, then new requests for the `salt://minion-id/` protocol will send files that are pushed by `cp.push` from `minion-id` to the master.

Note: All of the files that are pushed to the master are going to be available to all of the minions. If this is not what you want, please remove `minion` from `fileserver_backend` in the master config file.

Note: Having directories with the same name as your minions in the root that can be accessed like `salt://minion-id/` might cause confusion.

Commandline Example

Lets assume that we are going to generate SSH keys on a minion called `minion-source` and put the public part in `~/.ssh/authorized_keys` of root user of a minion called `minion-destination`.

First, lets make sure that `/root/.ssh` exists and has the right permissions:

```
[root@salt-master file]# salt '*' file.mkdir dir_path=/root/.ssh user=root group=root mode=700
minion-source:
    None
minion-destination:
    None
```

We create an RSA key pair without a passphrase [1]:

[1] Yes, that was the actual key on my server, but the server is already destroyed.

```
[root@salt-master file]# salt 'minion-source' cmd.run 'ssh-keygen -N "" -f /root/.ssh/id_rsa'
minion-source:
    Generating public/private rsa key pair.
    Your identification has been saved in /root/.ssh/id_rsa.
    Your public key has been saved in /root/.ssh/id_rsa.pub.
    The key fingerprint is:
    9b:cd:1c:b9:c2:93:8e:ad:a3:52:a0:8b:0a:cc:d4:9b root@minion-source
    The key's randomart image is:
    +--[ RSA 2048]----+
    |                 |
    |                 |
    |                 |
    |  o        .     |
    | o o    S o      |
    |=   +  . B o     |
    |o+ E    B =      |
    |+ .   .+ o       |
    |o  ...ooo        |
    +-----------------+
```

and we send the public part to the master to be available to all minions:

```
[root@salt-master file]# salt 'minion-source' cp.push /root/.ssh/id_rsa.pub
minion-source:
    True
```

now it can be seen by everyone:

```
[root@salt-master file]# salt 'minion-destination' cp.list_master_dirs
minion-destination:
    - .
    - etc
    - minion-source/root
    - minion-source/root/.ssh
```

Lets copy that as the only authorized key to `minion-destination`:

```
[root@salt-master file]# salt 'minion-destination' cp.get_file salt://minion-source/root/.ssh/id_rsa
minion-destination:
    /root/.ssh/authorized_keys
```

Or we can use a more elegant and salty way to add an SSH key:

```
[root@salt-master file]# salt 'minion-destination' ssh.set_auth_key_from_file user=root source=salt:/
minion-destination:
    new
```

3.4.3 Automatic Updates / Frozen Deployments

New in version 0.10.3.d.

Salt has support for the Esky application freezing and update tool. This tool allows one to build a complete zipfile out of the salt scripts and all their dependencies - including shared objects / DLLs.

Getting Started

To build frozen applications, suitable build environment will be needed for each platform. You should probably set up a virtualenv in order to limit the scope of Q/A.

This process does work on Windows. Directions are available at https://github.com/saltstack/salt-windows-install for details on installing Salt in Windows. Only the 32-bit Python and dependencies have been tested, but they have been tested on 64-bit Windows.

Install `bbfreeze`, and then `esky` from PyPI in order to enable the `bdist_esky` command in `setup.py`. Salt itself must also be installed, in addition to its dependencies.

Building and Freezing

Once you have your tools installed and the environment configured, use `setup.py` to prepare the distribution files.

```
python setup.py sdist
python setup.py bdist
```

Once the distribution files are in place, Esky can be used traverse the module tree and pack all the scripts up into a redistributable.

```
python setup.py bdist_esky
```

There will be an appropriately versioned `salt-VERSION.zip` in `dist/` if everything went smoothly.

Windows

`C:\Python27\lib\site-packages\zmq` will need to be added to the PATH variable. This helps bbfreeze find the zmq DLL so it can pack it up.

Using the Frozen Build

Unpack the zip file in the desired install location. Scripts like `salt-minion` and `salt-call` will be in the root of the zip file. The associated libraries and bootstrapping will be in the directories at the same level. (Check the Esky documentation for more information)

To support updating your minions in the wild, put the builds on a web server that the minions can reach. `salt.modules.saltutil.update()` will trigger an update and (optionally) a restart of the minion service under the new version.

Troubleshooting

A Windows minion isn't responding

The process dispatch on Windows is slower than it is on *nix. It may be necessary to add `-t 15' to salt commands to give minions plenty of time to return.

Windows and the Visual Studio Redist

The Visual C++ 2008 32-bit redistributable will need to be installed on all Windows minions. Esky has an option to pack the library into the zipfile, but OpenSSL does not seem to acknowledge the new location. If a no

`OPENSSL_Applink` error appears on the console when trying to start a frozen minion, the redistributable is not installed.

Mixed Linux environments and Yum

The Yum Python module doesn't appear to be available on any of the standard Python package mirrors. If RHEL/CentOS systems need to be supported, the frozen build should created on that platform to support all the Linux nodes. Remember to build the virtualenv with `--system-site-packages` so that the `yum` module is included.

Automatic (Python) module discovery

Automatic (Python) module discovery does not work with the late-loaded scheme that Salt uses for (Salt) modules. Any misbehaving modules will need to be explicitly added to the `freezer_includes` in Salt's `setup.py`. Always check the zipped application to make sure that the necessary modules were included.

3.4.4 Multi Master Tutorial

As of Salt 0.16.0, the ability to connect minions to multiple masters has been made available. The multi-master system allows for redundancy of Salt masters and facilitates multiple points of communication out to minions. When using a multi-master setup, all masters are running hot, and any active master can be used to send commands out to the minions.

Note: If you need failover capabilities with multiple masters, there is also a MultiMaster-PKI setup available, that uses a different topology MultiMaster-PKI with Failover Tutorial

---

In 0.16.0, the masters do not share any information, keys need to be accepted on both masters, and shared files need to be shared manually or use tools like the git fileserver backend to ensure that the `file_roots` are kept consistent.

Summary of Steps

1. Create a redundant master server
2. Copy primary master key to redundant master
3. Start redundant master
4. Configure minions to connect to redundant master
5. Restart minions
6. Accept keys on redundant master

Prepping a Redundant Master

The first task is to prepare the redundant master. If the redundant master is already running, stop it. There is only one requirement when preparing a redundant master, which is that masters share the same private key. When the first master was created, the master's identifying key pair was generated and placed in the master's `pki_dir`. The default location of the master's key pair is `/etc/salt/pki/master/`. Take the private key, `master.pem`, and copy it to the same location on the redundant master. Do the same for the master's public key, `master.pub`. Assuming that no minions have yet been connected to the new redundant master, it is safe to delete any existing key in this location and replace it.

Note: There is no logical limit to the number of redundant masters that can be used.

Once the new key is in place, the redundant master can be safely started.

Configure Minions

Since minions need to be master-aware, the new master needs to be added to the minion configurations. Simply update the minion configurations to list all connected masters:

```
master:
  - saltmaster1.example.com
  - saltmaster2.example.com
```

Now the minion can be safely restarted.

Now the minions will check into the original master and also check into the new redundant master. Both masters are first-class and have rights to the minions.

Note: Minions can automatically detect failed masters and attempt to reconnect to reconnect to them quickly. To enable this functionality, set *master_alive_interval* in the minion config and specify a number of seconds to poll the masters for connection status.

If this option is not set, minions will still reconnect to failed masters but the first command sent after a master comes back up may be lost while the minion authenticates.

Sharing Files Between Masters

Salt does not automatically share files between multiple masters. A number of files should be shared or sharing of these files should be strongly considered.

Minion Keys

Minion keys can be accepted the normal way using **salt-key** on both masters. Keys accepted, deleted, or rejected on one master will NOT be automatically managed on redundant masters; this needs to be taken care of by running salt-key on both masters or sharing the `/etc/salt/pki/master/{minions,minions_pre,minions_rejected}` directories between masters.

Note: While sharing the **/etc/salt/pki/master** directory will work, it is strongly discouraged, since allowing access to the **master.pem** key outside of Salt creates a *SERIOUS* security risk.

File_Roots

The `file_roots` contents should be kept consistent between masters. Otherwise state runs will not always be consistent on minions since instructions managed by one master will not agree with other masters.

The recommended way to sync these is to use a fileserver backend like gitfs or to keep these files on shared storage.

Pillar_Roots

Pillar roots should be given the same considerations as `file_roots`.

Master Configurations

While reasons may exist to maintain separate master configurations, it is wise to remember that each master maintains independent control over minions. Therefore, access controls should be in sync between masters unless a valid reason otherwise exists to keep them inconsistent.

These access control options include but are not limited to:

- external_auth
- client_acl
- peer
- peer_run

3.4.5 Multi-Master-PKI Tutorial With Failover

This tutorial will explain, how to run a salt-environment where a single minion can have multiple masters and fail-over between them if its current master fails.

The individual steps are

- setup the master(s) to sign its auth-replies
- setup minion(s) to verify master-public-keys
- enable multiple masters on minion(s)
- enable master-check on minion(s)

> Please note, that it is advised to have good knowledge of the salt- authentication and communication-process to understand this tutorial. All of the settings described here, go on top of the default authentication/communication process.

Motivation

The default behaviour of a salt-minion is to connect to a master and accept the masters public key. With each publication, the master sends his public-key for the minion to check and if this public-key ever changes, the minion complains and exits. Practically this means, that there can only be a single master at any given time.

Would it not be much nicer, if the minion could have any number of masters (1:n) and jump to the next master if its current master died because of a network or hardware failure?

Note: There is also a MultiMaster-Tutorial with a different approach and topology than this one, that might also suite your needs or might even be better suited Multi-Master Tutorial

---

It is also desirable, to add some sort of authenticity-check to the very first public key a minion receives from a master. Currently a minions takes the first masters public key for granted.

The Goal

Setup the master to sign the public key it sends to the minions and enable the minions to verify this signature for authenticity.

Prepping the master to sign its public key

For signing to work, both master and minion must have the signing and/or verification settings enabled. If the master signs the public key but the minion does not verify it, the minion will complain and exit. The same happens, when the master does not sign but the minion tries to verify.

The easiest way to have the master sign its public key is to set

```
master_sign_pubkey: True
```

After restarting the salt-master service, the master will automatically generate a new key-pair

```
master_sign.pem
master_sign.pub
```

A custom name can be set for the signing key-pair by setting

```
master_sign_key_name: <name_without_suffix>
```

The master will then generate that key-pair upon restart and use it for creating the public keys signature attached to the auth-reply.

The computation is done for every auth-request of a minion. If many minions auth very often, it is advised to use conf_master:*master_pubkey_signature* and conf_master:*master_use_pubkey_signature* settings described below.

If multiple masters are in use and should sign their auth-replies, the signing key-pair master_sign.* has to be copied to each master. Otherwise a minion will fail to verify the masters public when connecting to a different master than it did initially. That is because the public keys signature was created with a different signing key-pair.

Prepping the minion to verify received public keys

The minion must have the public key (and only that one!) available to be able to verify a signature it receives. That public key (defaults to master_sign.pub) must be copied from the master to the minions pki-directory.

```
/etc/salt/pki/minion/master_sign.pub

DO NOT COPY THE master_sign.pem FILE. IT MUST STAY ON THE MASTER AND
ONLY THERE!
```

When that is done, enable the signature checking in the minions configuration

```
verify_master_pubkey_sign: True
```

and restart the minion. For the first try, the minion should be run in manual debug mode.

```
$ salt-minion -l debug
```

Upon connecting to the master, the following lines should appear on the output:

```
[DEBUG   ] Attempting to authenticate with the Salt Master at 172.16.0.10
[DEBUG   ] Loaded minion key: /etc/salt/pki/minion/minion.pem
[DEBUG   ] salt.crypt.verify_signature: Loading public key
[DEBUG   ] salt.crypt.verify_signature: Verifying signature
```

```
[DEBUG   ] Successfully verified signature of master public key with verification public key master_s
[INFO    ] Received signed and verified master pubkey from master 172.16.0.10
[DEBUG   ] Decrypting the current master AES key
```

If the signature verification fails, something went wrong and it will look like this

```
[DEBUG   ] Attempting to authenticate with the Salt Master at 172.16.0.10
[DEBUG   ] Loaded minion key: /etc/salt/pki/minion/minion.pem
[DEBUG   ] salt.crypt.verify_signature: Loading public key
[DEBUG   ] salt.crypt.verify_signature: Verifying signature
[DEBUG   ] Failed to verify signature of public key
[CRITICAL] The Salt Master server's public key did not authenticate!
```

In a case like this, it should be checked, that the verification pubkey (master_sign.pub) on the minion is the same as the one on the master.

Once the verification is successful, the minion can be started in daemon mode again.

For the paranoid among us, its also possible to verify the public whenever it is received from the master. That is, for every single auth-attempt which can be quite frequent. For example just the start of the minion will force the signature to be checked 6 times for various things like auth, mine, highstate, etc.

If that is desired, enable the setting

```
always_verify_signature: True
```

Multiple Masters For A Minion

Configuring multiple masters on a minion is done by specifying two settings:

- a list of masters addresses
- what type of master is defined

```
master:
    - 172.16.0.10
    - 172.16.0.11
    - 172.16.0.12
```

```
master_type: failover
```

This tells the minion that all the master above are available for it to connect to. When started with this configuration, it will try the master in the order they are defined. To randomize that order, set

```
master_shuffle: True
```

The master-list will then be shuffled before the first connection attempt.

The first master that accepts the minion, is used by the minion. If the master does not yet know the minion, that counts as accepted and the minion stays on that master.

For the minion to be able to detect if its still connected to its current master enable the check for it

```
master_alive_interval: <seconds>
```

If the loss of the connection is detected, the minion will temporarily remove the failed master from the list and try one of the other masters defined (again shuffled if that is enabled).

Testing the setup

At least two running masters are needed to test the failover setup.

Both masters should be running and the minion should be running on the command line in debug mode

```
$ salt-minion -l debug
```

The minion will connect to the first master from its master list

```
[DEBUG   ] Attempting to authenticate with the Salt Master at 172.16.0.10
[DEBUG   ] Loaded minion key: /etc/salt/pki/minion/minion.pem
[DEBUG   ] salt.crypt.verify_signature: Loading public key
[DEBUG   ] salt.crypt.verify_signature: Verifying signature
[DEBUG   ] Successfully verified signature of master public key with verification public key master_s
[INFO    ] Received signed and verified master pubkey from master 172.16.0.10
[DEBUG   ] Decrypting the current master AES key
```

A test.ping on the master the minion is currently connected to should be run to test connectivity.

If successful, that master should be turned off. A firewall-rule denying the minions packets will also do the trick.

Depending on the configured conf_minion:*master_alive_interval*, the minion will notice the loss of the connection and log it to its logfile.

```
[INFO    ] Connection to master 172.16.0.10 lost
[INFO    ] Trying to tune in to next master from master-list
```

The minion will then remove the current master from the list and try connecting to the next master

```
[INFO    ] Removing possibly failed master 172.16.0.10 from list of masters
[WARNING ] Master ip address changed from 172.16.0.10 to 172.16.0.11
[DEBUG   ] Attempting to authenticate with the Salt Master at 172.16.0.11
```

If everything is configured correctly, the new masters public key will be verified successfully

```
[DEBUG   ] Loaded minion key: /etc/salt/pki/minion/minion.pem
[DEBUG   ] salt.crypt.verify_signature: Loading public key
[DEBUG   ] salt.crypt.verify_signature: Verifying signature
[DEBUG   ] Successfully verified signature of master public key with verification public key master_s
```

the authentication with the new master is successful

```
[INFO    ] Received signed and verified master pubkey from master 172.16.0.11
[DEBUG   ] Decrypting the current master AES key
[DEBUG   ] Loaded minion key: /etc/salt/pki/minion/minion.pem
[INFO    ] Authentication with master successful!
```

and the minion can be pinged again from its new master.

Performance Tuning

With the setup described above, the master computes a signature for every auth-request of a minion. With many minions and many auth-requests, that can chew up quite a bit of CPU-Power.

To avoid that, the master can use a pre-created signature of its public-key. The signature is saved as a base64 encoded string which the master reads once when starting and attaches only that string to auth-replies.

DO ME HERE Enabling this also gives paranoid users the possibility, to have the signing key-pair on a different system than the actual salt-master and create the public keys signature there. Probably on a system with more restrictive firewall rules, without internet access, less users, etc.

That signature can be created with

```
$ salt-key --gen-signature
```

This will create a default signature file in the master pki-directory

```
/etc/salt/pki/master/master_pubkey_signature
```

It is a simple text-file with the binary-signature converted to base64.

If no signing-pair is present yet, this will auto-create the signing pair and the signature file in one call

```
$ salt-key --gen-signature --auto-create
```

Telling the master to use the pre-created signature is done with

```
master_use_pubkey_signature: True
```

That requires the file `master_pubkey_signature' to be present in the masters pki-directory with the correct signature.

If the signature file is named differently, its name can be set with

```
master_pubkey_signature: <filename>
```

With many masters and many public-keys (default and signing), it is advised to use the salt-masters hostname for the signature-files name. Signatures can be easily confused because they do not provide any information about the key the signature was created from.

Verifying that everything works is done the same way as above.

How the signing and verification works

The default key-pair of the salt-master is

```
/etc/salt/pki/master/master.pem
/etc/salt/pki/master/master.pub
```

To be able to create a signature of a message (in this case a public-key), another key-pair has to be added to the setup. Its default name is:

```
master_sign.pem
master_sign.pub
```

The combination of the master.* and master_sign.* key-pairs give the possibility of generating signatures. The signature of a given message is unique and can be verified, if the public-key of the signing-key-pair is available to the recipient (the minion).

The signature of the masters public-key in master.pub is computed with

```
master_sign.pem
master.pub
M2Crypto.EVP.sign_update()
```

This results in a binary signature which is converted to base64 and attached to the auth-reply send to the minion.

With the signing-pairs public-key available to the minion, the attached signature can be verified with

```
master_sign.pub
master.pub
M2Cryptos EVP.verify_update().
```

When running multiple masters, either the signing key-pair has to be present on all of them, or the master_pubkey_signature has to be pre-computed for each master individually (because they all have different public-keys).

DO NOT PUT THE SAME master.pub ON ALL MASTERS FOR EASE OF USE.

3.4.6 Preseed Minion with Accepted Key

In some situations, it is not convenient to wait for a minion to start before accepting its key on the master. For instance, you may want the minion to bootstrap itself as soon as it comes online. You may also want to to let your developers provision new development machines on the fly.

See also:

Many ways to preseed minion keys

Salt has other ways to generate and pre-accept minion keys in addition to the manual steps outlined below.

salt-cloud performs these same steps automatically when new cloud VMs are created (unless instructed not to).

salt-api exposes an HTTP call to Salt's REST API to `generate and download the new minion keys as a tarball`.

There is a general four step process to do this:

1. Generate the keys on the master:

```
root@saltmaster# salt-key --gen-keys=[key_name]
```

Pick a name for the key, such as the minion's id.

2. Add the public key to the accepted minion folder:

```
root@saltmaster# cp key_name.pub /etc/salt/pki/master/minions/[minion_id]
```

It is necessary that the public key file has the same name as your minion id. This is how Salt matches minions with their keys. Also note that the pki folder could be in a different location, depending on your OS or if specified in the master config file.

3. Distribute the minion keys.

There is no single method to get the keypair to your minion. The difficulty is finding a distribution method which is secure. For Amazon EC2 only, an AWS best practice is to use IAM Roles to pass credentials. (See blog post, http://blogs.aws.amazon.com/security/post/Tx610S2MLVZWEA/Using-IAM-roles-to-distribute-non-AWS-credentials-to-your-EC2-instances)

Security Warning

Since the minion key is already accepted on the master, distributing the private key poses a potential security risk. A malicious party will have access to your entire state tree and other sensitive data if they gain access to a preseeded minion key.

---

4. Preseed the Minion with the keys

You will want to place the minion keys before starting the salt-minion daemon:

```
/etc/salt/pki/minion/minion.pem
/etc/salt/pki/minion/minion.pub
```

Once in place, you should be able to start salt-minion and run `salt-call state.highstate` or any other salt commands that require master authentication.

3.4.7 Salt Bootstrap

The Salt Bootstrap script allows for a user to install the Salt Minion or Master on a variety of system distributions and versions. This shell script known as `bootstrap-salt.sh` runs through a series of checks to determine the operating system type and version. It then installs the Salt binaries using the appropriate methods. The Salt Bootstrap script installs the minimum number of packages required to run Salt. This means that in the event you run the bootstrap to install via package, Git will not be installed. Installing the minimum number of packages helps ensure the script stays as lightweight as possible, assuming the user will install any other required packages after the Salt binaries are present on the system. The script source is available on GitHub: https://github.com/saltstack/salt-bootstrap

Supported Operating Systems

- Amazon Linux 2012.09
- Arch
- CentOS 5/6
- Debian 6.x/7.x/8(git installations only)
- Fedora 17/18
- FreeBSD 9.1/9.2/10
- Gentoo
- Linaro
- Linux Mint 13/14
- OpenSUSE 12.x
- Oracle Linux 5/5
- Red Hat 5/6
- Red Hat Enterprise 5/6
- Scientific Linux 5/6
- SmartOS
- SuSE 11 SP1/11 SP2
- Ubuntu 10.x/11.x/12.x/13.04/13.10
- Elementary OS 0.2

Note: In the event you do not see your distribution or version available please review the develop branch on Github as it main contain updates that are not present in the stable release: https://github.com/saltstack/salt-bootstrap/tree/develop

Example Usage

If you're looking for the *one-liner* to install salt, please scroll to the bottom and use the instructions for Installing via an Insecure One-Liner

---

Note: In every two-step example, you would be well-served to examine the downloaded file and examine it to ensure that it does what you expect.

Using `curl` to install latest git:

```
curl -L https://bootstrap.saltstack.com -o install_salt.sh
sudo sh install_salt.sh git develop
```

Using `wget` to install your distribution's stable packages:

```
wget -O install_salt.sh https://bootstrap.saltstack.com
sudo sh install_salt.sh
```

Install a specific version from git using `wget`:

```
wget -O install_salt.sh https://bootstrap.saltstack.com
sudo sh install_salt.sh -P git v0.16.4
```

If you already have python installed, `python 2.6`, then it's as easy as:

```
python -m urllib "https://bootstrap.saltstack.com" > install_salt.sh
sudo sh install_salt.sh git develop
```

All python versions should support the following one liner:

```
python -c 'import urllib; print urllib.urlopen("https://bootstrap.saltstack.com").read()' > install_s
sudo sh install_salt.sh git develop
```

On a FreeBSD base system you usually don't have either of the above binaries available. You **do** have `fetch` available though:

```
fetch -o install_salt.sh https://bootstrap.saltstack.com
sudo sh install_salt.sh
```

If all you want is to install a `salt-master` using latest git:

```
curl -o install_salt.sh -L https://bootstrap.saltstack.com
sudo sh install_salt.sh -M -N git develop
```

If you want to install a specific release version (based on the git tags):

```
curl -o install_salt.sh -L https://bootstrap.saltstack.com
sudo sh install_salt.sh git v0.16.4
```

To install a specific branch from a git fork:

```
curl -o install_salt.sh -L https://bootstrap.saltstack.com
sudo sh install_salt.sh -g https://github.com/myuser/salt.git git mybranch
```

Installing via an Insecure One-Liner

The following examples illustrate how to install Salt via a one-liner.

Note: Warning! These methods do not involve a verification step and assume that the delivered file is trustworthy.

Examples

Installing the latest develop branch of Salt:

```
curl -L https://bootstrap.saltstack.com | sudo sh -s -- git develop
```

Any of the example above which use two-lines can be made to run in a single-line configuration with minor modifications.

Example Usage

The Salt Bootstrap script has a wide variety of options that can be passed as well as several ways of obtaining the bootstrap script itself.

For example, using `curl` to install your distribution's stable packages:

```
curl -L https://bootstrap.saltstack.com | sudo sh
```

Using `wget` to install your distribution's stable packages:

```
wget -O - https://bootstrap.saltstack.com | sudo sh
```

Installing the latest version available from git with `curl`:

```
curl -L https://bootstrap.saltstack.com | sudo sh -s -- git develop
```

Install a specific version from git using `wget`:

```
wget -O - https://bootstrap.saltstack.com | sh -s -- -P git v0.16.4
```

If you already have python installed, `python 2.6`, then it's as easy as:

```
python -m urllib "https://bootstrap.saltstack.com" | sudo sh -s -- git develop
```

All python versions should support the following one liner:

```
python -c 'import urllib; print urllib.urlopen("https://bootstrap.saltstack.com").read()' | \
sudo  sh -s -- git develop
```

On a FreeBSD base system you usually don't have either of the above binaries available. You **do** have `fetch` available though:

```
fetch -o - https://bootstrap.saltstack.com | sudo sh
```

If all you want is to install a `salt-master` using latest git:

```
curl -L https://bootstrap.saltstack.com | sudo sh -s -- -M -N git develop
```

If you want to install a specific release version (based on the git tags):

```
curl -L https://bootstrap.saltstack.com | sudo sh -s -- git v0.16.4
```

Downloading the develop branch (from here standard command line options may be passed):

```
wget https://bootstrap.saltstack.com/develop
```

Command Line Options

Here's a summary of the command line options:

```
$ sh bootstrap-salt.sh -h

  Usage :  bootstrap-salt.sh [options] <install-type> <install-type-args>

  Installation types:
    - stable (default)
    - daily  (ubuntu specific)
    - git

  Examples:
    $ bootstrap-salt.sh
    $ bootstrap-salt.sh stable
    $ bootstrap-salt.sh daily
    $ bootstrap-salt.sh git
    $ bootstrap-salt.sh git develop
    $ bootstrap-salt.sh git v0.17.0
    $ bootstrap-salt.sh git 8c3fadf15ec183e5ce8c63739850d543617e4357

  Options:
  -h  Display this message
  -v  Display script version
  -n  No colours.
  -D  Show debug output.
  -c  Temporary configuration directory
  -g  Salt repository URL. (default: git://github.com/saltstack/salt.git)
  -k  Temporary directory holding the minion keys which will pre-seed
      the master.
  -M  Also install salt-master
  -S  Also install salt-syndic
  -N  Do not install salt-minion
  -X  Do not start daemons after installation
  -C  Only run the configuration function. This option automatically
      bypasses any installation.
  -P  Allow pip based installations. On some distributions the required salt
      packages or its dependencies are not available as a package for that
      distribution. Using this flag allows the script to use pip as a last
      resort method. NOTE: This only works for functions which actually
      implement pip based installations.
  -F  Allow copied files to overwrite existing(config, init.d, etc)
  -U  If set, fully upgrade the system prior to bootstrapping salt
  -K  If set, keep the temporary files in the temporary directories specified
      with -c and -k.
  -I  If set, allow insecure connections while downloading any files. For
      example, pass '--no-check-certificate' to 'wget' or '--insecure' to 'curl'
  -A  Pass the salt-master DNS name or IP. This will be stored under
      ${BS_SALT_ETC_DIR}/minion.d/99-master-address.conf
  -i  Pass the salt-minion id. This will be stored under
      ${BS_SALT_ETC_DIR}/minion_id
  -L  Install the Apache Libcloud package if possible(required for salt-cloud)
  -p  Extra-package to install while installing salt dependencies. One package
      per -p flag. You're responsible for providing the proper package name.
```

3.4.8 Git Fileserver Backend Walkthrough

Note: This walkthrough assumes basic knowledge of Salt. To get up to speed, check out the *Salt Walkthrough*.

The gitfs backend allows Salt to serve files from git repositories. It can be enabled by adding `git` to the `fileserver_backend` list, and configuring one or more repositories in `gitfs_remotes`.

Branches and tags become Salt fileserver environments.

Installing Dependencies

Beginning with version 2014.7.0, both pygit2 and Dulwich are supported as alternatives to GitPython. The desired provider can be configured using the `gitfs_provider` parameter in the master config file.

If `gitfs_provider` is not configured, then Salt will prefer pygit2 if a suitable version is available, followed by GitPython and Dulwich.

pygit2

The minimum supported version of pygit2 is 0.20.3. Availability for this version of pygit2 is still limited, though the SaltStack team is working to get compatible versions available for as many platforms as possible.

For the Fedora/EPEL versions which have a new enough version packaged, the following command would be used to install pygit2:

```
# yum install python-pygit2
```

Provided a valid version is packaged for Debian/Ubuntu (which is not currently the case), the package name would be the same, and the following command would be used to install it:

```
# apt-get install python-pygit2
```

If pygit2 is not packaged for the platform on which the Master is running, the pygit2 website has installation instructions here. Keep in mind however that following these instructions will install libgit2 and pygit2 without system packages. Additionally, keep in mind that *SSH authentication in pygit2* requires libssh2 (*not* libssh) development libraries to be present before libgit2 is built.

GitPython

GitPython 0.3.0 or newer is required to use GitPython for gitfs. For RHEL-based Linux distros, a compatible version is available in EPEL, and can be easily installed on the master using yum:

```
# yum install GitPython
```

Ubuntu 14.04 LTS and Debian Wheezy (7.x) also have a compatible version packaged:

```
# apt-get install python-git
```

If your master is running an older version (such as Ubuntu 12.04 LTS or Debian Squeeze), then you will need to install GitPython using either pip or easy_install (it is recommended to use pip). Version 0.3.2.RC1 is now marked as the stable release in PyPI, so it should be a simple matter of running `pip install GitPython` (or `easy_install GitPython`) as root.

Warning: Keep in mind that if GitPython has been previously installed on the master using pip (even if it was subsequently uninstalled), then it may still exist in the build cache (typically `/tmp/pip-build-root/GitPython`) if the cache is not cleared after installation. The package in the build cache will override any requirement specifiers, so if you try upgrading to version 0.3.2.RC1 by running `pip install 'GitPython==0.3.2.RC1'` then it will ignore this and simply install the version from the cache directory. Therefore, it may be necessary to delete the GitPython directory from the build cache in order to ensure that the specified version is installed.

Dulwich

Dulwich 0.9.4 or newer is required to use Dulwich as backend for gitfs.

Dulwich is available in EPEL, and can be easily installed on the master using yum:

```
# yum install python-dulwich
```

For APT-based distros such as Ubuntu and Debian:

```
# apt-get install python-dulwich
```

Important: If switching to Dulwich from GitPython/pygit2, or switching from GitPython/pygit2 to Dulwich, it is necessary to clear the gitfs cache to avoid unpredictable behavior. This is probably a good idea whenever switching to a new `gitfs_provider`, but it is less important when switching between GitPython and pygit2.

Beginning in version 2015.5.0, the gitfs cache can be easily cleared using the `fileserver.clear_cache` runner.

```
salt-run fileserver.clear_cache backend=git
```

If the Master is running an earlier version, then the cache can be cleared by removing the `gitfs` and `file_lists/gitfs` directories (both paths relative to the master cache directory, usually `/var/cache/salt/master`).

```
rm -rf /var/cache/salt/master{,/file_lists}/gitfs
```

Simple Configuration

To use the gitfs backend, only two configuration changes are required on the master:

1. Include `git` in the `fileserver_backend` list in the master config file:

```
fileserver_backend:
  - git
```

2. Specify one or more `git://`, `https://`, `file://`, or `ssh://` URLs in `gitfs_remotes` to configure which repositories to cache and search for requested files:

```
gitfs_remotes:
  - https://github.com/saltstack-formulas/salt-formula.git
```

SSH remotes can also be configured using scp-like syntax:

```
gitfs_remotes:
  - git@github.com:user/repo.git
  - ssh://user@domain.tld/path/to/repo.git
```

Information on how to authenticate to SSH remotes can be found *here*.

Note: Dulwich does not recognize `ssh://` URLs, `git+ssh://` must be used instead. Salt version 2015.5.0 and later will automatically add the `git+` to the beginning of these URLs before fetching, but earlier Salt versions will fail to fetch unless the URL is specified using `git+ssh://`.

3. Restart the master to load the new configuration.

Note: In a master/minion setup, files from a gitfs remote are cached once by the master, so minions do not need direct access to the git repository.

Multiple Remotes

The `gitfs_remotes` option accepts an ordered list of git remotes to cache and search, in listed order, for requested files.

A simple scenario illustrates this cascading lookup behavior:

If the `gitfs_remotes` option specifies three remotes:

```
gitfs_remotes:
  - git://github.com/example/first.git
  - https://github.com/example/second.git
  - file:///root/third
```

And each repository contains some files:

```
first.git:
    top.sls
    edit/vim.sls
    edit/vimrc
    nginx/init.sls

second.git:
    edit/dev_vimrc
    haproxy/init.sls

third:
    haproxy/haproxy.conf
    edit/dev_vimrc
```

Salt will attempt to lookup the requested file from each gitfs remote repository in the order in which they are defined in the configuration. The **git://github.com/example/first.git** remote will be searched first. If the requested file is found, then it is served and no further searching is executed. For example:

- A request for the file **salt://haproxy/init.sls** will be served from the **https://github.com/example/second.git** git repo.
- A request for the file **salt://haproxy/haproxy.conf** will be served from the **file:///root/third** repo.

Note: This example is purposefully contrived to illustrate the behavior of the gitfs backend. This example should not be read as a recommended way to lay out files and git repos.

The **file://** prefix denotes a git repository in a local directory. However, it will still use the given **file://** URL as a remote, rather than copying the git repo to the salt cache. This means that any refs you want accessible must exist as *local* refs in the specified repo.

Warning: Salt versions prior to 2014.1.0 are not tolerant of changing the order of remotes or modifying the URI of existing remotes. In those versions, when modifying remotes it is a good idea to remove the gitfs cache directory (`/var/cache/salt/master/gitfs`) before restarting the salt-master service.

Per-remote Configuration Parameters

New in version 2014.7.0.

The following master config parameters are global (that is, they apply to all configured gitfs remotes):

- `gitfs_base`
- `gitfs_root`
- `gitfs_mountpoint` (new in 2014.7.0)
- `gitfs_user` (**pygit2 only**, new in 2014.7.0)
- `gitfs_password` (**pygit2 only**, new in 2014.7.0)
- `gitfs_insecure_auth` (**pygit2 only**, new in 2014.7.0)
- `gitfs_pubkey` (**pygit2 only**, new in 2014.7.0)
- `gitfs_privkey` (**pygit2 only**, new in 2014.7.0)
- `gitfs_passphrase` (**pygit2 only**, new in 2014.7.0)

These parameters can now be overridden on a per-remote basis. This allows for a tremendous amount of customization. Here's some example usage:

```
gitfs_provider: pygit2
gitfs_base: develop

gitfs_remotes:
  - https://foo.com/foo.git
  - https://foo.com/bar.git:
    - root: salt
    - mountpoint: salt://foo/bar/baz
    - base: salt-base
  - http://foo.com/baz.git:
    - root: salt/states
    - user: joe
    - password: mysupersecretpassword
    - insecure_auth: True
```

Important: There are two important distinctions which should be noted for per-remote configuration:

1. The URL of a remote which has per-remote configuration must be suffixed with a colon.
2. Per-remote configuration parameters are named like the global versions, with the `gitfs_` removed from the beginning.

In the example configuration above, the following is true:

1. The first and third gitfs remotes will use the `develop` branch/tag as the `base` environment, while the second one will use the `salt-base` branch/tag as the `base` environment.
2. The first remote will serve all files in the repository. The second remote will only serve files from the `salt` directory (and its subdirectories), while the third remote will only serve files from the `salt/states` directory (and its subdirectories).

3. The files from the second remote will be located under `salt://foo/bar/baz`, while the files from the first and third remotes will be located under the root of the Salt fileserver namespace (`salt://`).
4. The third remote overrides the default behavior of *not authenticating to insecure (non-HTTPS) remotes*.

Serving from a Subdirectory

The `gitfs_root` parameter allows files to be served from a subdirectory within the repository. This allows for only part of a repository to be exposed to the Salt fileserver.

Assume the below layout:

```
.gitignore
README.txt
foo/
foo/bar/
foo/bar/one.txt
foo/bar/two.txt
foo/bar/three.txt
foo/baz/
foo/baz/top.sls
foo/baz/edit/vim.sls
foo/baz/edit/vimrc
foo/baz/nginx/init.sls
```

The below configuration would serve only the files under `foo/baz`, ignoring the other files in the repository:

```
gitfs_remotes:
  - git://mydomain.com/stuff.git

gitfs_root: foo/baz
```

The root can also be configured on a *per-remote basis*.

Mountpoints

New in version 2014.7.0.

The `gitfs_mountpoint` parameter will prepend the specified path to the files served from gitfs. This allows an existing repository to be used, rather than needing to reorganize a repository or design it around the layout of the Salt fileserver.

Before the addition of this feature, if a file being served up via gitfs was deeply nested within the root directory (for example, `salt://webapps/foo/files/foo.conf`, it would be necessary to ensure that the file was properly located in the remote repository, and that all of the the parent directories were present (for example, the directories `webapps/foo/files/` would need to exist at the root of the repository).

The below example would allow for a file `foo.conf` at the root of the repository to be served up from the Salt fileserver path `salt://webapps/foo/files/foo.conf`.

```
gitfs_remotes:
  - https://mydomain.com/stuff.git

gitfs_mountpoint: salt://webapps/foo/files
```

Mountpoints can also be configured on a *per-remote basis*.

Using gitfs Alongside Other Backends

Sometimes it may make sense to use multiple backends; for instance, if `sls` files are stored in git but larger files are stored directly on the master.

The cascading lookup logic used for multiple remotes is also used with multiple backends. If the `fileserver_backend` option contains multiple backends:

```
fileserver_backend:
  - roots
  - git
```

Then the `roots` backend (the default backend of files in `/srv/salt`) will be searched first for the requested file; then, if it is not found on the master, each configured git remote will be searched.

Branches, Environments, and Top Files

When using the gitfs backend, branches, and tags will be mapped to environments using the branch/tag name as an identifier.

There is one exception to this rule: the `master` branch is implicitly mapped to the `base` environment.

So, for a typical `base`, `qa`, `dev` setup, the following branches could be used:

```
master
qa
dev
```

`top.sls` files from different branches will be merged into one at runtime. Since this can lead to overly complex configurations, the recommended setup is to have the `top.sls` file only in the master branch and use environment-specific branches for state definitions.

To map a branch other than `master` as the `base` environment, use the `gitfs_base` parameter.

```
gitfs_base: salt-base
```

The base can also be configured on a *per-remote basis*.

Environment Whitelist/Blacklist

New in version 2014.7.0.

The `gitfs_env_whitelist` and `gitfs_env_blacklist` parameters allow for greater control over which branches/tags are exposed as fileserver environments. Exact matches, globs, and regular expressions are supported, and are evaluated in that order. If using a regular expression, `^` and `$` must be omitted, and the expression must match the entire branch/tag.

```
gitfs_env_whitelist:
  - base
  - v1.*
  - 'mybranch\d+'
```

Note: `v1.*`, in this example, will match as both a glob and a regular expression (though it will have been matched as a glob, since globs are evaluated before regular expressions).

The behavior of the blacklist/whitelist will differ depending on which combination of the two options is used:

- If only `gitfs_env_whitelist` is used, then **only** branches/tags which match the whitelist will be available as environments
- If only `gitfs_env_blacklist` is used, then the branches/tags which match the blacklist will **not** be available as environments
- If both are used, then the branches/tags which match the whitelist, but do **not** match the blacklist, will be available as environments.

Authentication

pygit2

New in version 2014.7.0.

Both HTTPS and SSH authentication are supported as of version 0.20.3, which is the earliest version of pygit2 supported by Salt for gitfs.

Note: The examples below make use of per-remote configuration parameters, a feature new to Salt 2014.7.0. More information on these can be found *here*.

HTTPS For HTTPS repositories which require authentication, the username and password can be provided like so:

```
gitfs_remotes:
  - https://domain.tld/myrepo.git:
    - user: git
    - password: mypassword
```

If the repository is served over HTTP instead of HTTPS, then Salt will by default refuse to authenticate to it. This behavior can be overridden by adding an `insecure_auth` parameter:

```
gitfs_remotes:
  - http://domain.tld/insecure_repo.git:
    - user: git
    - password: mypassword
    - insecure_auth: True
```

SSH SSH repositories can be configured using the `ssh://` protocol designation, or using scp-like syntax. So, the following two configurations are equivalent:

- `ssh://git@github.com/user/repo.git`
- `git@github.com:user/repo.git`

Both `gitfs_pubkey` and `gitfs_privkey` (or their *per-remote counterparts*) must be configured in order to authenticate to SSH-based repos. If the private key is protected with a passphrase, it can be configured using `gitfs_passphrase` (or simply `passphrase` if being configured *per-remote*). For example:

```
gitfs_remotes:
  - git@github.com:user/repo.git:
    - pubkey: /root/.ssh/id_rsa.pub
    - privkey: /root/.ssh/id_rsa
    - passphrase: myawesomepassphrase
```

Finally, the SSH host key must be *added to the known_hosts file*.

GitPython

With GitPython, only passphrase-less SSH public key authentication is supported. **The auth parameters (pubkey, privkey, etc.) shown in the pygit2 authentication examples above do not work with GitPython.**

```
gitfs_remotes:
  - ssh://git@github.com/example/salt-states.git
```

Since GitPython wraps the git CLI, the private key must be located in `~/.ssh/id_rsa` for the user under which the Master is running, and should have permissions of `0600`. Also, in the absence of a user in the repo URL, GitPython will (just as SSH does) attempt to login as the current user (in other words, the user under which the Master is running, usually `root`).

If a key needs to be used, then `~/.ssh/config` can be configured to use the desired key. Information on how to do this can be found by viewing the manpage for `ssh_config`. Here's an example entry which can be added to the `~/.ssh/config` to use an alternate key for gitfs:

```
Host github.com
    IdentityFile /root/.ssh/id_rsa_gitfs
```

The `Host` parameter should be a hostname (or hostname glob) that matches the domain name of the git repository.

It is also necessary to *add the SSH host key to the known_hosts file*. The exception to this would be if strict host key checking is disabled, which can be done by adding `StrictHostKeyChecking no` to the entry in `~/.ssh/config`

```
Host github.com
    IdentityFile /root/.ssh/id_rsa_gitfs
    StrictHostKeyChecking no
```

However, this is generally regarded as insecure, and is not recommended.

Adding the SSH Host Key to the known_hosts File

To use SSH authentication, it is necessary to have the remote repository's SSH host key in the `~/.ssh/known_hosts` file. If the master is also a minion, this can be done using the `ssh.set_known_host` function:

```
# salt mymaster ssh.set_known_host user=root hostname=github.com
mymaster:
    ----------
    new:
        ----------
        enc:
            ssh-rsa
        fingerprint:
            16:27:ac:a5:76:28:2d:36:63:1b:56:4d:eb:df:a6:48
        hostname:
            |1|OiefWWqOD4kwO3BhoIGa0loR5AA=|BIXVtmcTbPER+68HvXmceodDcfI=
        key:
            AAAAB3NzaC1yc2EAAAABIwAAAQEAq2A7hRGmdnm9tUDbO9IDSwBK6TbQa+PXYPCPy6rbTrTtw7PHkccKrpp0yVhp5
    old:
        None
    status:
        updated
```

If not, then the easiest way to add the key is to su to the user (usually `root`) under which the salt-master runs and attempt to login to the server via SSH:

```
$ su
Password:
# ssh github.com
The authenticity of host 'github.com (192.30.252.128)' can't be established.
RSA key fingerprint is 16:27:ac:a5:76:28:2d:36:63:1b:56:4d:eb:df:a6:48.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'github.com,192.30.252.128' (RSA) to the list of known hosts.
Permission denied (publickey).
```

It doesn't matter if the login was successful, as answering `yes` will write the fingerprint to the known_hosts file.

Verifying the Fingerprint To verify that the correct fingerprint was added, it is a good idea to look it up. One way to do this is to use nmap:

```
$ nmap github.com --script ssh-hostkey

Starting Nmap 5.51 ( http://nmap.org ) at 2014-08-18 17:47 CDT
Nmap scan report for github.com (192.30.252.129)
Host is up (0.17s latency).
Not shown: 996 filtered ports
PORT     STATE SERVICE
22/tcp   open  ssh
| ssh-hostkey: 1024 ad:1c:08:a4:40:e3:6f:9c:f5:66:26:5d:4b:33:5d:8c (DSA)
|_2048 16:27:ac:a5:76:28:2d:36:63:1b:56:4d:eb:df:a6:48 (RSA)
80/tcp   open  http
443/tcp  open  https
9418/tcp open  git

Nmap done: 1 IP address (1 host up) scanned in 28.78 seconds
```

Another way is to check one's own known_hosts file, using this one-liner:

```
$ ssh-keygen -l -f /dev/stdin <<<`ssh-keyscan -t rsa github.com 2>/dev/null` | awk '{print $2}'
16:27:ac:a5:76:28:2d:36:63:1b:56:4d:eb:df:a6:48
```

Refreshing gitfs Upon Push

By default, Salt updates the remote fileserver backends every 60 seconds. However, if it is desirable to refresh quicker than that, the *Reactor System* can be used to signal the master to update the fileserver on each push, provided that the git server is also a Salt minion. There are three steps to this process:

1. On the master, create a file **/srv/reactor/update_fileserver.sls**, with the following contents:

```
update_fileserver:
  runner.fileserver.update
```

2. Add the following reactor configuration to the master config file:

```
reactor:
  - 'salt/fileserver/gitfs/update':
    - /srv/reactor/update_fileserver.sls
```

3. On the git server, add a post-receive hook with the following contents:

```
#!/usr/bin/env sh

salt-call event.fire_master update salt/fileserver/gitfs/update
```

The `update` argument right after `event.fire_master` in this example can really be anything, as it represents the data being passed in the event, and the passed data is ignored by this reactor.

Similarly, the tag name `salt/fileserver/gitfs/update` can be replaced by anything, so long as the usage is consistent.

Using Git as an External Pillar Source

Git repositories can also be used to provide *Pillar* data, using the *External Pillar* system. Note that this is different from gitfs, and is not yet at feature parity with it.

To define a git external pillar, add a section like the following to the salt master config file:

```
ext_pillar:
  - git: <branch> <repo> [root=<gitroot>]
```

Changed in version 2014.7.0: The optional `root` parameter was added

The `<branch>` param is the branch containing the pillar SLS tree. The `<repo>` param is the URI for the repository. To add the `master` branch of the specified repo as an external pillar source:

```
ext_pillar:
  - git: master https://domain.com/pillar.git
```

Use the `root` parameter to use pillars from a subdirectory of a git repository:

```
ext_pillar:
  - git: master https://domain.com/pillar.git root=subdirectory
```

More information on the git external pillar can be found in the `salt.pillar.get_pillar docs`.

Why aren't my custom modules/states/etc. syncing to my Minions?

In versions 0.16.3 and older, when using the *git fileserver backend*, certain versions of GitPython may generate errors when fetching, which Salt fails to catch. While not fatal to the fetch process, these interrupt the fileserver update that takes place before custom types are synced, and thus interrupt the sync itself. Try disabling the git fileserver backend in the master config, restarting the master, and attempting the sync again.

This issue is worked around in Salt 0.16.4 and newer.

3.4.9 The MacOS X (Maverick) Developer Step By Step Guide To Salt Installation

This document provides a step-by-step guide to installing a Salt cluster consisting of one master, and one minion running on a local VM hosted on Mac OS X.

Note: This guide is aimed at developers who wish to run Salt in a virtual machine. The official (Linux) walkthrough can be found here.

The 5 Cent Salt Intro

Since you're here you've probably already heard about Salt, so you already know Salt lets you configure and run commands on hordes of servers easily. Here's a brief overview of a Salt cluster:

- Salt works by having a ``master'' server sending commands to one or multiple ``minion'' servers [2]. The master server is the ``command center''. It is going to be the place where you store your configuration files, aka: ``which server is the db, which is the web server, and what libraries and software they should have installed''. The minions receive orders from the master. Minions are the servers actually performing work for your business.
- Salt has two types of configuration files:

  1. the ``salt communication channels'' or ``meta'' or ``config'' configuration files (not official names): one for the master (usually is /etc/salt/master , **on the master server**), and one for minions (default is /etc/salt/minion or /etc/salt/minion.conf, **on the minion servers**). Those files are used to determine things like the Salt Master IP, port, Salt folder locations, etc.. If these are configured incorrectly, your minions will probably be unable to receive orders from the master, or the master will not know which software a given minion should install.

  2. the ``business'' or ``service'' configuration files (once again, not an official name): these are configuration files, ending with ".sls" extension, that describe which software should run on which server, along with particular configuration properties for the software that is being installed. These files should be created in the /srv/salt folder by default, but their location can be changed using ... /etc/salt/master configuration file!

Note: This tutorial contains a third important configuration file, not to be confused with the previous two: the virtual machine provisioning configuration file. This in itself is not specifically tied to Salt, but it also contains some Salt configuration. More on that in step 3. Also note that all configuration files are YAML files. So indentation matters.

Before Digging In, The Architecture Of The Salt Cluster

Salt Master

The ``Salt master'' server is going to be the Mac OS machine, directly. Commands will be run from a terminal app, so Salt will need to be installed on the Mac. This is going to be more convenient for toying around with configuration files.

Salt Minion

We'll only have one ``Salt minion'' server. It is going to be running on a Virtual Machine running on the Mac, using VirtualBox. It will run an Ubuntu distribution.

3.4.10 Step 1 - Configuring The Salt Master On Your Mac

official documentation

Because Salt has a lot of dependencies that are not built in Mac OS X, we will use Homebrew to install Salt. Homebrew is a package manager for Mac, it's great, use it (for this tutorial at least!). Some people spend a lot of time installing libs by hand to better understand dependencies, and then realize how useful a package manager is once they're configuring a brand new machine and have to do it all over again. It also lets you *uninstall* things easily.

Note: Brew is a Ruby program (Ruby is installed by default with your Mac). Brew downloads, compiles, and links software. The linking phase is when compiled software is deployed on your machine. It may conflict with manually installed software, especially in the /usr/local directory. It's ok, remove the manually installed version then refresh the link by typing `brew link 'packageName'`. Brew has a `brew doctor` command that can help you

[2] Salt also works with ``masterless'' configuration where a minion is autonomous (in which case salt can be seen as a local configuration tool), or in ``multiple master'' configuration. See the documentation for more on that.

troubleshoot. It's a great command, use it often. Brew requires xcode command line tools. When you run brew the first time it asks you to install them if they're not already on your system. Brew installs software in /usr/local/bin (system bins are in /usr/bin). In order to use those bins you need your $PATH to search there first. Brew tells you if your $PATH needs to be fixed.

Tip: Use the keyboard shortcut `cmd + shift + period` in the ``open'' Mac OS X dialog box to display hidden files and folders, such as .profile.

Install Homebrew

Install Homebrew here http://brew.sh/ Or just type

```
ruby -e "$(curl -fsSL https://raw.github.com/Homebrew/homebrew/go/install)"
```

Now type the following commands in your terminal (you may want to type `brew doctor` after each to make sure everything's fine):

```
brew install python
brew install swig
brew install zmq
```

Note: zmq is ZeroMQ. It's a fantastic library used for server to server network communication and is at the core of Salt efficiency.

Install Salt

You should now have everything ready to launch this command:

```
pip install salt
```

Note: There should be no need for `sudo pip install salt`. Brew installed Python for your user, so you should have all the access. In case you would like to check, type `which python` to ensure that it's /usr/local/bin/python, and `which pip` which should be /usr/local/bin/pip.

Now type `python` in a terminal then, `import salt`. There should be no errors. Now exit the Python terminal using `exit()`.

Create The Master Configuration

If the default /etc/salt/master configuration file was not created, copy-paste it from here: http://docs.saltstack.com/ref/configuration/examples.html#configuration-examples-master

---

Note: `/etc/salt/master` is a file, not a folder.

Salt Master configuration changes. The Salt master needs a few customization to be able to run on Mac OS X:

```
sudo launchctl limit maxfiles 4096 8192
```

In the /etc/salt/master file, change max_open_files to 8192 (or just add the line: `max_open_files: 8192` (no quote) if it doesn't already exists).

You should now be able to launch the Salt master:

```
sudo salt-master --log-level=all
```

There should be no errors when running the above command.

Note: This command is supposed to be a daemon, but for toying around, we'll keep it running on a terminal to monitor the activity.

Now that the master is set, let's configure a minion on a VM.

3.4.11 Step 2 - Configuring The Minion VM

The Salt minion is going to run on a Virtual Machine. There are a lot of software options that let you run virtual machines on a mac, But for this tutorial we're going to use VirtualBox. In addition to virtualBox, we will use Vagrant, which allows you to create the base VM configuration.

Vagrant lets you build ready to use VM images, starting from an OS image and customizing it using ``provisioners''. In our case, we'll use it to:

- Download the base Ubuntu image
- Install salt on that Ubuntu image (Salt is going to be the ``provisioner'' for the VM).
- Launch the VM
- SSH into the VM to debug
- Stop the VM once you're done.

Install VirtualBox

Go get it here: https://www.virtualBox.org/wiki/Downloads (click on VirtualBox for OS X hosts => x86/amd64)

Install Vagrant

Go get it here: http://downloads.vagrantup.com/ and choose the latest version (1.3.5 at time of writing), then the .dmg file. Double-click to install it. Make sure the `vagrant` command is found when run in the terminal. Type `vagrant`. It should display a list of commands.

Create The Minion VM Folder

Create a folder in which you will store your minion's VM. In this tutorial, it's going to be a minion folder in the $home directory.

```
cd $home
mkdir minion
```

Initialize Vagrant

From the minion folder, type

```
vagrant init
```

This command creates a default Vagrantfile configuration file. This configuration file will be used to pass configuration parameters to the Salt provisioner in Step 3.

Import Precise64 Ubuntu Box

```
vagrant box add precise64 http://files.vagrantup.com/precise64.box
```

Note: This box is added at the global Vagrant level. You only need to do it once as each VM will use this same file.

Modify the Vagrantfile

Modify ./minion/Vagrantfile to use th precise64 box. Change the `config.vm.box` line to:

```
config.vm.box = "precise64"
```

Uncomment the line creating a host-only IP. This is the ip of your minion (you can change it to something else if that IP is already in use):

```
config.vm.network :private_network, ip: "192.168.33.10"
```

At this point you should have a VM that can run, although there won't be much in it. Let's check that.

Checking The VM

From the $home/minion folder type:

```
vagrant up
```

A log showing the VM booting should be present. Once it's done you'll be back to the terminal:

```
ping 192.168.33.10
```

The VM should respond to your ping request.

Now log into the VM in ssh using Vagrant again:

```
vagrant ssh
```

You should see the shell prompt change to something similar to `vagrant@precise64:~$` meaning you're inside the VM. From there, enter the following:

```
ping 10.0.2.2
```

Note: That ip is the ip of your VM host (the Mac OS X OS). The number is a VirtualBox default and is displayed in the log after the Vagrant ssh command. We'll use that IP to tell the minion where the Salt master is. Once you're done, end the ssh session by typing `exit`.

It's now time to connect the VM to the salt master

3.4.12 Step 3 - Connecting Master and Minion

Creating The Minion Configuration File

Create the `/etc/salt/minion` file. In that file, put the following lines, giving the ID for this minion, and the IP of the master:

```
master: 10.0.2.2
id: 'minion1'
file_client: remote
```

Minions authenticate with the master using keys. Keys are generated automatically if you don't provide one and can accept them later on. However, this requires accepting the minion key every time the minion is destroyed or created (which could be quite often). A better way is to create those keys in advance, feed them to the minion, and authorize them once.

Preseed minion keys

From the minion folder on your Mac run:

```
sudo salt-key --gen-keys=minion1
```

This should create two files: minion1.pem, and minion1.pub. Since those files have been created using sudo, but will be used by vagrant, you need to change ownership:

```
sudo chown youruser:yourgroup minion1.pem
sudo chown youruser:yourgroup minion1.pub
```

Then copy the .pub file into the list of accepted minions:

```
sudo cp minion1.pub /etc/salt/pki/master/minions/minion1
```

Modify Vagrantfile to Use Salt Provisioner

Let's now modify the Vagrantfile used to provision the Salt VM. Add the following section in the Vagrantfile (note: it should be at the same indentation level as the other properties):

```
# salt-vagrant config
config.vm.provision :salt do |salt|
    salt.run_highstate = true
    salt.minion_config = "/etc/salt/minion"
    salt.minion_key = "./minion1.pem"
    salt.minion_pub = "./minion1.pub"
end
```

Now destroy the vm and recreate it from the /minion folder:

```
vagrant destroy
vagrant up
```

If everything is fine you should see the following message:

```
"Bootstrapping Salt... (this may take a while)
Salt successfully configured and installed!"
```

Checking Master-Minion Communication

To make sure the master and minion are talking to each other, enter the following:

```
sudo salt '*' test.ping
```

You should see your minion answering the ping. It's now time to do some configuration.

3.4.13 Step 4 - Configure Services to Install On the Minion

In this step we'll use the Salt master to instruct our minion to install Nginx.

Checking the system's original state

First, make sure that an HTTP server is not installed on our minion. When opening a browser directed at `http://192.168.33.10/` You should get an error saying the site cannot be reached.

Initialize the top.sls file

System configuration is done in the /srv/salt/top.sls file (and subfiles/folders), and then applied by running the `state.highstate` command to have the Salt master give orders so minions will update their instructions and run the associated commands.

First Create an empty file on your Salt master (Mac OS X machine):

```
touch /srv/salt/top.sls
```

When the file is empty, or if no configuration is found for our minion an error is reported:

```
sudo salt 'minion1' state.highstate
```

Should return an error stating: ``No Top file or external nodes data matches found''.

Create The Nginx Configuration

Now is finally the time to enter the real meat of our server's configuration. For this tutorial our minion will be treated as a web server that needs to have Nginx installed.

Insert the following lines into the `/srv/salt/top.sls` file (which should current be empty).

```
base:
  'minion1':
    - bin.nginx
```

Now create a `/srv/salt/bin/nginx.sls` file containing the following:

```
nginx:
  pkg.installed:
    - name: nginx
  service.running:
    - enable: True
    - reload: True
```

Check Minion State

Finally run the state.highstate command again:

```
sudo salt 'minion1' state.highstate
```

You should see a log showing that the Nginx package has been installed and the service configured. To prove it, open your browser and navigate to http://192.168.33.10/, you should see the standard Nginx welcome page.

Congratulations!

Where To Go From Here

A full description of configuration management within Salt (sls files among other things) is available here: http://docs.saltstack.com/index.html#configuration-management

3.4.14 Writing Salt Tests

Note: THIS TUTORIAL IS A WORK IN PROGRESS

Salt comes with a powerful integration and unit test suite. The test suite allows for the fully automated run of integration and/or unit tests from a single interface. The integration tests are surprisingly easy to write and can be written to be either destructive or non-destructive.

Getting Set Up For Tests

To walk through adding an integration test, start by getting the latest development code and the test system from GitHub:

Note: The develop branch often has failing tests and should always be considered a staging area. For a checkout that tests should be running perfectly on, please check out a specific release tag (such as v2014.1.4).

```
git clone git@github.com:saltstack/salt.git
pip install git+https://github.com/saltstack/salt-testing.git#egg=SaltTesting
```

Now that a fresh checkout is available run the test suite

Destructive vs Non-destructive

Since Salt is used to change the settings and behavior of systems, often, the best approach to run tests is to make actual changes to an underlying system. This is where the concept of destructive integration tests comes into play. Tests can be written to alter the system they are running on. This capability is what fills in the gap needed to properly test aspects of system management like package installation.

To write a destructive test import and use the *destructiveTest* decorator for the test method:

```
import integration
from salttesting.helpers import destructiveTest

class PkgTest(integration.ModuleCase):
    @destructiveTest
    def test_pkg_install(self):
```

```
        ret = self.run_function('pkg.install', name='finch')
        self.assertSaltTrueReturn(ret)
        ret = self.run_function('pkg.purge', name='finch')
        self.assertSaltTrueReturn(ret)
```

Automated Test Runs

SaltStack maintains a Jenkins server which can be viewed at http://jenkins.saltstack.com. The tests executed from this Jenkins server create fresh virtual machines for each test run, then execute the destructive tests on the new clean virtual machine. This allows for the execution of tests across supported platforms.

3.5 Salt Virt

3.5.1 Salt as a Cloud Controller

In Salt 0.14.0, an advanced cloud control system were introduced, allow private cloud vms to be managed directly with Salt. This system is generally referred to as **Salt Virt**.

The Salt Virt system already exists and is installed within Salt itself, this means that beside setting up Salt, no additional salt code needs to be deployed.

The main goal of Salt Virt is to facilitate a very fast and simple cloud. The cloud that can scale and fully featured. Salt Virt comes with the ability to set up and manage complex virtual machine networking, powerful image, and disk management, as well as virtual machine migration with and without shared storage.

This means that Salt Virt can be used to create a cloud from a blade center and a SAN, but can also create a cloud out of a swarm of Linux Desktops without a single shared storage system. Salt Virt can make clouds from truly commodity hardware, but can also stand up the power of specialized hardware as well.

Setting up Hypervisors

The first step to set up the hypervisors involves getting the correct software installed and setting up the hypervisor network interfaces.

Installing Hypervisor Software

Salt Virt is made to be hypervisor agnostic but currently the only fully implemented hypervisor is KVM via libvirt.

The required software for a hypervisor is libvirt and kvm. For advanced features install libguestfs or qemu-nbd.

Note: Libguestfs and qemu-nbd allow for virtual machine images to be mounted before startup and get pre-seeded with configurations and a salt minion

---

This sls will set up the needed software for a hypervisor, and run the routines to set up the libvirt pki keys.

Note: Package names and setup used is Red Hat specific, different package names will be required for different platforms

---

```
libvirt:
  pkg.installed: []
  file.managed:
    - name: /etc/sysconfig/libvirtd
    - contents: 'LIBVIRTD_ARGS="--listen"'
    - require:
      - pkg: libvirt
  libvirt.keys:
    - require:
      - pkg: libvirt
  service.running:
    - name: libvirtd
    - require:
      - pkg: libvirt
      - network: br0
      - libvirt: libvirt
    - watch:
      - file: libvirt

libvirt-python:
  pkg.installed: []

libguestfs:
  pkg.installed:
    - pkgs:
      - libguestfs
      - libguestfs-tools
```

Hypervisor Network Setup

The hypervisors will need to be running a network bridge to serve up network devices for virtual machines, this formula will set up a standard bridge on a hypervisor connecting the bridge to eth0:

```
eth0:
  network.managed:
    - enabled: True
    - type: eth
    - bridge: br0

br0:
  network.managed:
    - enabled: True
    - type: bridge
    - proto: dhcp
    - require:
      - network: eth0
```

Virtual Machine Network Setup

Salt Virt comes with a system to model the network interfaces used by the deployed virtual machines; by default a single interface is created for the deployed virtual machine and is bridged to `br0`. To get going with the default networking setup, ensure that the bridge interface named `br0` exists on the hypervisor and is bridged to an active network device.

Note: To use more advanced networking in Salt Virt, read the *Salt Virt Networking* document:

Salt Virt Networking

Libvirt State

One of the challenges of deploying a libvirt based cloud is the distribution of libvirt certificates. These certificates allow for virtual machine migration. Salt comes with a system used to auto deploy these certificates. Salt manages the signing authority key and generates keys for libvirt clients on the master, signs them with the certificate authority and uses pillar to distribute them. This is managed via the `libvirt` state. Simply execute this formula on the minion to ensure that the certificate is in place and up to date:

Note: The above formula includes the calls needed to set up libvirt keys.

```
libvirt_keys:
  libvirt.keys
```

Getting Virtual Machine Images Ready

Salt Virt, requires that virtual machine images be provided as these are not generated on the fly. Generating these virtual machine images differs greatly based on the underlying platform.

Virtual machine images can be manually created using KVM and running through the installer, but this process is not recommended since it is very manual and prone to errors.

Virtual Machine generation applications are available for many platforms:

vm-builder: https://wiki.debian.org/VMBuilder

See also:

vmbuilder-formula

Once virtual machine images are available, the easiest way to make them available to Salt Virt is to place them in the Salt file server. Just copy an image into `/srv/salt` and it can now be used by Salt Virt.

For purposes of this demo, the file name `centos.img` will be used.

Existing Virtual Machine Images

Many existing Linux distributions distribute virtual machine images which can be used with Salt Virt. Please be advised that NONE OF THESE IMAGES ARE SUPPORTED BY SALTSTACK.

CentOS These images have been prepared for OpenNebula but should work without issue with Salt Virt, only the raw qcow image file is needed: http://wiki.centos.org/Cloud/OpenNebula

Fedora Linux Images for Fedora Linux can be found here: http://fedoraproject.org/en/get-fedora#clouds

Ubuntu Linux Images for Ubuntu Linux can be found here: http://cloud-images.ubuntu.com/

Using Salt Virt

With hypervisors set up and virtual machine images ready, Salt can start issuing cloud commands.

Start by running a Salt Virt hypervisor info command:

```
salt-run virt.hyper_info
```

This will query what the running hypervisor stats are and display information for all configured hypervisors. This command will also validate that the hypervisors are properly configured.

Now that hypervisors are available a virtual machine can be provisioned. The `virt.init` routine will create a new virtual machine:

```
salt-run virt.init centos1 2 512 salt://centos.img
```

This command assumes that the CentOS virtual machine image is sitting in the root of the Salt fileserver. Salt Virt will now select a hypervisor to deploy the new virtual machine on and copy the virtual machine image down to the hypervisor.

Once the VM image has been copied down the new virtual machine will be seeded. Seeding the VMs involves setting pre-authenticated Salt keys on the new VM and if needed, will install the Salt Minion on the new VM before it is started.

Note: The biggest bottleneck in starting VMs is when the Salt Minion needs to be installed. Making sure that the source VM images already have Salt installed will GREATLY speed up virtual machine deployment.

Now that the new VM has been prepared, it can be seen via the `virt.query` command:

```
salt-run virt.query
```

This command will return data about all of the hypervisors and respective virtual machines.

Now that the new VM is booted it should have contacted the Salt Master, a `test.ping` will reveal if the new VM is running.

Migrating Virtual Machines

Salt Virt comes with full support for virtual machine migration, and using the libvirt state in the above formula makes migration possible.

A few things need to be available to support migration. Many operating systems turn on firewalls when originally set up, the firewall needs to be opened up to allow for libvirt and kvm to cross communicate and execution migration routines. On Red Hat based hypervisors in particular port 16514 needs to be opened on hypervisors:

```
iptables -A INPUT -m state --state NEW -m tcp -p tcp --dport 16514 -j ACCEPT
```

Note: More in-depth information regarding distribution specific firewall settings can read in:

Opening the Firewall up for Salt

Salt also needs an additional flag to be turned on as well. The `virt.tunnel` option needs to be turned on. This flag tells Salt to run migrations securely via the libvirt TLS tunnel and to use port 16514. Without `virt.tunnel` libvirt tries to bind to random ports when running migrations. To turn on `virt.tunnel` simple apply it to the master config file:

```
virt.tunnel: True
```

Once the master config has been updated, restart the master and send out a call to the minions to refresh the pillar to pick up on the change:

```
salt \* saltutil.refresh_modules
```

Now, migration routines can be run! To migrate a VM, simply run the Salt Virt migrate routine:

```
salt-run virt.migrate centos <new hypervisor>
```

VNC Consoles

Salt Virt also sets up VNC consoles by default, allowing for remote visual consoles to be oped up. The information from a `virt.query` routine will display the vnc console port for the specific vms:

```
centos
  CPU: 2
  Memory: 524288
  State: running
  Graphics: vnc - hyper6:5900
  Disk - vda:
    Size: 2.0G
    File: /srv/salt-images/ubuntu2/system.qcow2
    File Format: qcow2
  Nic - ac:de:48:98:08:77:
    Source: br0
    Type: bridge
```

The line *Graphics: vnc - hyper6:5900* holds the key. First the port named, in this case 5900, will need to be available in the hypervisor's firewall. Once the port is open, then the console can be easily opened via vncviewer:

```
vncviewer hyper6:5900
```

By default there is no VNC security set up on these ports, which suggests that keeping them firewalled and mandating that SSH tunnels be used to access these VNC interfaces. Keep in mind that activity on a VNC interface that is accessed can be viewed by any other user that accesses that same VNC interface, and any other user logging in can also operate with the logged in user on the virtual machine.

Conclusion

Now with Salt Virt running, new hypervisors can be seamlessly added just by running the above states on new bare metal machines, and these machines will be instantly available to Salt Virt.

3.6 Halite

3.6.1 Installing and Configuring Halite

In this tutorial, we'll walk through installing and setting up Halite. The current version of Halite is considered pre-alpha and is supported only in Salt `v2014.1.0` or greater. Additional information is available on GitHub: https://github.com/saltstack/halite

Before beginning this tutorial, ensure that the salt-master is installed. To install the salt-master, please review the installation documentation: http://docs.saltstack.com/topics/installation/index.html

Note: Halite only works with Salt versions greater than 2014.1.0.

Installing Halite Via Package

On CentOS, RHEL, or Fedora:

```
$ yum install python-halite
```

Note: By default python-halite only installs CherryPy. If you would like to use a different webserver please review the instructions below to install pip and your server of choice. The package does not modify the master configuration with `/etc/salt/master`.

Installing Halite Using pip

To begin the installation of Halite from PyPI, you'll need to install pip. The Salt package, as well as the bootstrap, do not install pip by default.

On CentOS, RHEL, or Fedora:

```
$ yum install python-pip
```

On Debian:

```
$ apt-get install python-pip
```

Once you have pip installed, use it to install halite:

```
$ pip install -U halite
```

Depending on the webserver you want to run halite through, you'll need to install that piece as well. On RHEL based distros, use one of the following:

```
$ pip install cherrypy
```

```
$ pip install paste
```

```
$ yum install python-devel
$ yum install gcc
$ pip install gevent
$ pip install pyopenssl
```

On Debian based distributions:

```
$ pip install CherryPy
```

```
$ pip install paste
```

```
$ apt-get install gcc
$ apt-get install python-dev
$ apt-get install libevent-dev
$ pip install gevent
$ pip install pyopenssl
```

Configuring Halite Permissions

Configuring Halite access permissions is easy. By default, you only need to ensure that the @runner group is configured. In the `/etc/salt/master` file, uncomment and modify the following lines:

```
external_auth:
  pam:
    testuser:
      - .*
      - '@runner'
```

Note: You cannot use the root user for pam login; it will fail to authenticate.

Halite uses the runner manage.present to get the status of minions, so runner permissions are required. For example:

```
external_auth:
  pam:
    mytestuser:
      - .*
      - '@runner'
      - '@wheel'
```

Currently Halite allows, but does not require, any wheel modules.

Configuring Halite Settings

Once you've configured the permissions for Halite, you'll need to set up the Halite settings in the /etc/salt/master file. Halite supports CherryPy, Paste, and Gevent out of the box.

To configure cherrypy, add the following to the bottom of your /etc/salt/master file:

```
halite:
  level: 'debug'
  server: 'cherrypy'
  host: '0.0.0.0'
  port: '8080'
  cors: False
  tls: True
  certpath: '/etc/pki/tls/certs/localhost.crt'
  keypath: '/etc/pki/tls/certs/localhost.key'
  pempath: '/etc/pki/tls/certs/localhost.pem'
```

If you wish to use paste:

```
halite:
  level: 'debug'
  server: 'paste'
  host: '0.0.0.0'
  port: '8080'
  cors: False
  tls: True
  certpath: '/etc/pki/tls/certs/localhost.crt'
  keypath: '/etc/pki/tls/certs/localhost.key'
  pempath: '/etc/pki/tls/certs/localhost.pem'
```

To use gevent:

```
halite:
  level: 'debug'
  server: 'gevent'
  host: '0.0.0.0'
  port: '8080'
  cors: False
  tls: True
  certpath: '/etc/pki/tls/certs/localhost.crt'
  keypath: '/etc/pki/tls/certs/localhost.key'
  pempath: '/etc/pki/tls/certs/localhost.pem'
```

The ``cherrypy'' and ``gevent'' servers require the certpath and keypath files to run tls/ssl. The .crt file holds the public cert and the .key file holds the private key. Whereas the ``paste'' server requires a single .pem file that contains both the cert and key. This can be created simply by concatenating the .crt and .key files.

If you want to use a self-signed cert, you can create one using the Salt.tls module:

Note: The following command needs to be run on your salt master.

```
salt-call tls.create_self_signed_cert tls
```

Note that certs generated by the above command can be found under the `/etc/pki/tls/certs/` directory. When using self-signed certs, browsers will need approval before accepting the cert. If the web application page has been cached with a non-HTTPS version of the app, then the browser cache will have to be cleared before it will recognize and prompt to accept the self-signed certificate.

Starting Halite

Once you've configured the halite section of your /etc/salt/master, you can restart the salt-master service, and your halite instance will be available. Depending on your configuration, the instance will be available either at https://localhost:8080/app, https://domain:8080/app, or https://123.456.789.012:8080/app .

Note: halite requires an HTML 5 compliant browser.

All logs relating to halite are logged to the default /var/log/salt/master file.

3.7 Using Salt at scale

3.7.1 Using salt at scale

The focus of this tutorial will be building a Salt infrastructure for handling large numbers of minions. This will include tuning, topology, and best practices.

For how to install the saltmaster please go here: Installing saltstack

Note: This tutorial is intended for large installations, although these same settings won't hurt, it may not be worth the complexity to smaller installations.

When used with minions, the term `many' refers to at least a thousand and `a few' always means 500.

For simplicity reasons, this tutorial will default to the standard ports used by salt.

The Master

The most common problems on the salt-master are:

1. too many minions authing at once
2. too many minions re-authing at once
3. too many minions re-connecting at once
4. too many minions returning at once
5. too few resources (CPU/HDD)

The first three are all ``thundering herd'' problems. To mitigate these issues we must configure the minions to back-off appropriately when the master is under heavy load.

The fourth is caused by masters with little hardware resources in combination with a possible bug in ZeroMQ. At least thats what it looks like till today (Issue 118651, Issue 5948, Mail thread)

To fully understand each problem, it is important to understand, how salt works.

Very briefly, the saltmaster offers two services to the minions.

- a job publisher on port 4505
- an open port 4506 to receive the minions returns

All minions are always connected to the publisher on port 4505 and only connect to the open return port 4506 if necessary. On an idle master, there will only be connections on port 4505.

Too many minions authing

When the minion service is first started up, it will connect to its master's publisher on port 4505. If too many minions are started at once, this can cause a ``thundering herd''. This can be avoided by not starting too many minions at once.

The connection itself usually isn't the culprit, the more likely cause of master-side issues is the authentication that the minion must do with the master. If the master is too heavily loaded to handle the auth request it will time it out. The minion will then wait *acceptance_wait_time* to retry. If *acceptance_wait_time_max* is set then the minion will increase its wait time by the *acceptance_wait_time* each subsequent retry until reaching *acceptance_wait_time_max*.

Too many minions re-authing

This is most likely to happen in the testing phase, when all minion keys have already been accepted, the framework is being tested and parameters change frequently in the masters configuration file.

In a few cases (master restart, remove minion key, etc.) the salt-master generates a new AES-key to encrypt its publications with. The minions aren't notified of this but will realize this on the next pub job they receive. When the minion receives such a job it will then re-auth with the master. Since Salt does minion-side filtering this means that all the minions will re-auth on the next command published on the master-- causing another ``thundering herd''. This can be avoided by setting the

```
random_reauth_delay: 60
```

in the minions configuration file to a higher value and stagger the amount of re-auth attempts. Increasing this value will of course increase the time it takes until all minions are reachable via salt commands.

Too many minions re-connecting

By default the zmq socket will re-connect every 100ms which for some larger installations may be too quick. This will control how quickly the TCP session is re-established, but has no bearing on the auth load.

To tune the minions sockets reconnect attempts, there are a few values in the sample configuration file (default values)

```
recon_default: 100ms
recon_max: 5000
recon_randomize: True
```

- recon_default: the default value the socket should use, i.e. 100ms
- recon_max: the max value that the socket should use as a delay before trying to reconnect
- recon_randomize: enables randomization between recon_default and recon_max

To tune this values to an existing environment, a few decision have to be made.

1. How long can one wait, before the minions should be online and reachable via salt?
2. How many reconnects can the master handle without a syn flood?

These questions can not be answered generally. Their answers depend on the hardware and the administrators requirements.

Here is an example scenario with the goal, to have all minions reconnect within a 60 second time-frame on a salt-master service restart.

```
recon_default: 1000
recon_max: 59000
recon_randomize: True
```

Each minion will have a randomized reconnect value between `recon_default' and `recon_default + recon_max', which in this example means between 1000ms and 60000ms (or between 1 and 60 seconds). The generated random-value will be doubled after each attempt to reconnect (ZeroMQ default behavior).

Lets say the generated random value is 11 seconds (or 11000ms).

```
reconnect 1: wait 11 seconds
reconnect 2: wait 22 seconds
reconnect 3: wait 33 seconds
reconnect 4: wait 44 seconds
reconnect 5: wait 55 seconds
reconnect 6: wait time is bigger than 60 seconds (recon_default + recon_max)
reconnect 7: wait 11 seconds
reconnect 8: wait 22 seconds
reconnect 9: wait 33 seconds
reconnect x: etc.
```

With a thousand minions this will mean

```
1000/60 = ~16
```

round about 16 connection attempts a second. These values should be altered to values that match your environment. Keep in mind though, that it may grow over time and that more minions might raise the problem again.

Too many minions returning at once

This can also happen during the testing phase, if all minions are addressed at once with

```
$ salt * test.ping
```

it may cause thousands of minions trying to return their data to the salt-master open port 4506. Also causing a flood of syn-flood if the master can't handle that many returns at once.

This can be easily avoided with salts batch mode:

```
$ salt * test.ping -b 50
```

This will only address 50 minions at once while looping through all addressed minions.

Too few resources

The masters resources always have to match the environment. There is no way to give good advise without knowing the environment the master is supposed to run in. But here are some general tuning tips for different situations:

The master is CPU bound

Salt uses RSA-Key-Pairs on the masters and minions end. Both generate 4096 bit key-pairs on first start. While the key-size for the master is currently not configurable, the minions keysize can be configured with different key-sizes. For example with a 2048 bit key:

```
keysize: 2048
```

With thousands of decryptions, the amount of time that can be saved on the masters end should not be neglected. See here for reference: Pull Request 9235 how much influence the key-size can have.

Downsizing the salt-masters key is not that important, because the minions do not encrypt as many messages as the master does.

The master is disk IO bound

By default, the master saves every minion's return for every job in its job-cache. The cache can then be used later, to lookup results for previous jobs. The default directory for this is:

```
cachedir: /var/cache/salt
```

and then in the `/proc` directory.

Each job return for every minion is saved in a single file. Over time this directory can grow quite large, depending on the number of published jobs. The amount of files and directories will scale with the number of jobs published and the retention time defined by

```
keep_jobs: 24
```

```
250 jobs/day * 2000 minions returns = 500.000 files a day
```

If no job history is needed, the job cache can be disabled:

```
job_cache: False
```

If the job cache is necessary there are (currently) 2 options:

- ext_job_cache: this will have the minions store their return data directly into a returner (not sent through the master)

- master_job_cache (New in *2014.7.0*): this will make the master store the job data using a returner (instead of the local job cache on disk).

CHAPTER 4

Targeting Minions

Targeting minions is specifying which minions should run a command or execute a state by matching against hostnames, or system information, or defined groups, or even combinations thereof.

For example the command `salt web1 apache.signal restart` to restart the Apache httpd server specifies the machine `web1` as the target and the command will only be run on that one minion.

Similarly when using States, the following *top file* specifies that only the `web1` minion should execute the contents of `webserver.sls`:

```
base:
  'web1':
    - webserver
```

There are many ways to target individual minions or groups of minions in Salt:

4.1 Matching the `minion id`

Each minion needs a unique identifier. By default when a minion starts for the first time it chooses its FQDN (fully qualified domain name) as that identifier. The minion id can be overridden via the minion's `id` configuration setting.

Tip: minion id and minion keys

The *minion id* is used to generate the minion's public/private keys and if it ever changes the master must then accept the new key as though the minion was a new host.

4.1.1 Globbing

The default matching that Salt utilizes is `shell-style globbing` around the *minion id*. This also works for states in the *top file*.

Note: You must wrap **`salt`** calls that use globbing in single-quotes to prevent the shell from expanding the globs before Salt is invoked.

Match all minions:

```
salt '*' test.ping
```

Match all minions in the example.net domain or any of the example domains:

```
salt '*.example.net' test.ping
salt '*.example.*' test.ping
```

Match all the webN minions in the example.net domain (web1.example.net, web2.example.net ... webN.example.net):

```
salt 'web?.example.net' test.ping
```

Match the web1 through web5 minions:

```
salt 'web[1-5]' test.ping
```

Match the web1 and web3 minions:

```
salt 'web[1,3]' test.ping
```

Match the web-x, web-y, and web-z minions:

```
salt 'web-[x-z]' test.ping
```

Note: For additional targeting methods please review the *compound matchers* documentation.

4.1.2 Regular Expressions

Minions can be matched using Perl-compatible `regular expressions` (which is globbing on steroids and a ton of caffeine).

Match both web1-prod and web1-devel minions:

```
salt -E 'web1-(prod|devel)' test.ping
```

When using regular expressions in a State's *top file*, you must specify the matcher as the first option. The following example executes the contents of webserver.sls on the above-mentioned minions.

```
base:
  'web1-(prod|devel)':
  - match: pcre
  - webserver
```

4.1.3 Lists

At the most basic level, you can specify a flat list of minion IDs:

```
salt -L 'web1,web2,web3' test.ping
```

4.2 Grains

Salt comes with an interface to derive information about the underlying system. This is called the grains interface, because it presents salt with grains of information.

The grains interface is made available to Salt modules and components so that the right salt minion commands are automatically available on the right systems.

It is important to remember that grains are bits of information loaded when the salt minion starts, so this information is static. This means that the information in grains is unchanging, therefore the nature of the data is static. So grains information are things like the running kernel, or the operating system.

Note: Grains resolve to lowercase letters. For example, `FOO`, and `foo` target the same grain.

Match all CentOS minions:

```
salt -G 'os:CentOS' test.ping
```

Match all minions with 64-bit CPUs, and return number of CPU cores for each matching minion:

```
salt -G 'cpuarch:x86_64' grains.item num_cpus
```

Additionally, globs can be used in grain matches, and grains that are nested in a *dictionary* can be matched by adding a colon for each level that is traversed. For example, the following will match hosts that have a grain called `ec2_tags`, which itself is a *dict* with a key named `environment`, which has a value that contains the word `production`:

```
salt -G 'ec2_tags:environment:*production*'
```

4.2.1 Listing Grains

Available grains can be listed by using the `grains.ls' module:

```
salt '*' grains.ls
```

Grains data can be listed by using the `grains.items' module:

```
salt '*' grains.items
```

4.2.2 Grains in the Minion Config

Grains can also be statically assigned within the minion configuration file. Just add the option `grains` and pass options to it:

```
grains:
  roles:
    - webserver
    - memcache
  deployment: datacenter4
  cabinet: 13
  cab_u: 14-15
```

Then status data specific to your servers can be retrieved via Salt, or used inside of the State system for matching. It also makes targeting, in the case of the example above, simply based on specific data about your deployment.

4.2.3 Grains in /etc/salt/grains

If you do not want to place your custom static grains in the minion config file, you can also put them in `/etc/salt/grains` on the minion. They are configured in the same way as in the above example, only without a top-level `grains:` key:

```
roles:
  - webserver
  - memcache
deployment: datacenter4
cabinet: 13
cab_u: 14-15
```

4.2.4 Matching Grains in the Top File

With correctly configured grains on the Minion, the *top file* used in Pillar or during Highstate can be made very efficient. For example, consider the following configuration:

```
'node_type:web':
  - match: grain
  - webserver

'node_type:postgres':
  - match: grain
  - database

'node_type:redis':
  - match: grain
  - redis

'node_type:lb':
  - match: grain
  - lb
```

For this example to work, you would need to have defined the grain `node_type` for the minions you wish to match. This simple example is nice, but too much of the code is similar. To go one step further, Jinja templating can be used to simplify the *top file*.

```
{% set node_type = salt['grains.get']('node_type', '') %}

{% if node_type %}
  'node_type:{{ self }}':
    - match: grain
    - {{ self }}
{% endif %}
```

Using Jinja templating, only one match entry needs to be defined.

Note: The example above uses the `grains.get` function to account for minions which do not have the `node_type` grain set.

4.2.5 Writing Grains

The grains interface is derived by executing all of the ``public'' functions found in the modules located in the grains package or the custom grains directory. The functions in the modules of the grains must return a Python *dict*, where the keys in the *dict* are the names of the grains and the values are the values.

Custom grains should be placed in a `_grains` directory located under the `file_roots` specified by the master config file. The default path would be `/srv/salt/_grains`. Custom grains will be distributed to the minions when `state.highstate` is run, or by executing the `saltutil.sync_grains` or `saltutil.sync_all` functions.

Grains are easy to write, and only need to return a dictionary. A common approach would be code something similar to the following:

```
#!/usr/bin/env python
def yourfunction():
     # initialize a grains dictionary
     grains = {}
     # Some code for logic that sets grains like
     grains['yourcustomgrain']=True
     grains['anothergrain']='somevalue'
     return grains
```

Before adding a grain to Salt, consider what the grain is and remember that grains need to be static data. If the data is something that is likely to change, consider using *Pillar* instead.

Warning: Custom grains will not be available in the top file until after the first *highstate*. To make custom grains available on a minion's first highstate, it is recommended to use *this example* to ensure that the custom grains are synced when the minion starts.

4.2.6 Precedence

Core grains can be overridden by custom grains. As there are several ways of defining custom grains, there is an order of precedence which should be kept in mind when defining them. The order of evaluation is as follows:

1. Core grains.
2. Custom grain modules in `_grains` directory, synced to minions.
3. Custom grains in `/etc/salt/grains`.
4. Custom grains in `/etc/salt/minion`.

Each successive evaluation overrides the previous ones, so any grains defined by custom grains modules synced to minions that have the same name as a core grain will override that core grain. Similarly, grains from `/etc/salt/grains` override both core grains and custom grain modules, and grains in `/etc/salt/minion` will override *any* grains of the same name.

4.2.7 Examples of Grains

The core module in the grains package is where the main grains are loaded by the Salt minion and provides the principal example of how to write grains:

https://github.com/saltstack/salt/blob/develop/salt/grains/core.py

4.2.8 Syncing Grains

Syncing grains can be done a number of ways, they are automatically synced when `state.highstate` is called, or (as noted above) the grains can be manually synced and reloaded by calling the `saltutil.sync_grains` or `saltutil.sync_all` functions.

4.3 Subnet/IP Address Matching

Minions can easily be matched based on IP address, or by subnet (using CIDR notation).

```
salt -S 192.168.40.20 test.ping
salt -S 10.0.0.0/24 test.ping
```

Note: Only IPv4 matching is supported at this time.

4.4 Compound matchers

Compound matchers allow very granular minion targeting using any of Salt's matchers. The default matcher is a `glob` match, just as with CLI and *top file* matching. To match using anything other than a glob, prefix the match string with the appropriate letter from the table below, followed by an `@` sign.

Letter	Match Type	Example
G	Grains glob	`G@os:Ubuntu`
E	PCRE Minion ID	`E@web\d+\.(dev|qa|prod)\.loc`
P	Grains PCRE	`P@os:(RedHat|Fedora|CentOS)`
L	List of minions	`L@minion1.example.com,minion3.domain.com` or `bl*.domain.com`
I	Pillar glob	`I@pdata:foobar`
S	Subnet/IP address	`S@192.168.1.0/24` or `S@192.168.1.100`
R	Range cluster	`R@%foo.bar`

Matchers can be joined using boolean `and`, `or`, and `not` operators.

For example, the following string matches all Debian minions with a hostname that begins with `webserv`, as well as any minions that have a hostname which matches the `regular expression` `web-dc1-srv.*`:

```
salt -C 'webserv* and G@os:Debian or E@web-dc1-srv.*' test.ping
```

That same example expressed in a *top file* looks like the following:

```
base:
  'webserv* and G@os:Debian or E@web-dc1-srv.*':
    - match: compound
    - webserver
```

Note that a leading `not` is not supported in compound matches. Instead, something like the following must be done:

```
salt -C '* and not G@kernel:Darwin' test.ping
```

Excluding a minion based on its ID is also possible:

```
salt -C '* and not web-dc1-srv' test.ping
```

4.4.1 Precedence Matching

Matches can be grouped together with parentheses to explicitly declare precedence amongst groups.

```
salt -C '( ms-1 or G@id:ms-3 ) and G@id:ms-3' test.ping
```

Note: Be certain to note that spaces are required between the parentheses and targets. Failing to obey this rule may result in incorrect targeting!

4.5 Node groups

Nodegroups are declared using a compound target specification. The compound target documentation can be found *here*.

The `nodegroups` master config file parameter is used to define nodegroups. Here's an example nodegroup configuration within `/etc/salt/master`:

```
nodegroups:
  group1: 'L@foo.domain.com,bar.domain.com,baz.domain.com or bl*.domain.com'
  group2: 'G@os:Debian and foo.domain.com'
  group3: 'G@os:Debian and N@group1'
```

Note: The `L` within group1 is matching a list of minions, while the `G` in group2 is matching specific grains. See the *compound matchers* documentation for more details.

Note: Nodgroups can reference other nodegroups as seen in `group3`. Ensure that you do not have circular references. Circular references will be detected and cause partial expansion with a logged error message.

To match a nodegroup on the CLI, use the `-N` command-line option:

```
salt -N group1 test.ping
```

To match a nodegroup in your *top file*, make sure to put `- match: nodegroup` on the line directly following the nodegroup name.

```
base:
  group1:
    - match: nodegroup
    - webserver
```

Note: When adding or modifying nodegroups to a master configuration file, the master must be restarted for those changes to be fully recognized.

A limited amount of functionality, such as targeting with -N from the command-line may be available without a restart.

4.6 Batch Size

The `-b` (or `--batch-size`) option allows commands to be executed on only a specified number of minions at a time. Both percentages and finite numbers are supported.

```
salt '*' -b 10 test.ping

salt -G 'os:RedHat' --batch-size 25% apache.signal restart
```

This will only run test.ping on 10 of the targeted minions at a time and then restart apache on 25% of the minions matching `os:RedHat` at a time and work through them all until the task is complete. This makes jobs like rolling web server restarts behind a load balancer or doing maintenance on BSD firewalls using carp much easier with salt.

The batch system maintains a window of running minions, so, if there are a total of 150 minions targeted and the batch size is 10, then the command is sent to 10 minions, when one minion returns then the command is sent to one additional minion, so that the job is constantly running on 10 minions.

4.7 SECO Range

SECO range is a cluster-based metadata store developed and maintained by Yahoo!

The Range project is hosted here:

https://github.com/ytoolshed/range

Learn more about range here:

https://github.com/ytoolshed/range/wiki/

4.7.1 Prerequisites

To utilize range support in Salt, a range server is required. Setting up a range server is outside the scope of this document. Apache modules are included in the range distribution.

With a working range server, cluster files must be defined. These files are written in YAML and define hosts contained inside a cluster. Full documentation on writing YAML range files is here:

https://github.com/ytoolshed/range/wiki/%22yamlfile%22-module-file-spec

Additionally, the Python seco range libraries must be installed on the salt master. One can verify that they have been installed correctly via the following command:

```
python -c 'import seco.range'
```

If no errors are returned, range is installed successfully on the salt master.

4.7.2 Preparing Salt

Range support must be enabled on the salt master by setting the hostname and port of the range server inside the master configuration file:

```
range_server: my.range.server.com:80
```

Following this, the master must be restarted for the change to have an effect.

4.7.3 Targeting with Range

Once a cluster has been defined, it can be targeted with a salt command by using the `-R` or `--range` flags.

For example, given the following range YAML file being served from a range server:

```
$ cat /etc/range/test.yaml
CLUSTER: host1..100.test.com
APPS:
  - frontend
  - backend
  - mysql
```

One might target host1 through host100 in the test.com domain with Salt as follows:

```
salt --range %test:CLUSTER test.ping
```

The following salt command would target three hosts: `frontend`, `backend`, and `mysql`:

```
salt --range %test:APPS test.ping
```

CHAPTER 5

Storing Static Data in the Pillar

Pillar is an interface for Salt designed to offer global values that can be distributed to all minions. Pillar data is managed in a similar way as the Salt State Tree.

Pillar was added to Salt in version 0.9.8

Note: Storing sensitive data

Unlike state tree, pillar data is only available for the targeted minion specified by the matcher type. This makes it useful for storing sensitive data specific to a particular minion.

5.1 Declaring the Master Pillar

The Salt Master server maintains a pillar_roots setup that matches the structure of the file_roots used in the Salt file server. Like the Salt file server the `pillar_roots` option in the master config is based on environments mapping to directories. The pillar data is then mapped to minions based on matchers in a top file which is laid out in the same way as the state top file. Salt pillars can use the same matcher types as the standard top file.

The configuration for the `pillar_roots` in the master config file is identical in behavior and function as `file_roots`:

```
pillar_roots:
  base:
    - /srv/pillar
```

This example configuration declares that the base environment will be located in the `/srv/pillar` directory. It must not be in a subdirectory of the state tree.

The top file used matches the name of the top file used for States, and has the same structure:

`/srv/pillar/top.sls`

```
base:
  '*':
    - packages
```

In the above top file, it is declared that in the `base` environment, the glob matching all minions will have the pillar data found in the `packages` pillar available to it. Assuming the `pillar_roots` value of `/srv/pillar` taken from above, the `packages` pillar would be located at `/srv/pillar/packages.sls`.

Another example shows how to use other standard top matching types to deliver specific salt pillar data to minions with different properties.

Here is an example using the `grains` matcher to target pillars to minions by their `os` grain:

```
dev:
  'os:Debian':
    - match: grain
    - servers
```

`/srv/pillar/packages.sls`

```
{% if grains['os'] == 'RedHat' %}
apache: httpd
git: git
{% elif grains['os'] == 'Debian' %}
apache: apache2
git: git-core
{% endif %}

company: Foo Industries
```

The above pillar sets two key/value pairs. If a minion is running RedHat, then the `apache` key is set to `httpd` and the `git` key is set to the value of `git`. If the minion is running Debian, those values are changed to `apache2` and `git-core` respctively. All minions that have this pillar targeting to them via a top file will have the key of `company` with a value of `Foo Industries`.

Consequently this data can be used from within modules, renderers, State SLS files, and more via the shared pillar *dict*:

```
apache:
  pkg.installed:
    - name: {{ pillar['apache'] }}
```

```
git:
  pkg.installed:
    - name: {{ pillar['git'] }}
```

Finally, the above states can utilize the values provided to them via Pillar. All pillar values targeted to a minion are available via the `pillar' dictionary. As seen in the above example, Jinja substitution can then be utilized to access the keys and values in the Pillar dictionary.

Note that you cannot just list key/value-information in `top.sls`. Instead, target a minion to a pillar file and then list the keys and values in the pillar. Here is an example top file that illustrates this point:

```
base:
  '*':
    - common_pillar
```

And the actual pillar file at `/srv/pillar/common_pillar.sls':

```
foo: bar
boo: baz
```

5.2 Pillar namespace flattened

The separate pillar files all share the same namespace. Given a `top.sls` of:

```
base:
  '*':
```

```
    - packages
    - services
```

a `packages.sls` file of:

```
bind: bind9
```

and a `services.sls` file of:

```
bind: named
```

Then a request for the `bind` pillar will only return `named`; the `bind9` value is not available. It is better to structure your pillar files with more hierarchy. For example your `package.sls` file could look like:

```
packages:
  bind: bind9
```

5.3 Pillar Namespace Merges

With some care, the pillar namespace can merge content from multiple pillar files under a single key, so long as conflicts are avoided as described above.

For example, if the above example were modified as follows, the values are merged below a single key:

```
base:
  '*':
    - packages
    - services
```

And a `packages.sls` file like:

```
bind:
  package-name: bind9
  version: 9.9.5
```

And a `services.sls` file like:

```
bind:
  port: 53
  listen-on: any
```

The resulting pillar will be as follows:

```
$ salt-call pillar.get bind
local:
    ----------
    listen-on:
        any
    package-name:
        bind9
    port:
        53
    version:
        9.9.5
```

Note: Remember: conflicting keys will be overwritten in a non-deterministic manner!

5.4 Including Other Pillars

New in version 0.16.0.

Pillar SLS files may include other pillar files, similar to State files. Two syntaxes are available for this purpose. The simple form simply includes the additional pillar as if it were part of the same file:

```
include:
  - users
```

The full include form allows two additional options -- passing default values to the templating engine for the included pillar file as well as an optional key under which to nest the results of the included pillar:

```
include:
  - users:
      defaults:
          sudo: ['bob', 'paul']
      key: users
```

With this form, the included file (users.sls) will be nested within the `users' key of the compiled pillar. Additionally, the `sudo' value will be available as a template variable to users.sls.

5.5 Viewing Minion Pillar

Once the pillar is set up the data can be viewed on the minion via the `pillar` module, the pillar module comes with functions, `pillar.items` and `pillar.raw`. `pillar.items` will return a freshly reloaded pillar and `pillar.raw` will return the current pillar without a refresh:

```
salt '*' pillar.items
```

Note: Prior to version 0.16.2, this function is named `pillar.data`. This function name is still supported for backwards compatibility.

5.6 Pillar ``get'' Function

New in version 0.14.0.

The `pillar.get` function works much in the same way as the `get` method in a python dict, but with an enhancement: nested dict components can be extracted using a *:* delimiter.

If a structure like this is in pillar:

```
foo:
  bar:
    baz: qux
```

Extracting it from the raw pillar in an sls formula or file template is done this way:

```
{{ pillar['foo']['bar']['baz'] }}
```

Now, with the new `pillar.get` function the data can be safely gathered and a default can be set, allowing the template to fall back if the value is not available:

```
{{ salt['pillar.get']('foo:bar:baz', 'qux') }}
```

This makes handling nested structures much easier.

Note: `pillar.get()` vs `salt['pillar.get']()`

It should be noted that within templating, the `pillar` variable is just a dictionary. This means that calling `pillar.get()` inside of a template will just use the default dictionary `.get()` function which does not include the extra `:` delimiter functionality. It must be called using the above syntax (`salt['pillar.get']('foo:bar:baz', 'qux')`) to get the salt function, instead of the default dictionary behavior.

5.7 Refreshing Pillar Data

When pillar data is changed on the master the minions need to refresh the data locally. This is done with the `saltutil.refresh_pillar` function.

```
salt '*' saltutil.refresh_pillar
```

This function triggers the minion to asynchronously refresh the pillar and will always return `None`.

5.8 Targeting with Pillar

Pillar data can be used when targeting minions. This allows for ultimate control and flexibility when targeting minions.

```
salt -I 'somekey:specialvalue' test.ping
```

Like with *Grains*, it is possible to use globbing as well as match nested values in Pillar, by adding colons for each level that is being traversed. The below example would match minions with a pillar named `foo`, which is a dict containing a key `bar`, with a value beginning with `baz`:

```
salt -I 'foo:bar:baz*' test.ping
```

5.9 Set Pillar Data at the Command Line

Pillar data can be set at the command line like the following example:

```
salt '*' state.highstate pillar='{"cheese": "spam"}'
```

This will create a dict with a key of `cheese' and a value of `spam'. A list can be created like this:

```
salt '*' state.highstate pillar='["cheese", "milk", "bread"]'
```

5.10 Master Config In Pillar

For convenience the data stored in the master configuration file is made available in all minion's pillars. This makes global configuration of services and systems very easy but may not be desired if sensitive data is stored in the master configuration.

To disable the master config from being added to the pillar set `pillar_opts` to `False`:

```
pillar_opts: False
```

5.11 Master Provided Pillar Error

By default if there is an error rendering a pillar, the detailed error is hidden and replaced with:

```
Rendering SLS 'my.sls' failed. Please see master log for details.
```

The error is protected because it's possible to contain templating data which would give that minion information it shouldn't know, like a password!

To have the master provide the detailed error that could potentially carry protected data set `pillar_safe_render_error` to `False`:

```
pillar_safe_render_error: True
```

CHAPTER 6

Reactor System

Salt version 0.11.0 introduced the reactor system. The premise behind the reactor system is that with Salt's events and the ability to execute commands, a logic engine could be put in place to allow events to trigger actions, or more accurately, reactions.

This system binds sls files to event tags on the master. These sls files then define reactions. This means that the reactor system has two parts. First, the reactor option needs to be set in the master configuration file. The reactor option allows for event tags to be associated with sls reaction files. Second, these reaction files use highdata (like the state system) to define reactions to be executed.

6.1 Event System

A basic understanding of the event system is required to understand reactors. The event system is a local ZeroMQ PUB interface which fires salt events. This event bus is an open system used for sending information notifying Salt and other systems about operations.

The event system fires events with a very specific criteria. Every event has a **tag**. Event tags allow for fast top level filtering of events. In addition to the tag, each event has a data structure. This data structure is a dict, which contains information about the event.

6.2 Mapping Events to Reactor SLS Files

Reactor SLS files and event tags are associated in the master config file. By default this is /etc/salt/master, or /etc/salt/master.d/reactor.conf.

New in version 2014.7.0: Added Reactor support for `salt://` file paths.

In the master config section `reactor:' is a list of event tags to be matched and each event tag has a list of reactor SLS files to be run.

```
reactor:                            # Master config section "reactor"

  - 'salt/minion/*/start':          # Match tag "salt/minion/*/start"
    - /srv/reactor/start.sls        # Things to do when a minion starts
    - /srv/reactor/monitor.sls      # Other things to do

  - 'salt/cloud/*/destroyed':       # Globs can be used to matching tags
    - /srv/reactor/destroy/*.sls    # Globs can be used to match file names
```

```
  - 'myco/custom/event/tag':        # React to custom event tags
    - salt://reactor/mycustom.sls   # Put reactor files under file_roots
```

Reactor sls files are similar to state and pillar sls files. They are by default yaml + Jinja templates and are passed familiar context variables.

They differ because of the addition of the `tag` and `data` variables.

- The `tag` variable is just the tag in the fired event.
- The `data` variable is the event's data dict.

Here is a simple reactor sls:

```
{% if data['id'] == 'mysql1' %}
highstate_run:
  local.state.highstate:
    - tgt: mysql1
{% endif %}
```

This simple reactor file uses Jinja to further refine the reaction to be made. If the `id` in the event data is `mysql1` (in other words, if the name of the minion is `mysql1`) then the following reaction is defined. The same data structure and compiler used for the state system is used for the reactor system. The only difference is that the data is matched up to the salt command API and the runner system. In this example, a command is published to the `mysql1` minion with a function of `state.highstate`. Similarly, a runner can be called:

```
{% if data['data']['overstate'] == 'refresh' %}
overstate_run:
  runner.state.over
{% endif %}
```

This example will execute the state.overstate runner and initiate an overstate execution.

6.3 Fire an event

To fire an event from a minion call `event.send`

```
salt-call event.send 'foo' '{overstate: refresh}'
```

After this is called, any reactor sls files matching event tag `foo` will execute with `{{ data['data']['overstate'] }}` equal to `'refresh'`.

See `salt.modules.event` for more information.

6.4 Knowing what event is being fired

The best way to see exactly what events are fired and what data is available in each event is to use the `state.event runner`.

See also:

Common Salt Events

Example usage:

```
salt-run state.event pretty=True
```

Example output:

```
salt/job/20150213001905721678/new       {
    "_stamp": "2015-02-13T00:19:05.724583",
    "arg": [],
    "fun": "test.ping",
    "jid": "20150213001905721678",
    "minions": [
        "jerry"
    ],
    "tgt": "*",
    "tgt_type": "glob",
    "user": "root"
}
salt/job/20150213001910749506/ret/jerry {
    "_stamp": "2015-02-13T00:19:11.136730",
    "cmd": "_return",
    "fun": "saltutil.find_job",
    "fun_args": [
        "20150213001905721678"
    ],
    "id": "jerry",
    "jid": "20150213001910749506",
    "retcode": 0,
    "return": {},
    "success": true
}
```

6.5 Debugging the Reactor

The best window into the Reactor is to run the master in the foreground with debug logging enabled. The output will include when the master sees the event, what the master does in response to that event, and it will also include the rendered SLS file (or any errors generated while rendering the SLS file).

1. Stop the master.
2. Start the master manually:

```
salt-master -l debug
```

3. Look for log entries in the form:

```
[DEBUG   ] Gathering reactors for tag foo/bar
[DEBUG   ] Compiling reactions for tag foo/bar
[DEBUG   ] Rendered data from file: /path/to/the/reactor_file.sls:
<... Rendered output appears here. ...>
```

The rendered output is the result of the Jinja parsing and is a good way to view the result of referencing Jinja variables. If the result is empty then Jinja produced an empty result and the Reactor will ignore it.

6.6 Understanding the Structure of Reactor Formulas

I.e., when to use `arg` and `kwarg` and when to specify the function arguments directly.

While the reactor system uses the same basic data structure as the state system, the functions that will be called using that data structure are different functions than are called via Salt's state system. The Reactor can call Runner

modules using the *runner* prefix, Wheel modules using the *wheel* prefix, and can also cause minions to run Execution modules using the *local* prefix.

Changed in version 2014.7.0: The `cmd` prefix was renamed to `local` for consistency with other parts of Salt. A backward-compatible alias was added for `cmd`.

The Reactor runs on the master and calls functions that exist on the master. In the case of Runner and Wheel functions the Reactor can just call those functions directly since they exist on the master and are run on the master.

In the case of functions that exist on minions and are run on minions, the Reactor still needs to call a function on the master in order to send the necessary data to the minion so the minion can execute that function.

The Reactor calls functions exposed in *Salt's Python API documentation*. and thus the structure of Reactor files very transparently reflects the function signatures of those functions.

6.6.1 Calling Execution modules on Minions

The Reactor sends commands down to minions in the exact same way Salt's CLI interface does. It calls a function locally on the master that sends the name of the function as well as a list of any arguments and a dictionary of any keyword arguments that the minion should use to execute that function.

Specifically, the Reactor calls the async version of `this function`. You can see that function has `arg' and `kwarg' parameters which are both values that are sent down to the minion.

Executing remote commands maps to the **LocalClient** interface which is used by the **salt** command. This interface more specifically maps to the **cmd_async** method inside of the **LocalClient** class. This means that the arguments passed are being passed to the **cmd_async** method, not the remote method. A field starts with **local** to use the **LocalClient** subsystem. The result is, to execute a remote command, a reactor formula would look like this:

```
clean_tmp:
  local.cmd.run:
    - tgt: '*'
    - arg:
      - rm -rf /tmp/*
```

The `arg` option takes a list of arguments as they would be presented on the command line, so the above declaration is the same as running this salt command:

```
salt '*' cmd.run 'rm -rf /tmp/*'
```

Use the `expr_form` argument to specify a matcher:

```
clean_tmp:
  local.cmd.run:
    - tgt: 'os:Ubuntu'
    - expr_form: grain
    - arg:
      - rm -rf /tmp/*


clean_tmp:
  local.cmd.run:
    - tgt: 'G@roles:hbase_master'
    - expr_form: compound
    - arg:
      - rm -rf /tmp/*
```

Any other parameters in the `LocalClient().cmd()` method can be specified as well.

6.6.2 Calling Runner modules and Wheel modules

Calling Runenr modules and wheel modules from the Reactor uses a more direct syntax since the function is being executed locally instead of sending a command to a remote system to be executed there. There are no `arg' or `kwarg' parameters (unless the Runenr function or Wheel function accepts a paramter with either of those names.)

For example:

```
clear_the_grains_cache_for_all_minions:
  runner.cache.clear_grains
```

If `the runner takes arguments` then they can be specified as well:

```
spin_up_more_web_machines:
  runner.cloud.profile:
    - prof: centos_6
    - instances:
      - web11       # These VM names would be generated via Jinja in a
      - web12       # real-world example.
```

6.6.3 Passing event data to Minions or Orchestrate as Pillar

An interesting trick to pass data from the Reactor script to `state.highstate` or `state.sls` is to pass it as inline Pillar data since both functions take a keyword argument named `pillar`.

The following example uses Salt's Reactor to listen for the event that is fired when the key for a new minion is accepted on the master using `salt-key`.

`/etc/salt/master.d/reactor.conf`:

```
reactor:
  - 'salt/key':
    - /srv/salt/haproxy/react_new_minion.sls
```

The Reactor then fires a `state.sls` command targeted to the HAProxy servers and passes the ID of the new minion from the event to the state file via inline Pillar.

`/srv/salt/haproxy/react_new_minion.sls`:

```
{% if data['act'] == 'accept' and data['id'].startswith('web') %}
add_new_minion_to_pool:
  local.state.sls:
    - tgt: 'haproxy*'
    - arg:
      - haproxy.refresh_pool
    - kwarg:
        pillar:
          new_minion: {{ data['id'] }}
{% endif %}
```

The above command is equivalent to the following command at the CLI:

```
salt 'haproxy*' state.sls haproxy.refresh_pool 'pillar={new_minion: minionid}'
```

This works with Orchestrate files as well:

```
call_some_orchestrate_file:
  runner.state.orchestrate:
    - mods: some_orchestrate_file
```

```
    - pillar:
        stuff: things
```

Which is equivalent to the following command at the CLI:

```
salt-run state.orchestrate some_orchestrate_file pillar='{stuff: things}'
```

Finally, that data is available in the state file using the normal Pillar lookup syntax. The following example is grabbing web server names and IP addresses from *Salt Mine*. If this state is invoked from the Reactor then the custom Pillar value from above will be available and the new minion will be added to the pool but with the `disabled` flag so that HAProxy won't yet direct traffic to it.

`/srv/salt/haproxy/refresh_pool.sls`:

```
{% set new_minion = salt['pillar.get']('new_minion') %}

listen web *:80
    balance source
    {% for server,ip in salt['mine.get']('web*', 'network.interfaces', ['eth0']).items() %}
    {% if server == new_minion %}
    server {{ server }} {{ ip }}:80 disabled
    {% else %}
    server {{ server }} {{ ip }}:80 check
    {% endif %}
    {% endfor %}
```

6.7 A Complete Example

In this example, we're going to assume that we have a group of servers that will come online at random and need to have keys automatically accepted. We'll also add that we don't want all servers being automatically accepted. For this example, we'll assume that all hosts that have an id that starts with `ink' will be automatically accepted and have state.highstate executed. On top of this, we're going to add that a host coming up that was replaced (meaning a new key) will also be accepted.

Our master configuration will be rather simple. All minions that attempte to authenticate will match the **tag** of **salt/auth**. When it comes to the minion key being accepted, we get a more refined **tag** that includes the minion id, which we can use for matching.

`/etc/salt/master.d/reactor.conf`:

```
reactor:
  - 'salt/auth':
    - /srv/reactor/auth-pending.sls
  - 'salt/minion/ink*/start':
    - /srv/reactor/auth-complete.sls
```

In this sls file, we say that if the key was rejected we will delete the key on the master and then also tell the master to ssh in to the minion and tell it to restart the minion, since a minion process will die if the key is rejected.

We also say that if the key is pending and the id starts with ink we will accept the key. A minion that is waiting on a pending key will retry authentication every ten seconds by default.

`/srv/reactor/auth-pending.sls`:

```
{# Ink server faild to authenticate -- remove accepted key #}
{% if not data['result'] and data['id'].startswith('ink') %}
minion_remove:
```

```
  wheel.key.delete:
    - match: {{ data['id'] }}
minion_rejoin:
  local.cmd.run:
    - tgt: salt-master.domain.tld
    - arg:
      - ssh -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no "{{ data['id'] }}" 'sleep 10
{% endif %}

{# Ink server is sending new key -- accept this key #}
{% if 'act' in data and data['act'] == 'pend' and data['id'].startswith('ink') %}
minion_add:
  wheel.key.accept:
    - match: {{ data['id'] }}
{% endif %}
```

No if statements are needed here because we already limited this action to just Ink servers in the master configuration.

`/srv/reactor/auth-complete.sls`:

```
{# When an Ink server connects, run state.highstate. #}
highstate_run:
  local.state.highstate:
    - tgt: {{ data['id'] }}
    - ret: smtp_return
```

The above will also return the highstate result data using the *smtp_return* returner. The returner needs to be configured on the minion for this to work. See `salt.returners.smtp_return` documentation for that.

6.8 Syncing Custom Types on Minion Start

Salt will sync all custom types (by running a `saltutil.sync_all`) on every highstate. However, there is a chicken-and-egg issue where, on the initial highstate, a minion will not yet have these custom types synced when the top file is first compiled. This can be worked around with a simple reactor which watches for `minion_start` events, which each minion fires when it first starts up and connects to the master.

On the master, create **/srv/reactor/sync_grains.sls** with the following contents:

```
sync_grains:
  local.saltutil.sync_grains:
    - tgt: {{ data['id'] }}
```

And in the master config file, add the following reactor configuration:

```
reactor:
  - 'minion_start':
    - /srv/reactor/sync_grains.sls
```

This will cause the master to instruct each minion to sync its custom grains when it starts, making these grains available when the initial highstate is executed.

Other types can be synced by replacing `local.saltutil.sync_grains` with `local.saltutil.sync_modules`, `local.saltutil.sync_all`, or whatever else suits the intended use case.

CHAPTER 7

The Salt Mine

The Salt Mine is used to collect arbitrary data from minions and store it on the master. This data is then made available to all minions via the `salt.modules.mine` module.

The data is gathered on the minion and sent back to the master where only the most recent data is maintained (if long term data is required use returners or the external job cache).

7.1 Mine Functions

To enable the Salt Mine the *mine_functions* option needs to be applied to a minion. This option can be applied via the minion's configuration file, or the minion's Pillar. The *mine_functions* option dictates what functions are being executed and allows for arguments to be passed in. If no arguments are passed, an empty list must be added:

```
mine_functions:
  test.ping: []
  network.ip_addrs:
    interface: eth0
    cidr: '10.0.0.0/8'
```

7.1.1 Mine Functions Aliases

Function aliases can be used to provide usage intentions or to allow multiple calls of the same function with different arguments.

New in version 2014.7.

```
mine_functions:
  network.ip_addrs: [eth0]
  networkplus.internal_ip_addrs: []
  internal_ip_addrs:
    mine_function: network.ip_addrs
    cidr: 192.168.0.0/16
  loopback_ip_addrs:
    mine_function: network.ip_addrs
    lo: True
```

7.2 Mine Interval

The Salt Mine functions are executed when the minion starts and at a given interval by the scheduler. The default interval is every 60 minutes and can be adjusted for the minion via the *mine_interval* option:

```
mine_interval: 60
```

7.3 Mine in Salt-SSH

As of the 2015.5.0 release of salt, salt-ssh supports `mine.get`.

Because the minions cannot provide their own `mine_functions` configuration, we retrieve the args for specified mine functions in one of three places, searched in the following order:

1. Roster data
2. Pillar
3. Master config

The `mine_functions` are formatted exactly the same as in normal salt, just stored in a different location. Here is an example of a flat roster containing `mine_functions`:

```
test:
  host: 104.237.131.248
  user: root
  mine_functions:
    cmd.run: ['echo "hello!"']
    network.ip_addrs:
      interface: eth0
```

Note: Because of the differences in the architecture of salt-ssh, `mine.get` calls are somewhat inefficient. Salt must make a new salt-ssh call to each of the minions in question to retrieve the requested data, much like a publish call. However, unlike publish, it must run the requested function as a wrapper function, so we can retrieve the function args from the pillar of the minion in question. This results in a non-trivial delay in retrieving the requested data.

7.4 Example

One way to use data from Salt Mine is in a State. The values can be retrieved via Jinja and used in the SLS file. The following example is a partial HAProxy configuration file and pulls IP addresses from all minions with the ``web'' grain to add them to the pool of load balanced servers.

`/srv/pillar/top.sls`:

```
base:
  'G@roles:web':
    - web
```

`/srv/pillar/web.sls`:

```
mine_functions:
  network.ip_addrs: [eth0]
```

`/etc/salt/minion.d/mine.conf`:

```
mine_interval: 5
```

/srv/salt/haproxy.sls:

```
haproxy_config:
  file.managed:
    - name: /etc/haproxy/config
    - source: salt://haproxy_config
    - template: jinja
```

/srv/salt/haproxy_config:

```
<...file contents snipped...>

{% for server, addrs in salt['mine.get']('roles:web', 'network.ip_addrs', expr_form='grain').items()
server {{ server }} {{ addrs[0] }}:80 check
{% endfor %}

<...file contents snipped...>
```

CHAPTER 8

External Authentication System

Salt's External Authentication System (eAuth) allows for Salt to pass through command authorization to any external authentication system, such as PAM or LDAP.

8.1 Access Control System

The external authentication system allows for specific users to be granted access to execute specific functions on specific minions. Access is configured in the master configuration file and uses the *access control system*:

```
external_auth:
  pam:
    thatch:
      - 'web*':
        - test.*
        - network.*
    steve:
      - .*
```

The above configuration allows the user `thatch` to execute functions in the test and network modules on the minions that match the web* target. User `steve` is given unrestricted access to minion commands.

Note: The PAM module does not allow authenticating as `root`.

To allow access to *wheel modules* or *runner modules* the following `@` syntax must be used:

```
external_auth:
  pam:
    thatch:
      - '@wheel'    # to allow access to all wheel modules
      - '@runner'   # to allow access to all runner modules
      - '@jobs'     # to allow access to the jobs runner and/or wheel module
```

Note: The runner/wheel markup is different, since there are no minions to scope the acl to.

Note: Globs will not match wheel or runners! They must be explicitly allowed with @wheel or @runner.

The external authentication system can then be used from the command-line by any user on the same system as the master with the `-a` option:

```
$ salt -a pam web\* test.ping
```

The system will ask the user for the credentials required by the authentication system and then publish the command.

To apply permissions to a group of users in an external authentication system, append a % to the ID:

```
external_auth:
  pam:
    admins%:
      - '*':
        - 'pkg.*'
```

8.2 Tokens

With external authentication alone, the authentication credentials will be required with every call to Salt. This can be alleviated with Salt tokens.

Tokens are short term authorizations and can be easily created by just adding a `-T` option when authenticating:

```
$ salt -T -a pam web\* test.ping
```

Now a token will be created that has a expiration of 12 hours (by default). This token is stored in a file named `.salt_token` in the active user's home directory.

Once the token is created, it is sent with all subsequent communications. User authentication does not need to be entered again until the token expires.

Token expiration time can be set in the Salt master config file.

8.3 LDAP and Active Directory

Salt supports both user and group authentication for LDAP (and Active Directory accessed via its LDAP interface)

LDAP configuration happens in the Salt master configuration file.

Server configuration values and their defaults:

```
auth.ldap.server: localhost
auth.ldap.port: 389
auth.ldap.tls: False
auth.ldap.scope: 2
auth.ldap.uri: ''
auth.ldap.tls: False
auth.ldap.no_verify: False
auth.ldap.anonymous: False
auth.ldap.groupou: 'Groups'
auth.ldap.groupclass: 'posixGroup'
auth.ldap.accountattributename: 'memberUid'

# These are only for Active Directory
auth.ldap.activedirectory: False
auth.ldap.persontype: 'person'
```

Salt also needs to know which Base DN to search for users and groups and the DN to bind to:

```
auth.ldap.basedn: dc=saltstack,dc=com
auth.ldap.binddn: cn=admin,dc=saltstack,dc=com
```

To bind to a DN, a password is required

```
auth.ldap.bindpw: mypassword
```

Salt uses a filter to find the DN associated with a user. Salt substitutes the `{{ username }}` value for the username when querying LDAP

```
auth.ldap.filter: uid={{ username }}
```

For OpenLDAP, to determine group membership, one can specify an OU that contains group data. This is prepended to the basedn to create a search path. Then the results are filtered against `auth.ldap.groupclass`, default `posixGroup`, and the account's `name' attribute, `memberUid` by default.

```
auth.ldap.groupou: Groups
```

Active Directory handles group membership differently, and does not utilize the `groupou` configuration variable. AD needs the following options in the master config:

```
auth.ldap.activedirectory: True
auth.ldap.filter: sAMAccountName={{username}}
auth.ldap.accountattributename: sAMAccountName
auth.ldap.groupclass: group
auth.ldap.persontype: person
```

To determine group membership in AD, the username and password that is entered when LDAP is requested as the eAuth mechanism on the command line is used to bind to AD's LDAP interface. If this fails, then it doesn't matter what groups the user belongs to, he or she is denied access. Next, the distinguishedName of the user is looked up with the following LDAP search:

```
(&(<value of auth.ldap.accountattributename>={{username}})
  (objectClass=<value of auth.ldap.persontype>)
)
```

This should return a distinguishedName that we can use to filter for group membership. Then the following LDAP quey is executed:

```
(&(member=<distinguishedName from search above>)
  (objectClass=<value of auth.ldap.groupclass>)
)
```

```
external_auth:
  ldap:
    test_ldap_user:
      - '*':
        - test.ping
```

To configure an LDAP group, append a % to the ID:

```
external_auth:
ldap:
    test_ldap_group%:
      - '*':
        - test.echo
```

CHAPTER 9

Job Management

New in version 0.9.7.

Since Salt executes jobs running on many systems, Salt needs to be able to manage jobs running on many systems.

9.1 The Minion proc System

Salt Minions maintain a *proc* directory in the Salt `cachedir`. The *proc* directory maintains files named after the executed job ID. These files contain the information about the current running jobs on the minion and allow for jobs to be looked up. This is located in the *proc* directory under the cachedir, with a default configuration it is under `/var/cache/salt/proc`.

9.2 Functions in the saltutil Module

Salt 0.9.7 introduced a few new functions to the *saltutil* module for managing jobs. These functions are:

1. `running` Returns the data of all running jobs that are found in the *proc* directory.
2. `find_job` Returns specific data about a certain job based on job id.
3. `signal_job` Allows for a given jid to be sent a signal.
4. `term_job` Sends a termination signal (SIGTERM, 15) to the process controlling the specified job.
5. `kill_job` Sends a kill signal (SIGKILL, 9) to the process controlling the specified job.

These functions make up the core of the back end used to manage jobs at the minion level.

9.3 The jobs Runner

A convenience runner front end and reporting system has been added as well. The jobs runner contains functions to make viewing data easier and cleaner.

The jobs runner contains a number of functions...

9.3.1 active

The active function runs saltutil.running on all minions and formats the return data about all running jobs in a much more usable and compact format. The active function will also compare jobs that have returned and jobs that are still running, making it easier to see what systems have completed a job and what systems are still being waited on.

```
# salt-run jobs.active
```

9.3.2 lookup_jid

When jobs are executed the return data is sent back to the master and cached. By default it is cached for 24 hours, but this can be configured via the `keep_jobs` option in the master configuration. Using the lookup_jid runner will display the same return data that the initial job invocation with the salt command would display.

```
# salt-run jobs.lookup_jid <job id number>
```

9.3.3 list_jobs

Before finding a historic job, it may be required to find the job id. list_jobs will parse the cached execution data and display all of the job data for jobs that have already, or partially returned.

```
# salt-run jobs.list_jobs
```

9.4 Scheduling Jobs

In Salt versions greater than 0.12.0, the scheduling system allows incremental executions on minions or the master. The schedule system exposes the execution of any execution function on minions or any runner on the master.

Scheduling is enabled via the `schedule` option on either the master or minion config files, or via a minion's pillar data. Schedules that are impletemented via pillar data, only need to refresh the minion's pillar data, for example by using `saltutil.refresh_pillar`. Schedules implemented in the master or minion config have to restart the application in order for the schedule to be implemented.

Note: The scheduler executes different functions on the master and minions. When running on the master the functions reference runner functions, when running on the minion the functions specify execution functions.

A scheduled run has no output on the minion unless the config is set to info level or higher. Refer to *minion logging settings*.

Specify `maxrunning` to ensure that there are no more than N copies of a particular routine running. Use this for jobs that may be long-running and could step on each other or otherwise double execute. The default for `maxrunning` is 1.

States are executed on the minion, as all states are. You can pass positional arguments and provide a yaml dict of named arguments.

```
schedule:
  job1:
    function: state.sls
    seconds: 3600
    args:
      - httpd
```

```
    kwargs:
      test: True
```

This will schedule the command: state.sls httpd test=True every 3600 seconds (every hour)

```
schedule:
  job1:
    function: state.sls
    seconds: 3600
    args:
      - httpd
    kwargs:
      test: True
    splay: 15
```

This will schedule the command: state.sls httpd test=True every 3600 seconds (every hour) splaying the time between 0 and 15 seconds

```
schedule:
  job1:
    function: state.sls
    seconds: 3600
    args:
      - httpd
    kwargs:
      test: True
    splay:
      start: 10
      end: 15
```

This will schedule the command: state.sls httpd test=True every 3600 seconds (every hour) splaying the time between 10 and 15 seconds

New in version 2014.7.0.

Frequency of jobs can also be specified using date strings supported by the python dateutil library. This requires python-dateutil to be installed on the minion.

```
schedule:
  job1:
    function: state.sls
    args:
      - httpd
    kwargs:
      test: True
    when: 5:00pm
```

This will schedule the command: state.sls httpd test=True at 5:00pm minion localtime.

```
schedule:
  job1:
    function: state.sls
    args:
      - httpd
    kwargs:
      test: True
    when:
        - Monday 5:00pm
        - Tuesday 3:00pm
        - Wednesday 5:00pm
```

```
        - Thursday 3:00pm
        - Friday 5:00pm
```

This will schedule the command: state.sls httpd test=True at 5pm on Monday, Wednesday, and Friday, and 3pm on Tuesday and Thursday.

```
schedule:
  job1:
    function: state.sls
    seconds: 3600
    args:
      - httpd
    kwargs:
      test: True
    range:
        start: 8:00am
        end: 5:00pm
```

This will schedule the command: state.sls httpd test=True every 3600 seconds (every hour) between the hours of 8am and 5pm. The range parameter must be a dictionary with the date strings using the dateutil format. This requires python-dateutil to be installed on the minion.

New in version 2014.7.0.

The scheduler also supports ensuring that there are no more than N copies of a particular routine running. Use this for jobs that may be long-running and could step on each other or pile up in case of infrastructure outage.

The default for maxrunning is 1.

```
schedule:
  long_running_job:
      function: big_file_transfer
      jid_include: True
```

9.5 States

```
schedule:
  log-loadavg:
    function: cmd.run
    seconds: 3660
    args:
      - 'logger -t salt < /proc/loadavg'
    kwargs:
      stateful: False
      shell: \bin\sh
```

9.6 Highstates

To set up a highstate to run on a minion every 60 minutes set this in the minion config or pillar:

```
schedule:
  highstate:
    function: state.highstate
    minutes: 60
```

Time intervals can be specified as seconds, minutes, hours, or days.

9.7 Runners

Runner executions can also be specified on the master within the master configuration file:

```
schedule:
  overstate:
    function: state.over
    seconds: 35
    minutes: 30
    hours: 3
```

The above configuration will execute the state.over runner every 3 hours, 30 minutes and 35 seconds, or every 12,635 seconds.

9.8 Scheduler With Returner

The scheduler is also useful for tasks like gathering monitoring data about a minion, this schedule option will gather status data and send it to a MySQL returner database:

```
schedule:
  uptime:
    function: status.uptime
    seconds: 60
    returner: mysql
  meminfo:
    function: status.meminfo
    minutes: 5
    returner: mysql
```

Since specifying the returner repeatedly can be tiresome, the `schedule_returner` option is available to specify one or a list of global returners to be used by the minions when scheduling.

In Salt versions greater than 0.12.0, the scheduling system allows incremental executions on minions or the master. The schedule system exposes the execution of any execution function on minions or any runner on the master.

Scheduling is enabled via the `schedule` option on either the master or minion config files, or via a minion's pillar data. Schedules that are impletemented via pillar data, only need to refresh the minion's pillar data, for example by using `saltutil.refresh_pillar`. Schedules implemented in the master or minion config have to restart the application in order for the schedule to be implemented.

Note: The scheduler executes different functions on the master and minions. When running on the master the functions reference runner functions, when running on the minion the functions specify execution functions.

A scheduled run has no output on the minion unless the config is set to info level or higher. Refer to *minion logging settings*.

Specify `maxrunning` to ensure that there are no more than N copies of a particular routine running. Use this for jobs that may be long-running and could step on each other or otherwise double execute. The default for `maxrunning` is 1.

States are executed on the minion, as all states are. You can pass positional arguments and provide a yaml dict of named arguments.

```
schedule:
  job1:
    function: state.sls
    seconds: 3600
    args:
      - httpd
    kwargs:
      test: True
```

This will schedule the command: state.sls httpd test=True every 3600 seconds (every hour)

```
schedule:
  job1:
    function: state.sls
    seconds: 3600
    args:
      - httpd
    kwargs:
      test: True
    splay: 15
```

This will schedule the command: state.sls httpd test=True every 3600 seconds (every hour) splaying the time between 0 and 15 seconds

```
schedule:
  job1:
    function: state.sls
    seconds: 3600
    args:
      - httpd
    kwargs:
      test: True
    splay:
      start: 10
      end: 15
```

This will schedule the command: state.sls httpd test=True every 3600 seconds (every hour) splaying the time between 10 and 15 seconds

New in version 2014.7.0.

Frequency of jobs can also be specified using date strings supported by the python dateutil library. This requires python-dateutil to be installed on the minion.

```
schedule:
  job1:
    function: state.sls
    args:
      - httpd
    kwargs:
      test: True
    when: 5:00pm
```

This will schedule the command: state.sls httpd test=True at 5:00pm minion localtime.

```
schedule:
  job1:
    function: state.sls
    args:
      - httpd
```

```
    kwargs:
      test: True
    when:
        - Monday 5:00pm
        - Tuesday 3:00pm
        - Wednesday 5:00pm
        - Thursday 3:00pm
        - Friday 5:00pm
```

This will schedule the command: state.sls httpd test=True at 5pm on Monday, Wednesday, and Friday, and 3pm on Tuesday and Thursday.

```
schedule:
  job1:
    function: state.sls
    seconds: 3600
    args:
      - httpd
    kwargs:
      test: True
    range:
        start: 8:00am
        end: 5:00pm
```

This will schedule the command: state.sls httpd test=True every 3600 seconds (every hour) between the hours of 8am and 5pm. The range parameter must be a dictionary with the date strings using the dateutil format. This requires python-dateutil to be installed on the minion.

New in version 2014.7.0.

The scheduler also supports ensuring that there are no more than N copies of a particular routine running. Use this for jobs that may be long-running and could step on each other or pile up in case of infrastructure outage.

The default for maxrunning is 1.

```
schedule:
  long_running_job:
      function: big_file_transfer
      jid_include: True
```

9.8.1 States

```
schedule:
  log-loadavg:
    function: cmd.run
    seconds: 3660
    args:
      - 'logger -t salt < /proc/loadavg'
    kwargs:
      stateful: False
      shell: \bin\sh
```

9.8.2 Highstates

To set up a highstate to run on a minion every 60 minutes set this in the minion config or pillar:

```
schedule:
  highstate:
    function: state.highstate
    minutes: 60
```

Time intervals can be specified as seconds, minutes, hours, or days.

9.8.3 Runners

Runner executions can also be specified on the master within the master configuration file:

```
schedule:
  overstate:
    function: state.over
    seconds: 35
    minutes: 30
    hours: 3
```

The above configuration will execute the state.over runner every 3 hours, 30 minutes and 35 seconds, or every 12,635 seconds.

9.8.4 Scheduler With Returner

The scheduler is also useful for tasks like gathering monitoring data about a minion, this schedule option will gather status data and send it to a MySQL returner database:

```
schedule:
  uptime:
    function: status.uptime
    seconds: 60
    returner: mysql
  meminfo:
    function: status.meminfo
    minutes: 5
    returner: mysql
```

Since specifying the returner repeatedly can be tiresome, the `schedule_returner` option is available to specify one or a list of global returners to be used by the minions when scheduling.

CHAPTER 10

Salt Event System

The Salt Event System is used to fire off events enabling third party applications or external processes to react to behavior within Salt.

The event system is comprised of a two primary components:

- The event sockets which publishes events.
- The event library which can listen to events and send events into the salt system.

10.1 Event types

10.1.1 Salt Master Events

These events are fired on the Salt Master event bus. This list is **not** comprehensive.

Authentication events

`salt/auth`
Fired when a minion performs an authentication check with the master.

Variables

- **`id`** -- The minion ID.
- **`act`** -- The current status of the minion key: `accept`, `pend`, `reject`.
- **`pub`** -- The minion public key.

Note: Minions fire auth events on fairly regular basis for a number of reasons. Writing reactors to respond to events through the auth cycle can lead to infinite reactor event loops (minion tries to auth, reactor responds by doing something that generates another auth event, minion sends auth event, etc.). Consider reacting to `salt/key` or `salt/minion/<MID>/start` or firing a custom event tag instead.

Start events

`salt/minion/<MID>/start`
Fired every time a minion connects to the Salt master.

Variables **`id`** -- The minion ID.

Key events

salt/key

Fired when accepting and rejecting minions keys on the Salt master.

Variables

- **id** -- The minion ID.
- **act** -- The new status of the minion key: `accept`, `pend`, `reject`.

Warning: If a master is in `auto_accept mode`, `salt/key` events will not be fired when the keys are accepted. In addition, pre-seeding keys (like happens through *Salt-Cloud*) will not cause firing of these events.

Job events

salt/job/<JID>/new

Fired as a new job is sent out to minions.

Variables

- **jid** -- The job ID.
- **tgt** -- The target of the job: `*`, a minion ID, `G@os_family:RedHat`, etc.
- **tgt_type** -- The type of targeting used: `glob`, `grain`, `compound`, etc.
- **fun** -- The function to run on minions: `test.ping`, `network.interfaces`, etc.
- **arg** -- A list of arguments to pass to the function that will be called.
- **minions** -- A list of minion IDs that Salt expects will return data for this job.
- **user** -- The name of the user that ran the command as defined in Salt's Client ACL or external auth.

salt/job/<JID>/ret/<MID>

Fired each time a minion returns data for a job.

Variables

- **id** -- The minion ID.
- **jid** -- The job ID.
- **retcode** -- The return code for the job.
- **fun** -- The function the minion ran. E.g., `test.ping`.
- **return** -- The data returned from the execution module.

Presence events

salt/presence/present

Events fired on a regular interval about currently connected, newly connected, or recently disconnected minions. Requires the `presence_events` setting to be enabled.

Variables **present** -- A list of minions that are currently connected to the Salt master.

salt/presence/change

Fired when the Presence system detects new minions connect or disconnect.

Variables

- **new** -- A list of minions that have connected since the last presence event.
- **lost** -- A list of minions that have disconnected since the last presence event.

Cloud Events

Unlike other Master events, `salt-cloud` events are not fired on behalf of a Salt Minion. Instead, `salt-cloud` events are fired on behalf of a VM. This is because the minion-to-be may not yet exist to fire events to or also may have been destroyed.

This behavior is reflected by the `name` variable in the event data for `salt-cloud` events as compared to the `id` variable for Salt Minion-triggered events.

salt/cloud/<VM NAME>/creating
Fired when salt-cloud starts the VM creation process.

Variables

- **name** -- the name of the VM being created.
- **event** -- description of the event.
- **provider** -- the cloud provider of the VM being created.
- **profile** -- the cloud profile for the VM being created.

salt/cloud/<VM NAME>/deploying
Fired when the VM is available and salt-cloud begins deploying Salt to the new VM.

Variables

- **name** -- the name of the VM being created.
- **event** -- description of the event.
- **kwargs** -- options available as the deploy script is invoked: `conf_file`, `deploy_command`, `display_ssh_output`, `host`, `keep_tmp`, `key_filename`, `make_minion`, `minion_conf`, `name`, `parallel`, `preseed_minion_keys`, `script`, `script_args`, `script_env`, `sock_dir`, `start_action`, `sudo`, `tmp_dir`, `tty`, `username`

salt/cloud/<VM NAME>/requesting
Fired when salt-cloud sends the request to create a new VM.

Variables

- **event** -- description of the event.
- **location** -- the location of the VM being requested.
- **kwargs** -- options available as the VM is being requested: `Action`, `ImageId`, `InstanceType`, `KeyName`, `MaxCount`, `MinCount`, `SecurityGroup.1`

salt/cloud/<VM NAME>/querying
Fired when salt-cloud queries data for a new instance.

Variables

- **event** -- description of the event.
- **instance_id** -- the ID of the new VM.

salt/cloud/<VM NAME>/tagging
Fired when salt-cloud tags a new instance.

Variables

- **event** -- description of the event.
- **tags** -- tags being set on the new instance.

salt/cloud/<VM NAME>/waiting_for_ssh
Fired while the salt-cloud deploy process is waiting for ssh to become available on the new instance.

Variables

- **event** -- description of the event.
- **ip_address** -- IP address of the new instance.

salt/cloud/<VM NAME>/deploy_script
Fired once the deploy script is finished.

Variables **event** -- description of the event.

salt/cloud/<VM NAME>/created
Fired once the new instance has been fully created.

Variables

- **name** -- the name of the VM being created.
- **event** -- description of the event.
- **instance_id** -- the ID of the new instance.
- **provider** -- the cloud provider of the VM being created.
- **profile** -- the cloud profile for the VM being created.

salt/cloud/<VM NAME>/destroying
Fired when salt-cloud requests the destruction of an instance.

Variables

- **name** -- the name of the VM being created.
- **event** -- description of the event.
- **instance_id** -- the ID of the new instance.

salt/cloud/<VM NAME>/destroyed
Fired when an instance has been destroyed.

Variables

- **name** -- the name of the VM being created.
- **event** -- description of the event.
- **instance_id** -- the ID of the new instance.

10.2 Listening for Events

Salt's Event Bus is used heavily within Salt and it is also written to integrate heavily with existing tooling and scripts. There is a variety of ways to consume it.

10.2.1 From the CLI

The quickest way to watch the event bus is by calling the `state.event runner`:

```
salt-run state.event pretty=True
```

That runner is designed to interact with the event bus from external tools and shell scripts. See the documentation for more examples.

10.2.2 Remotely via the REST API

Salt's event bus can be consumed `salt.netapi.rest_cherrypy.app.Events` as an HTTP stream from external tools or services.

```
curl -SsNk https://salt-api.example.com:8000/events?token=05A3
```

10.2.3 From Python

Python scripts can access the event bus only as the same system user that Salt is running as.

The event system is accessed via the event library and can only be accessed by the same system user that Salt is running as. To listen to events a SaltEvent object needs to be created and then the get_event function needs to be run. The SaltEvent object needs to know the location that the Salt Unix sockets are kept. In the configuration this is the `sock_dir` option. The `sock_dir` option defaults to ``/var/run/salt/master'' on most systems.

The following code will check for a single event:

```
import salt.config
import salt.utils.event

opts = salt.config.client_config('/etc/salt/master')

event = salt.utils.event.get_event(
        'master',
        sock_dir=opts['sock_dir'],
        transport=opts['transport'],
        opts=opts)

data = event.get_event()
```

Events will also use a ``tag''. Tags allow for events to be filtered by prefix. By default all events will be returned. If only authentication events are desired, then pass the tag ``salt/auth''.

The `get_event` method has a default poll time assigned of 5 seconds. To change this time set the ``wait'' option.

The following example will only listen for auth events and will wait for 10 seconds instead of the default 5.

```
data = event.get_event(wait=10, tag='salt/auth')
```

To retrieve the tag as well as the event data, pass `full=True`:

```
evdata = event.get_event(wait=10, tag='salt/job', full=True)

tag, data = evdata['tag'], evdata['data']
```

Instead of looking for a single event, the `iter_events` method can be used to make a generator which will continually yield salt events.

The iter_events method also accepts a tag but not a wait time:

```
for data in event.iter_events(tag='salt/auth'):
    print(data)
```

And finally event tags can be globbed, such as they can be in the Reactor, using the fnmatch library.

```
import fnmatch

import salt.config
import salt.utils.event

opts = salt.config.client_config('/etc/salt/master')

sevent = salt.utils.event.get_event(
        'master',
        sock_dir=opts['sock_dir'],
        transport=opts['transport'],
        opts=opts)

while True:
    ret = sevent.get_event(full=True)
    if ret is None:
        continue

    if fnmatch.fnmatch(ret['tag'], 'salt/job/*/ret/*'):
        do_something_with_job_return(ret['data'])
```

10.3 Firing Events

It is possible to fire events on either the minion's local bus or to fire events intended for the master.

To fire a local event from the minion on the command line call the `event.fire` execution function:

```
salt-call event.fire '{"data": "message to be sent in the event"}' 'tag'
```

To fire an event to be sent up to the master from the minion call the `event.send` execution function. Remember YAML can be used at the CLI in function arguments:

```
salt-call event.send 'myco/mytag/success' '{success: True, message: "It works!"}'
```

If a process is listening on the minion, it may be useful for a user on the master to fire an event to it:

```
# Job on minion
import salt.utils.event

event = salt.utils.event.MinionEvent(**__opts__)

for evdata in event.iter_events(tag='customtag/'):
    return evdata # do your processing here...
```

```
salt minionname event.fire '{"data": "message for the minion"}' 'customtag/african/unladen'
```

10.4 Firing Events from Python

10.4.1 From Salt execution modules

Events can be very useful when writing execution modules, in order to inform various processes on the master when a certain task has taken place. This is easily done using the normal cross-calling syntax:

```
# /srv/salt/_modules/my_custom_module.py

def do_something():
    '''
    Do something and fire an event to the master when finished

    CLI Example::

        salt '*' my_custom_module:do_something
    '''
    # do something!
    __salt__['event.send']('myco/my_custom_module/finished', {
        'finished': True,
        'message': "The something is finished!",
    })
```

10.4.2 From Custom Python Scripts

Firing events from custom Python code is quite simple and mirrors how it is done at the CLI:

```
import salt.client

caller = salt.client.Caller()

caller.sminion.functions['event.send'](
    'myco/myevent/success',
    {
        'success': True,
        'message': "It works!",
    }
)
```

CHAPTER 11

Salt Syndic

The Salt Syndic interface is a powerful tool which allows for the construction of Salt command topologies. A basic Salt setup has a Salt Master commanding a group of Salt Minions. The Syndic interface is a special passthrough minion, it is run on a master and connects to another master, then the master that the Syndic minion is listening to can control the minions attached to the master running the syndic.

The intent for supporting many layouts is not presented with the intent of supposing the use of any single topology, but to allow a more flexible method of controlling many systems.

11.1 Configuring the Syndic

Since the Syndic only needs to be attached to a higher level master the configuration is very simple. On a master that is running a syndic to connect to a higher level master the `syndic_master` option needs to be set in the master config file. The `syndic_master` option contains the hostname or IP address of the master server that can control the master that the syndic is running on.

The master that the syndic connects to sees the syndic as an ordinary minion, and treats it as such. the higher level master will need to accept the syndic's minion key like any other minion. This master will also need to set the `order_masters` value in the configuration to `True`. The `order_masters` option in the config on the higher level master is very important, to control a syndic extra information needs to be sent with the publications, the `order_masters` option makes sure that the extra data is sent out.

To sum up, you have those configuration options available on the master side:

- `syndic_master`: MasterOfMaster ip/address
- `syndic_master_port`: MasterOfMaster ret_port
- `syndic_log_file`: path to the logfile (absolute or not)
- `syndic_pidfile`: path to the pidfile (absolute or not)

Each Syndic must provide its own `file_roots` directory. Files will not be automatically transferred from the master-master.

11.2 Running the Syndic

The Syndic is a separate daemon that needs to be started on the master that is controlled by a higher master. Starting the Syndic daemon is the same as starting the other Salt daemons.

```
# salt-syndic
```

Note: If you have an exceptionally large infrastructure or many layers of syndics, you may find that the CLI doesn't wait long enough for the syndics to return their events. If you think this is the case, you can set the `syndic_wait` value in the upper master config. The default value is `1`, and should work for the majority of deployments.

11.3 Topology

The `salt-syndic` is little more than a command and event forwarder. When a command is issued from a higher-level master, it will be received by the configured syndics on lower-level masters, and propagated to to their minions, and other syndics that are bound to them further down in the hierarchy. When events and job return data are generated by minions, they aggregated back, through the same syndic(s), to the master which issued the command.

The master sitting at the top of the hierarchy (the Master of Masters) will *not* be running the `salt-syndic` daemon. It will have the `salt-master` daemon running, and optionally, the `salt-minion` daemon. Each syndic connected to an upper-level master will have both the `salt-master` and the `salt-syndic` daemon running, and optionally, the `salt-minion` daemon.

Nodes on the lowest points of the hierarchy (minions which do not propagate data to another level) will only have the `salt-minion` daemon running. There is no need for either `salt-master` or `salt-syndic` to be running on a standard minion.

11.4 Syndic and the CLI

In order for the high-level master to return information from minions that are below the syndic(s), the CLI requires a short wait time in order to allow the syndic(s) to gather responses from their minions. This value is defined in the ``syndic_wait` and has a default of five seconds.

While it is possible to run a syndic without a minion installed on the same machine, it is recommended, for a faster CLI response time, to do so. Without a minion installed on the syndic, the timeout value of `syndic_wait` increases significantly - about three-fold. With a minion installed on the syndic, the CLI timeout resides at the value defined in `syndic_wait`.

Note: To reduce the amount of time the CLI waits for minions to respond, install a minion on the syndic or tune the value of the `syndic_wait` configuration.

CHAPTER 12

Salt Proxy Minion Documentation

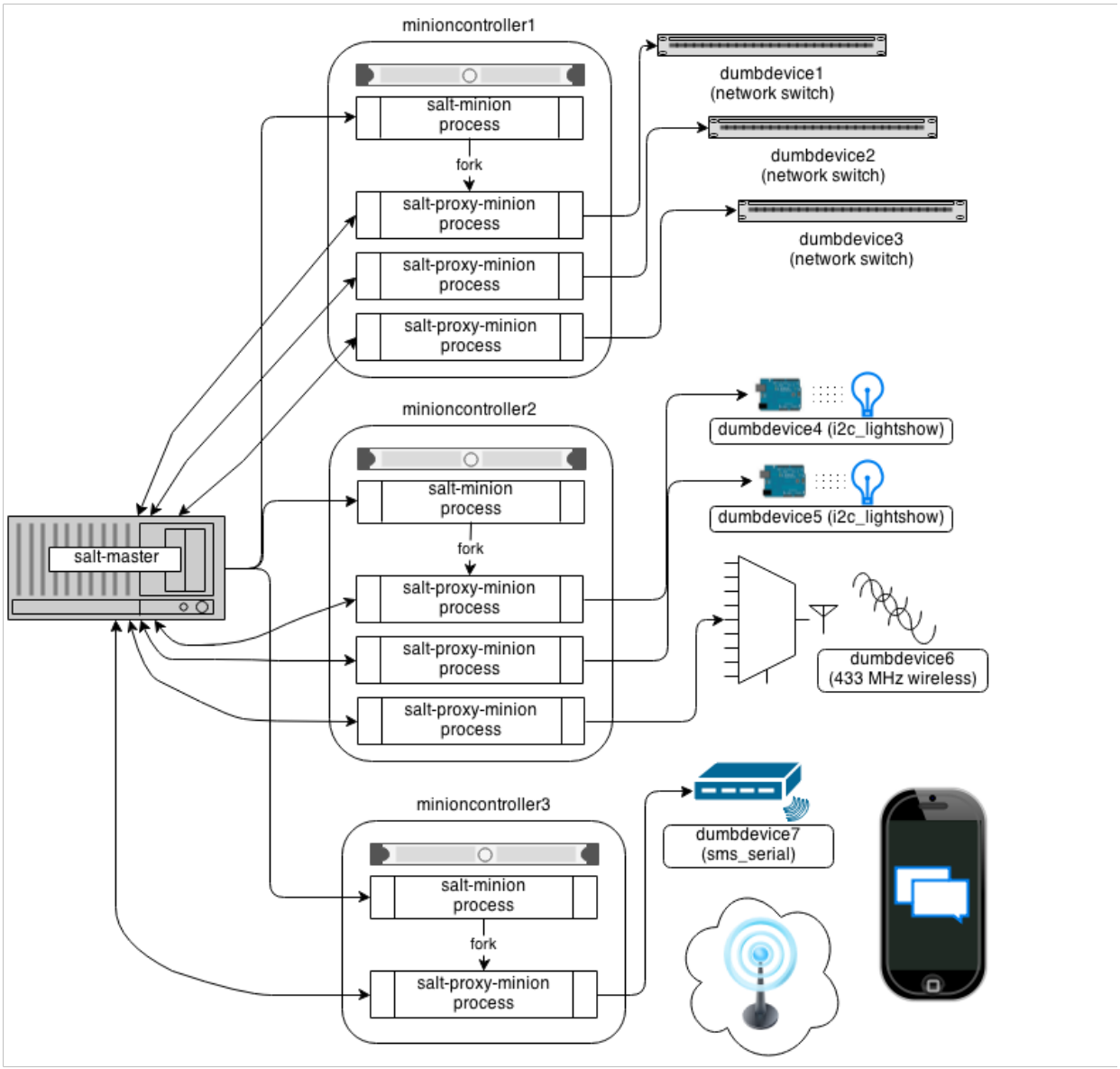
Proxy minions are a developing Salt feature that enables controlling devices that, for whatever reason, cannot run a standard salt-minion. Examples include network gear that has an API but runs a proprietary OS, devices with limited CPU or memory, or devices that could run a minion, but for security reasons, will not.

Proxy minions are not an ``out of the box'' feature. Because there are an infinite number of controllable devices, you will most likely have to write the interface yourself. Fortunately, this is only as difficult as the actual interface to the proxied device. Devices that have an existing Python module (PyUSB for example) would be relatively simple to interface. Code to control a device that has an HTML REST-based interface should be easy. Code to control your typical housecat would be excellent source material for a PhD thesis.

Salt proxy-minions provide the `plumbing' that allows device enumeration and discovery, control, status, remote execution, and state management.

12.1 Getting Started

The following diagram may be helpful in understanding the structure of a Salt installation that includes proxy-minions:



The key thing to remember is the left-most section of the diagram. Salt's nature is to have a minion connect to a master, then the master may control the minion. However, for proxy minions, the target device cannot run a minion, and thus must rely on a separate minion to fire up the proxy-minion and make the initial and persistent connection.

After the proxy minion is started and initiates its connection to the `dumb' device, it connects back to the salt-master and ceases to be affiliated in any way with the minion that started it.

To create support for a proxied device one needs to create four things:

1. The proxytype connection class (located in salt/proxy).
2. The grains support code (located in salt/grains).
3. *Salt modules* specific to the controlled device.
4. *Salt states* specific to the controlled device.

12.1.1 Configuration parameters on the master

Proxy minions require no configuration parameters in /etc/salt/master.

Salt's Pillar system is ideally suited for configuring proxy-minions. Proxies can either be designated via a pillar file in pillar_roots, or through an external pillar. External pillars afford the opportunity for interfacing with a configuration management system, database, or other knowledgeable system that that may already contain all the details of proxy targets. To use static files in pillar_roots, pattern your files after the following examples, which are based on the diagram above:

/srv/pillar/top.sls

```
base:
  minioncontroller1:
    - networkswitches
  minioncontroller2:
    - reallydumbdevices
  minioncontroller3:
    - smsgateway
```

/srv/pillar/networkswitches.sls

```
proxy:
  dumbdevice1:
    proxytype: networkswitch
    host: 172.23.23.5
    username: root
    passwd: letmein
  dumbdevice2:
    proxytype: networkswitch
    host: 172.23.23.6
    username: root
    passwd: letmein
  dumbdevice3:
    proxytype: networkswitch
    host: 172.23.23.7
    username: root
    passwd: letmein
```

/srv/pillar/reallydumbdevices.sls

```
proxy:
  dumbdevice4:
    proxytype: i2c_lightshow
    i2c_address: 1
  dumbdevice5:
    proxytype: i2c_lightshow
    i2c_address: 2
  dumbdevice6:
    proxytype: 433mhz_wireless
```

/srv/pillar/smsgateway.sls

```
proxy:
  minioncontroller3:
    dumbdevice7:
      proxytype: sms_serial
      deventry: /dev/tty04
```

Note the contents of each minioncontroller key may differ widely based on the type of device that the proxy-minion is managing.

In the above example

- dumbdevices 1, 2, and 3 are network switches that have a management interface available at a particular IP address.
- dumbdevices 4 and 5 are very low-level devices controlled over an i2c bus. In this case the devices are physically connected to machine `minioncontroller2', and are addressable on the i2c bus at their respective i2c addresses.
- dumbdevice6 is a 433 MHz wireless transmitter, also physically connected to minioncontroller2
- dumbdevice7 is an SMS gateway connected to machine minioncontroller3 via a serial port.

Because of the way pillar works, each of the salt-minions that fork off the proxy minions will only see the keys specific to the proxies it will be handling. In other words, from the above example, only minioncontroller1 will see the connection information for dumbdevices 1, 2, and 3. Minioncontroller2 will see configuration data for dumbdevices 4, 5, and 6, and minioncontroller3 will be privy to dumbdevice7.

Also, in general, proxy-minions are lightweight, so the machines that run them could conceivably control a large number of devices. The example above is just to illustrate that it is possible for the proxy services to be spread across many machines if necessary, or intentionally run on machines that need to control devices because of some physical interface (e.g. i2c and serial above). Another reason to divide proxy services might be security. In more secure environments only certain machines may have a network path to certain devices.

Now our salt-minions know if they are supposed to spawn a proxy-minion process to control a particular device. That proxy-minion process will initiate a connection back to the master to enable control.

12.1.2 Proxytypes

A proxytype is a Python class called `Proxyconn' that encapsulates all the code necessary to interface with a device. Proxytypes are located inside the salt.proxy module. At a minimum a proxytype object must implement the following methods:

`proxytype(self)`: Returns a string with the name of the proxy type.

`proxyconn(self, **kwargs)`: Provides the primary way to connect and communicate with the device. Some proxyconns instantiate a particular object that opens a network connection to a device and leaves the connection open for communication. Others simply abstract a serial connection or even implement endpoints to communicate via REST over HTTP.

`id(self, opts)`: Returns a unique, unchanging id for the controlled device. This is the ``name'' of the device, and is used by the salt-master for targeting and key authentication.

Optionally, the class may define a `shutdown(self, opts)` method if the controlled device should be informed when the minion goes away cleanly.

It is highly recommended that the `test.ping` execution module also be defined for a proxytype. The code for `ping` should contact the controlled device and make sure it is really available.

Here is an example proxytype used to interface to Juniper Networks devices that run the Junos operating system. Note the additional library requirements--most of the ``hard part'' of talking to these devices is handled by the jnpr.junos, jnpr.junos.utils, and jnpr.junos.cfg modules.

```
# Import python libs
import logging
import os

import jnpr.junos
import jnpr.junos.utils
```

```
import jnpr.junos.cfg
HAS_JUNOS = True

class Proxyconn(object):


    def __init__(self, details):
        self.conn = jnpr.junos.Device(user=details['username'], host=details['host'], password=detai
        self.conn.open()
        self.conn.bind(cu=jnpr.junos.cfg.Resource)


    def proxytype(self):
        return 'junos'


    def id(self, opts):
        return self.conn.facts['hostname']


    def ping(self):
        return self.conn.connected


    def shutdown(self, opts):

        print('Proxy module {} shutting down!!'.format(opts['id']))
        try:
            self.conn.close()
        except Exception:
            pass
```

Grains are data about minions. Most proxied devices will have a paltry amount of data as compared to a typical Linux server. Because proxy-minions are started by a regular minion, they inherit a sizeable number of grain settings which can be useful, especially when targeting (PYTHONPATH, for example).

All proxy minions set a grain called `proxy'. If it is present, you know the minion is controlling another device. To add more grains to your proxy minion for a particular device, create a file in salt/grains named [proxytype].py and place inside it the different functions that need to be run to collect the data you are interested in. Here's an example:

12.2 The __proxyenabled__ directive

Salt states and execution modules, by, and large, cannot ``automatically'' work with proxied devices. Execution modules like `pkg` or `sqlite3` have no meaning on a network switch or a housecat. For a state/execution module to be available to a proxy-minion, the `__proxyenabled__` variable must be defined in the module as an array containing the names of all the proxytypes that this module can support. The array can contain the special value `*` to indicate that the module supports all proxies.

If no `__proxyenabled__` variable is defined, then by default, the state/execution module is unavailable to any proxy.

Here is an excerpt from a module that was modified to support proxy-minions:

```
def ping():

    if 'proxyobject' in __opts__:
```

```
        if 'ping' in __opts__['proxyobject'].__attr__():
            return __opts['proxyobject'].ping()
        else:
            return False
    else:
        return True
```

And then in salt.proxy.junos we find

```
def ping(self):

   return self.connected
```

The Junos API layer lacks the ability to do a traditional `ping', so the example simply checks the connection object field that indicates if the ssh connection was successfully made to the device.

CHAPTER 13

Windows Software Repository

The Salt Windows Software Repository provides a package manager and software repository similar to what is provided by yum and apt on Linux.

It permits the installation of software using the installers on remote windows machines. In many senses, the operation is similar to that of the other package managers salt is aware of:

- the `pkg.installed` and similar states work on Windows.
- the `pkg.install` and similar module functions work on Windows.
- each windows machine needs to have `pkg.refresh_db` executed against it to pick up the latest version of the package database.

High level differences to yum and apt are:

- The repository metadata (sls files) is hosted through either salt or git.
- Packages can be downloaded from within the salt repository, a git repository or from http(s) or ftp urls.
- No dependencies are managed. Dependencies between packages needs to be managed manually.

13.1 Operation

The install state/module function of the windows package manager works roughly as follows:

1. Execute `pkg.list_pkgs` and store the result
2. Check if any action needs to be taken. (i.e. compare required package and version against `pkg.list_pkgs` results)
3. If so, run the installer command.
4. Execute `pkg.list_pkgs` and compare to the result stored from before installation.
5. Success/Failure/Changes will be reported based on the differences between the original and final `pkg.list_pkgs` results.

If there are any problems in using the package manager it is likely to be due to the data in your sls files not matching the difference between the pre and post `pkg.list_pkgs` results.

13.2 Usage

By default, the Windows software repository is found at `/srv/salt/win/repo` This can be changed in the master config file (default location is `/etc/salt/master`) by modifying the `win_repo` variable. Each piece of software should have its own directory which contains the installers and a package definition file. This package definition file is a YAML file named `init.sls`.

The package definition file should look similar to this example for Firefox: `/srv/salt/win/repo/firefox/init.sls`

```
Firefox:
  17.0.1:
    installer: 'salt://win/repo/firefox/English/Firefox Setup 17.0.1.exe'
    full_name: Mozilla Firefox 17.0.1 (x86 en-US)
    locale: en_US
    reboot: False
    install_flags: ' -ms'
    uninstaller: '%ProgramFiles(x86)%/Mozilla Firefox/uninstall/helper.exe'
    uninstall_flags: ' /S'
  16.0.2:
    installer: 'salt://win/repo/firefox/English/Firefox Setup 16.0.2.exe'
    full_name: Mozilla Firefox 16.0.2 (x86 en-US)
    locale: en_US
    reboot: False
    install_flags: ' -ms'
    uninstaller: '%ProgramFiles(x86)%/Mozilla Firefox/uninstall/helper.exe'
    uninstall_flags: ' /S'
  15.0.1:
    installer: 'salt://win/repo/firefox/English/Firefox Setup 15.0.1.exe'
    full_name: Mozilla Firefox 15.0.1 (x86 en-US)
    locale: en_US
    reboot: False
    install_flags: ' -ms'
    uninstaller: '%ProgramFiles(x86)%/Mozilla Firefox/uninstall/helper.exe'
    uninstall_flags: ' /S'
```

More examples can be found here: https://github.com/saltstack/salt-winrepo

The version number and `full_name` need to match the output from `pkg.list_pkgs` so that the status can be verified when running highstate. Note: It is still possible to successfully install packages using `pkg.install` even if they don't match which can make this hard to troubleshoot.

```
salt 'test-2008' pkg.list_pkgs
test-2008
    ----------
    7-Zip 9.20 (x64 edition):
        9.20.00.0
    Microsoft .NET Framework 4 Client Profile:
        4.0.30319,4.0.30319
    Microsoft .NET Framework 4 Extended:
        4.0.30319,4.0.30319
    Microsoft Visual C++ 2008 Redistributable - x64 9.0.21022:
        9.0.21022
    Mozilla Firefox 17.0.1 (x86 en-US):
        17.0.1
    Mozilla Maintenance Service:
        17.0.1
    NSClient++ (x64):
```

```
        0.3.8.76
    Notepad++:
        6.4.2
    Salt Minion 0.16.0:
        0.16.0
```

If any of these preinstalled packages already exist in winrepo the full_name will be automatically renamed to their package name during the next update (running highstate or installing another package).

```
test-2008:
    ----------
    7zip:
        9.20.00.0
    Microsoft .NET Framework 4 Client Profile:
        4.0.30319,4.0.30319
    Microsoft .NET Framework 4 Extended:
        4.0.30319,4.0.30319
    Microsoft Visual C++ 2008 Redistributable - x64 9.0.21022:
        9.0.21022
    Mozilla Maintenance Service:
        17.0.1
    Notepad++:
        6.4.2
    Salt Minion 0.16.0:
        0.16.0
    firefox:
        17.0.1
    nsclient:
        0.3.9.328
```

Add `msiexec: True` if using an MSI installer requiring the use of `msiexec /i` to install and `msiexec /x` to uninstall.

The `install_flags` and `uninstall_flags` are flags passed to the software installer to cause it to perform a silent install. These can often be found by adding `/?` or `/h` when running the installer from the command line. A great resource for finding these silent install flags can be found on the WPKG project's wiki:

```
7zip:
  9.20.00.0:
    installer: salt://win/repo/7zip/7z920-x64.msi
    full_name: 7-Zip 9.20 (x64 edition)
    reboot: False
    install_flags: ' /q '
    msiexec: True
    uninstaller: salt://win/repo/7zip/7z920-x64.msi
    uninstall_flags: ' /qn'
```

13.3 Generate Repo Cache File

Once the sls file has been created, generate the repository cache file with the winrepo runner:

```
salt-run winrepo.genrepo
```

Then update the repository cache file on your minions, exactly how it's done for the Linux package managers:

```
salt '*' pkg.refresh_db
```

13.4 Install Windows Software

Now you can query the available version of Firefox using the Salt pkg module.

```
salt '*' pkg.available_version Firefox

{'Firefox': {'15.0.1': 'Mozilla Firefox 15.0.1 (x86 en-US)',
             '16.0.2': 'Mozilla Firefox 16.0.2 (x86 en-US)',
             '17.0.1': 'Mozilla Firefox 17.0.1 (x86 en-US)'}}
```

As you can see, there are three versions of Firefox available for installation. You can refer a software package by its `name` or its `full_name` surround by single quotes.

```
salt '*' pkg.install 'Firefox'
```

The above line will install the latest version of Firefox.

```
salt '*' pkg.install 'Firefox' version=16.0.2
```

The above line will install version 16.0.2 of Firefox.

If a different version of the package is already installed it will be replaced with the version in winrepo (only if the package itself supports live updating).

You can also specify the full name:

```
salt '*' pkg.install 'Mozilla Firefox 17.0.1 (x86 en-US)'
```

13.5 Uninstall Windows Software

Uninstall software using the pkg module:

```
salt '*' pkg.remove 'Firefox'

salt '*' pkg.purge 'Firefox'
```

`pkg.purge` just executes `pkg.remove` on Windows. At some point in the future `pkg.purge` may direct the installer to remove all configs and settings for software packages that support that option.

13.6 Standalone Minion Salt Windows Repo Module

In order to facilitate managing a Salt Windows software repo with Salt on a Standalone Minion on Windows, a new module named winrepo has been added to Salt. winrepo matches what is available in the salt runner and allows you to manage the Windows software repo contents. Example: `salt '*' winrepo.genrepo`

13.7 Git Hosted Repo

Windows software package definitions can also be hosted in one or more git repositories. The default repo is one hosted on Github.com by SaltStack,Inc., which includes package definitions for open source software. This repo points to the HTTP or ftp locations of the installer files. Anyone is welcome to send a pull request to this repo to add new package definitions. Browse the repo here: https://github.com/saltstack/salt-winrepo .

Configure which git repos the master can search for package definitions by modifying or extending the `win_gitrepos` configuration option list in the master config.

Checkout each git repo in `win_gitrepos`, compile your package repository cache and then refresh each minion's package cache:

```
salt-run winrepo.update_git_repos
salt-run winrepo.genrepo
salt '*' pkg.refresh_db
```

13.8 Troubleshooting

13.8.1 Incorrect name/version

If the package seems to install properly, but salt reports a failure then it is likely you have a version or `full_name` mismatch.

Check the exact `full_name` and version used by the package. Use `pkg.list_pkgs` to check that the names and version exactly match what is installed.

13.8.2 Changes to sls files not being picked up

Ensure you have (re)generated the repository cache file and then updated the repository cache on the relevant minions:

```
salt-run winrepo.genrepo
salt 'MINION' pkg.refresh_db
```

13.8.3 Packages management under Windows 2003

On windows server 2003, you need to install optional windows component ``wmi windows installer provider'' to have full list of installed packages. If you don't have this, salt-minion can't report some installed software.

CHAPTER 14

Windows-specific Behaviour

Salt is capable of managing Windows systems, however due to various differences between the operating systems, there are some things you need to keep in mind.

This document will contain any quirks that apply across Salt or generally across multiple module functions. Any Windows-specific behavior for particular module functions will be documented in the module function documentation. Therefore this document should be read in conjunction with the module function documentation.

14.1 Group parameter for files

Salt was originally written for managing Unix-based systems, and therefore the file module functions were designed around that security model. Rather than trying to shoehorn that model on to Windows, Salt ignores these parameters and makes non-applicable module functions unavailable instead.

One of the commonly ignored parameters is the `group` parameter for managing files. Under Windows, while files do have a `primary group' property, this is rarely used. It generally has no bearing on permissions unless intentionally configured and is most commonly used to provide Unix compatibility (e.g. Services For Unix, NFS services).

Because of this, any file module functions that typically require a group, do not under Windows. Attempts to directly use file module functions that operate on the group (e.g. `file.chgrp`) will return a pseudo-value and cause a log message to appear. No group parameters will be acted on.

If you do want to access and change the `primary group' property and understand the implications, use the `file.get_pgid` or `file.get_pgroup` functions or the `pgroup` parameter on the `file.chown` module function.

14.2 Dealing with case-insensitive but case-preserving names

Windows is case-insensitive, but however preserves the case of names and it is this preserved form that is returned from system functions. This causes some issues with Salt because it assumes case-sensitive names. These issues generally occur in the state functions and can cause bizarre looking errors.

To avoid such issues, always pretend Windows is case-sensitive and use the right case for names, e.g. specify `user=Administrator` instead of `user=administrator`.

Follow issue 11801 for any changes to this behavior.

14.3 Dealing with various username forms

Salt does not understand the various forms that Windows usernames can come in, e.g. username, mydomainusername, username@mydomain.tld can all refer to the same user. In fact, Salt generally only considers the raw username value, i.e. the username without the domain or host information.

Using these alternative forms will likely confuse Salt and cause odd errors to happen. Use only the raw username value in the correct case to avoid problems.

Follow issue 11801 for any changes to this behavior.

14.4 Specifying the None group

Each Windows system has built-in _None_ group. This is the default `primary group' for files for users not on a domain environment.

Unfortunately, the word _None_ has special meaning in Python - it is a special value indicating `nothing', similar to `null` or `nil` in other languages.

To specify the None group, it must be specified in quotes, e.g. `./salt '*' file.chpgrp C:\path\to\file "'None'"`.

14.5 Symbolic link loops

Under Windows, if any symbolic link loops are detected or if there are too many levels of symlinks (defaults to 64), an error is always raised.

For some functions, this behavior is different to the behavior on Unix platforms. In general, avoid symlink loops on either platform.

14.6 Modifying security properties (ACLs) on files

There is no support in Salt for modifying ACLs, and therefore no support for changing file permissions, besides modifying the owner/user.

CHAPTER 15

Salt Cloud

15.1 Getting Started

15.1.1 Install Salt Cloud

Salt Cloud is now part of Salt proper. It was merged in as of *Salt version 2014.1.0*.

On Ubuntu, install Salt Cloud by using following command:

```
sudo add-apt-repository ppa:saltstack/salt
sudo apt-get install salt-cloud
```

If using Salt Cloud on OS X, `curl-ca-bundle` must be installed. Presently, this package is not available via `brew`, but it is available using MacPorts:

```
sudo port install curl-ca-bundle
```

Salt Cloud depends on `apache-libcloud`. Libcloud can be installed via pip with `pip install apache-libcloud`.

Installing Salt Cloud for development

Installing Salt for development enables Salt Cloud development as well, just make sure `apache-libcloud` is installed as per above paragraph.

See these instructions: *Installing Salt for development*.

15.2 Using Salt Cloud

15.2.1 Salt Cloud basic usage

Salt Cloud needs, at least, one configured *Provider* and *Profile* to be functional.

Creating a VM

To create a VM with salt cloud, use command:

```
salt-cloud -p <profile> name_of_vm
```

Assuming there is a profile configured as following:

```
fedora_rackspace:
    provider: rackspace
    image: Fedora 17
    size: 256 server
    script: bootstrap-salt
```

Then, the command to create new VM named `fedora_http_01` is:

```
salt-cloud -p fedora_rackspace fedora_http_01
```

Destroying a VM

To destroy a created-by-salt-cloud VM, use command:

```
salt-cloud -d name_of_vm
```

For example, to delete the VM created on above example, use:

```
salt-cloud -d fedora_http_01
```

15.2.2 VM Profiles

Salt cloud designates virtual machines inside the profile configuration file. The profile configuration file defaults to `/etc/salt/cloud.profiles` and is a yaml configuration. The syntax for declaring profiles is simple:

```
fedora_rackspace:
    provider: rackspace
    image: Fedora 17
    size: 256 server
    script: bootstrap-salt
```

It should be noted that the `script` option defaults to `bootstrap-salt`, and does not normally need to be specified. Further examples in this document will not show the `script` option.

A few key pieces of information need to be declared and can change based on the public cloud provider. A number of additional parameters can also be inserted:

```
centos_rackspace:
    provider: rackspace
    image: CentOS 6.2
    size: 1024 server
    minion:
        master: salt.example.com
        append_domain: webs.example.com
        grains:
            role: webserver
```

The image must be selected from available images. Similarly, sizes must be selected from the list of sizes. To get a list of available images and sizes use the following command:

```
salt-cloud --list-images openstack
salt-cloud --list-sizes openstack
```

Some parameters can be specified in the main Salt cloud configuration file and then are applied to all cloud profiles. For instance if only a single cloud provider is being used then the provider option can be declared in the Salt cloud configuration file.

Multiple Configuration Files

In addition to `/etc/salt/cloud.profiles`, profiles can also be specified in any file matching `cloud.profiles.d/*conf` which is a sub-directory relative to the profiles configuration file(with the above configuration file as an example, `/etc/salt/cloud.profiles.d/*.conf`). This allows for more extensible configuration, and plays nicely with various configuration management tools as well as version control systems.

Larger Example

```
rhel_ec2:
    provider: ec2
    image: ami-e565ba8c
    size: t1.micro
    minion:
        cheese: edam

ubuntu_ec2:
    provider: ec2
    image: ami-7e2da54e
    size: t1.micro
    minion:
        cheese: edam

ubuntu_rackspace:
    provider: rackspace
    image: Ubuntu 12.04 LTS
    size: 256 server
    minion:
        cheese: edam

fedora_rackspace:
    provider: rackspace
    image: Fedora 17
    size: 256 server
    minion:
        cheese: edam

cent_linode:
    provider: linode
    image: CentOS 6.2 64bit
    size: Linode 512

cent_gogrid:
    provider: gogrid
    image: 12834
    size: 512MB

cent_joyent:
    provider: joyent
    image: centos-6
    size: Small 1GB
```

15.2.3 Cloud Map File

A number of options exist when creating virtual machines. They can be managed directly from profiles and the command line execution, or a more complex map file can be created. The map file allows for a number of virtual machines to be created and associated with specific profiles.

Map files have a simple format, specify a profile and then a list of virtual machines to make from said profile:

```
fedora_small:
  - web1
  - web2
  - web3
fedora_high:
  - redis1
  - redis2
  - redis3
cent_high:
  - riak1
  - riak2
  - riak3
```

This map file can then be called to roll out all of these virtual machines. Map files are called from the salt-cloud command with the -m option:

```
$ salt-cloud -m /path/to/mapfile
```

Remember, that as with direct profile provisioning the -P option can be passed to create the virtual machines in parallel:

```
$ salt-cloud -m /path/to/mapfile -P
```

A map file can also be enforced to represent the total state of a cloud deployment by using the `--hard` option. When using the hard option any vms that exist but are not specified in the map file will be destroyed:

```
$ salt-cloud -m /path/to/mapfile -P -H
```

Be careful with this argument, it is very dangerous! In fact, it is so dangerous that in order to use it, you must explicitly enable it in the main configuration file.

```
enable_hard_maps: True
```

A map file can include grains and minion configuration options:

```
fedora_small:
  - web1:
      minion:
        log_level: debug
      grains:
        cheese: tasty
        omelet: du fromage
  - web2:
      minion:
        log_level: warn
      grains:
        cheese: more tasty
        omelet: with peppers
```

A map file may also be used with the various query options:

```
$ salt-cloud -m /path/to/mapfile -Q
{'ec2': {'web1': {'id': 'i-e6aqfegb',
                  'image': None,
                  'private_ips': [],
                  'public_ips': [],
                  'size': None,
                  'state': 0}},
         'web2': {'Absent'}}
```

...or with the delete option:

```
$ salt-cloud -m /path/to/mapfile -d
The following virtual machines are set to be destroyed:
  web1
  web2

Proceed? [N/y]
```

Setting up New Salt Masters

Bootstrapping a new master in the map is as simple as:

```
fedora_small:
  - web1:
      make_master: True
  - web2
  - web3
```

Notice that **ALL** bootstrapped minions from the map will answer to the newly created salt-master.

To make any of the bootstrapped minions answer to the bootstrapping salt-master as opposed to the newly created salt-master, as an example:

```
fedora_small:
  - web1:
      make_master: True
      minion:
        master: <the local master ip address>
        local_master: True
  - web2
  - web3
```

The above says the minion running on the newly created salt-master responds to the local master, ie, the master used to bootstrap these VMs.

Another example:

```
fedora_small:
  - web1:
      make_master: True
  - web2
  - web3:
      minion:
        master: <the local master ip address>
        local_master: True
```

The above example makes the `web3` minion answer to the local master, not the newly created master.

15.2.4 Cloud Actions

Once a VM has been created, there are a number of actions that can be performed on it. The ``reboot'' action can be used across all providers, but all other actions are specific to the cloud provider. In order to perform an action, you may specify it from the command line, including the name(s) of the VM to perform the action on:

```
$ salt-cloud -a reboot vm_name
$ salt-cloud -a reboot vm1 vm2 vm2
```

Or you may specify a map which includes all VMs to perform the action on:

```
$ salt-cloud -a reboot -m /path/to/mapfile
```

The following is a list of actions currently supported by salt-cloud:

```
all providers:
    - reboot
ec2:
    - start
    - stop
joyent:
    - stop
```

Another useful reference for viewing more salt-cloud actions is the :ref:Salt Cloud Feature Matrix <salt-cloud-feature-matrix>

15.2.5 Cloud Functions

Cloud functions work much the same way as cloud actions, except that they don't perform an operation on a specific instance, and so do not need a machine name to be specified. However, since they perform an operation on a specific cloud provider, that provider must be specified.

```
$ salt-cloud -f show_image ec2 image=ami-fd20ad94
```

There are three universal salt-cloud functions that are extremely useful for gathering information about instances on a provider basis:

*`list_nodes`: Returns some general information about the instances for the given provider. *`list_nodes_full`: Returns all information about the instances for the given provider. *`list_nodes_select`: Returns select information about the instances for the given provider.

```
$ salt-cloud -f list_nodes linode
$ salt-cloud -f list_nodes_full linode
$ salt-cloud -f list_nodes_select linode
```

Another useful reference for viewing salt-cloud functions is the :ref:Salt Cloud Feature Matrix <salt-cloud-feature-matrix>

15.3 Core Configuration

15.3.1 Core Configuration

A number of core configuration options and some options that are global to the VM profiles can be set in the cloud configuration file. By default this file is located at `/etc/salt/cloud`.

Thread Pool Size

When salt cloud is operating in parallel mode via the `-P` argument, you can control the thread pool size by specifying the `pool_size` parameter with a positive integer value.

By default, the thread pool size will be set to the number of VMs that salt cloud is operating on.

```
pool_size: 10
```

Minion Configuration

The default minion configuration is set up in this file. Minions created by salt-cloud derive their configuration from this file. Almost all parameters found in *Configuring the Salt Minion* can be used here.

```
minion:
  master: saltmaster.example.com
```

In particular, this is the location to specify the location of the salt master and its listening port, if the port is not set to the default.

Cloud Configuration Syntax

The data specific to interacting with public clouds is set up here.

Cloud provider configuration syntax can live in several places. The first is in `/etc/salt/cloud`:

```
# /etc/salt/cloud
providers:
  my-aws-migrated-config:
    id: HJGRYCILJLKJYG
    key: 'kdjgfsgm;woormgl/aserigjksjdhasdfgn'
    keyname: test
    securitygroup: quick-start
    private_key: /root/test.pem
    provider: aws
```

Cloud provider configuration data can also be housed in `/etc/salt/cloud.providers` or any file matching `/etc/salt/cloud.providers.d/*.conf`. All files in any of these locations will be parsed for cloud provider data.

Using the example configuration above:

```
# /etc/salt/cloud.providers
# or could be /etc/salt/cloud.providers.d/*.conf
my-aws-config:
  id: HJGRYCILJLKJYG
  key: 'kdjgfsgm;woormgl/aserigjksjdhasdfgn'
  keyname: test
  securitygroup: quick-start
  private_key: /root/test.pem
  provider: aws
```

Note: Salt Cloud provider configurations within `/etc/cloud.provider.d/ should not specify the ``providers` starting key.

It is also possible to have multiple cloud configuration blocks within the same alias block. For example:

```
production-config:
  - id: HJGRYCILJLKJYG
    key: 'kdjgfsgm;woormgl/aserigjksjdhasdfgn'
    keyname: test
    securitygroup: quick-start
    private_key: /root/test.pem
    provider: aws

  - user: example_user
    apikey: 123984bjjas87034
    provider: rackspace
```

However, using this configuration method requires a change with profile configuration blocks. The provider alias needs to have the provider key value appended as in the following example:

```
rhel_aws_dev:
  provider: production-config:aws
  image: ami-e565ba8c
  size: t1.micro

rhel_aws_prod:
  provider: production-config:aws
  image: ami-e565ba8c
  size: High-CPU Extra Large Instance

database_prod:
  provider: production-config:rackspace
  image: Ubuntu 12.04 LTS
  size: 256 server
```

Notice that because of the multiple entries, one has to be explicit about the provider alias and name, from the above example, `production-config: aws`.

This data interactions with the `salt-cloud` binary regarding its `--list-location`, `--list-images`, and `--list-sizes` which needs a cloud provider as an argument. The argument used should be the configured cloud provider alias. If the provider alias has multiple entries, `<provider-alias>: <provider-name>` should be used.

To allow for a more extensible configuration, `--providers-config`, which defaults to `/etc/salt/cloud.providers`, was added to the cli parser. It allows for the providers' configuration to be added on a per-file basis.

Pillar Configuration

It is possible to configure cloud providers using pillars. This is only used when inside the cloud module. You can setup a variable called `cloud` that contains your profile and provider to pass that information to the cloud servers instead of having to copy the full configuration to every minion. In your pillar file, you would use something like this:

```
cloud:
  ssh_key_name: saltstack
  ssh_key_file: /root/.ssh/id_rsa
  update_cachedir: True
  diff_cache_events: True
  change_password: True

  providers:
```

```
    my-nova:
      identity_url: https://identity.api.rackspacecloud.com/v2.0/
      compute_region: IAD
      user: myuser
      api_key: apikey
      tenant: 123456
      provider: nova

    my-openstack:
      identity_url: https://identity.api.rackspacecloud.com/v2.0/tokens
      user: user2
      apikey: apikey2
      tenant: 654321
      compute_region: DFW
      provider: openstack
      compute_name: cloudServersOpenStack

  profiles:
    ubuntu-nova:
      provider: my-nova
      size: performance1-8
      image: bb02b1a3-bc77-4d17-ab5b-421d89850fca
      script_args: git develop

    ubuntu-openstack:
      provider: my-openstack
      size: performance1-8
      image: bb02b1a3-bc77-4d17-ab5b-421d89850fca
      script_args: git develop
```

Cloud Configurations

Rackspace

Rackspace cloud requires two configuration options; a `user` and an `apikey`:

```
my-rackspace-config:
  user: example_user
  apikey: 123984bjjas87034
  provider: rackspace-config
```

Note: In the cloud profile that uses this provider configuration, the syntax for the `provider` required field would be `provider: my-rackspace-config`.

Amazon AWS

A number of configuration options are required for Amazon AWS including `id`, `key`, `keyname`, `sercurity-group`, and `private_key`:

```
my-aws-quick-start:
  id: HJGRYCILJLKJYG
  key: 'kdjgfsgm;woormgl/aserigjksjdhasdfgn'
  keyname: test
  securitygroup: quick-start
```

```
  private_key: /root/test.pem
  provider: aws

my-aws-default:
  id: HJGRYCILJLKJYG
  key: 'kdjgfsgm;woormgl/aserigjksjdhasdfgn'
  keyname: test
  securitygroup: default
  private_key: /root/test.pem
  provider: aws
```

Note: In the cloud profile that uses this provider configuration, the syntax for the `provider` required field would be either `provider: my-aws-quick-start` or `provider: my-aws-default`.

Linode

Linode requires a single API key, but the default root password also needs to be set:

```
my-linode-config:
  apikey: asldkgfakl;sdfjsjaslfjaklsdjf;askldjfaaklsjdfhasldsadfghdkf
  password: F00barbaz
  ssh_pubkey: ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIKHEOLLbeXgaqRQT9NBAopVz366SdYc0KKX33vAnq+2R user@h
  ssh_key_file: ~/.ssh/id_ed25519
  provider: linode
```

The password needs to be 8 characters and contain lowercase, uppercase, and numbers.

Note: In the cloud profile that uses this provider configuration, the syntax for the `provider` required field would be `provider: my-linode-config`

Joyent Cloud

The Joyent cloud requires three configuration parameters: The username and password that are used to log into the Joyent system, as well as the location of the private SSH key associated with the Joyent account. The SSH key is needed to send the provisioning commands up to the freshly created virtual machine.

```
my-joyent-config:
  user: fred
  password: saltybacon
  private_key: /root/joyent.pem
  provider: joyent
```

Note: In the cloud profile that uses this provider configuration, the syntax for the `provider` required field would be `provider: my-joyent-config`

GoGrid

To use Salt Cloud with GoGrid, log into the GoGrid web interface and create an API key. Do this by clicking on ``My Account'' and then going to the API Keys tab.

The `apikey` and the `sharedsecret` configuration parameters need to be set in the configuration file to enable interfacing with GoGrid:

```
my-gogrid-config:
  apikey: asdff7896asdh789
  sharedsecret: saltybacon
  provider: gogrid
```

Note: In the cloud profile that uses this provider configuration, the syntax for the `provider` required field would be `provider: my-gogrid-config`.

OpenStack

OpenStack configuration differs between providers, and at the moment several options need to be specified. This module has been officially tested against the HP and the Rackspace implementations, and some examples are provided for both.

```
# For HP
my-openstack-hp-config:
  identity_url:
  'https://region-a.geo-1.identity.hpcloudsvc.com:35357/v2.0/'
  compute_name: Compute
  compute_region: 'az-1.region-a.geo-1'
  tenant: myuser-tenant1
  user: myuser
  ssh_key_name: mykey
  ssh_key_file: '/etc/salt/hpcloud/mykey.pem'
  password: mypass
  provider: openstack

# For Rackspace
my-openstack-rackspace-config:
  identity_url: 'https://identity.api.rackspacecloud.com/v2.0/tokens'
  compute_name: cloudServersOpenStack
  protocol: ipv4
  compute_region: DFW
  protocol: ipv4
  user: myuser
  tenant: 5555555
  password: mypass
  provider: openstack
```

If you have an API key for your provider, it may be specified instead of a password:

```
my-openstack-hp-config:
  apikey: 901d3f579h23c8v73q9

my-openstack-rackspace-config:
  apikey: 901d3f579h23c8v73q9
```

Note: In the cloud profile that uses this provider configuration, the syntax for the `provider` required field would be either `provider: my-openstack-hp-config` or `provider: my-openstack-rackspace-config`.

You will certainly need to configure the `user`, `tenant`, and either `password` or `apikey`.

If your OpenStack instances only have private IP addresses and a CIDR range of private addresses are not reachable from the salt-master, you may set your preference to have Salt ignore it:

```
my-openstack-config:
  ignore_cidr: 192.168.0.0/16
```

For in-house OpenStack Essex installation, libcloud needs the service_type :

```
my-openstack-config:
  identity_url: 'http://control.openstack.example.org:5000/v2.0/'
  compute_name : Compute Service
  service_type : compute
```

DigitalOcean

Using Salt for DigitalOcean requires a `client_key` and an `api_key`. These can be found in the DigitalOcean web interface, in the ``My Settings'' section, under the API Access tab. .. code-block:: yaml

my-digitalocean-config: provider: digital_ocean personal_access_token: xxx location: New York 1

Note: In the cloud profile that uses this provider configuration, the syntax for the `provider` required field would be `provider: my-digital-ocean-config`.

Parallels

Using Salt with Parallels requires a `user`, `password` and `URL`. These can be obtained from your cloud provider.

```
my-parallels-config:
  user: myuser
  password: xyzzy
  url: https://api.cloud.xmission.com:4465/paci/v1.0/
  provider: parallels
```

Note: In the cloud profile that uses this provider configuration, the syntax for the `provider` required field would be `provider: my-parallels-config`.

Proxmox

Using Salt with Proxmox requires a `user`, `password`, and `URL`. These can be obtained from your cloud provider. Both PAM and PVE users can be used.

```
my-proxmox-config:
  provider: proxmox
  user: saltcloud@pve
  password: xyzzy
  url: your.proxmox.host
```

Note: In the cloud profile that uses this provider configuration, the syntax for the `provider` required field would be `provider: my-proxmox-config`.

LXC

The lxc driver uses saltify to install salt and attach the lxc container as a new lxc minion. As soon as we can, we manage baremetal operation over SSH. You can also destroy those containers via this driver.

```
devhost10-lxc:
  target: devhost10
  provider: lxc
```

And in the map file:

```
devhost10-lxc:
  provider: devhost10-lxc
  from_container: ubuntu
  backing: lvm
  sudo: True
  size: 3g
  ip: 10.0.3.9
  minion:
    master: 10.5.0.1
    master_port: 4506
  lxc_conf:
    - lxc.utsname: superlxc
```

Note: In the cloud profile that uses this provider configuration, the syntax for the `provider` required field would be ``provdier: devhost10-lxc`.

Saltify

The Saltify driver is a new, experimental driver for installing Salt on existing machines (virtual or bare metal). Because it does not use an actual cloud provider, it needs no configuration in the main cloud config file. However, it does still require a profile to be set up, and is most useful when used inside a map file. The key parameters to be set are `ssh_host`, `ssh_username` and either `ssh_keyfile` or `ssh_password`. These may all be set in either the profile or the map. An example configuration might use the following in cloud.profiles:

```
make_salty:
  provider: saltify
```

And in the map file:

```
make_salty:
  - myinstance:
    ssh_host: 54.262.11.38
    ssh_username: ubuntu
    ssh_keyfile: '/etc/salt/mysshkey.pem'
    sudo: True
```

Note: In the cloud profile that uses this provider configuration, the syntax for the `provider` required field would be `provider: make_salty`.

Extending Profiles and Cloud Providers Configuration

As of 0.8.7, the option to extend both the profiles and cloud providers configuration and avoid duplication was added. The extends feature works on the current profiles configuration, but, regarding the cloud providers configuration,

only works in the new syntax and respective configuration files, i.e. `/etc/salt/salt/cloud.providers` or `/etc/salt/cloud.providers.d/*.conf`.

Note: Extending cloud profiles and providers is not recursive. For example, a profile that is extended by a second profile is possible, but the second profile cannot be extended by a third profile.

Also, if a profile (or provider) is extending another profile and each contains a list of values, the lists from the extending profile will override the list from the original profile. The lists are not merged together.

Extending Profiles

Some example usage on how to use `extends` with profiles. Consider `/etc/salt/salt/cloud.profiles` containing:

```
development-instances:
  provider: my-ec2-config
  size: t1.micro
  ssh_username: ec2_user
  securitygroup:
    - default
  deploy: False

Amazon-Linux-AMI-2012.09-64bit:
  image: ami-54cf5c3d
  extends: development-instances

Fedora-17:
  image: ami-08d97e61
  extends: development-instances

CentOS-5:
  provider: my-aws-config
  image: ami-09b61d60
  extends: development-instances
```

The above configuration, once parsed would generate the following profiles data:

```
[{'deploy': False,
  'image': 'ami-08d97e61',
  'profile': 'Fedora-17',
  'provider': 'my-ec2-config',
  'securitygroup': ['default'],
  'size': 't1.micro',
  'ssh_username': 'ec2_user'},
 {'deploy': False,
  'image': 'ami-09b61d60',
  'profile': 'CentOS-5',
  'provider': 'my-aws-config',
  'securitygroup': ['default'],
  'size': 't1.micro',
  'ssh_username': 'ec2_user'},
 {'deploy': False,
  'image': 'ami-54cf5c3d',
  'profile': 'Amazon-Linux-AMI-2012.09-64bit',
  'provider': 'my-ec2-config',
  'securitygroup': ['default'],
  'size': 't1.micro',
```

```
  'ssh_username': 'ec2_user'},
 {'deploy': False,
  'profile': 'development-instances',
  'provider': 'my-ec2-config',
  'securitygroup': ['default'],
  'size': 't1.micro',
  'ssh_username': 'ec2_user'}]
```

Pretty cool right?

Extending Providers

Some example usage on how to use `extends` within the cloud providers configuration. Consider `/etc/salt/salt/cloud.providers` containing:

```
my-develop-envs:
  - id: HJGRYCILJLKJYG
    key: 'kdjgfsgm;woormgl/aserigjksjdhasdfgn'
    keyname: test
    securitygroup: quick-start
    private_key: /root/test.pem
    location: ap-southeast-1
    availability_zone: ap-southeast-1b
    provider: aws

  - user: myuser@mycorp.com
    password: mypass
    ssh_key_name: mykey
    ssh_key_file: '/etc/salt/ibm/mykey.pem'
    location: Raleigh
    provider: ibmsce


my-productions-envs:
  - extends: my-develop-envs:ibmsce
    user: my-production-user@mycorp.com
    location: us-east-1
    availability_zone: us-east-1
```

The above configuration, once parsed would generate the following providers data:

```
'providers': {
    'my-develop-envs': [
        {'availability_zone': 'ap-southeast-1b',
         'id': 'HJGRYCILJLKJYG',
         'key': 'kdjgfsgm;woormgl/aserigjksjdhasdfgn',
         'keyname': 'test',
         'location': 'ap-southeast-1',
         'private_key': '/root/test.pem',
         'provider': 'aws',
         'securitygroup': 'quick-start'
        },
        {'location': 'Raleigh',
         'password': 'mypass',
         'provider': 'ibmsce',
         'ssh_key_file': '/etc/salt/ibm/mykey.pem',
         'ssh_key_name': 'mykey',
```

```
        'user': 'myuser@mycorp.com'
        }
    ],
    'my-productions-envs': [
        {'availability_zone': 'us-east-1',
         'location': 'us-east-1',
         'password': 'mypass',
         'provider': 'ibmsce',
         'ssh_key_file': '/etc/salt/ibm/mykey.pem',
         'ssh_key_name': 'mykey',
         'user': 'my-production-user@mycorp.com'
        }
    ]
}
```

15.4 Windows Configuration

15.4.1 Spinning up Windows Minions

It is possible to use Salt Cloud to spin up Windows instances, and then install Salt on them. This functionality is available on all cloud providers that are supported by Salt Cloud. However, it may not necessarily be available on all Windows images.

Requirements

Salt Cloud makes use of *impacket* and *winexe* to set up the Windows Salt Minion installer.

impacket is usually available as either the *impacket* or the *python-impacket* package, depending on the distribution. More information on *impacket* can be found at the project home:

- impacket project home

winexe is less commonly available in distribution-specific repositories. However, it is currently being built for various distributions in 3rd party channels:

- RPMs at pbone.net
- OpenSuse Build Service

Additionally, a copy of the Salt Minion Windows installer must be present on the system on which Salt Cloud is running. This installer may be downloaded from saltstack.com:

- SaltStack Download Area

Firewall Settings

Because Salt Cloud makes use of *smbclient* and *winexe*, port 445 must be open on the target image. This port is not generally open by default on a standard Windows distribution, and care must be taken to use an image in which this port is open, or the Windows firewall is disabled.

If supported by the cloud provider, a PowerShell script may be used to open up this port automatically, using the cloud provider's *userdata*. The following script would open up port 445, and apply the changes:

```
<powershell>
New-NetFirewallRule -Name "SMB445" -DisplayName "SMB445" -Protocol TCP -LocalPort 445
Set-Item (dir wsman:\localhost\Listener\*\Port -Recurse).pspath 445 -Force
Restart-Service winrm
</powershell>
```

For EC2, this script may be saved as a file, and specified in the provider or profile configuration as *userdata_file*. For instance:

```
userdata_file: /etc/salt/windows-firewall.ps1
```

Configuration

Configuration is set as usual, with some extra configuration settings. The location of the Windows installer on the machine that Salt Cloud is running on must be specified. This may be done in any of the regular configuration files (main, providers, profiles, maps). For example:

Setting the installer in `/etc/salt/cloud.providers`:

```
my-softlayer:
  provider: softlayer
  user: MYUSER1138
  apikey: 'e3b68aa711e6deadc62d5b76355674beef7cc3116062ddbacafe5f7e465bfdc9'
  minion:
    master: saltmaster.example.com
  win_installer: /root/Salt-Minion-2014.7.0-AMD64-Setup.exe
  win_username: Administrator
  win_password: letmein
```

The default Windows user is *Administrator*, and the default Windows password is blank.

Auto-Generated Passwords on EC2

On EC2, when the *win_password* is set to *auto*, Salt Cloud will query EC2 for an auto-generated password. This password is expected to take at least 4 minutes to generate, adding additional time to the deploy process.

When the EC2 API is queried for the auto-generated password, it will be returned in a message encrypted with the specified *keyname*. This requires that the appropriate *private_key* file is also specified. Such a profile configuration might look like:

```
windows-server-2012:
  provider: my-ec2-config
  image: ami-c49c0dac
  size: m1.small
  securitygroup: windows
  keyname: mykey
  private_key: /root/mykey.pem
  userdata_file: /etc/salt/windows-firewall.ps1
  win_installer: /root/Salt-Minion-2014.7.0-AMD64-Setup.exe
  win_username: Administrator
  win_password: auto
```

15.5 Cloud Provider Specifics

15.5.1 Getting Started With Aliyun ECS

The Aliyun ECS (Elastic Computer Service) is one of the most popular public cloud providers in China. This cloud provider can be used to manage aliyun instance using salt-cloud.

http://www.aliyun.com/

Dependencies

This driver requires the Python `requests` library to be installed.

Configuration

Using Salt for Aliyun ECS requires aliyun access key id and key secret. These can be found in the aliyun web interface, in the ``User Center'' section, under ``My Service'' tab.

```
# Note: This example is for /etc/salt/cloud.providers or any file in the
# /etc/salt/cloud.providers.d/ directory.

my-aliyun-config:
  # aliyun Access Key ID
  id: wDGEwGregedg3435gDgxd
  # aliyun Access Key Secret
  key: GDd45t43RDBTrkkkg43934t34qT43t4dgegerGEgg
  location: cn-qingdao
  provider: aliyun
```

Profiles

Cloud Profiles

Set up an initial profile at `/etc/salt/cloud.profiles` or in the `/etc/salt/cloud.profiles.d/` directory:

```
aliyun_centos:
    provider: my-aliyun-config
    size: ecs.t1.small
    location: cn-qingdao
    securitygroup: G1989096784427999
    image: centos6u3_64_20G_aliaegis_20130816.vhd
```

Sizes can be obtained using the `--list-sizes` option for the `salt-cloud` command:

```
# salt-cloud --list-sizes my-aliyun-config
my-aliyun-config:
    ----------
    aliyun:
        ----------
        ecs.c1.large:
            ----------
            CpuCoreCount:
                8
```

```
            InstanceTypeId:
                ecs.c1.large
            MemorySize:
                16.0

...SNIP...
```

Images can be obtained using the `--list-images` option for the `salt-cloud` command:

```
# salt-cloud --list-images my-aliyun-config
my-aliyun-config:
    ----------
    aliyun:
        ----------
        centos5u8_64_20G_aliaegis_20131231.vhd:
            ----------
            Architecture:
                x86_64
            Description:

            ImageId:
                centos5u8_64_20G_aliaegis_20131231.vhd
            ImageName:
                CentOS 5.8 64
            ImageOwnerAlias:
                system
            ImageVersion:
                1.0
            OSName:
                CentOS  5.8 64
            Platform:
                CENTOS5
            Size:
                20
            Visibility:
                public
...SNIP...
```

Locations can be obtained using the `--list-locations` option for the `salt-cloud` command:

```
my-aliyun-config:
    ----------
    aliyun:
        ----------
        cn-beijing:
            ----------
            LocalName:
                
            RegionId:
                cn-beijing
        cn-hangzhou:
            ----------
            LocalName:
                
            RegionId:
                cn-hangzhou
        cn-hongkong:
            ----------
            LocalName:
```

```
                ▯▯
            RegionId:
                cn-hongkong
        cn-qingdao:
            ----------
            LocalName:
                ▯▯
            RegionId:
                cn-qingdao
```

Security Group can be obtained using the `-f list_securitygroup` option for the `salt-cloud` command:

```
# salt-cloud --location=cn-qingdao -f list_securitygroup my-aliyun-config
my-aliyun-config:
    ----------
    aliyun:
        ----------
        G1989096784427999:
            ----------
            Description:
                G1989096784427999
            SecurityGroupId:
                G1989096784427999
```

Note: Aliyun ECS REST API documentation is available from Aliyun ECS API.

15.5.2 Getting Started With Azure

New in version 2014.1.0.

Azure is a cloud service by Microsoft providing virtual machines, SQL services, media services, and more. This document describes how to use Salt Cloud to create a virtual machine on Azure, with Salt installed.

More information about Azure is located at http://www.windowsazure.com/.

Dependencies

- The Azure Python SDK.
- A Microsoft Azure account
- OpenSSL (to generate the certificates)
- Salt

Configuration

Set up the provider config at `/etc/salt/cloud.providers.d/azure.conf`:

```
# Note: This example is for /etc/salt/cloud.providers.d/azure.conf

my-azure-config:
  provider: azure
  subscription_id: 3287abc8-f98a-c678-3bde-326766fd3617
  certificate_path: /etc/salt/azure.pem
```

```
  # Set up the location of the salt master
  #
  minion:
    master: saltmaster.example.com

  # Optional
  management_host: management.core.windows.net
```

The certificate used must be generated by the user. OpenSSL can be used to create the management certificates. Two certificates are needed: a .cer file, which is uploaded to Azure, and a .pem file, which is stored locally.

To create the .pem file, execute the following command:

```
openssl req -x509 -nodes -days 365 -newkey rsa:1024 -keyout /etc/salt/azure.pem -out /etc/salt/azure
```

To create the .cer file, execute the following command:

```
openssl x509 -inform pem -in /etc/salt/azure.pem -outform der -out /etc/salt/azure.cer
```

After creating these files, the .cer file will need to be uploaded to Azure via the ``Upload a Management Certificate'' action of the ``Management Certificates'' tab within the ``Settings'' section of the management portal.

Optionally, a `management_host` may be configured, if necessary for the region.

Cloud Profiles

Set up an initial profile at `/etc/salt/cloud.profiles`:

```
azure-ubuntu:
  provider: my-azure-config
  image: 'b39f27a8b8c64d52b05eac6a62ebad85__Ubuntu-12_04_3-LTS-amd64-server-20131003-en-us-30GB'
  size: Small
  location: 'East US'
  ssh_username: azureuser
  ssh_password: verybadpass
  slot: production
  media_link: 'http://portalvhdabcdefghijklmn.blob.core.windows.net/vhds'
```

These options are described in more detail below. Once configured, the profile can be realized with a salt command:

```
salt-cloud -p azure-ubuntu newinstance
```

This will create an salt minion instance named `newinstance` in Azure. If the command was executed on the salt-master, its Salt key will automatically be signed on the master.

Once the instance has been created with salt-minion installed, connectivity to it can be verified with Salt:

```
salt newinstance test.ping
```

Profile Options

The following options are currently available for Azure.

provider

The name of the provider as configured in */etc/salt/cloud.providers.d/azure.conf*.

image

The name of the image to use to create a VM. Available images can be viewed using the following command:

```
salt-cloud --list-images my-azure-config
```

size

The name of the size to use to create a VM. Available sizes can be viewed using the following command:

```
salt-cloud --list-sizes my-azure-config
```

location

The name of the location to create a VM in. Available locations can be viewed using the following command:

```
salt-cloud --list-locations my-azure-config
```

ssh_username

The user to use to log into the newly-created VM to install Salt.

ssh_password

The password to use to log into the newly-created VM to install Salt.

slot

The environment to which the hosted service is deployed. Valid values are *staging* or *production*. When set to *production*, the resulting URL of the new VM will be *<vm_name>.cloudapp.net*. When set to *staging*, the resulting URL will contain a generated hash instead.

media_link

This is the URL of the container that will store the disk that this VM uses. Currently, this container must already exist. If a VM has previously been created in the associated account, a container should already exist. In the web interface, go into the Storage area and click one of the available storage selections. Click the Containers link, and then copy the URL from the container that will be used. It generally looks like:

```
http://portalvhdabcdefghijklmn.blob.core.windows.net/vhds
```

Show Instance

This action is a thin wrapper around `--full-query`, which displays details on a single instance only. In an environment with several machines, this will save a user from having to sort through all instance data, just to examine a single instance.

```
salt-cloud -a show_instance myinstance
```

15.5.3 Getting Started With DigitalOcean

DigitalOcean is a public cloud provider that specializes in Linux instances.

Configuration

Using Salt for DigitalOcean requires a personal_access_token, an ssh_key_file, and at least one SSH key name in ssh_key_names, more can be added by comma-separating them. The personal_access_token can be found in the Digital Ocean web interface, in the ``Apps & API'' section. The SSH key name can be found under the ``SSH Keys'' section.

```
# Note: This example is for /etc/salt/cloud.providers or any file in the
# /etc/salt/cloud.providers.d/ directory.

my-digitalocean-config:
  provider: digital_ocean
  personal_access_token: xxx
  ssh_key_file: /path/to/ssh/key/file
  ssh_key_names: my-key-name,my-key-name-2
  location: New York 1
```

Profiles

Cloud Profiles

Set up an initial profile at `/etc/salt/cloud.profiles` or in the `/etc/salt/cloud.profiles.d/` directory:

```
digitalocean-ubuntu:
    provider: my-digitalocean-config
    image: Ubuntu 14.04 x32
    size: 512MB
    location: New York 1
    private_networking: True
    backups_enabled: True
    ipv6: True
```

Sizes can be obtained using the `--list-sizes` option for the `salt-cloud` command:

```
# salt-cloud --list-sizes my-digitalocean-config
my-digitalocean-config:
    ----------
    digital_ocean:
        ----------
        512MB:
            ----------
            cost_per_hour:
                0.00744
            cost_per_month:
                5.0
            cpu:
                1
```

```
            disk:
                20
            id:
                66
            memory:
                512
            name:
                512MB
            slug:
                None
...SNIP...
```

Images can be obtained using the `--list-images` option for the `salt-cloud` command:

```
# salt-cloud --list-images my-digitalocean-config
my-digitalocean-config:
    ----------
    digital_ocean:
        ----------
        Arch Linux 2013.05 x64:
            ----------
            distribution:
                Arch Linux
            id:
                350424
            name:
                Arch Linux 2013.05 x64
            public:
                True
            slug:
                None
...SNIP...
```

Note: DigitalOcean's concept of `Applications` is nothing more than a pre-configured instance (same as a normal Droplet). You will find examples such `Docker 0.7 Ubuntu 13.04 x64` and `Wordpress on Ubuntu 12.10` when using the `--list-images` option. These names can be used just like the rest of the standard instances when specifying an image in the cloud profile configuration.

Note: If your domain's DNS is managed with DigitalOcean, you can automatically create A-records for newly created droplets. Use `create_dns_record: True` in your config to enable this. Add `delete_dns_record: True` to also delete records when a droplet is destroyed.

Note: Additional documentation is available from DigitalOcean.

15.5.4 Getting Started With AWS EC2

Amazon EC2 is a very widely used public cloud platform and one of the core platforms Salt Cloud has been built to support.

Previously, the suggested provider for AWS EC2 was the `aws` provider. This has been deprecated in favor of the `ec2` provider. Configuration using the old `aws` provider will still function, but that driver is no longer in active development.

Dependencies

This driver requires the Python `requests` library to be installed.

Configuration

The following example illustrates some of the options that can be set. These parameters are discussed in more detail below.

```
# Note: This example is for /etc/salt/cloud.providers or any file in the
# /etc/salt/cloud.providers.d/ directory.

my-ec2-southeast-public-ips:
  # Set up the location of the salt master
  #
  minion:
    master: saltmaster.example.com

  # Set up grains information, which will be common for all nodes
  # using this provider
  grains:
    node_type: broker
    release: 1.0.1

  # Specify whether to use public or private IP for deploy script.
  #
  # Valid options are:
  #     private_ips - The salt-cloud command is run inside the EC2
  #     public_ips - The salt-cloud command is run outside of EC2
  #
  ssh_interface: public_ips

  # Optionally configure the Windows credential validation number of
  # retries and delay between retries.  This defaults to 10 retries
  # with a one second delay betwee retries
  win_deploy_auth_retries: 10
  win_deploy_auth_retry_delay: 1

  # Set the EC2 access credentials (see below)
  #
  id: HJGRYCILJLKJYG
  key: 'kdjgfsgm;woormgl/aserigjksjdhasdfgn'

  # Make sure this key is owned by root with permissions 0400.
  #
  private_key: /etc/salt/my_test_key.pem
  keyname: my_test_key
  securitygroup: default

  # Optionally configure default region
  # Use salt-cloud --list-locations <provider> to obtain valid regions
  #
  location: ap-southeast-1
  availability_zone: ap-southeast-1b

  # Configure which user to use to run the deploy script. This setting is
  # dependent upon the AMI that is used to deploy. It is usually safer to
```

```
  # configure this individually in a profile, than globally. Typical users
  # are:
  #
  # Amazon Linux -> ec2-user
  # RHEL         -> ec2-user
  # CentOS       -> ec2-user
  # Ubuntu       -> ubuntu
  #
  ssh_username: ec2-user

  # Optionally add an IAM profile
  iam_profile: 'arn:aws:iam::123456789012:instance-profile/ExampleInstanceProfile'

  provider: ec2


my-ec2-southeast-private-ips:
  # Set up the location of the salt master
  #
  minion:
    master: saltmaster.example.com

  # Specify whether to use public or private IP for deploy script.
  #
  # Valid options are:
  #     private_ips - The salt-master is also hosted with EC2
  #     public_ips - The salt-master is hosted outside of EC2
  #
  ssh_interface: private_ips

  # Optionally configure the Windows credential validation number of
  # retries and delay between retries.  This defaults to 10 retries
  # with a one second delay betwee retries
  win_deploy_auth_retries: 10
  win_deploy_auth_retry_delay: 1

  # Set the EC2 access credentials (see below)
  #
  id: HJGRYCILJLKJYG
  key: 'kdjgfsgm;woormgl/aserigjksjdhasdfgn'

  # Make sure this key is owned by root with permissions 0400.
  #
  private_key: /etc/salt/my_test_key.pem
  keyname: my_test_key
  securitygroup: default

  # Optionally configure default region
  #
  location: ap-southeast-1
  availability_zone: ap-southeast-1b

  # Configure which user to use to run the deploy script. This setting is
  # dependent upon the AMI that is used to deploy. It is usually safer to
  # configure this individually in a profile, than globally. Typical users
  # are:
  #
  # Amazon Linux -> ec2-user
```

```
# RHEL         -> ec2-user
# CentOS       -> ec2-user
# Ubuntu       -> ubuntu
#
ssh_username: ec2-user

# Optionally add an IAM profile
iam_profile: 'my other profile name'

provider: ec2
```

Access Credentials

The `id` and `key` settings may be found in the Security Credentials area of the AWS Account page:

https://portal.aws.amazon.com/gp/aws/securityCredentials

Both are located in the Access Credentials area of the page, under the Access Keys tab. The `id` setting is labeled Access Key ID, and the `key` setting is labeled Secret Access Key.

Windows Deploy Timeouts

For Windows instances, it may take longer than normal for the instance to be ready. In these circumstances, the provider configuration can be configured with a `win_deploy_auth_retries` and/or a `win_deploy_auth_retry_delay` setting, which default to 10 retries and a one second delay between retries. These retries and timeouts relate to validating the Administrator password once AWS provides the credentials via the AWS API.

Key Pairs

In order to create an instance with Salt installed and configured, a key pair will need to be created. This can be done in the EC2 Management Console, in the Key Pairs area. These key pairs are unique to a specific region. Keys in the us-east-1 region can be configured at:

https://console.aws.amazon.com/ec2/home?region=us-east-1#s=KeyPairs

Keys in the us-west-1 region can be configured at

https://console.aws.amazon.com/ec2/home?region=us-west-1#s=KeyPairs

...and so on. When creating a key pair, the browser will prompt to download a pem file. This file must be placed in a directory accessible by Salt Cloud, with permissions set to either 0400 or 0600.

Security Groups

An instance on EC2 needs to belong to a security group. Like key pairs, these are unique to a specific region. These are also configured in the EC2 Management Console. Security groups for the us-east-1 region can be configured at:

https://console.aws.amazon.com/ec2/home?region=us-east-1#s=SecurityGroups

...and so on.

A security group defines firewall rules which an instance will adhere to. If the salt-master is configured outside of EC2, the security group must open the SSH port (usually port 22) in order for Salt Cloud to install Salt.

IAM Profile

Amazon EC2 instances support the concept of an instance profile, which is a logical container for the IAM role. At the time that you launch an EC2 instance, you can associate the instance with an instance profile, which in turn corresponds to the IAM role. Any software that runs on the EC2 instance is able to access AWS using the permissions associated with the IAM role.

Scaffolding the profile is a 2-step configuration process:

1. Configure an IAM Role from the IAM Management Console.
2. Attach this role to a new profile. It can be done with the AWS CLI:

```
> aws iam create-instance-profile --instance-profile-name PROFILE_NAME
> aws iam add-role-to-instance-profile --instance-profile-name PROFILE_NAME --role-name ROLE
```

Once the profile is created, you can use the **PROFILE_NAME** to configure your cloud profiles.

Cloud Profiles

Set up an initial profile at `/etc/salt/cloud.profiles`:

```
base_ec2_private:
  provider: my-ec2-southeast-private-ips
  image: ami-e565ba8c
  size: t1.micro
  ssh_username: ec2-user

base_ec2_public:
  provider: my-ec2-southeast-public-ips
  image: ami-e565ba8c
  size: t1.micro
  ssh_username: ec2-user

base_ec2_db:
  provider: my-ec2-southeast-public-ips
  image: ami-e565ba8c
  size: m1.xlarge
  ssh_username: ec2-user
  volumes:
    - { size: 10, device: /dev/sdf }
    - { size: 10, device: /dev/sdg, type: io1, iops: 1000 }
    - { size: 10, device: /dev/sdh, type: io1, iops: 1000 }
  # optionally add tags to profile:
  tag: {'Environment': 'production', 'Role': 'database'}
  # force grains to sync after install
  sync_after_install: grains

base_ec2_vpc:
  provider: my-ec2-southeast-public-ips
  image: ami-a73264ce
  size: m1.xlarge
  ssh_username: ec2-user
  script:  /etc/salt/cloud.deploy.d/user_data.sh
  network_interfaces:
    - DeviceIndex: 0
      PrivateIpAddresses:
        - Primary: True
```

```
      #auto assign public ip (not EIP)
      AssociatePublicIpAddress: True
      SubnetId: subnet-813d4bbf
      SecurityGroupId:
        - sg-750af413
  volumes:
    - { size: 10, device: /dev/sdf }
    - { size: 10, device: /dev/sdg, type: io1, iops: 1000 }
    - { size: 10, device: /dev/sdh, type: io1, iops: 1000 }
  del_root_vol_on_destroy: True
  del_all_vol_on_destroy: True
  tag: {'Environment': 'production', 'Role': 'database'}
  sync_after_install: grains
```

The profile can now be realized with a salt command:

```
# salt-cloud -p base_ec2 ami.example.com
# salt-cloud -p base_ec2_public ami.example.com
# salt-cloud -p base_ec2_private ami.example.com
```

This will create an instance named `ami.example.com` in EC2. The minion that is installed on this instance will have an `id` of `ami.example.com`. If the command was executed on the salt-master, its Salt key will automatically be signed on the master.

Once the instance has been created with salt-minion installed, connectivity to it can be verified with Salt:

```
# salt 'ami.example.com' test.ping
```

Required Settings

The following settings are always required for EC2:

```
# Set the EC2 login data
my-ec2-config:
  id: HJGRYCILJLKJYG
  key: 'kdjgfsgm;woormgl/aserigjksjdhasdfgn'
  keyname: test
  securitygroup: quick-start
  private_key: /root/test.pem
  provider: ec2
```

Optional Settings

EC2 allows a location to be set for servers to be deployed in. Availability zones exist inside regions, and may be added to increase specificity.

```
my-ec2-config:
  # Optionally configure default region
  location: ap-southeast-1
  availability_zone: ap-southeast-1b
```

EC2 instances can have a public or private IP, or both. When an instance is deployed, Salt Cloud needs to log into it via SSH to run the deploy script. By default, the public IP will be used for this. If the salt-cloud command is run from another EC2 instance, the private IP should be used.

```
my-ec2-config:
  # Specify whether to use public or private IP for deploy script
  # private_ips or public_ips
  ssh_interface: public_ips
```

Many EC2 instances do not allow remote access to the root user by default. Instead, another user must be used to run the deploy script using sudo. Some common usernames include ec2-user (for Amazon Linux), ubuntu (for Ubuntu instances), admin (official Debian) and bitnami (for images provided by Bitnami).

```
my-ec2-config:
  # Configure which user to use to run the deploy script
  ssh_username: ec2-user
```

Multiple usernames can be provided, in which case Salt Cloud will attempt to guess the correct username. This is mostly useful in the main configuration file:

```
my-ec2-config:
  ssh_username:
    - ec2-user
    - ubuntu
    - admin
    - bitnami
```

Multiple security groups can also be specified in the same fashion:

```
my-ec2-config:
  securitygroup:
    - default
    - extra
```

Your instances may optionally make use of EC2 Spot Instances. The following example will request that spot instances be used and your maximum bid will be $0.10. Keep in mind that different spot prices may be needed based on the current value of the various EC2 instance sizes. You can check current and past spot instance pricing via the EC2 API or AWS Console.

```
my-ec2-config:
  spot_config:
    spot_price: 0.10
```

By default, the spot instance type is set to `one-time', meaning it will be launched and, if it's ever terminated for whatever reason, it will not be recreated. If you would like your spot instances to be relaunched after a termination (by your or AWS), set the `type` to `persistent'.

NOTE: Spot instances are a great way to save a bit of money, but you do run the risk of losing your spot instances if the current price for the instance size goes above your maximum bid.

The following parameters may be set in the cloud configuration file to control various aspects of the spot instance launching:

- `wait_for_spot_timeout`: seconds to wait before giving up on spot instance launch (default=600)
- `wait_for_spot_interval`: seconds to wait in between polling requests to determine if a spot instance is available (default=30)
- `wait_for_spot_interval_multiplier`: a multiplier to add to the interval in between requests, which is useful if AWS is throttling your requests (default=1)
- `wait_for_spot_max_failures`: maximum number of failures before giving up on launching your spot instance (default=10)

If you find that you're being throttled by AWS while polling for spot instances, you can set the following in your core cloud configuration file that will double the polling interval after each request to AWS.

```
wait_for_spot_interval: 1
wait_for_spot_interval_multiplier: 2
```

See the AWS Spot Instances documentation for more information.

Block device mappings enable you to specify additional EBS volumes or instance store volumes when the instance is launched. This setting is also available on each cloud profile. Note that the number of instance stores varies by instance type. If more mappings are provided than are supported by the instance type, mappings will be created in the order provided and additional mappings will be ignored. Consult the AWS documentation for a listing of the available instance stores, device names, and mount points.

```
my-ec2-config:
  block_device_mappings:
    - DeviceName: /dev/sdb
      VirtualName: ephemeral0
    - DeviceName: /dev/sdc
      VirtualName: ephemeral1
```

You can also use block device mappings to change the size of the root device at the provisioning time. For example, assuming the root device is `/dev/sda', you can set its size to 100G by using the following configuration.

```
my-ec2-config:
  block_device_mappings:
    - DeviceName: /dev/sda
      Ebs.VolumeSize: 100
      Ebs.VolumeType: gp2
      Ebs.SnapshotId: dummy0
```

Existing EBS volumes may also be attached (not created) to your instances or you can create new EBS volumes based on EBS snapshots. To simply attach an existing volume use the `volume_id` parameter.

```
device: /dev/xvdj
mount_point: /mnt/my_ebs
volume_id: vol-12345abcd
```

Or, to create a volume from an EBS snapshot, use the `snapshot` parameter.

```
device: /dev/xvdj
mount_point: /mnt/my_ebs
snapshot: snap-abcd12345
```

Note that `volume_id` will take precedence over the `snapshot` parameter.

Tags can be set once an instance has been launched.

```
my-ec2-config:
    tag:
        tag0: value
        tag1: value
```

Modify EC2 Tags

One of the features of EC2 is the ability to tag resources. In fact, under the hood, the names given to EC2 instances by salt-cloud are actually just stored as a tag called Name. Salt Cloud has the ability to manage these tags:

```
salt-cloud -a get_tags mymachine
salt-cloud -a set_tags mymachine tag1=somestuff tag2='Other stuff'
salt-cloud -a del_tags mymachine tag1,tag2,tag3
```

It is possible to manage tags on any resource in EC2 with a Resource ID, not just instances:

```
salt-cloud -f get_tags my_ec2 resource_id=af5467ba
salt-cloud -f set_tags my_ec2 resource_id=af5467ba tag1=somestuff
salt-cloud -f del_tags my_ec2 resource_id=af5467ba tag1,tag2,tag3
```

Rename EC2 Instances

As mentioned above, EC2 instances are named via a tag. However, renaming an instance by renaming its tag will cause the salt keys to mismatch. A rename function exists which renames both the instance, and the salt keys.

```
salt-cloud -a rename mymachine newname=yourmachine
```

EC2 Termination Protection

EC2 allows the user to enable and disable termination protection on a specific instance. An instance with this protection enabled cannot be destroyed.

```
salt-cloud -a enable_term_protect mymachine
salt-cloud -a disable_term_protect mymachine
```

Rename on Destroy

When instances on EC2 are destroyed, there will be a lag between the time that the action is sent, and the time that Amazon cleans up the instance. During this time, the instance still retails a Name tag, which will cause a collision if the creation of an instance with the same name is attempted before the cleanup occurs. In order to avoid such collisions, Salt Cloud can be configured to rename instances when they are destroyed. The new name will look something like:

```
myinstance-DEL20f5b8ad4eb64ed88f2c428df80a1a0c
```

In order to enable this, add rename_on_destroy line to the main configuration file:

```
my-ec2-config:
  rename_on_destroy: True
```

Listing Images

Normally, images can be queried on a cloud provider by passing the `--list-images` argument to Salt Cloud. This still holds true for EC2:

```
salt-cloud --list-images my-ec2-config
```

However, the full list of images on EC2 is extremely large, and querying all of the available images may cause Salt Cloud to behave as if frozen. Therefore, the default behavior of this option may be modified, by adding an `owner` argument to the provider configuration:

```
owner: aws-marketplace
```

The possible values for this setting are `amazon`, `aws-marketplace`, `self`, `<AWS account ID>` or `all`. The default setting is `amazon`. Take note that `all` and `aws-marketplace` may cause Salt Cloud to appear as if it is freezing, as it tries to handle the large amount of data.

It is also possible to perform this query using different settings without modifying the configuration files. To do this, call the `avail_images` function directly:

```
salt-cloud -f avail_images my-ec2-config owner=aws-marketplace
```

EC2 Images

The following are lists of available AMI images, generally sorted by OS. These lists are on 3rd-party websites, are not managed by Salt Stack in any way. They are provided here as a reference for those who are interested, and contain no warranty (express or implied) from anyone affiliated with Salt Stack. Most of them have never been used, much less tested, by the Salt Stack team.

- Arch Linux
- FreeBSD
- Fedora
- CentOS
- Ubuntu
- Debian
- OmniOS
- All Images on Amazon

show_image

This is a function that describes an AMI on EC2. This will give insight as to the defaults that will be applied to an instance using a particular AMI.

```
$ salt-cloud -f show_image ec2 image=ami-fd20ad94
```

show_instance

This action is a thin wrapper around `--full-query`, which displays details on a single instance only. In an environment with several machines, this will save a user from having to sort through all instance data, just to examine a single instance.

```
$ salt-cloud -a show_instance myinstance
```

ebs_optimized

This argument enables switching of the EbsOptimized setting which default to `false'. Indicates whether the instance is optimized for EBS I/O. This optimization provides dedicated throughput to Amazon EBS and an optimized configuration stack to provide optimal Amazon EBS I/O performance. This optimization isn't available with all instance types. Additional usage charges apply when using an EBS-optimized instance.

This setting can be added to the profile or map file for an instance.

If set to True, this setting will enable an instance to be EbsOptimized

```
ebs_optimized: True
```

This can also be set as a cloud provider setting in the EC2 cloud configuration:

```
my-ec2-config:
  ebs_optimized: True
```

del_root_vol_on_destroy

This argument overrides the default DeleteOnTermination setting in the AMI for the EBS root volumes for an instance. Many AMIs contain `false' as a default, resulting in orphaned volumes in the EC2 account, which may unknowingly be charged to the account. This setting can be added to the profile or map file for an instance.

If set, this setting will apply to the root EBS volume

```
del_root_vol_on_destroy: True
```

This can also be set as a cloud provider setting in the EC2 cloud configuration:

```
my-ec2-config:
  del_root_vol_on_destroy: True
```

del_all_vols_on_destroy

This argument overrides the default DeleteOnTermination setting in the AMI for the not-root EBS volumes for an instance. Many AMIs contain `false' as a default, resulting in orphaned volumes in the EC2 account, which may unknowingly be charged to the account. This setting can be added to the profile or map file for an instance.

If set, this setting will apply to any (non-root) volumes that were created by salt-cloud using the `volumes' setting.

The volumes will not be deleted under the following conditions * If a volume is detached before terminating the instance * If a volume is created without this setting and attached to the instance

```
del_all_vols_on_destroy: True
```

This can also be set as a cloud provider setting in the EC2 cloud configuration:

```
my-ec2-config:
  del_all_vols_on_destroy: True
```

The setting for this may be changed on all volumes of an existing instance using one of the following commands:

```
salt-cloud -a delvol_on_destroy myinstance
salt-cloud -a keepvol_on_destroy myinstance
salt-cloud -a show_delvol_on_destroy myinstance
```

The setting for this may be changed on a volume on an existing instance using one of the following commands:

```
salt-cloud -a delvol_on_destroy myinstance device=/dev/sda1
salt-cloud -a delvol_on_destroy myinstance volume_id=vol-1a2b3c4d
salt-cloud -a keepvol_on_destroy myinstance device=/dev/sda1
salt-cloud -a keepvol_on_destroy myinstance volume_id=vol-1a2b3c4d
salt-cloud -a show_delvol_on_destroy myinstance device=/dev/sda1
salt-cloud -a show_delvol_on_destroy myinstance volume_id=vol-1a2b3c4d
```

EC2 Termination Protection

EC2 allows the user to enable and disable termination protection on a specific instance. An instance with this protection enabled cannot be destroyed. The EC2 driver adds a show_term_protect action to the regular EC2 functionality.

```
salt-cloud -a show_term_protect mymachine
salt-cloud -a enable_term_protect mymachine
salt-cloud -a disable_term_protect mymachine
```

Alternate Endpoint

Normally, EC2 endpoints are build using the region and the service_url. The resulting endpoint would follow this pattern:

```
ec2.<region>.<service_url>
```

This results in an endpoint that looks like:

```
ec2.us-east-1.amazonaws.com
```

There are other projects that support an EC2 compatibility layer, which this scheme does not account for. This can be overridden by specifying the endpoint directly in the main cloud configuration file:

```
my-ec2-config:
  endpoint: myendpoint.example.com:1138/services/Cloud
```

Volume Management

The EC2 driver has several functions and actions for management of EBS volumes.

Creating Volumes

A volume may be created, independent of an instance. A zone must be specified. A size or a snapshot may be specified (in GiB). If neither is given, a default size of 10 GiB will be used. If a snapshot is given, the size of the snapshot will be used.

```
salt-cloud -f create_volume ec2 zone=us-east-1b
salt-cloud -f create_volume ec2 zone=us-east-1b size=10
salt-cloud -f create_volume ec2 zone=us-east-1b snapshot=snap12345678
salt-cloud -f create_volume ec2 size=10 type=standard
salt-cloud -f create_volume ec2 size=10 type=io1 iops=1000
```

Attaching Volumes

Unattached volumes may be attached to an instance. The following values are required; name or instance_id, volume_id, and device.

```
salt-cloud -a attach_volume myinstance volume_id=vol-12345 device=/dev/sdb1
```

Show a Volume

The details about an existing volume may be retrieved.

```
salt-cloud -a show_volume myinstance volume_id=vol-12345
salt-cloud -f show_volume ec2 volume_id=vol-12345
```

Detaching Volumes

An existing volume may be detached from an instance.

```
salt-cloud -a detach_volume myinstance volume_id=vol-12345
```

Deleting Volumes

A volume that is not attached to an instance may be deleted.

```
salt-cloud -f delete_volume ec2 volume_id=vol-12345
```

Managing Key Pairs

The EC2 driver has the ability to manage key pairs.

Creating a Key Pair

A key pair is required in order to create an instance. When creating a key pair with this function, the return data will contain a copy of the private key. This private key is not stored by Amazon, will not be obtainable past this point, and should be stored immediately.

```
salt-cloud -f create_keypair ec2 keyname=mykeypair
```

Show a Key Pair

This function will show the details related to a key pair, not including the private key itself (which is not stored by Amazon).

```
salt-cloud -f show_keypair ec2 keyname=mykeypair
```

Delete a Key Pair

This function removes the key pair from Amazon.

```
salt-cloud -f delete_keypair ec2 keyname=mykeypair
```

Launching instances into a VPC

Simple launching into a VPC

In the amazon web interface, identify the id of the subnet into which your image should be created. Then, edit your cloud.profiles file like so:-

```
profile-id:
  provider: provider-name
  subnetid: subnet-XXXXXXXX
  image: ami-XXXXXXXX
  size: m1.medium
  ssh_username: ubuntu
  securitygroupid:
    - sg-XXXXXXXX
```

Specifying interface properties

New in version 2014.7.0.

Launching into a VPC allows you to specify more complex configurations for the network interfaces of your virtual machines, for example:-

```
profile-id:
  provider: provider-name
  image: ami-XXXXXXXX
  size: m1.medium
  ssh_username: ubuntu

  # Do not include either 'subnetid' or 'securitygroupid' here if you are
  # going to manually specify interface configuration
  #
  network_interfaces:
    - DeviceIndex: 0
      SubnetId: subnet-XXXXXXXX
      SecurityGroupId:
        - sg-XXXXXXXX

      # Uncomment this to associate an existing Elastic IP Address with
      # this network interface:
      #
      # associate_eip: eni-XXXXXXXX

      # You can allocate more than one IP address to an interface. Use the
      # 'ip addr list' command to see them.
      #
      # SecondaryPrivateIpAddressCount: 2

      # Uncomment this to allocate a new Elastic IP Address to this
      # interface (will be associated with the primary private ip address
      # of the interface
      #
      # allocate_new_eip: True

      # Uncomment this instead to allocate a new Elastic IP Address to
      # both the primary private ip address and each of the secondary ones
```

```
    #
    allocate_new_eips: True
```

Note that it is an error to assign a `subnetid' or `securitygroupid' to a profile where the interfaces are manually configured like this. These are both really properties of each network interface, not of the machine itself.

15.5.5 Getting Started With GoGrid

GoGrid is a public cloud provider supporting Linux and Windows.

Dependencies

- Libcloud >= 0.13.2

Configuration

To use Salt Cloud with GoGrid log into the GoGrid web interface and create an API key. Do this by clicking on ``My Account'' and then going to the API Keys tab.

The `apikey` and the `sharedsecret` configuration parameters need to be set in the configuration file to enable interfacing with GoGrid:

```
# Note: This example is for /etc/salt/cloud.providers or any file in the
# /etc/salt/cloud.providers.d/ directory.

my-gogrid-config:
  provider: gogrid
  apikey: asdff7896asdh789
  sharedsecret: saltybacon
```

Profiles

Cloud Profiles

Set up an initial profile at `/etc/salt/cloud.profiles` or in the `/etc/salt/cloud.profiles.d/` directory:

```
gogrid_512:
  provider: my-gogrid-config
  size: 512MB
  image: CentOS 6.2 (64-bit) w/ None
```

Sizes can be obtained using the `--list-sizes` option for the `salt-cloud` command:

```
# salt-cloud --list-sizes my-gogrid-config
my-gogrid-config:
    ----------
    gogrid:
        ----------
        512MB:
            ----------
            bandwidth:
                None
```

```
            disk:
                30
            driver:
            get_uuid:
            id:
                512MB
            name:
                512MB
            price:
                0.095
            ram:
                512
            uuid:
                bde1e4d7c3a643536e42a35142c7caac34b060e9
...SNIP...
```

Images can be obtained using the `--list-images` option for the `salt-cloud` command:

```
# salt-cloud --list-images my-gogrid-config
my-gogrid-config:
    ----------
    gogrid:
        ----------
        CentOS 6.4 (64-bit) w/ None:
            ----------
            driver:
            extra:
                ----------
            get_uuid:
            id:
                18094
            name:
                CentOS 6.4 (64-bit) w/ None
            uuid:
                bfd4055389919e01aa6261828a96cf54c8dcc2c4
...SNIP...
```

15.5.6 Getting Started With Google Compute Engine

Google Compute Engine (GCE) is Google-infrastructure as a service that lets you run your large-scale computing workloads on virtual machines. This document covers how to use Salt Cloud to provision and manage your virtual machines hosted within Google's infrastructure.

You can find out more about GCE and other Google Cloud Platform services at https://cloud.google.com.

Dependencies

- Libcloud >= 0.14.0-beta3
- PyCrypto >= 2.1.
- A Google Cloud Platform account with Compute Engine enabled
- A registered Service Account for authorization
- Oh, and obviously you'll need salt

Google Compute Engine Setup

1. Sign up for Google Cloud Platform

   Go to https://cloud.google.com and use your Google account to sign up for Google Cloud Platform and complete the guided instructions.

2. Create a Project

   Next, go to the console at https://cloud.google.com/console and create a new Project. Make sure to select your new Project if you are not automatically directed to the Project.

   Projects are a way of grouping together related users, services, and billing. You may opt to create multiple Projects and the remaining instructions will need to be completed for each Project if you wish to use GCE and Salt Cloud to manage your virtual machines.

3. Enable the Google Compute Engine service

   In your Project, either just click *Compute Engine* to the left, or go to the *APIs & auth* section and *APIs* link and enable the Google Compute Engine service.

4. Create a Service Account

   To set up authorization, navigate to *APIs & auth* section and then the *Credentials* link and click the *CREATE NEW CLIENT ID* button. Select *Service Account* and click the *Create Client ID* button. This will automatically download a `.json` file, which should be ignored. Look for a new *Service Account* section in the page and record the generated email address for the matching key/fingerprint. The email address will be used in the `service_account_email_address` of the `/etc/salt/cloud` file.

5. Key Format

   In the new *Service Account* section, click *Generate new P12 key*, which will automatically download a `.p12` private key file. The `.p12` private key needs to be converted to a format compatible with libcloud. This new Google-generated private key was encrypted using *notasecret* as a passphrase. Use the following command and record the location of the converted private key and record the location for use in the `service_account_private_key` of the `/etc/salt/cloud` file:

```
openssl pkcs12 -in ORIG.p12 -passin pass:notasecret \
-nodes -nocerts | openssl rsa -out NEW.pem
```

Configuration

Set up the cloud config at `/etc/salt/cloud`:

```
# Note: This example is for /etc/salt/cloud

providers:
  gce-config:
    # Set up the Project name and Service Account authorization
    #
    project: "your-project-id"
    service_account_email_address: "123-a5gt@developer.gserviceaccount.com"
    service_account_private_key: "/path/to/your/NEW.pem"

    # Set up the location of the salt master
    #
    minion:
      master: saltmaster.example.com

    # Set up grains information, which will be common for all nodes
```

```
    # using this provider
    grains:
      node_type: broker
      release: 1.0.1

    provider: gce
```

Note: The value provided for `project` must not contain underscores or spaces and is labeled as ``Project ID'' on the Google Developers Console.

Cloud Profiles

Set up an initial profile at `/etc/salt/cloud.profiles`:

```
all_settings:
  image: centos-6
  size: n1-standard-1
  location: europe-west1-b
  network: default
  tags: '["one", "two", "three"]'
  metadata: '{"one": "1", "2": "two"}'
  use_persistent_disk: True
  delete_boot_pd: False
  deploy: True
  make_master: False
  provider: gce-config
```

The profile can be realized now with a salt command:

```
salt-cloud -p all_settings gce-instance
```

This will create an salt minion instance named `gce-instance` in GCE. If the command was executed on the salt-master, its Salt key will automatically be signed on the master.

Once the instance has been created with salt-minion installed, connectivity to it can be verified with Salt:

```
salt 'ami.example.com' test.ping
```

GCE Specific Settings

Consult the sample profile below for more information about GCE specific settings. Some of them are mandatory and are properly labeled below but typically also include a hard-coded default.

```
all_settings:

  # Image is used to define what Operating System image should be used
  # to for the instance.  Examples are Debian 7 (wheezy) and CentOS 6.
  #
  # MANDATORY
  #
  image: centos-6

  # A 'size', in GCE terms, refers to the instance's 'machine type'.  See
  # the on-line documentation for a complete list of GCE machine types.
  #
```

```
  # MANDATORY
  #
  size: n1-standard-1

  # A 'location', in GCE terms, refers to the instance's 'zone'.  GCE
  # has the notion of both Regions (e.g. us-central1, europe-west1, etc)
  # and Zones (e.g. us-central1-a, us-central1-b, etc).
  #
  # MANDATORY
  #
  location: europe-west1-b

  # Use this setting to define the network resource for the instance.
  # All GCE projects contain a network named 'default' but it's possible
  # to use this setting to create instances belonging to a different
  # network resource.
  #
  network: default

  # GCE supports instance/network tags and this setting allows you to
  # set custom tags.  It should be a list of strings and must be
  # parse-able by the python ast.literal_eval() function to convert it
  # to a python list.
  #
  tags: '["one", "two", "three"]'

  # GCE supports instance metadata and this setting allows you to
  # set custom metadata.  It should be a hash of key/value strings and
  # parse-able by the python ast.literal_eval() function to convert it
  # to a python dictionary.
  #
  metadata: '{"one": "1", "2": "two"}'

  # Use this setting to ensure that when new instances are created,
  # they will use a persistent disk to preserve data between instance
  # terminations and re-creations.
  #
  use_persistent_disk: True

  # In the event that you wish the boot persistent disk to be permanently
  # deleted when you destroy an instance, set delete_boot_pd to True.
  #
  delete_boot_pd: False

  # Specify whether to use public or private IP for deploy script.
  # Valid options are:
  #     private_ips - The salt-master is also hosted with GCE
  #     public_ips - The salt-master is hosted outside of GCE
  ssh_interface: public_ips

  # Per instance setting: Used a named fixed IP address to this host.
  # Valid options are:
  #     ephemeral - The host will use a GCE ephemeral IP
  #     None - No external IP will be configured on this host.
  # Optionally, pass the name of a GCE address to use a fixed IP address.
  # If the address does not already exist, it will be created.
  external_ip: "ephemeral"
```

GCE instances do not allow remote access to the root user by default. Instead, another user must be used to run the deploy script using sudo. Append something like this to `/etc/salt/cloud.profiles`:

```
all_settings:
    ...

    # SSH to GCE instances as gceuser
    ssh_username: gceuser

    # Use the local private SSH key file located here
    ssh_keyfile: /etc/cloud/google_compute_engine
```

If you have not already used this SSH key to login to instances in this GCE project you will also need to add the public key to your projects metadata at https://cloud.google.com/console. You could also add it via the metadata setting too:

```
all_settings:
    ...

    metadata: '{"one": "1", "2": "two",
                "sshKeys": "gceuser:ssh-rsa <Your SSH Public Key> gceuser@host"}'
```

Single instance details

This action is a thin wrapper around `--full-query`, which displays details on a single instance only. In an environment with several machines, this will save a user from having to sort through all instance data, just to examine a single instance.

```
salt-cloud -a show_instance myinstance
```

Destroy, persistent disks, and metadata

As noted in the provider configuration, it's possible to force the boot persistent disk to be deleted when you destroy the instance. The way that this has been implemented is to use the instance metadata to record the cloud profile used when creating the instance. When `destroy` is called, if the instance contains a `salt-cloud-profile` key, it's value is used to reference the matching profile to determine if `delete_boot_pd` is set to `True`.

Be aware that any GCE instances created with salt cloud will contain this custom `salt-cloud-profile` metadata entry.

List various resources

It's also possible to list several GCE resources similar to what can be done with other providers. The following commands can be used to list GCE zones (locations), machine types (sizes), and images.

```
salt-cloud --list-locations gce
salt-cloud --list-sizes gce
salt-cloud --list-images gce
```

Persistent Disk

The Compute Engine provider provides functions via salt-cloud to manage your Persistent Disks. You can create and destroy disks as well as attach and detach them from running instances.

Create

When creating a disk, you can create an empty disk and specify its size (in GB), or specify either an `image' or `snapshot'.

```
salt-cloud -f create_disk gce disk_name=pd location=us-central1-b size=200
```

Delete

Deleting a disk only requires the name of the disk to delete

```
salt-cloud -f delete_disk gce disk_name=old-backup
```

Attach

Attaching a disk to an existing instance is really an `action' and requires both an instance name and disk name. It's possible to use this ation to create bootable persistent disks if necessary. Compute Engine also supports attaching a persistent disk in READ_ONLY mode to multiple instances at the same time (but then cannot be attached in READ_WRITE to any instance).

```
salt-cloud -a attach_disk myinstance disk_name=pd mode=READ_WRITE boot=yes
```

Detach

Detaching a disk is also an action against an instance and only requires the name of the disk. Note that this does *not* safely sync and umount the disk from the instance. To ensure no data loss, you must first make sure the disk is unmounted from the instance.

```
salt-cloud -a detach_disk myinstance disk_name=pd
```

Show disk

It's also possible to look up the details for an existing disk with either a function or an action.

```
salt-cloud -a show_disk myinstance disk_name=pd
salt-cloud -f show_disk gce disk_name=pd
```

Create snapshot

You can take a snapshot of an existing disk's content. The snapshot can then in turn be used to create other persistent disks. Note that to prevent data corruption, it is strongly suggested that you unmount the disk prior to taking a snapshot. You must name the snapshot and provide the name of the disk.

```
salt-cloud -f create_snapshot gce name=backup-20140226 disk_name=pd
```

Delete snapshot

You can delete a snapshot when it's no longer needed by specifying the name of the snapshot.

```
salt-cloud -f delete_snapshot gce name=backup-20140226
```

Show snapshot

Use this function to look up information about the snapshot.

```
salt-cloud -f show_snapshot gce name=backup-20140226
```

Networking

Compute Engine supports multiple private networks per project. Instances within a private network can easily communicate with each other by an internal DNS service that resolves instance names. Instances within a private network can also communicate with either directly without needing special routing or firewall rules even if they span different regions/zones.

Networks also support custom firewall rules. By default, traffic between instances on the same private network is open to all ports and protocols. Inbound SSH traffic (port 22) is also allowed but all other inbound traffic is blocked.

Create network

New networks require a name and CIDR range. New instances can be created and added to this network by setting the network name during create. It is not possible to add/remove existing instances to a network.

```
salt-cloud -f create_network gce name=mynet cidr=10.10.10.0/24
```

Destroy network

Destroy a network by specifying the name. Make sure that there are no instances associated with the network prior to deleting it or you'll have a bad day.

```
salt-cloud -f delete_network gce name=mynet
```

Show network

Specify the network name to view information about the network.

```
salt-cloud -f show_network gce name=mynet
```

Create address

Create a new named static IP address in a region.

```
salt-cloud -f create_address gce name=my-fixed-ip region=us-central1
```

Delete address

Delete an existing named fixed IP address.

```
salt-cloud -f delete_address gce name=my-fixed-ip region=us-central1
```

Show address

View details on a named address.

```
salt-cloud -f show_address gce name=my-fixed-ip region=us-central1
```

Create firewall

You'll need to create custom firewall rules if you want to allow other traffic than what is described above. For instance, if you run a web service on your instances, you'll need to explicitly allow HTTP and/or SSL traffic. The firewall rule must have a name and it will use the `default' network unless otherwise specified with a `network' attribute. Firewalls also support instance tags for source/destination

```
salt-cloud -f create_fwrule gce name=web allow=tcp:80,tcp:443,icmp
```

Delete firewall

Deleting a firewall rule will prevent any previously allowed traffic for the named firewall rule.

```
salt-cloud -f delete_fwrule gce name=web
```

Show firewall

Use this function to review an existing firewall rule's information.

```
salt-cloud -f show_fwrule gce name=web
```

Load Balancer

Compute Engine possess a load-balancer feature for splitting traffic across multiple instances. Please reference the documentation for a more complete discription.

The load-balancer functionality is slightly different than that described in Google's documentation. The concept of *TargetPool* and *ForwardingRule* are consolidated in salt-cloud/libcloud. HTTP Health Checks are optional.

HTTP Health Check

HTTP Health Checks can be used as a means to toggle load-balancing across instance members, or to detect if an HTTP site is functioning. A common use-case is to set up a health check URL and if you want to toggle traffic on/off to an instance, you can temporarily have it return a non-200 response. A non-200 response to the load-balancer's health check will keep the LB from sending any new traffic to the ``down'' instance. Once the instance's health check URL beings returning 200-responses, the LB will again start to send traffic to it. Review Compute Engine's

documentation for allowable parameters. You can use the following salt-cloud functions to manage your HTTP health checks.

```
salt-cloud -f create_hc gce name=myhc path=/ port=80
salt-cloud -f delete_hc gce name=myhc
salt-cloud -f show_hc gce name=myhc
```

Load-balancer

When creating a new load-balancer, it requires a name, region, port range, and list of members. There are other optional parameters for protocol, and list of health checks. Deleting or showing details about the LB only requires the name.

```
salt-cloud -f create_lb gce name=lb region=... ports=80 members=w1,w2,w3
salt-cloud -f delete_lb gce name=lb
salt-cloud -f show_lb gce name=lb
```

You can also create a load balancer using a named fixed IP addressby specifying the name of the address. If the address does not exist yet it will be created.

```
salt-cloud -f create_lb gce name=my-lb region=us-central1 ports=234 members=s1,s2,s3 address=my-lb-ip
```

Attach and Detach LB

It is possible to attach or detach an instance from an existing load-balancer. Both the instance and load-balancer must exist before using these functions.

```
salt-cloud -f attach_lb gce name=lb member=w4
salt-cloud -f detach_lb gce name=lb member=oops
```

15.5.7 Getting Started With HP Cloud

HP Cloud is a major public cloud platform and uses the libcloud *openstack* driver. The current version of OpenStack that HP Cloud uses is Havana. When an instance is booted, it must have a floating IP added to it in order to connect to it and further below you will see an example that adds context to this statement.

Set up a cloud provider configuration file

To use the *openstack* driver for HP Cloud, set up the cloud provider configuration file as in the example shown below:

`/etc/salt/cloud.providers.d/hpcloud.conf`:

```
hpcloud-config:
  # Set the location of the salt-master
  #
  minion:
    master: saltmaster.example.com

  # Configure HP Cloud using the OpenStack plugin
  #
  identity_url: https://region-b.geo-1.identity.hpcloudsvc.com:35357/v2.0/tokens
  compute_name: Compute
  protocol: ipv4
```

```
  # Set the compute region:
  #
  compute_region: region-b.geo-1

  # Configure HP Cloud authentication credentials
  #
  user: myname
  tenant: myname-project1
  password: xxxxxxxxx

  # keys to allow connection to the instance launched
  #
  ssh_key_name: yourkey
  ssh_key_file: /path/to/key/yourkey.priv

  provider: openstack
```

The subsequent example that follows is using the openstack driver.

Compute Region

Originally, HP Cloud, in its OpenStack Essex version (1.0), had 3 availability zones in one region, US West (region-a.geo-1), which each behaved each as a region.

This has since changed, and the current OpenStack Havana version of HP Cloud (1.1) now has simplified this and now has two regions to choose from:

```
region-a.geo-1 -> US West
region-b.geo-1 -> US East
```

Authentication

The `user` is the same user as is used to log into the HP Cloud management UI. The `tenant` can be found in the upper left under ``Project/Region/Scope''. It is often named the same as `user` albeit with a `-project1` appended. The `password` is of course what you created your account with. The management UI also has other information such as being able to select US East or US West.

Set up a cloud profile config file

The profile shown below is a know working profile for an Ubuntu instance. The profile configuration file is stored in the following location:

`/etc/salt/cloud.profiles.d/hp_ae1_ubuntu.conf`:

```
hp_ae1_ubuntu:
    provider: hp_ae1
    image: 9302692b-b787-4b52-a3a6-daebb79cb498
    ignore_cidr: 10.0.0.1/24
    networks:
      - floating: Ext-Net
    size: standard.small
    ssh_key_file: /root/keys/test.key
    ssh_key_name: test
    ssh_username: ubuntu
```

Some important things about the example above:

- The `image` parameter can use either the image name or image ID which you can obtain by running in the example below (this case US East):

```
# salt-cloud --list-images hp_ae1
```

- The parameter `ignore_cidr` specifies a range of addresses to ignore when trying to connect to the instance. In this case, it's the range of IP addresses used for an private IP of the instance.
- The parameter `networks` is very important to include. In previous versions of Salt Cloud, this is what made it possible for salt-cloud to be able to attach a floating IP to the instance in order to connect to the instance and set up the minion. The current version of salt-cloud doesn't require it, though having it is of no harm either. Newer versions of salt-cloud will use this, and without it, will attempt to find a list of floating IP addresses to use regardless.
- The `ssh_key_file` and `ssh_key_name` are the keys that will make it possible to connect to the instance to set up the minion
- The `ssh_username` parameter, in this case, being that the image used will be ubuntu, will make it possible to not only log in but install the minion

Launch an instance

To instantiate a machine based on this profile (example):

```
# salt-cloud -p hp_ae1_ubuntu ubuntu_instance_1
```

After several minutes, this will create an instance named ubuntu_instance_1 running in HP Cloud in the US East region and will set up the minion and then return information about the instance once completed.

Manage the instance

Once the instance has been created with salt-minion installed, connectivity to it can be verified with Salt:

```
# salt ubuntu_instance_1 ping
```

SSH to the instance

Additionally, the instance can be accessed via SSH using the floating IP assigned to it

```
# ssh ubuntu@<floating ip>
```

Using a private IP

Alternatively, in the cloud profile, using the private IP to log into the instance to set up the minion is another option, particularly if salt-cloud is running within the cloud on an instance that is on the same network with all the other instances (minions)

The example below is a modified version of the previous example. Note the use of `ssh_interface`:

```
hp_ae1_ubuntu:
    provider: hp_ae1
    image: 9302692b-b787-4b52-a3a6-daebb79cb498
    size: standard.small
```

```
    ssh_key_file: /root/keys/test.key
    ssh_key_name: test
    ssh_username: ubuntu
    ssh_interface: private_ips
```

With this setup, salt-cloud will use the private IP address to ssh into the instance and set up the salt-minion

15.5.8 Getting Started With Joyent

Joyent is a public cloud provider supporting SmartOS, Linux, FreeBSD, and Windows.

Dependencies

This driver requires the Python `requests` library to be installed.

Configuration

The Joyent cloud requires three configuration parameters. The user name and password that are used to log into the Joyent system, and the location of the private ssh key associated with the Joyent account. The ssh key is needed to send the provisioning commands up to the freshly created virtual machine.

```
# Note: This example is for /etc/salt/cloud.providers or any file in the
# /etc/salt/cloud.providers.d/ directory.

my-joyent-config:
    provider: joyent
    user: fred
    password: saltybacon
    private_key: /root/mykey.pem
    keyname: mykey
```

Profiles

Cloud Profiles

Set up an initial profile at `/etc/salt/cloud.profiles` or in the `/etc/salt/cloud.profiles.d/` directory:

```
joyent_512
  provider: my-joyent-config
  size: Extra Small 512 MB
  image: Arch Linux 2013.06
```

Sizes can be obtained using the `--list-sizes` option for the `salt-cloud` command:

```
# salt-cloud --list-sizes my-joyent-config
my-joyent-config:
    ----------
    joyent:
        ----------
        Extra Small 512 MB:
            ----------
            default:
```

```
                false
            disk:
                15360
            id:
                Extra Small 512 MB
            memory:
                512
            name:
                Extra Small 512 MB
            swap:
                1024
            vcpus:
                1
...SNIP...
```

Images can be obtained using the `--list-images` option for the `salt-cloud` command:

```
# salt-cloud --list-images my-joyent-config
my-joyent-config:
    ----------
    joyent:
        ----------
        base:
            ----------
            description:
                A 32-bit SmartOS image with just essential packages
                installed. Ideal for users who are comfortable with setting
                up their own environment and tools.
            disabled:
                False
            files:
                ----------
                - compression:
                    bzip2
                - sha1:
                    40cdc6457c237cf6306103c74b5f45f5bf2d9bbe
                - size:
                    82492182
            name:
                base
            os:
                smartos
            owner:
                352971aa-31ba-496c-9ade-a379feaecd52
            public:
                True
...SNIP...
```

SmartDataCenter

This driver can also be used with the Joyent SmartDataCenter project. More details can be found at:

Using SDC requires that an api_host_suffix is set. The default value for this is *.api.joyentcloud.com*. All characters, including the leading ., should be included:

```
api_host_suffix: .api.myhostname.com
```

Miscellaneous Configuration

The following configuration items can be set in either `provider` or `profile` confuration files.

use_ssl

When set to `True` (the default), attach `https://` to any URL that does not already have `http://` or `https://` included at the beginning. The best practice is to leave the protocol out of the URL, and use this setting to manage it.

verify_ssl

When set to `True` (the default), the underlying web library will verify the SSL certificate. This should only be set to `False` for debugging.`

15.5.9 Getting Started With LXC

The LXC module is designed to install Salt in an LXC container on a controlled and possibly remote minion.

In other words, Salt will connect to a minion, then from that minion:

- Provision and configure a container for networking access
- Use *saltify* to deploy salt and re-attach to master

Limitations

- You can only act on one minion and one provider at a time.
- Listing images must be targeted to a particular LXC provider (nothing will be outputted with `all`)

Operation

Salt's LXC support does not use lxc.init. This enables it to tie minions to a master in a more generic fashion (if any masters are defined) and allows other custom association code.

Order of operation:

- Create the LXC container using `the LXC execution module` on the desired minion (clone or template)
- Change LXC config options (if any need to be changed)
- Start container
- Change base passwords if any
- Change base DNS configuration if necessary
- Wait for LXC container to be up and ready for ssh
- Test SSH connection and bailout in error
- Via SSH (with the help of saltify), upload deploy script and seeds, then re-attach the minion.

Provider configuration

Here is a simple provider configuration:

```
# Note: This example goes in /etc/salt/cloud.providers or any file in the
# /etc/salt/cloud.providers.d/ directory.
devhost10-lxc:
  target: devhost10
  provider: lxc
```

Profile configuration

Here are the options to configure your containers:

```
``target``
    Host minion id to install the lxc Container into
``profile``
    Name of the profile containing the LXC configuration

Container creation/clone options:
    Create a container by cloning:
        ``from_container``
            Name of an original container using clone
        ``snapshot``
            Do we use snapshots on cloned filesystems
    Create a container from scratch using an LXC template:
        image
            template to use
        backing
            Backing store type (None, lvm, brtfs)
        lvname
            LVM logical volume name, if any
        fstype
            Type of filesystem
size
    Size of the containera (for brtfs, or lvm)
vgname
    LVM Volume Group name, if any
users
    Names of the users to be pre-created inside the container
ssh_username
    Username of the SSH systems administrator inside the container
sudo
    Do we use sudo
ssh_gateway
    if the minion is not in your 'topmaster' network, use
    that gateway to connect to the lxc container.
    This may be the public ip of the hosting minion
ssh_gateway_key
    When using gateway, the ssh key of the gateway user (passed to saltify)
ssh_gateway_port
    When using gateway, the ssh port of the gateway (passed to saltify)
ssh_gateway_user
    When using gateway, user to login as via SSH (passed to saltify)
password
    password for root and sysadmin (see "users" parameter above)
mac
```

```
    mac address to assign to the container's network interface
ip
    IP address to assign to the container's network interface
netmask
    netmask for the network interface's IP
bridge
    bridge under which the container's network interface will be enslaved
dnsservers
    List of DNS servers to use--this is optional.  If present, DNS
    servers will be restricted to that list if used
lxc_conf_unset
    Configuration variables to unset in this container's LXC configuration
lxc_conf
    LXC configuration variables to add in this container's LXC configuration
minion
    minion configuration (see :doc:`Minion Configuration in Salt Cloud </topics/cloud/config>`)
```

```
# Note: This example would go in /etc/salt/cloud.profile or any file in the
# /etc/salt/cloud.profile.d/ directory.
devhost10-lxc:
  provider: devhost10-lxc
  from_container: ubuntu
  backing: lvm
  sudo: True
  size: 3g
  ip: 10.0.3.9
  minion:
    master: 10.5.0.1
    master_port: 4506
  lxc_conf:
    - lxc.utsname: superlxc
```

Driver Support

- Container creation
- Image listing (LXC templates)
- Running container information (IP addresses, etc.)

15.5.10 Getting Started With Linode

Linode is a public cloud provider with a focus on Linux instances.

Dependencies

- linode-python >= 1.1.1

OR

- Libcloud >= 0.13.2

This driver supports accessing Linode via linode-python or Apache Libcloud. Linode-python is recommended, it is more full-featured than Libcloud. In particular using linode-python enables stopping, starting, and cloning machines.

Driver selection is automatic. If linode-python is present it will be used. If it is absent, salt-cloud will fall back to Libcloud. If neither are present salt-cloud will abort.

NOTE: linode-python 1.1.1 or later is recommended. Earlier versions of linode-python should work but leak sensitive information into the debug logs.

Linode-python can be downloaded from https://github.com/tjfontaine/linode-python or installed via pip.

Configuration

Linode requires a single API key, but the default root password for new instances also needs to be set:

```
# Note: This example is for /etc/salt/cloud.providers or any file in the
# /etc/salt/cloud.providers.d/ directory.

my-linode-config:
  apikey: asldkgfakl;sdfjsjaslfjaklsdjf;askldjfaaklsjdfhasldsadfghdkf
  password: F00barbaz
  ssh_pubkey: ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIKHEOLLbeXgaqRQT9NBAopVz366SdYc0KKX33vAnq+2R user@h
  ssh_key_file: ~/.ssh/id_ed25519
  provider: linode
```

The password needs to be 8 characters and contain lowercase, uppercase, and numbers.

Profiles

Cloud Profiles

Set up an initial profile at `/etc/salt/cloud.profiles` or in the `/etc/salt/cloud.profiles.d/` directory:

```
linode_1024:
  provider: my-linode-config
  size: Linode 1024
  image: Arch Linux 2013.06
```

Sizes can be obtained using the `--list-sizes` option for the `salt-cloud` command:

```
# salt-cloud --list-sizes my-linode-config
my-linode-config:
    ----------
    linode:
        ----------
        Linode 1024:
            ----------
            bandwidth:
                2000
            disk:
                49152
            driver:
            get_uuid:
            id:
                1
            name:
                Linode 1024
            price:
                20.0
```

```
            ram:
                1024
            uuid:
                03e18728ce4629e2ac07c9cbb48afffb8cb499c4
...SNIP...
```

Images can be obtained using the `--list-images` option for the `salt-cloud` command:

```
# salt-cloud --list-images my-linode-config
my-linode-config:
    ----------
    linode:
        ----------
        Arch Linux 2013.06:
            ----------
            driver:
            extra:
                ----------
                64bit:
                    1
                pvops:
                    1
            get_uuid:
            id:
                112
            name:
                Arch Linux 2013.06
            uuid:
                8457f92eaffc92b7666b6734a96ad7abe1a8a6dd
...SNIP...
```

Cloning

When salt-cloud accesses Linode via linode-python it can clone machines.

It is safest to clone a stopped machine. To stop a machine run

```
salt-cloud -a stop machine_to_clone
```

To create a new machine based on another machine, add an entry to your linode cloud profile that looks like this:

```
li-clone:
  provider: linode
  clonefrom: machine_to_clone
  script_args: -C
```

Then run salt-cloud as normal, specifying *-p li-clone*. The profile name can be anything--it doesn't have to be *li-clone*.

Clonefrom: is the name of an existing machine in Linode from which to clone. *Script_args: -C* is necessary to avoid re-deploying Salt via salt-bootstrap. *-C* will just re-deploy keys so the new minion will not have a duplicate key or minion_id on the master.

15.5.11 Getting Started With OpenStack

OpenStack is one the most popular cloud projects. It's an open source project to build public and/or private clouds. You can use Salt Cloud to launch OpenStack instances.

Dependencies

- Libcloud >= 0.13.2

Configuration

- Using the new format, set up the cloud configuration at `/etc/salt/cloud.providers` or `/etc/salt/cloud.providers.d/openstack.conf`:

```
my-openstack-config:
  # Set the location of the salt-master
  #
  minion:
    master: saltmaster.example.com

  # Configure the OpenStack driver
  #
  identity_url: http://identity.youopenstack.com/v2.0/tokens
  compute_name: nova
  protocol: ipv4

  compute_region: RegionOne

  # Configure Openstack authentication credentials
  #
  user: myname
  password: 123456
  # tenant is the project name
  tenant: myproject

  provider: openstack

  # skip SSL certificate validation (default false)
  insecure: false
```

Using nova client to get information from OpenStack

One of the best ways to get information about OpenStack is using the novaclient python package (available in pypi as python-novaclient). The client configuration is a set of environment variables that you can get from the Dashboard. Log in and then go to Project -> Access & security -> API Access and download the ``OpenStack RC file''. Then:

```
source /path/to/your/rcfile
nova credentials
nova endpoints
```

In the `nova endpoints` output you can see the information about `compute_region` and `compute_name`.

Compute Region

It depends on the OpenStack cluster that you are using. Please, have a look at the previous sections.

Authentication

The `user` and `password` is the same user as is used to log into the OpenStack Dashboard.

Profiles

Here is an example of a profile:

```
openstack_512:
  provider: my-openstack-config
  size: m1.tiny
  image: cirros-0.3.1-x86_64-uec
  ssh_key_file: /tmp/test.pem
  ssh_key_name: test
  ssh_interface: private_ips
```

The following list explains some of the important properties.

size can be one of the options listed in the output of `nova flavor-list`.

image can be one of the options listed in the output of `nova image-list`.

ssh_key_file The SSH private key that the salt-cloud uses to SSH into the VM after its first booted in order to execute a command or script. This private key's *public key* must be the openstack public key inserted into the authorized_key's file of the VM's root user account.

ssh_key_name The name of the openstack SSH public key that is inserted into the authorized_keys file of the VM's root user account. Prior to using this public key, you must use openstack commands or the horizon web UI to load that key into the tenant's account. Note that this openstack tenant must be the one you defined in the cloud provider.

ssh_interface This option allows you to create a VM without a public IP. If this option is omitted and the VM does not have a public IP, then the salt-cloud waits for a certain period of time and then destroys the VM.

For more information concerning cloud profiles, see *here*.

change_password

If no ssh_key_file is provided, and the server already exists, change_password will use the api to change the root password of the server so that it can be bootstrapped.

```
change_password: True
```

userdata_file

Use *userdata_file* to specify the userdata file to upload for use with cloud-init if available.

```
userdata_file: /etc/salt/cloud-init/packages.yml
```

15.5.12 Getting Started With Parallels

Parallels Cloud Server is a product by Parallels that delivers a cloud hosting solution. The PARALLELS module for Salt Cloud enables you to manage instances hosted by a provider using PCS. Further information can be found at:

http://www.parallels.com/products/pcs/

- Using the old format, set up the cloud configuration at `/etc/salt/cloud`:

```
# Set up the location of the salt master
#
minion:
    master: saltmaster.example.com

# Set the PARALLELS access credentials (see below)
#
PARALLELS.user: myuser
PARALLELS.password: badpass

# Set the access URL for your PARALLELS provider
#
PARALLELS.url: https://api.cloud.xmission.com:4465/paci/v1.0/
```

- Using the new format, set up the cloud configuration at `/etc/salt/cloud.providers` or `/etc/salt/cloud.providers.d/parallels.conf`:

```
my-parallels-config:
  # Set up the location of the salt master
  #
  minion:
    master: saltmaster.example.com

  # Set the PARALLELS access credentials (see below)
  #
  user: myuser
  password: badpass

  # Set the access URL for your PARALLELS provider
  #
  url: https://api.cloud.xmission.com:4465/paci/v1.0/
  provider: parallels
```

Access Credentials

The `user`, `password`, and `url` will be provided to you by your cloud provider. These are all required in order for the PARALLELS driver to work.

Cloud Profiles

Set up an initial profile at `/etc/salt/cloud.profiles` or `/etc/salt/cloud.profiles.d/parallels.conf`:

- Using the old cloud configuration format:

```
parallels-ubuntu:
    provider: parallels
    image: ubuntu-12.04-x86_64
```

- Using the new cloud configuration format and the cloud configuration example from above:

```
parallels-ubuntu:
    provider: my-parallels-config
    image: ubuntu-12.04-x86_64
```

The profile can be realized now with a salt command:

```
# salt-cloud -p parallels-ubuntu myubuntu
```

This will create an instance named `myubuntu` on the cloud provider. The minion that is installed on this instance will have an `id` of `myubuntu`. If the command was executed on the salt-master, its Salt key will automatically be signed on the master.

Once the instance has been created with salt-minion installed, connectivity to it can be verified with Salt:

```
# salt myubuntu test.ping
```

Required Settings

The following settings are always required for PARALLELS:

- Using the old cloud configuration format:

```
PARALLELS.user: myuser
PARALLELS.password: badpass
PARALLELS.url: https://api.cloud.xmission.com:4465/paci/v1.0/
```

- Using the new cloud configuration format:

```
my-parallels-config:
  user: myuser
  password: badpass
  url: https://api.cloud.xmission.com:4465/paci/v1.0/
  provider: parallels
```

Optional Settings

Unlike other cloud providers in Salt Cloud, Parallels does not utilize a `size` setting. This is because Parallels allows the end-user to specify a more detailed configuration for their instances, than is allowed by many other cloud providers. The following options are available to be used in a profile, with their default settings listed.

```
# Description of the instance. Defaults to the instance name.
desc: <instance_name>

# How many CPU cores, and how fast they are (in MHz)
cpu_number: 1
cpu_power: 1000

# How many megabytes of RAM
ram: 256

# Bandwidth available, in kbps
bandwidth: 100

# How many public IPs will be assigned to this instance
ip_num: 1

# Size of the instance disk (in GiB)
disk_size: 10

# Username and password
ssh_username: root
password: <value from PARALLELS.password>
```

```
# The name of the image, from ``salt-cloud --list-images parallels``
image: ubuntu-12.04-x86_64
```

15.5.13 Getting Started With Proxmox

Proxmox Virtual Environment is a complete server virtualization management solution, based on KVM virtualization and OpenVZ containers. Further information can be found at:

http://www.proxmox.org/

Dependencies

- IPy >= 0.81
- requests >= 2.2.1

Please note: This module allows you to create both OpenVZ and KVM but installing Salt on it will only be done when the VM is an OpenVZ container rather than a KVM virtual machine.

- Set up the cloud configuration at `/etc/salt/cloud.providers` or `/etc/salt/cloud.providers.d/proxmox.conf`:

```
my-proxmox-config:
  # Set up the location of the salt master
  #
  minion:
    master: saltmaster.example.com

  # Set the PROXMOX access credentials (see below)
  #
  user: myuser@pve
  password: badpass

  # Set the access URL for your PROXMOX provider
  #
  url: your.proxmox.host
  provider: proxmox
```

Access Credentials

The `user`, `password`, and `url` will be provided to you by your cloud provider. These are all required in order for the PROXMOX driver to work.

Cloud Profiles

Set up an initial profile at `/etc/salt/cloud.profiles` or `/etc/salt/cloud.profiles.d/proxmox.conf`:

- Configure a profile to be used:

```
proxmox-ubuntu:
    provider: proxmox
    image: local:vztmpl/ubuntu-12.04-standard_12.04-1_amd64.tar.gz
    technology: openvz
    host: myvmhost
```

```
    ip_address: 192.168.100.155
    password: topsecret
```

The profile can be realized now with a salt command:

```
# salt-cloud -p proxmox-ubuntu myubuntu
```

This will create an instance named `myubuntu` on the cloud provider. The minion that is installed on this instance will have a `hostname` of `myubuntu`. If the command was executed on the salt-master, its Salt key will automatically be signed on the master.

Once the instance has been created with salt-minion installed, connectivity to it can be verified with Salt:

```
# salt myubuntu test.ping
```

Required Settings

The following settings are always required for PROXMOX:

- Using the new cloud configuration format:

```
my-proxmox-config:
  provider: proxmox
  user: saltcloud@pve
  password: xyzzy
  url: your.proxmox.host
```

Optional Settings

Unlike other cloud providers in Salt Cloud, Proxmox does not utilize a `size` setting. This is because Proxmox allows the end-user to specify a more detailed configuration for their instances, than is allowed by many other cloud providers. The following options are available to be used in a profile, with their default settings listed.

```
# Description of the instance.
desc: <instance_name>

# How many CPU cores, and how fast they are (in MHz)
cpus: 1
cpuunits: 1000

# How many megabytes of RAM
memory: 256

# How much swap space in MB
swap: 256

# Whether to auto boot the vm after the host reboots
onboot: 1

# Size of the instance disk (in GiB)
disk: 10

# Host to create this vm on
host: myvmhost

# Nameservers. Defaults to host
```

```
nameserver: 8.8.8.8 8.8.4.4

# Username and password
ssh_username: root
password: <value from PROXMOX.password>

# The name of the image, from ``salt-cloud --list-images proxmox``
image: local:vztmpl/ubuntu-12.04-standard_12.04-1_amd64.tar.gz
```

15.5.14 Getting Started With Rackspace

Rackspace is a major public cloud platform which may be configured using either the *rackspace* or the *openstack* driver, depending on your needs.

Please note that the *rackspace* driver is only intended for 1st gen instances, aka, ``the old cloud'' at Rackspace. It is required for 1st gen instances, but will *not* work with OpenStack-based instances. Unless you explicitly have a reason to use it, it is highly recommended that you use the *openstack* driver instead.

Dependencies

- Libcloud >= 0.13.2

Configuration

To use the *openstack* driver (recommended), set up the cloud configuration at `/etc/salt/cloud.providers` or `/etc/salt/cloud.providers.d/rackspace.conf`:

```
my-rackspace-config:
  # Set the location of the salt-master
  #
  minion:
    master: saltmaster.example.com

  # Configure Rackspace using the OpenStack plugin
  #
  identity_url: 'https://identity.api.rackspacecloud.com/v2.0/tokens'
  compute_name: cloudServersOpenStack
  protocol: ipv4

  # Set the compute region:
  #
  compute_region: DFW

  # Configure Rackspace authentication credentials
  #
  user: myname
  tenant: 123456
  apikey: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

  provider: openstack
```

To use the *rackspace* driver, set up the cloud configuration at `/etc/salt/cloud.providers` or `/etc/salt/cloud.providers.d/rackspace.conf`:

```
my-rackspace-config:
  provider: rackspace
  # The Rackspace login user
  user: fred
  # The Rackspace user's apikey
  apikey: 901d3f579h23c8v73q9
```

The settings that follow are for using Rackspace with the *openstack* driver, and will not work with the *rackspace* driver.

Compute Region

Rackspace currently has six compute regions which may be used:

```
DFW -> Dallas/Forth Worth
ORD -> Chicago
SYD -> Sydney
LON -> London
IAD -> Northern Virginia
HKG -> Hong Kong
```

Note: Currently the LON region is only available with a UK account, and UK accounts cannot access other regions

Authentication

The `user` is the same user as is used to log into the Rackspace Control Panel. The `tenant` and `apikey` can be found in the API Keys area of the Control Panel. The `apikey` will be labeled as API Key (and may need to be generated), and `tenant` will be labeled as Cloud Account Number.

An initial profile can be configured in `/etc/salt/cloud.profiles` or `/etc/salt/cloud.profiles.d/rackspace.conf`:

```
openstack_512:
    provider: my-rackspace-config
    size: 512 MB Standard
    image: Ubuntu 12.04 LTS (Precise Pangolin)
```

To instantiate a machine based on this profile:

```
# salt-cloud -p openstack_512 myinstance
```

This will create a virtual machine at Rackspace with the name `myinstance`. This operation may take several minutes to complete, depending on the current load at the Rackspace data center.

Once the instance has been created with salt-minion installed, connectivity to it can be verified with Salt:

```
# salt myinstance test.ping
```

RackConnect Environments

Rackspace offers a hybrid hosting configuration option called RackConnect that allows you to use a physical firewall appliance with your cloud servers. When this service is in use the public_ip assigned by nova will be replaced by a NAT ip on the firewall. For salt-cloud to work properly it must use the newly assigned ``access ip'' instead of the Nova assigned public ip. You can enable that capability by adding this to your profiles:

```
openstack_512:
    provider: my-openstack-config
    size: 512 MB Standard
    image: Ubuntu 12.04 LTS (Precise Pangolin)
    rackconnect: True
```

Managed Cloud Environments

Rackspace offers a managed service level of hosting. As part of the managed service level you have the ability to choose from base of lamp installations on cloud server images. The post build process for both the base and the lamp installations used Chef to install things such as the cloud monitoring agent and the cloud backup agent. It also takes care of installing the lamp stack if selected. In order to prevent the post installation process from stomping over the bootstrapping you can add the below to your profiles.

```
openstack_512:
    provider: my-rackspace-config
    size: 512 MB Standard
    image: Ubuntu 12.04 LTS (Precise Pangolin)
    managedcloud: True
```

First and Next Generation Images

Rackspace provides two sets of virtual machine images, *first*, and *next* generation. As of `0.8.9` salt-cloud will default to using the *next* generation images. To force the use of first generation images, on the profile configuration please add:

```
FreeBSD-9.0-512:
  provider: my-rackspace-config
  size: 512 MB Standard
  image: FreeBSD 9.0
  force_first_gen: True
```

Private Subnets

By default salt-cloud will not add Rackspace private networks to new servers. To enable a private network to a server instantiated by salt cloud, add the following section to the provider file (typically `/etc/salt/cloud.providers.d/rackspace.conf`)

```
networks:
  - fixed:
    # This is the private network
    - private-network-id
    # This is Rackspace's "PublicNet"
    - 00000000-0000-0000-0000-000000000000
    # This is Rackspace's "ServiceNet"
    - 11111111-1111-1111-1111-111111111111
```

To get the Rackspace private network ID, go to Networking, Networks and hover over the private network name.

The order of the networks in the above code block does not map to the order of the ethernet devices on newly created servers. Public IP will always be first (eth0) followed by servicenet (eth1) and then private networks.

Enabling the private network per above gives the option of using the private subnet for all master-minion communication, including the bootstrap install of salt-minion. To enable the minion to use the private subnet, update the

master: line in the minion: section of the providers file. To configure the master to only listen on the private subnet IP, update the interface: line in the /etc/salt/master file to be the private subnet IP of the salt master.

15.5.15 Getting Started With SoftLayer

SoftLayer is a public cloud provider, and baremetal hardware hosting provider.

Dependencies

The SoftLayer driver for Salt Cloud requires the softlayer package, which is available at PyPI:

https://pypi.python.org/pypi/SoftLayer

This package can be installed using *pip* or *easy_install*:

```
# pip install softlayer
# easy_install softlayer
```

Configuration

Set up the cloud config at `/etc/salt/cloud.providers`:

```
# Note: These examples are for /etc/salt/cloud.providers

  my-softlayer:
    # Set up the location of the salt master
    minion:
      master: saltmaster.example.com

    # Set the SoftLayer access credentials (see below)
    user: MYUSER1138
    apikey: 'e3b68aa711e6deadc62d5b76355674beef7cc3116062ddbacafe5f7e465bfdc9'

    provider: softlayer


  my-softlayer-hw:
    # Set up the location of the salt master
    minion:
      master: saltmaster.example.com

    # Set the SoftLayer access credentials (see below)
    user: MYUSER1138
    apikey: 'e3b68aa711e6deadc62d5b76355674beef7cc3116062ddbacafe5f7e465bfdc9'

    provider: softlayer_hw
```

Access Credentials

The *user* setting is the same user as is used to log into the SoftLayer Administration area. The *apikey* setting is found inside the Admin area after logging in:

- Hover over the *Administrative* menu item.
- Click the *API Access* link.

- The *apikey* is located next to the *user* setting.

Profiles

Cloud Profiles

Set up an initial profile at `/etc/salt/cloud.profiles`:

```
base_softlayer_ubuntu:
  provider: my-softlayer
  image: UBUNTU_LATEST
  cpu_number: 1
  ram: 1024
  disk_size: 100
  local_disk: True
  hourly_billing: True
  domain: example.com
  location: sjc01
  # Optional
  max_net_speed: 1000
  private_vlan: 396
  private_network: True
  private_ssh: True
  # May be used _instead_of_ image
  global_identifier: 320d8be5-46c0-dead-cafe-13e3c51
```

Most of the above items are required; optional items are specified below.

image Images to build an instance can be found using the *--list-images* option:

```
# salt-cloud --list-images my-softlayer
```

The setting used will be labeled as *template*.

cpu_number This is the number of CPU cores that will be used for this instance. This number may be dependent upon the image that is used. For instance:

```
Red Hat Enterprise Linux 6 - Minimal Install (64 bit) (1 - 4 Core):
    ----------
    name:
        Red Hat Enterprise Linux 6 - Minimal Install (64 bit) (1 - 4 Core)
    template:
        REDHAT_6_64
Red Hat Enterprise Linux 6 - Minimal Install (64 bit) (5 - 100 Core):
    ----------
    name:
        Red Hat Enterprise Linux 6 - Minimal Install (64 bit) (5 - 100 Core)
    template:
        REDHAT_6_64
```

Note that the template (meaning, the *image* option) for both of these is the same, but the names suggests how many CPU cores are supported.

ram This is the amount of memory, in megabytes, that will be allocated to this instance.

disk_size The amount of disk space that will be allocated to this image, in megabytes.

local_disk When true the disks for the computing instance will be provisioned on the host which it runs, otherwise SAN disks will be provisioned.

hourly_billing When true the computing instance will be billed on hourly usage, otherwise it will be billed on a monthly basis.

domain The domain name that will be used in the FQDN (Fully Qualified Domain Name) for this instance. The *domain* setting will be used in conjunction with the instance name to form the FQDN.

location Images to build an instance can be found using the *--list-locations* option:

```
# salt-cloud --list-location my-softlayer
```

max_net_speed Specifies the connection speed for the instance's network components. This setting is optional. By default, this is set to 10.

public_vlan If it is necessary for an instance to be created within a specific frontend VLAN, the ID for that VLAN can be specified in either the provider or profile configuration.

This ID can be queried using the *list_vlans* function, as described below. This setting is optional.

private_vlan If it is necessary for an instance to be created within a specific backend VLAN, the ID for that VLAN can be specified in either the provider or profile configuration.

This ID can be queried using the *list_vlans* function, as described below. This setting is optional.

private_network If a server is to only be used internally, meaning it does not have a public VLAN associated with it, this value would be set to True. This setting is optional. The default is False.

private_ssh Whether to run the deploy script on the server using the public IP address or the private IP address. If set to True, Salt Cloud will attempt to SSH into the new server using the private IP address. The default is False. This settiong is optional.

global_identifier When creating an instance using a custom template, this option is set to the corresponding value obtained using the *list_custom_images* function. This option will not be used if an *image* is set, and if an *image* is not set, it is required.

The profile can be realized now with a salt command:

```
# salt-cloud -p base_softlayer_ubuntu myserver
```

Using the above configuration, this will create *myserver.example.com*.

Once the instance has been created with salt-minion installed, connectivity to it can be verified with Salt:

```
# salt 'myserver.example.com' test.ping
```

Cloud Profiles

Set up an initial profile at `/etc/salt/cloud.profiles`:

```
base_softlayer_hw_centos:
  provider: my-softlayer-hw
  # CentOS 6.0 - Minimal Install (64 bit)
  image: 13963
  # 2 x 2.0 GHz Core Bare Metal Instance - 2 GB Ram
  size: 1921
  # 250GB SATA II
  hdd: 19
  # San Jose 01
  location: 168642
  domain: example.com
  # Optional
  vlan: 396
  port_speed: 273
  banwidth: 248
```

Most of the above items are required; optional items are specified below.

image Images to build an instance can be found using the *--list-images* option:

```
# salt-cloud --list-images my-softlayer-hw
```

A list of *id`s and names will be provided. The `name* will describe the operating system and architecture. The *id* will be the setting to be used in the profile.

size Sizes to build an instance can be found using the *--list-sizes* option:

```
# salt-cloud --list-sizes my-softlayer-hw
```

A list of *id`s and names will be provided. The `name* will describe the speed and quantity of CPU cores, and the amount of memory that the hardware will contain. The *id* will be the setting to be used in the profile.

hdd There are currently two sizes of hard disk drive (HDD) that are available for hardware instances on SoftLayer:

```
19: 250GB SATA II
1267: 500GB SATA II
```

The *hdd* setting in the profile will be either 19 or 1267. Other sizes may be added in the future.

location Locations to build an instance can be found using the *--list-images* option:

```
# salt-cloud --list-locations my-softlayer-hw
```

A list of IDs and names will be provided. The *location* will describe the location in human terms. The *id* will be the setting to be used in the profile.

domain The domain name that will be used in the FQDN (Fully Qualified Domain Name) for this instance. The *domain* setting will be used in conjunction with the instance name to form the FQDN.

vlan If it is necessary for an instance to be created within a specific VLAN, the ID for that VLAN can be specified in either the provider or profile configuration.

This ID can be queried using the *list_vlans* function, as described below.

port_speed Specifies the speed for the instance's network port. This setting refers to an ID within the SoftLayer API, which sets the port speed. This setting is optional. The default is 273, or, 100 Mbps Public & Private Networks. The following settings are available:

- 273: 100 Mbps Public & Private Networks
- 274: 1 Gbps Public & Private Networks
- 21509: 10 Mbps Dual Public & Private Networks (up to 20 Mbps)
- 21513: 100 Mbps Dual Public & Private Networks (up to 200 Mbps)
- 2314: 1 Gbps Dual Public & Private Networks (up to 2 Gbps)
- 272: 10 Mbps Public & Private Networks

bandwidth Specifies the network bandwidth available for the instance. This setting refers to an ID within the SoftLayer API, which sets the bandwidth. This setting is optional. The default is 248, or, 5000 GB Bandwidth. The following settings are available:

- 248: 5000 GB Bandwidth
- 129: 6000 GB Bandwidth
- 130: 8000 GB Bandwidth
- 131: 10000 GB Bandwidth
- 36: Unlimited Bandwidth (10 Mbps Uplink)
- 125: Unlimited Bandwidth (100 Mbps Uplink)

Actions

The following actions are currently supported by the SoftLayer Salt Cloud driver.

show_instance

This action is a thin wrapper around *--full-query*, which displays details on a single instance only. In an environment with several machines, this will save a user from having to sort through all instance data, just to examine a single instance.

```
$ salt-cloud -a show_instance myinstance
```

Functions

The following functions are currently supported by the SoftLayer Salt Cloud driver.

list_vlans

This function lists all VLANs associated with the account, and all known data from the SoftLayer API concerning those VLANs.

```
$ salt-cloud -f list_vlans my-softlayer
$ salt-cloud -f list_vlans my-softlayer-hw
```

The *id* returned in this list is necessary for the *vlan* option when creating an instance.

list_custom_images

This function lists any custom templates associated with the account, that can be used to create a new instance.

```
$ salt-cloud -f list_custom_images my-softlayer
```

The *globalIdentifier* returned in this list is necessary for the *global_identifier* option when creating an image using a custom template.

Optional Products for SoftLayer HW

The softlayer_hw provider supports the ability to add optional products, which are supported by SoftLayer's API. These products each have an ID associated with them, that can be passed into Salt Cloud with the *optional_products* option:

```
softlayer_hw_test:
  provider: my-softlayer-hw
  # CentOS 6.0 - Minimal Install (64 bit)
  image: 13963
  # 2 x 2.0 GHz Core Bare Metal Instance - 2 GB Ram
  size: 1921
  # 250GB SATA II
  hdd: 19
  # San Jose 01
  location: 168642
  domain: example.com
  optional_products:
    # MySQL for Linux
    - id: 28
    # Business Continuance Insurance
    - id: 104
```

These values can be manually obtained by looking at the source of an order page on the SoftLayer web interface. For convenience, many of these values are listed here:

Public Secondary IP Addresses

- 22: 4 Public IP Addresses
- 23: 8 Public IP Addresses

Primary IPv6 Addresses

- 17129: 1 IPv6 Address

Public Static IPv6 Addresses

- 1481: /64 Block Static Public IPv6 Addresses

OS-Specific Addon

- 17139: XenServer Advanced for XenServer 6.x
- 17141: XenServer Enterprise for XenServer 6.x
- 2334: XenServer Advanced for XenServer 5.6
- 2335: XenServer Enterprise for XenServer 5.6
- 13915: Microsoft WebMatrix
- 21276: VMware vCenter 5.1 Standard

Control Panel Software

- 121: cPanel/WHM with Fantastico and RVskin
- 20778: Parallels Plesk Panel 11 (Linux) 100 Domain w/ Power Pack
- 20786: Parallels Plesk Panel 11 (Windows) 100 Domain w/ Power Pack
- 20787: Parallels Plesk Panel 11 (Linux) Unlimited Domain w/ Power Pack
- 20792: Parallels Plesk Panel 11 (Windows) Unlimited Domain w/ Power Pack
- 2340: Parallels Plesk Panel 10 (Linux) 100 Domain w/ Power Pack
- 2339: Parallels Plesk Panel 10 (Linux) Unlimited Domain w/ Power Pack
- 13704: Parallels Plesk Panel 10 (Windows) Unlimited Domain w/ Power Pack

Database Software

- 29: MySQL 5.0 for Windows
- 28: MySQL for Linux
- 21501: Riak 1.x
- 20893: MongoDB
- 30: Microsoft SQL Server 2005 Express
- 92: Microsoft SQL Server 2005 Workgroup
- 90: Microsoft SQL Server 2005 Standard
- 94: Microsoft SQL Server 2005 Enterprise
- 1330: Microsoft SQL Server 2008 Express
- 1340: Microsoft SQL Server 2008 Web
- 1337: Microsoft SQL Server 2008 Workgroup
- 1334: Microsoft SQL Server 2008 Standard
- 1331: Microsoft SQL Server 2008 Enterprise

- 2179: Microsoft SQL Server 2008 Express R2
- 2173: Microsoft SQL Server 2008 Web R2
- 2183: Microsoft SQL Server 2008 Workgroup R2
- 2180: Microsoft SQL Server 2008 Standard R2
- 2176: Microsoft SQL Server 2008 Enterprise R2

Anti-Virus & Spyware Protection

- 594: McAfee VirusScan Anti-Virus - Windows
- 414: McAfee Total Protection - Windows

Insurance

- 104: Business Continuance Insurance

Monitoring

- 55: Host Ping
- 56: Host Ping and TCP Service Monitoring

Notification

- 57: Email and Ticket

Advanced Monitoring

- 2302: Monitoring Package - Basic
- 2303: Monitoring Package - Advanced
- 2304: Monitoring Package - Premium Application

Response

- 58: Automated Notification
- 59: Automated Reboot from Monitoring
- 60: 24x7x365 NOC Monitoring, Notification, and Response

Intrusion Detection & Protection

- 413: McAfee Host Intrusion Protection w/Reporting

Hardware & Software Firewalls

- 411: APF Software Firewall for Linux
- 894: Microsoft Windows Firewall
- 410: 10Mbps Hardware Firewall
- 409: 100Mbps Hardware Firewall
- 408: 1000Mbps Hardware Firewall

15.5.16 Getting Started with VEXXHOST

VEXXHOST is an cloud computing provider which provides Canadian cloud computing services which are based in Monteral and uses the libcloud OpenStack driver. VEXXHOST currently runs the Havana release of OpenStack. When provisioning new instances, they automatically get a public IP and private IP address. Therefore, you do not need to assign a floating IP to access your instance once it's booted.

Cloud Provider Configuration

To use the *openstack* driver for the VEXXHOST public cloud, you will need to set up the cloud provider configuration file as in the example below:

`/etc/salt/cloud.providers.d/vexxhost.conf`: In order to use the VEXXHOST public cloud, you will need to setup a cloud provider configuration file as in the example below which uses the OpenStack driver.

```
vexxhost:
  # Set the location of the salt-master
  #
  minion:
    master: saltmaster.example.com

  # Configure VEXXHOST using the OpenStack plugin
  #
  identity_url: http://auth.api.thenebulacloud.com:5000/v2.0/tokens
  compute_name: nova

  # Set the compute region:
  #
  compute_region: na-yul-nhs1

  # Configure VEXXHOST authentication credentials
  #
  user: your-tenant-id
  password: your-api-key
  tenant: your-tenant-name

  # keys to allow connection to the instance launched
  #
  ssh_key_name: yourkey
  ssh_key_file: /path/to/key/yourkey.priv

  provider: openstack
```

Authentication

All of the authentication fields that you need can be found by logging into your VEXXHOST customer center. Once you've logged in, you will need to click on ``CloudConsole'' and then click on ``API Credentials''.

Cloud Profile Configuration

In order to get the correct image UUID and the instance type to use in the cloud profile, you can run the following command respectively:

```
# salt-cloud --list-images=vexxhost-config
# salt-cloud --list-sizes=vexxhost-config
```

Once you have that, you can go ahead and create a new cloud profile. This profile will build an Ubuntu 12.04 LTS *nb.2G* instance.

`/etc/salt/cloud.profiles.d/vh_ubuntu1204_2G.conf`:

```
vh_ubuntu1204_2G:
    provider: vexxhost
    image: 4051139f-750d-4d72-8ef0-074f2ccc7e5a
    size: nb.2G
```

Provision an instance

To create an instance based on the sample profile that we created above, you can run the following *salt-cloud* command.

```
# salt-cloud -p vh_ubuntu1204_2G vh_instance1
```

Typically, instances are provisioned in under 30 seconds on the VEXXHOST public cloud. After the instance provisions, it will be set up a minion and then return all the instance information once it's complete.

Once the instance has been setup, you can test connectivity to it by running the following command:

```
# salt vh_instance1 test.ping
```

You can now continue to provision new instances and they will all automatically be set up as minions of the master you've defined in the configuration file.

15.6 Miscellaneous Options

15.6.1 Miscellaneous Salt Cloud Options

This page describes various miscellaneous options available in Salt Cloud

Deploy Script Arguments

Custom deploy scripts are unlikely to need custom arguments to be passed to them, but salt-bootstrap has been extended quite a bit, and this may be necessary. script_args can be specified in either the profile or the map file, to pass arguments to the deploy script:

```
ec2-amazon:
    provider: ec2
    image: ami-1624987f
    size: t1.micro
    ssh_username: ec2-user
    script: bootstrap-salt
    script_args: -c /tmp/
```

This has also been tested to work with pipes, if needed:

```
script_args: | head
```

Use SFTP to transfer files

Some distributions do not have scp distributed with the ssh package. The solution is to use sftp with the *use_sftp* flag

```
use_sftp: True
```

Sync After Install

Salt allows users to create custom modules, grains, and states which can be synchronised to minions to extend Salt with further functionality.

This option will inform Salt Cloud to synchronise your custom modules, grains, states or all these to the minion just after it has been created. For this to happen, the following line needs to be added to the main cloud configuration file:

```
sync_after_install: all
```

The available options for this setting are:

```
modules
grains
states
all
```

Setting up New Salt Masters

It has become increasingly common for users to set up multi-hierarchal infrastructures using Salt Cloud. This sometimes involves setting up an instance to be a master in addition to a minion. With that in mind, you can now lay down master configuration on a machine by specifying master options in the profile or map file.

```
make_master: True
```

This will cause Salt Cloud to generate master keys for the instance, and tell salt-bootstrap to install the salt-master package, in addition to the salt-minion package.

The default master configuration is usually appropriate for most users, and will not be changed unless specific master configuration has been added to the profile or map:

```
master:
    user: root
    interface: 0.0.0.0
```

Delete SSH Keys

When Salt Cloud deploys an instance, the SSH pub key for the instance is added to the known_hosts file for the user that ran the salt-cloud command. When an instance is deployed, a cloud provider generally recycles the IP address for the instance. When Salt Cloud attempts to deploy an instance using a recycled IP address that has previously been accessed from the same machine, the old key in the known_hosts file will cause a conflict.

In order to mitigate this issue, Salt Cloud can be configured to remove old keys from the known_hosts file when destroying the node. In order to do this, the following line needs to be added to the main cloud configuration file:

```
delete_sshkeys: True
```

Keeping /tmp/ Files

When Salt Cloud deploys an instance, it uploads temporary files to /tmp/ for salt-bootstrap to put in place. After the script has run, they are deleted. To keep these files around (mostly for debugging purposes), the --keep-tmp option can be added:

```
salt-cloud -p myprofile mymachine --keep-tmp
```

For those wondering why /tmp/ was used instead of /root/, this had to be done for images which require the use of sudo, and therefore do not allow remote root logins, even for file transfers (which makes /root/ unavailable).

Hide Output From Minion Install

By default Salt Cloud will stream the output from the minion deploy script directly to STDOUT. Although this can been very useful, in certain cases you may wish to switch this off. The following config option is there to enable or disable this output:

```
display_ssh_output: False
```

Connection Timeout

There are several stages when deploying Salt where Salt Cloud needs to wait for something to happen. The VM getting it's IP address, the VM's SSH port is available, etc.

If you find that the Salt Cloud defaults are not enough and your deployment fails because Salt Cloud did not wait log enough, there are some settings you can tweak.

Note

All values should be provided in seconds

You can tweak these settings globally, per cloud provider, or event per profile definition.

wait_for_ip_timeout

The amount of time Salt Cloud should wait for a VM to start and get an IP back from the cloud provider. Default: 5 minutes.

wait_for_ip_interval

The amount of time Salt Cloud should sleep while querying for the VM's IP. Default: 5 seconds.

ssh_connect_timeout

The amount of time Salt Cloud should wait for a successful SSH connection to the VM. Default: 5 minutes.

wait_for_passwd_timeout

The amount of time until an ssh connection can be established via password or ssh key. Default 15 seconds.

wait_for_passwd_maxtries

The number of attempts to connect to the VM until we abandon. Default 15 attempts

wait_for_fun_timeout

Some cloud drivers check for an available IP or a successful SSH connection using a function, namely, SoftLayer, and SoftLayer-HW. So, the amount of time Salt Cloud should retry such functions before failing. Default: 5 minutes.

wait_for_spot_timeout

The amount of time Salt Cloud should wait before an EC2 Spot instance is available. This setting is only available for the EC2 cloud driver.

Salt Cloud Cache

Salt Cloud can maintain a cache of node data, for supported providers. The following options manage this functionality.

update_cachedir

On supported cloud providers, whether or not to maintain a cache of nodes returned from a --full-query. The data will be stored in `msgpack` format under `<SALT_CACHEDIR>/cloud/active/<DRIVER>/<PROVIDER>/<NODE_NAME>.p`. This setting can be True or False.

diff_cache_events

When the cloud cachedir is being managed, if differences are encountered between the data that is returned live from the cloud provider and the data in the cache, fire events which describe the changes. This setting can be True or False.

Some of these events will contain data which describe a node. Because some of the fields returned may contain sensitive data, the `cache_event_strip_fields` configuration option exists to strip those fields from the event return.

```
cache_event_strip_fields:
  - password
  - priv_key
```

The following are events that can be fired based on this data.

salt/cloud/minionid/cache_node_new A new node was found on the cloud provider which was not listed in the cloud cachedir. A dict describing the new node will be contained in the event.

salt/cloud/minionid/cache_node_missing A node that was previously listed in the cloud cachedir is no longer available on the cloud provider.

salt/cloud/minionid/cache_node_diff One or more pieces of data in the cloud cachedir has changed on the cloud provider. A dict containing both the old and the new data will be contained in the event.

SSH Known Hosts

Normally when bootstrapping a VM, salt-cloud will ignore the SSH host key. This is because it does not know what the host key is before starting (because it doesn't exist yet). If strict host key checking is turned on without the key in the `known_hosts` file, then the host will never be available, and cannot be bootstrapped.

If a provider is able to determine the host key before trying to bootstrap it, that provider's driver can add it to the `known_hosts` file, and then turn on strict host key checking. This can be set up in the main cloud configuration file (normally `/etc/salt/cloud`) or in the provider-specific configuration file:

```
known_hosts_file: /path/to/.ssh/known_hosts
```

If this is not set, it will default to `/dev/null`, and strict host key checking will be turned off.

It is highly recommended that this option is *not* set, unless the user has verified that the provider supports this functionality, and that the image being used is capable of providing the necessary information. At this time, only the EC2 driver supports this functionality.

SSH Agent

New in version 2015.5.0.

If the ssh key is not stored on the server salt-cloud is being run on, set ssh_agent, and salt-cloud will use the forwarded ssh-agent to authenticate.

```
ssh_agent: True
```

File Map Upload

New in version 2014.7.0.

The `file_map` option allows an arbitrary group of files to be uploaded to the target system before running the deploy script. This functionality requires a provider uses salt.utils.cloud.bootstrap(), which is currently limited to the ec2, gce, openstack and nova drivers.

The `file_map` can be configured globally in `/etc/salt/cloud`, or in any cloud provider or profile file. For example, to upload an extra package or a custom deploy script, a cloud profile using `file_map` might look like:

```
ubuntu14:
  provider: ec2-config
  image: ami-98aa1cf0
  size: t1.micro
  ssh_username: root
  securitygroup: default
  file_map:
    /local/path/to/custom/script: /remote/path/to/use/custom/script
    /local/path/to/package: /remote/path/to/store/package
```

15.7 Troubleshooting Steps

15.7.1 Troubleshooting Salt Cloud

This page describes various steps for troubleshooting problems that may arise while using Salt Cloud.

Virtual Machines Are Created, But Do Not Respond

Are TCP ports 4505 and 4506 open on the master? This is easy to overlook on new masters. Information on how to open firewall ports on various platforms can be found *here*.

Generic Troubleshooting Steps

This section describes a set of instructions that are useful to a large number of situations, and are likely to solve most issues that arise.

Version Compatibility

One of the most common issues that Salt Cloud users run into is import errors. These are often caused by version compatibility issues with Salt.

Salt 0.16.x works with Salt Cloud 0.8.9 or greater.

Salt 0.17.x requires Salt Cloud 0.8.11.

Releases after 0.17.x (0.18 or greater) should not encounter issues as Salt Cloud has been merged into Salt itself.

Debug Mode

Frequently, running Salt Cloud in debug mode will reveal information about a deployment which would otherwise not be obvious:

```
salt-cloud -p myprofile myinstance -l debug
```

Keep in mind that a number of messages will appear that look at first like errors, but are in fact intended to give developers factual information to assist in debugging. A number of messages that appear will be for cloud providers that you do not have configured; in these cases, the message usually is intended to confirm that they are not configured.

Salt Bootstrap

By default, Salt Cloud uses the Salt Bootstrap script to provision instances:

This script is packaged with Salt Cloud, but may be updated without updating the Salt package:

```
salt-cloud -u
```

The Bootstrap Log

If the default deploy script was used, there should be a file in the `/tmp/` directory called `bootstrap-salt.log`. This file contains the full output from the deployment, including any errors that may have occurred.

Keeping Temp Files

Salt Cloud uploads minion-specific files to instances once they are available via SSH, and then executes a deploy script to put them into the correct place and install Salt. The `--keep-tmp` option will instruct Salt Cloud not to remove those files when finished with them, so that the user may inspect them for problems:

```
salt-cloud -p myprofile myinstance --keep-tmp
```

By default, Salt Cloud will create a directory on the target instance called `/tmp/.saltcloud/`. This directory should be owned by the user that is to execute the deploy script, and should have permissions of `0700`.

Most cloud providers are configured to use `root` as the default initial user for deployment, and as such, this directory and all files in it should be owned by the `root` user.

The `/tmp/.saltcloud/` directory should the following files:

- A `deploy.sh` script. This script should have permissions of `0755`.
- A `.pem` and `.pub` key named after the minion. The `.pem` file should have permissions of `0600`. Ensure that the `.pem` and `.pub` files have been properly copied to the `/etc/salt/pki/minion/` directory.
- A file called `minion`. This file should have been copied to the `/etc/salt/` directory.
- Optionally, a file called `grains`. This file, if present, should have been copied to the `/etc/salt/` directory.

Unprivileged Primary Users

Some providers, most notably EC2, are configured with a different primary user. Some common examples are `ec2-user`, `ubuntu`, `fedora`, and `bitnami`. In these cases, the `/tmp/.saltcloud/` directory and all files in it should be owned by this user.

Some providers, such as EC2, are configured to not require these users to provide a password when using the `sudo` command. Because it is more secure to require `sudo` users to provide a password, other providers are configured that way.

If this instance is required to provide a password, it needs to be configured in Salt Cloud. A password for sudo to use may be added to either the provider configuration or the profile configuration:

```
sudo_password: mypassword
```

/tmp/ is Mounted as noexec

It is more secure to mount the /tmp/ directory with a noexec option. This is uncommon on most cloud providers, but very common in private environments. To see if the /tmp/ directory is mounted this way, run the following command:

```
mount | grep tmp
```

The if the output of this command includes a line that looks like this, then the /tmp/ directory is mounted as noexec:

```
tmpfs on /tmp type tmpfs (rw,noexec)
```

If this is the case, then the deploy_command will need to be changed in order to run the deploy script through the sh command, rather than trying to execute it directly. This may be specified in either the provider or the profile config:

```
deploy_command: sh /tmp/.saltcloud/deploy.sh
```

Please note that by default, Salt Cloud will place its files in a directory called /tmp/.saltcloud/. This may be also be changed in the provider or profile configuration:

```
tmp_dir: /tmp/.saltcloud/
```

If this directory is changed, then the deploy_command need to be changed in order to reflect the tmp_dir configuration.

Executing the Deploy Script Manually

If all of the files needed for deployment were successfully uploaded to the correct locations, and contain the correct permissions and ownerships, the deploy script may be executed manually in order to check for other issues:

```
cd /tmp/.saltcloud/
./deploy.sh
```

15.8 Extending Salt Cloud

15.8.1 Writing Cloud Provider Modules

Salt Cloud runs on a module system similar to the main Salt project. The modules inside saltcloud exist in the salt/cloud/clouds directory of the salt source.

There are two basic types of cloud modules. If a cloud provider is supported by libcloud, then using it is the fastest route to getting a module written. The Apache Libcloud project is located at:

http://libcloud.apache.org/

Not every cloud provider is supported by libcloud. Additionally, not every feature in a supported cloud provider is necessary supported by libcloud. In either of these cases, a module can be created which does not rely on libcloud.

All Modules

The following functions are required by all modules, whether or not they are based on libcloud.

The __virtual__() Function

This function determines whether or not to make this cloud module available upon execution. Most often, it uses `get_configured_provider()` to determine if the necessary configuration has been set up. It may also check for necessary imports, to decide whether to load the module. In most cases, it will return a `True` or `False` value. If the name of the driver used does not match the filename, then that name should be returned instead of `True`. An example of this may be seen in the Azure module:

https://github.com/saltstack/salt/tree/develop/salt/cloud/clouds/msazure.py

The get_configured_provider() Function

This function uses `config.is_provider_configured()` to determine wither all required information for this driver has been configured. The last value in the list of required settings should be followed by a comma.

Libcloud Based Modules

Writing a cloud module based on libcloud has two major advantages. First of all, much of the work has already been done by the libcloud project. Second, most of the functions necessary to Salt have already been added to the Salt Cloud project.

The create() Function

The most important function that does need to be manually written is the `create()` function. This is what is used to request a virtual machine to be created by the cloud provider, wait for it to become available, and then (optionally) log in and install Salt on it.

A good example to follow for writing a cloud provider module based on libcloud is the module provided for Linode:

https://github.com/saltstack/salt/tree/develop/salt/cloud/clouds/linode.py

The basic flow of a `create()` function is as follows:

- Send a request to the cloud provider to create a virtual machine.
- Wait for the virtual machine to become available.
- Generate kwargs to be used to deploy Salt.
- Log into the virtual machine and deploy Salt.
- Return a data structure that describes the newly-created virtual machine.

At various points throughout this function, events may be fired on the Salt event bus. Four of these events, which are described below, are required. Other events may be added by the user, where appropriate.

When the `create()` function is called, it is passed a data structure called `vm_`. This dict contains a composite of information describing the virtual machine to be created. A dict called `__opts__` is also provided by Salt, which contains the options used to run Salt Cloud, as well as a set of configuration and environment variables.

The first thing the `create()` function must do is fire an event stating that it has started the create process. This event is tagged `salt/cloud/<vm name>/creating`. The payload contains the names of the VM, profile and provider.

A set of kwargs is then usually created, to describe the parameters required by the cloud provider to request the virtual machine.

An event is then fired to state that a virtual machine is about to be requested. It is tagged as `salt/cloud/<vm name>/requesting`. The payload contains most or all of the parameters that will be sent to the cloud provider. Any private information (such as passwords) should not be sent in the event.

After a request is made, a set of deploy kwargs will be generated. These will be used to install Salt on the target machine. Windows options are supported at this point, and should be generated, even if the cloud provider does not currently support Windows. This will save time in the future if the provider does eventually decide to support Windows.

An event is then fired to state that the deploy process is about to begin. This event is tagged `salt/cloud/<vm name>/deploying`. The payload for the event will contain a set of deploy kwargs, useful for debugging purposed. Any private data, including passwords and keys (including public keys) should be stripped from the deploy kwargs before the event is fired.

If any Windows options have been passed in, the `salt.utils.cloud.deploy_windows()` function will be called. Otherwise, it will be assumed that the target is a Linux or Unix machine, and the `salt.utils.cloud.deploy_script()` will be called.

Both of these functions will wait for the target machine to become available, then the necessary port to log in, then a successful login that can be used to install Salt. Minion configuration and keys will then be uploaded to a temporary directory on the target by the appropriate function. On a Windows target, the Windows Minion Installer will be run in silent mode. On a Linux/Unix target, a deploy script (bootstrap-salt.sh, by default) will be run, which will auto-detect the operating system, and install Salt using its native package manager. These do not need to be handled by the developer in the cloud module.

The `salt.utils.cloud.validate_windows_cred()` function has been extended to take the number of retries and retry_delay parameters in case a specific cloud provider has a delay between providing the Windows credentials and the credentials being available for use. In their `create()` function, or as a a sub-function called during the creation process, developers should use the `win_deploy_auth_retries` and `win_deploy_auth_retry_delay` parameters from the provider configuration to allow the end-user the ability to customize the number of tries and delay between tries for their particular provider.

After the appropriate deploy function completes, a final event is fired which describes the virtual machine that has just been created. This event is tagged `salt/cloud/<vm name>/created`. The payload contains the names of the VM, profile, and provider.

Finally, a dict (queried from the provider) which describes the new virtual machine is returned to the user. Because this data is not fired on the event bus it can, and should, return any passwords that were returned by the cloud provider. In some cases (for example, Rackspace), this is the only time that the password can be queried by the user; post-creation queries may not contain password information (depending upon the provider).

The libcloudfuncs Functions

A number of other functions are required for all cloud providers. However, with libcloud-based modules, these are all provided for free by the libcloudfuncs library. The following two lines set up the imports:

```
from salt.cloud.libcloudfuncs import *   # pylint: disable=W0614,W0401
from salt.utils import namespaced_function
```

And then a series of declarations will make the necessary functions available within the cloud module.

```
get_size = namespaced_function(get_size, globals())
get_image = namespaced_function(get_image, globals())
avail_locations = namespaced_function(avail_locations, globals())
avail_images = namespaced_function(avail_images, globals())
avail_sizes = namespaced_function(avail_sizes, globals())
script = namespaced_function(script, globals())
destroy = namespaced_function(destroy, globals())
```

```
list_nodes = namespaced_function(list_nodes, globals())
list_nodes_full = namespaced_function(list_nodes_full, globals())
list_nodes_select = namespaced_function(list_nodes_select, globals())
show_instance = namespaced_function(show_instance, globals())
```

If necessary, these functions may be replaced by removing the appropriate declaration line, and then adding the function as normal.

These functions are required for all cloud modules, and are described in detail in the next section.

Non-Libcloud Based Modules

In some cases, using libcloud is not an option. This may be because libcloud has not yet included the necessary driver itself, or it may be that the driver that is included with libcloud does not contain all of the necessary features required by the developer. When this is the case, some or all of the functions in `libcloudfuncs` may be replaced. If they are all replaced, the libcloud imports should be absent from the Salt Cloud module.

A good example of a non-libcloud provider is the DigitalOcean module:

https://github.com/saltstack/salt/tree/develop/salt/cloud/clouds/digital_ocean.py

The `create()` Function

The `create()` function must be created as described in the libcloud-based module documentation.

The get_size() Function

This function is only necessary for libcloud-based modules, and does not need to exist otherwise.

The get_image() Function

This function is only necessary for libcloud-based modules, and does not need to exist otherwise.

The avail_locations() Function

This function returns a list of locations available, if the cloud provider uses multiple data centers. It is not necessary if the cloud provider only uses one data center. It is normally called using the `--list-locations` option.

```
salt-cloud --list-locations my-cloud-provider
```

The avail_images() Function

This function returns a list of images available for this cloud provider. There are not currently any known cloud providers that do not provide this functionality, though they may refer to images by a different name (for example, ``templates''). It is normally called using the `--list-images` option.

```
salt-cloud --list-images my-cloud-provider
```

The avail_sizes() Function

This function returns a list of sizes available for this cloud provider. Generally, this refers to a combination of RAM, CPU, and/or disk space. This functionality may not be present on some cloud providers. For example, the Parallels module breaks down RAM, CPU, and disk space into separate options, whereas in other providers, these options are baked into the image. It is normally called using the `--list-sizes` option.

```
salt-cloud --list-sizes my-cloud-provider
```

The script() Function

This function builds the deploy script to be used on the remote machine. It is likely to be moved into the `salt.utils.cloud` library in the near future, as it is very generic and can usually be copied wholesale from another module. An excellent example is in the Azure driver.

The destroy() Function

This function irreversibly destroys a virtual machine on the cloud provider. Before doing so, it should fire an event on the Salt event bus. The tag for this event is `salt/cloud/<vm name>/destroying`. Once the virtual machine has been destroyed, another event is fired. The tag for that event is `salt/cloud/<vm name>/destroyed`.

This function is normally called with the `-d` options:

```
salt-cloud -d myinstance
```

The list_nodes() Function

This function returns a list of nodes available on this cloud provider, using the following fields:

- id (str)
- image (str)
- size (str)
- state (str)
- private_ips (list)
- public_ips (list)

No other fields should be returned in this function, and all of these fields should be returned, even if empty. The private_ips and public_ips fields should always be of a list type, even if empty, and the other fields should always be of a str type. This function is normally called with the `-Q` option:

```
salt-cloud -Q
```

The list_nodes_full() Function

All information available about all nodes should be returned in this function. The fields in the list_nodes() function should also be returned, even if they would not normally be provided by the cloud provider. This is because some functions both within Salt and 3rd party will break if an expected field is not present. This function is normally called with the `-F` option:

```
salt-cloud -F
```

The list_nodes_select() Function

This function returns only the fields specified in the `query.selection` option in `/etc/salt/cloud`. Because this function is so generic, all of the heavy lifting has been moved into the `salt.utils.cloud` library.

A function to call `list_nodes_select()` still needs to be present. In general, the following code can be used as-is:

```
def list_nodes_select(call=None):
    '''
    Return a list of the VMs that are on the provider, with select fields
    '''
    return salt.utils.cloud.list_nodes_select(
        list_nodes_full('function'), __opts__['query.selection'], call,
    )
```

However, depending on the cloud provider, additional variables may be required. For instance, some modules use a `conn` object, or may need to pass other options into `list_nodes_full()`. In this case, be sure to update the function appropriately:

```
def list_nodes_select(conn=None, call=None):
    '''
    Return a list of the VMs that are on the provider, with select fields
    '''
    if not conn:
        conn = get_conn()   # pylint: disable=E0602

    return salt.utils.cloud.list_nodes_select(
        list_nodes_full(conn, 'function'),
        __opts__['query.selection'],
        call,
    )
```

This function is normally called with the `-S` option:

```
salt-cloud -S
```

The show_instance() Function

This function is used to display all of the information about a single node that is available from the cloud provider. The simplest way to provide this is usually to call `list_nodes_full()`, and return just the data for the requested node. It is normally called as an action:

```
salt-cloud -a show_instance myinstance
```

Actions and Functions

Extra functionality may be added to a cloud provider in the form of an `--action` or a `--function`. Actions are performed against a cloud instance/virtual machine, and functions are performed against a cloud provider.

Actions

Actions are calls that are performed against a specific instance or virtual machine. The `show_instance` action should be available in all cloud modules. Actions are normally called with the `-a` option:

```
salt-cloud -a show_instance myinstance
```

Actions must accept a `name` as a first argument, may optionally support any number of kwargs as appropriate, and must accept an argument of `call`, with a default of `None`.

Before performing any other work, an action should normally verify that it has been called correctly. It may then perform the desired feature, and return useful information to the user. A basic action looks like:

```
def show_instance(name, call=None):
'''
Show the details from EC2 concerning an AMI
'''
if call != 'action':
    raise SaltCloudSystemExit(
        'The show_instance action must be called with -a or --action.'
    )

return _get_node(name)
```

Please note that generic kwargs, if used, are passed through to actions as `kwargs` and not `**kwargs`. An example of this is seen in the Functions section.

Functions

Functions are called that are performed against a specific cloud provider. An optional function that is often useful is `show_image`, which describes an image in detail. Functions are normally called with the `-f` option:

```
salt-cloud -f show_image my-cloud-provider image='Ubuntu 13.10 64-bit'
```

A function may accept any number of kwargs as appropriate, and must accept an argument of `call` with a default of `None`.

Before performing any other work, a function should normally verify that it has been called correctly. It may then perform the desired feature, and return useful information to the user. A basic function looks like:

```
def show_image(kwargs, call=None):
    '''
    Show the details from EC2 concerning an AMI
    '''
    if call != 'function':
        raise SaltCloudSystemExit(
            'The show_image action must be called with -f or --function.'
        )

    params = {'ImageId.1': kwargs['image'],
              'Action': 'DescribeImages'}
    result = query(params)
    log.info(result)

    return result
```

Take note that generic kwargs are passed through to functions as `kwargs` and not `**kwargs`.

15.8.2 OS Support for Cloud VMs

Salt Cloud works primarily by executing a script on the virtual machines as soon as they become available. The script that is executed is referenced in the cloud profile as the `script`. In older versions, this was the `os` argument. This was changed in 0.8.2.

A number of legacy scripts exist in the deploy directory in the saltcloud source tree. The preferred method is currently to use the salt-bootstrap script. A stable version is included with each release tarball starting with 0.8.4. The most updated version can be found at:

https://github.com/saltstack/salt-bootstrap

If you do not specify a script argument, this script will be used at the default.

If the Salt Bootstrap script does not meet your needs, you may write your own. The script should be written in bash and is a Jinja template. Deploy scripts need to execute a number of functions to do a complete salt setup. These functions include:

1. Install the salt minion. If this can be done via system packages this method is HIGHLY preferred.
2. Add the salt minion keys before the minion is started for the first time. The minion keys are available as strings that can be copied into place in the Jinja template under the dict named ``vm''.
3. Start the salt-minion daemon and enable it at startup time.
4. Set up the minion configuration file from the ``minion'' data available in the Jinja template.

A good, well commented, example of this process is the Fedora deployment script:

https://github.com/saltstack/salt-cloud/blob/master/saltcloud/deploy/Fedora.sh

A number of legacy deploy scripts are included with the release tarball. None of them are as functional or complete as Salt Bootstrap, and are still included for academic purposes.

Other Generic Deploy Scripts

If you want to be assured of always using the latest Salt Bootstrap script, there are a few generic templates available in the deploy directory of your saltcloud source tree:

```
curl-bootstrap
curl-bootstrap-git
python-bootstrap
wget-bootstrap
wget-bootstrap-git
```

These are example scripts which were designed to be customized, adapted, and refit to meet your needs. One important use of them is to pass options to the salt-bootstrap script, such as updating to specific git tags.

Post-Deploy Commands

Once a minion has been deployed, it has the option to run a salt command. Normally, this would be the state.highstate command, which would finish provisioning the VM. Another common option is state.sls, or for just testing, test.ping. This is configured in the main cloud config file:

```
start_action: state.highstate
```

This is currently considered to be experimental functionality, and may not work well with all providers. If you experience problems with Salt Cloud hanging after Salt is deployed, consider using Startup States instead:

http://docs.saltstack.com/ref/states/startup.html

Skipping the Deploy Script

For whatever reason, you may want to skip the deploy script altogether. This results in a VM being spun up much faster, with absolutely no configuration. This can be set from the command line:

```
salt-cloud --no-deploy -p micro_aws my_instance
```

Or it can be set from the main cloud config file:

```
deploy: False
```

Or it can be set from the provider's configuration:

```
RACKSPACE.user: example_user
RACKSPACE.apikey: 123984bjjas87034
RACKSPACE.deploy: False
```

Or even on the VM's profile settings:

```
ubuntu_aws:
  provider: aws
  image: ami-7e2da54e
  size: t1.micro
  deploy: False
```

The default for deploy is True.

In the profile, you may also set the script option to `None`:

```
script: None
```

This is the slowest option, since it still uploads the None deploy script and executes it.

Updating Salt Bootstrap

Salt Bootstrap can be updated automatically with salt-cloud:

```
salt-cloud -u
salt-cloud --update-bootstrap
```

Bear in mind that this updates to the latest (unstable) version, so use with caution.

Keeping /tmp/ Files

When Salt Cloud deploys an instance, it uploads temporary files to /tmp/ for salt-bootstrap to put in place. After the script has run, they are deleted. To keep these files around (mostly for debugging purposes), the --keep-tmp option can be added:

```
salt-cloud -p myprofile mymachine --keep-tmp
```

For those wondering why /tmp/ was used instead of /root/, this had to be done for images which require the use of sudo, and therefore do not allow remote root logins, even for file transfers (which makes /root/ unavailable).

Deploy Script Arguments

Custom deploy scripts are unlikely to need custom arguments to be passed to them, but salt-bootstrap has been extended quite a bit, and this may be necessary. script_args can be specified in either the profile or the map file, to pass arguments to the deploy script:

```
aws-amazon:
    provider: aws
    image: ami-1624987f
    size: t1.micro
    ssh_username: ec2-user
    script: bootstrap-salt
    script_args: -c /tmp/
```

This has also been tested to work with pipes, if needed:

```
script_args: | head
```

15.9 Using Salt Cloud from Salt

15.9.1 Using the Salt Modules for Cloud

In addition to the `salt-cloud` command, Salt Cloud can be called from Salt, in a variety of different ways. Most users will be interested in either the execution module or the state module, but it is also possible to call Salt Cloud as a runner.

Because the actual work will be performed on a remote minion, the normal Salt Cloud configuration must exist on any target minion that needs to execute a Salt Cloud command. Because Salt Cloud now supports breaking out configuration into individual files, the configuration is easily managed using Salt's own `file.managed` state function. For example, the following directories allow this configuration to be managed easily:

```
/etc/salt/cloud.providers.d/
/etc/salt/cloud.profiles.d/
```

Minion Keys

Keep in mind that when creating minions, Salt Cloud will create public and private minion keys, upload them to the minion, and place the public key on the machine that created the minion. It will *not* attempt to place any public minion keys on the master, unless the minion which was used to create the instance is also the Salt Master. This is because granting arbitrary minions access to modify keys on the master is a serious security risk, and must be avoided.

Execution Module

The `cloud` module is available to use from the command line. At the moment, almost every standard Salt Cloud feature is available to use. The following commands are available:

list_images

This command is designed to show images that are available to be used to create an instance using Salt Cloud. In general they are used in the creation of profiles, but may also be used to create an instance directly (see below). Listing images requires a provider to be configured, and specified:

```
salt myminion cloud.list_images my-cloud-provider
```

list_sizes

This command is designed to show sizes that are available to be used to create an instance using Salt Cloud. In general they are used in the creation of profiles, but may also be used to create an instance directly (see below). This command is not available for all cloud providers; see the provider-specific documentation for details. Listing sizes requires a provider to be configured, and specified:

```
salt myminion cloud.list_sizes my-cloud-provider
```

list_locations

This command is designed to show locations that are available to be used to create an instance using Salt Cloud. In general they are used in the creation of profiles, but may also be used to create an instance directly (see below). This command is not available for all cloud providers; see the provider-specific documentation for details. Listing locations requires a provider to be configured, and specified:

```
salt myminion cloud.list_locations my-cloud-provider
```

query

This command is used to query all configured cloud providers, and display all instances associated with those accounts. By default, it will run a standard query, returning the following fields:

id The name or ID of the instance, as used by the cloud provider.

image The disk image that was used to create this instance.

private_ips Any public IP addresses currently assigned to this instance.

public_ips Any private IP addresses currently assigned to this instance.

size The size of the instance; can refer to RAM, CPU(s), disk space, etc., depending on the cloud provider.

state The running state of the instance; for example, `running`, `stopped`, `pending`, etc. This state is dependent upon the provider.

This command may also be used to perform a full query or a select query, as described below. The following usages are available:

```
salt myminion cloud.query
salt myminion cloud.query list_nodes
salt myminion cloud.query list_nodes_full
```

full_query

This command behaves like the `query` command, but lists all information concerning each instance as provided by the cloud provider, in addition to the fields returned by the `query` command.

```
salt myminion cloud.full_query
```

select_query

This command behaves like the `query` command, but only returned select fields as defined in the `/etc/salt/cloud` configuration file. A sample configuration for this section of the file might look like:

```
query.selection:
  - id
  - key_name
```

This configuration would only return the `id` and `key_name` fields, for those cloud providers that support those two fields. This would be called using the following command:

```
salt myminion cloud.select_query
```

profile

This command is used to create an instance using a profile that is configured on the target minion. Please note that the profile must be configured before this command can be used with it.

```
salt myminion cloud.profile ec2-centos64-x64 my-new-instance
```

Please note that the execution module does *not* run in parallel mode. Using multiple minions to create instances can effectively perform parallel instance creation.

create

This command is similar to the `profile` command, in that it is used to create a new instance. However, it does not require a profile to be pre-configured. Instead, all of the options that are normally configured in a profile are passed directly to Salt Cloud to create the instance:

```
salt myminion cloud.create my-ec2-config my-new-instance \
    image=ami-1624987f size='t1.micro' ssh_username=ec2-user \
    securitygroup=default delvol_on_destroy=True
```

Please note that the execution module does *not* run in parallel mode. Using multiple minions to create instances can effectively perform parallel instance creation.

destroy

This command is used to destroy an instance or instances. This command will search all configured providers and remove any instance(s) which matches the name(s) passed in here. The results of this command are *non-reversable* and should be used with caution.

```
salt myminion cloud.destroy myinstance
salt myminion cloud.destroy myinstance1,myinstance2
```

action

This command implements both the `action` and the `function` commands used in the standard `salt-cloud` command. If one of the standard `action` commands is used, an instance name must be provided. If one of the standard `function` commands is used, a provider configuration must be named.

```
salt myminion cloud.action start instance=myinstance
salt myminion cloud.action show_image provider=my-ec2-config \
    image=ami-1624987f
```

The actions available are largely dependent upon the module for the specific cloud provider. The following actions are available for all cloud providers:

list_nodes This is a direct call to the `query` function as described above, but is only performed against a single cloud provider. A provider configuration must be included.

list_nodes_select This is a direct call to the `full_query` function as described above, but is only performed against a single cloud provider. A provider configuration must be included.

list_nodes_select This is a direct call to the `select_query` function as described above, but is only performed against a single cloud provider. A provider configuration must be included.

show_instance This is a thin wrapper around `list_nodes`, which returns the full information about a single instance. An instance name must be provided.

State Module

A subset of the execution module is available through the `cloud` state module. Not all functions are currently included, because there is currently insufficient code for them to perform statefully. For example, a command to create an instance may be issued with a series of options, but those options cannot currently be statefully managed. Additional states to manage these options will be released at a later time.

cloud.present

This state will ensure that an instance is present inside a particular cloud provider. Any option that is normally specified in the `cloud.create` execution module and function may be declared here, but only the actual presence of the instance will be managed statefully.

```
my-instance-name:
  cloud.present:
    - provider: my-ec2-config
    - image: ami-1624987f
    - size: 't1.micro'
    - ssh_username: ec2-user
    - securitygroup: default
    - delvol_on_destroy: True
```

cloud.profile

This state will ensure that an instance is present inside a particular cloud provider. This function calls the `cloud.profile` execution module and function, but as with `cloud.present`, only the actual presence of the instance will be managed statefully.

```
my-instance-name:
  cloud.profile:
    - profile: ec2-centos64-x64
```

cloud.absent

This state will ensure that an instance (identified by name) does not exist in any of the cloud providers configured on the target minion. Please note that this state is *non-reversable* and may be considered especially destructive when issued as a cloud state.

```
my-instance-name:
  cloud.absent
```

Runner Module

The `cloud` runner module is executed on the master, and performs actions using the configuration and Salt modules on the master itself. This means that any public minion keys will also be properly accepted by the master.

Using the functions in the runner module is no different than using those in the execution module, outside of the behavior described in the above paragraph. The following functions are available inside the runner:

- list_images
- list_sizes
- list_locations
- query
- full_query
- select_query
- profile
- destroy
- action

Outside of the standard usage of `salt-run` itself, commands are executed as usual:

```
salt-run cloud.profile ec2-centos64-x86_64 my-instance-name
```

CloudClient

The execution, state, and runner modules ultimately all use the CloudClient library that ships with Salt. To use the CloudClient library locally (either on the master or a minion), create a client object and issue a command against it:

```
import salt.cloud
import pprint
client = salt.cloud.CloudClient('/etc/salt/cloud')
nodes = client.query()
pprint.pprint(nodes)
```

15.10 Feature Comparison

15.10.1 Feature Matrix

A number of features are available in most cloud providers, but not all are available everywhere. This may be because the feature isn't supported by the cloud provider itself, or it may only be that the feature has not yet been added to

Salt Cloud. In a handful of cases, it is because the feature does not make sense for a particular cloud provider (Saltify, for instance).

This matrix shows which features are available in which cloud providers, as far as Salt Cloud is concerned. This is not a comprehensive list of all features available in all cloud providers, and should not be used to make business decisions concerning choosing a cloud provider. In most cases, adding support for a feature to Salt Cloud requires only a little effort.

Legacy Drivers

Both AWS and Rackspace are listed as ``Legacy''. This is because those drivers have been replaced by other drivers, which are generally the preferred method for working with those providers.

The EC2 driver should be used instead of the AWS driver, when possible. The OpenStack driver should be used instead of the Rackspace driver, unless the user is dealing with instances in ``the old cloud'' in Rackspace.

Note for Developers

When adding new features to a particular cloud provider, please make sure to add the feature to this table. Additionally, if you notice a feature that is not properly listed here, pull requests to fix them is appreciated.

Standard Features

These are features that are available for almost every provider.

	AWS (Legacy)	Cloud-Stack	Digi-tal Ocean	EC2	GoGrid	JoyEnt	Lin-ode	Open-Stack	Par-al-lels	Rackspace (Legacy)	Saltify	Soft-layer	Soft-layer Hard-ware	Aliyun
Query	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes		Yes	Yes	Yes
Full Query	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes		Yes	Yes	Yes
Selec-tive Query	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes		Yes	Yes	Yes
List Sizes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes		Yes	Yes	Yes
List Im-ages	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes		Yes	Yes	Yes
List Loca-tions	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes		Yes	Yes	Yes
create	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
de-stroy	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes		Yes	Yes	Yes

Actions

These are features that are performed on a specific instance, and require an instance name to be passed in. For example:

```
# salt-cloud -a attach_volume ami.example.com
```

Actions	AWS (Legacy)	CloudStack	Digital Ocean	EC2	GoGrid	JoyEnt	Linode	OpenStack	Parallels	Rackspace (Legacy)	Saltify	Softlayer	Softlayer Hardware	Aliyun
attach_volume				Yes										
create_attach_volumes	Yes			Yes										
del_tags	Yes			Yes										
delvol_on_destroy				Yes										
detach_volume				Yes										
disable_term_protect	Yes			Yes										
enable_term_protect	Yes			Yes										
get_tags	Yes			Yes										
keepvol_on_destroy				Yes										
list_keypairs			Yes											
rename	Yes			Yes										
set_tags	Yes			Yes										
show_delvol_on_destroy				Yes										
show_instance			Yes	Yes					Yes			Yes	Yes	Yes
show_term_protect				Yes										
start	Yes			Yes		Yes			Yes					Yes
stop	Yes			Yes		Yes			Yes					Yes
take_action						Yes								

Functions

These are features that are performed against a specific cloud provider, and require the name of the provider to be passed in. For example:

```
# salt-cloud -f list_images my_digitalocean
```

Functions	AWS (Legacy)	CloudStack	Digital Ocean	EC2	GoGrid	JoyEnt	Linode	OpenStack
block_device_mappings	Yes							
create_keypair				Yes				
create_volume				Yes				
delete_key						Yes		
delete_keypair				Yes				
delete_volume				Yes				
get_image			Yes			Yes		
get_ip		Yes						
get_key		Yes						
get_keyid			Yes					
get_keypair		Yes						
get_networkid		Yes						

Table 15.1 -- continued from previous page

Functions	AWS (Legacy)	CloudStack	Digital Ocean	EC2	GoGrid	JoyEnt	Linode	OpenStack
get_node						Yes		
get_password		Yes						
get_size			Yes			Yes		
get_spot_config				Yes				
get_subnetid				Yes				
iam_profile	Yes			Yes				
import_key						Yes		
key_list						Yes		
keyname	Yes			Yes				
list_availability_zones				Yes				
list_custom_images								
list_keys						Yes		
list_nodes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
list_nodes_full	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
list_nodes_select	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
list_vlans								
rackconnect								Yes
reboot				Yes		Yes		
reformat_node						Yes		
securitygroup	Yes			Yes				
securitygroupid				Yes				
show_image				Yes				
show_key						Yes		
show_keypair			Yes	Yes				
show_volume				Yes				

15.11 Tutorials

15.11.1 Using Salt Cloud with the Event Reactor

One of the most powerful features of the Salt framework is the Event Reactor. As the Reactor was in development, Salt Cloud was regularly updated to take advantage of the Reactor upon completion. As such, various aspects of both the creation and destruction of instances with Salt Cloud fire events to the Salt Master, which can be used by the Event Reactor.

Event Structure

As of this writing, all events in Salt Cloud have a tag, which includes the ID of the instance being managed, and a payload which describes the task that is currently being handled. A Salt Cloud tag looks like:

```
salt/cloud/<minion_id>/<task>
```

For instance, the first event fired when creating an instance named `web1` would look like:

```
salt/cloud/web1/creating
```

Assuming this instance is using the `ec2-centos` profile, which is in turn using the `ec2-config` provider, the payload for this tag would look like:

```
{'name': 'web1',
 'profile': 'ec2-centos',
 'provider': 'ec2-config'}
```

Available Events

When an instance is created in Salt Cloud, whether by map, profile, or directly through an API, a minimum of five events are normally fired. More may be available, depending upon the cloud provider being used. Some of the common events are described below.

salt/cloud/<minion_id>/creating

This event states simply that the process to create an instance has begun. At this point in time, no actual work has begun. The payload for this event includes:

name profile provider

salt/cloud/<minion_id>/requesting

Salt Cloud is about to make a request to the cloud provider to create an instance. At this point, all of the variables required to make the request have been gathered, and the payload of the event will reflect those variables which do not normally pose a security risk. What is returned here is dependent upon the cloud provider. Some common variables are:

name image size location

salt/cloud/<minion_id>/querying

The instance has been successfully requested, but the necessary information to log into the instance (such as IP address) is not yet available. This event marks the beginning of the process to wait for this information.

The payload for this event normally only includes the `instance_id`.

salt/cloud/<minion_id>/waiting_for_ssh

The information required to log into the instance has been retrieved, but the instance is not necessarily ready to be accessed. Following this event, Salt Cloud will wait for the IP address to respond to a ping, then wait for the specified port (usually 22) to respond to a connection, and on Linux systems, for SSH to become available. Salt Cloud will attempt to issue the `date` command on the remote system, as a means to check for availability. If no `ssh_username` has been specified, a list of usernames (starting with `root`) will be attempted. If one or more usernames was configured for `ssh_username`, they will be added to the beginning of the list, in order.

The payload for this event normally only includes the `ip_address`.

salt/cloud/<minion_id>/deploying

The necessary port has been detected as available, and now Salt Cloud can log into the instance, upload any files used for deployment, and run the deploy script. Once the script has completed, Salt Cloud will log back into the instance and remove any remaining files.

A number of variables are used to deploy instances, and the majority of these will be available in the payload. Any keys, passwords or other sensitive data will be scraped from the payload. Most of the variables returned will be related to the profile or provider config, and any default values that could have been changed in the profile or provider, but weren't.

salt/cloud/<minion_id>/created

The deploy sequence has completed, and the instance is now available, Salted, and ready for use. This event is the final task for Salt Cloud, before returning instance information to the user and exiting.

The payload for this event contains little more than the initial `creating` event. This event is required in all cloud providers.

Configuring the Event Reactor

The Event Reactor is built into the Salt Master process, and as such is configured via the master configuration file. Normally this will will be a YAML file located at `/etc/salt/master`. Additionally, master configuration items can be stored, in YAML format, inside the `/etc/salt/master.d/` directory.

These configuration items may be stored in either location; however, they may only be stored in one location. For organizational and security purposes, it may be best to create a single configuration file, which contains only Event Reactor configuration, at `/etc/salt/master.d/reactor`.

The Event Reactor uses a top-level configuration item called `reactor`. This block contains a list of tags to be watched for, each of which also includes a list of `sls` files. For instance:

```
reactor:
  - 'salt/minion/*/start':
    - '/srv/reactor/custom-reactor.sls'
  - 'salt/cloud/*/created':
    - '/srv/reactor/cloud-alert.sls'
  - 'salt/cloud/*/destroyed':
    - '/srv/reactor/cloud-destroy-alert.sls'
```

The above configuration configures reactors for three different tags: one which is fired when a minion process has started and is available to receive commands, one which is fired when a cloud instance has been created, and one which is fired when a cloud instance is destroyed.

Note that each tag contains a wildcard (`*`) in it. For each of these tags, this will normally refer to a `minion_id`. This is not required of event tags, but is very common.

Reactor SLS Files

Reactor `sls` files should be placed in the `/srv/reactor/` directory for consistency between environments, but this is not currently enforced by Salt.

Reactor `sls` files follow a similar format to other `sls` files in Salt. By default they are written in YAML and can be templated using Jinja, but since they are processed through Salt's rendering system, any available renderer (JSON, Mako, Cheetah, etc.) can be used.

As with other `sls` files, each stanza will start with a declaration ID, followed by the function to run, and then any arguments for that function. For example:

```
# /srv/reactor/cloud-alert.sls
new_instance_alert:
  cmd.pagerduty.create_event:
```

```
    - tgt: alertserver
    - kwarg:
        description: "New instance: {{ data['name'] }}"
        details: "New cloud instance created on {{ data['provider'] }}"
        service_key: 1626dead5ecafe46231e968eb1be29c4
        profile: my-pagerduty-account
```

When the Event Reactor receives an event notifying it that a new instance has been created, this `sls` will create a new incident in PagerDuty, using the configured PagerDuty account.

The declaration ID in this example is `new_instance_alert`. The function called is `cmd.pagerduty.create_event`. The `cmd` portion of this function specifies that an execution module and function will be called, in this case, the `pagerduty.create_event` function.

Because an execution module is specified, a target (`tgt`) must be specified on which to call the function. In this case, a minion called `alertserver` has been used. Any arguments passed through to the function are declared in the `kwarg` block.

Example: Reactor-Based Highstate

When Salt Cloud creates an instance, by default it will install the Salt Minion onto the instance, along with any specified minion configuration, and automatically accept that minion's keys on the master. One of the configuration options that can be specified is `startup_states`, which is commonly set to `highstate`. This will tell the minion to immediately apply a highstate, as soon as it is able to do so.

This can present a problem with some system images on some cloud providers. For instance, Salt Cloud can be configured to log in as either the `root` user, or a user with `sudo` access. While some providers commonly use images that lock out remote `root` access and require a user with `sudo` privileges to log in (notably EC2, with their `ec2-user` login), most cloud providers fall back to `root` as the default login on all images, including for operating systems (such as Ubuntu) which normally disallow remote `root` login.

For users of these operating systems, it is understandable that a highstate would include configuration to block remote `root` logins again. However, Salt Cloud may not have finished cleaning up its deployment files by the time the minion process has started, and kicked off a highstate run. Users have reported errors from Salt Cloud getting locked out while trying to clean up after itself.

The goal of a startup state may be achieved using the Event Reactor. Because a minion fires an event when it is able to receive commands, this event can effectively be used inside the reactor system instead. The following will point the reactor system to the right `sls` file:

```
reactor:
  - 'salt/cloud/*/created':
    - '/srv/reactor/startup_highstate.sls'
```

And the following `sls` file will start a highstate run on the target minion:

```
# /srv/reactor/startup_highstate.sls
reactor_highstate:
  cmd.state.highstate:
    - tgt: {{ data['name'] }}
```

Because this event will not be fired until Salt Cloud has cleaned up after itself, the highstate run will not step on Salt Cloud's toes. And because every file on the minion is configurable, including `/etc/salt/minion`, the `startup_states` can still be configured for future minion restarts, if desired.

CHAPTER 16

netapi modules

16.1 Writing netapi modules

`netapi` modules, put simply, bind a port and start a service. They are purposefully open-ended and can be used to present a variety of external interfaces to Salt, and even present multiple interfaces at once.

See also:

The full list of netapi modules

16.1.1 Configuration

All `netapi` configuration is done in the *Salt master config* and takes a form similar to the following:

```
rest_cherrypy:
  port: 8000
  debug: True
  ssl_crt: /etc/pki/tls/certs/localhost.crt
  ssl_key: /etc/pki/tls/certs/localhost.key
```

16.1.2 The `__virtual__` function

Like all module types in Salt, `netapi` modules go through Salt's loader interface to determine if they should be loaded into memory and then executed.

The `__virtual__` function in the module makes this determination and should return `False` or a string that will serve as the name of the module. If the module raises an `ImportError` or any other errors, it will not be loaded.

16.1.3 The `start` function

The `start()` function will be called for each `netapi` module that is loaded. This function should contain the server loop that actually starts the service. This is started in a multiprocess.

16.1.4 Inline documentation

As with the rest of Salt, it is a best-practice to include liberal inline documentation in the form of a module docstring and docstrings on any classes, methods, and functions in your `netapi` module.

16.1.5 Loader "magic" methods

The loader makes the `__opts__` data structure available to any function in a `netapi` module.

16.2 Introduction to netapi modules

netapi modules provide API-centric access to Salt. Usually externally-facing services such as REST or WebSockets, XMPP, XMLRPC, etc.

In general netapi modules bind to a port and start a service. They are purposefully open-ended. A single module can be configured to run as well as multiple modules simultaneously.

netapi modules are enabled by adding configuration to your Salt Master config file and then starting the **`salt-api`** daemon. Check the docs for each module to see external requirements and configuration settings.

Communication with Salt and Salt satellite projects is done using Salt's own *Python API*. A list of available client interfaces is below.

---

salt-api

Prior to Salt's 2014.7.0 release, netapi modules lived in the separate sister projected `salt-api`. That project has been merged into the main Salt project.

See also:

The full list of netapi modules

16.3 Client interfaces

Salt's client interfaces expose executing functions by crafting a dictionary of values that are mapped to function arguments. This allows calling functions simply by creating a data structure. (And this is exactly how much of Salt's own internals work!)

class `salt.netapi.`**`NetapiClient`**(*opts*)

Provide a uniform method of accessing the various client interfaces in Salt in the form of low-data data structures. For example:

```
>>> client = NetapiClient(__opts__)
>>> lowstate = {'client': 'local', 'tgt': '*', 'fun': 'test.ping', 'arg': ''}
>>> client.run(lowstate)
```

`local`(**args*, ***kwargs*)

Run *execution modules* synchronously

See `salt.client.LocalClient.cmd()` for all available parameters.

Sends a command from the master to the targeted minions. This is the same interface that Salt's own CLI uses. Note the `arg` and `kwarg` parameters are sent down to the minion(s) and the given function, `fun`, is called with those parameters.

Returns Returns the result from the execution module

`local_async`(**args*, ***kwargs*)

Run *execution modules* asynchronously

Wraps `salt.client.LocalClient.run_job()`.

Returns job ID

local_batch(**args*, ***kwargs*)
Run *execution modules* against batches of minions

New in version 0.8.4.

Wraps `salt.client.LocalClient.cmd_batch()`

Returns Returns the result from the exeuction module for each batch of returns

runner(*fun*, *timeout=None*, ***kwargs*)
Run *runner modules <all-salt.runners>* synchronously

Wraps `salt.runner.RunnerClient.cmd_sync()`.

Note that runner functions must be called using keyword arguments. Positional arguments are not supported.

Returns Returns the result from the runner module

wheel(*fun*, ***kwargs*)
Run *wheel modules* synchronously

Wraps `salt.wheel.WheelClient.master_call()`.

Note that wheel functions must be called using keyword arguments. Positional arguments are not supported.

Returns Returns the result from the wheel module

CHAPTER 17

Salt Virt

The Salt Virt cloud controller capability was initially added to Salt in version 0.14.0 as an alpha technology.

The initial Salt Virt system supports core cloud operations:

- Virtual machine deployment
- Inspection of deployed VMs
- Virtual machine migration
- Network profiling
- Automatic VM integration with all aspects of Salt
- Image Pre-seeding

Many features are currently under development to enhance the capabilities of the Salt Virt systems.

Note: It is noteworthy that Salt was originally developed with the intent of using the Salt communication system as the backbone to a cloud controller. This means that the Salt Virt system is not an afterthought, simply a system that took the back seat to other development. The original attempt to develop the cloud control aspects of Salt was a project called butter. This project never took off, but was functional and proves the early viability of Salt to be a cloud controller.

17.1 Salt Virt Tutorial

A tutorial about how to get Salt Virt up and running has been added to the tutorial section:

Cloud Controller Tutorial

17.2 The Salt Virt Runner

The point of interaction with the cloud controller is the **virt** runner. The **virt** runner comes with routines to execute specific virtual machine routines.

Reference documentation for the virt runner is available with the runner module documentation:

`Virt Runner Reference`

17.3 Based on Live State Data

The Salt Virt system is based on using Salt to query live data about hypervisors and then using the data gathered to make decisions about cloud operations. This means that no external resources are required to run Salt Virt, and that the information gathered about the cloud is live and accurate.

17.4 Deploy from Network or Disk

17.4.1 Virtual Machine Disk Profiles

Salt Virt allows for the disks created for deployed virtual machines to be finely configured. The configuration is a simple data structure which is read from the `config.option` function, meaning that the configuration can be stored in the minion config file, the master config file, or the minion's pillar.

This configuration option is called `virt.disk`. The default `virt.disk` data structure looks like this:

```
virt.disk:
  default:
    - system:
      size: 8192
      format: qcow2
      model: virtio
```

Note: The format and model does not need to be defined, Salt will default to the optimal format used by the underlying hypervisor, in the case of kvm this it is **qcow2** and **virtio**.

This configuration sets up a disk profile called default. The default profile creates a single system disk on the virtual machine.

Define More Profiles

Many environments will require more complex disk profiles and may require more than one profile, this can be easily accomplished:

```
virt.disk:
  default:
    - system:
        size: 8192
  database:
    - system:
        size: 8192
    - data:
        size: 30720
  web:
    - system:
        size: 1024
    - logs:
        size: 5120
```

This configuration allows for one of three profiles to be selected, allowing virtual machines to be created with different storage needs of the deployed vm.

17.4.2 Virtual Machine Network Profiles

Salt Virt allows for the network devices created for deployed virtual machines to be finely configured. The configuration is a simple data structure which is read from the `config.option` function, meaning that the configuration can be stored in the minion config file, the master config file, or the minion's pillar.

This configuration option is called `virt.nic`. By default the `virt.nic` option is empty but defaults to a data structure which looks like this:

```
virt.nic:
  default:
    eth0:
      bridge: br0
      model: virtio
```

Note: The model does not need to be defined, Salt will default to the optimal model used by the underlying hypervisor, in the case of kvm this model is **virtio**

This configuration sets up a network profile called default. The default profile creates a single Ethernet device on the virtual machine that is bridged to the hypervisor's **br0** interface. This default setup does not require setting up the `virt.nic` configuration, and is the reason why a default install only requires setting up the **br0** bridge device on the hypervisor.

Define More Profiles

Many environments will require more complex network profiles and may require more than one profile, this can be easily accomplished:

```
virt.nic:
  dual:
    eth0:
      bridge: service_br
    eth1:
      bridge: storage_br
  single:
    eth0:
      bridge: service_br
  triple:
    eth0:
      bridge: service_br
    eth1:
      bridge: storage_br
    eth2:
      bridge: dmz_br
  all:
    eth0:
      bridge: service_br
    eth1:
      bridge: storage_br
    eth2:
      bridge: dmz_br
    eth3:
      bridge: database_br
  dmz:
    eth0:
      bridge: service_br
    eth1:
```

```
      bridge: dmz_br
  database:
    eth0:
      bridge: service_br
    eth1:
      bridge: database_br
```

This configuration allows for one of six profiles to be selected, allowing virtual machines to be created which attach to different network depending on the needs of the deployed vm.

CHAPTER 18

Understanding YAML

The default renderer for SLS files is the YAML renderer. YAML is a markup language with many powerful features. However, Salt uses a small subset of YAML that maps over very commonly used data structures, like lists and dictionaries. It is the job of the YAML renderer to take the YAML data structure and compile it into a Python data structure for use by Salt.

Though YAML syntax may seem daunting and terse at first, there are only three very simple rules to remember when writing YAML for SLS files.

18.1 Rule One: Indentation

YAML uses a fixed indentation scheme to represent relationships between data layers. Salt requires that the indentation for each level consists of exactly two spaces. Do not use tabs.

18.2 Rule Two: Colons

Python dictionaries are, of course, simply key-value pairs. Users from other languages may recognize this data type as hashes or associative arrays.

Dictionary keys are represented in YAML as strings terminated by a trailing colon. Values are represented by either a string following the colon, separated by a space:

```
my_key: my_value
```

In Python, the above maps to:

```
{'my_key': 'my_value'}
```

Alternatively, a value can be associated with a key through indentation.

```
my_key:
  my_value
```

Note: The above syntax is valid YAML but is uncommon in SLS files because most often, the value for a key is not singular but instead is a *list* of values.

In Python, the above maps to:

```
{'my_key': 'my_value'}
```

Dictionaries can be nested:

```
first_level_dict_key:
  second_level_dict_key: value_in_second_level_dict
```

And in Python:

```
{
    'first_level_dict_key': {
        'second_level_dict_key': 'value_in_second_level_dict'
    }
}
```

18.3 Rule Three: Dashes

To represent lists of items, a single dash followed by a space is used. Multiple items are a part of the same list as a function of their having the same level of indentation.

```
- list_value_one
- list_value_two
- list_value_three
```

Lists can be the value of a key-value pair. This is quite common in Salt:

```
my_dictionary:
  - list_value_one
  - list_value_two
  - list_value_three
```

In Python, the above maps to:

```
{'my_dictionary': ['list_value_one', 'list_value_two', 'list_value_three']}
```

18.4 Learning More

One easy way to learn more about how YAML gets rendered into Python data structures is to use an online YAML parser to see the Python output.

One excellent choice for experimenting with YAML parsing is: http://yaml-online-parser.appspot.com/

CHAPTER 19

Master Tops System

In 0.10.4 the *external_nodes* system was upgraded to allow for modular subsystems to be used to generate the top file data for a highstate run on the master.

The old *external_nodes* option has been removed. The master tops system contains a number of subsystems that are loaded via the Salt loader interfaces like modules, states, returners, runners, etc.

Using the new *master_tops* option is simple:

```
master_tops:
  ext_nodes: cobbler-external-nodes
```

for *Cobbler* or:

```
master_tops:
  reclass:
    inventory_base_uri: /etc/reclass
    classes_uri: roles
```

for *Reclass*.

It's also possible to create custom master_tops modules. These modules must go in a subdirectory called *tops* in the *extension_modules* directory. The *extension_modules* directory is not defined by default (the default */srv/salt/_modules* will NOT work as of this release)

Custom tops modules are written like any other execution module, see the source for the two modules above for examples of fully functional ones. Below is a degenerate example:

/etc/salt/master:

```
extension_modules: /srv/salt/modules
master_tops:
  customtop: True
```

/srv/salt/modules/tops/customtop.py:

```
import logging
import sys
# Define the module's virtual name
__virtualname__ = 'customtop'

log = logging.getLogger(__name__)

def __virtual__():
    return __virtualname__
```

```
def top(**kwargs):
    log.debug('Calling top in customtop')
    return {'base': ['test']}
```

salt minion state.show_top should then display something like:

```
$ salt minion state.show_top

minion
    ----------
    base:
      - test
```

CHAPTER 20

Salt SSH

Note: Salt ssh is considered production ready in version 2014.7.0

Note: On many systems, the `salt-ssh` executable will be in its own package, usually named `salt-ssh`.

In version 0.17.0 of Salt a new transport system was introduced, the ability to use SSH for Salt communication. This addition allows for Salt routines to be executed on remote systems entirely through ssh, bypassing the need for a Salt Minion to be running on the remote systems and the need for a Salt Master.

Note: The Salt SSH system does not supercede the standard Salt communication systems, it simply offers an SSH based alternative that does not require ZeroMQ and a remote agent. Be aware that since all communication with Salt SSH is executed via SSH it is substantially slower than standard Salt with ZeroMQ.

Salt SSH is very easy to use, simply set up a basic *roster* file of the systems to connect to and run `salt-ssh` commands in a similar way as standard `salt` commands.

Note: The Salt SSH eventually is supposed to support the same set of commands and functionality as standard `salt` command.

At the moment fileserver operations must be wrapped to ensure that the relevant files are delivered with the `salt-ssh` commands. The state module is an exception, which compiles the state run on the master, and in the process finds all the references to `salt://` paths and copies those files down in the same tarball as the state run. However, needed fileserver wrappers are still under development.

20.1 Salt SSH Roster

The roster system in Salt allows for remote minions to be easily defined.

Note: See the *Roster documentation* for more details.

Simply create the roster file, the default location is */etc/salt/roster*:

```
web1: 192.168.42.1
```

This is a very basic roster file where a Salt ID is being assigned to an IP address. A more elaborate roster can be created:

```
web1:
  host: 192.168.42.1 # The IP addr or DNS hostname
  user: fred         # Remote executions will be executed as user fred
  passwd: foobarbaz  # The password to use for login, if omitted, keys are used
  sudo: True         # Whether to sudo to root, not enabled by default
web2:
  host: 192.168.42.2
```

Note: sudo works only if NOPASSWD is set for user in /etc/sudoers: `fred ALL=(ALL) NOPASSWD: ALL`

20.2 Calling Salt SSH

The `salt-ssh` command can be easily executed in the same way as a salt command:

```
salt-ssh '*' test.ping
```

Commands with `salt-ssh` follow the same syntax as the `salt` command.

The standard salt functions are available! The output is the same as `salt` and many of the same flags are available. Please see http://docs.saltstack.com/ref/cli/salt-ssh.html for all of the available options.

20.2.1 Raw Shell Calls

By default `salt-ssh` runs Salt execution modules on the remote system, but `salt-ssh` can also execute raw shell commands:

```
salt-ssh '*' -r 'ifconfig'
```

20.3 States Via Salt SSH

The Salt State system can also be used with `salt-ssh`. The state system abstracts the same interface to the user in `salt-ssh` as it does when using standard `salt`. The intent is that Salt Formulas defined for standard `salt` will work seamlessly with `salt-ssh` and vice-versa.

The standard Salt States walkthroughs function by simply replacing `salt` commands with `salt-ssh`.

20.4 Targeting with Salt SSH

Due to the fact that the targeting approach differs in salt-ssh, only glob and regex targets are supported as of this writing, the remaining target systems still need to be implemented.

20.5 Configuring Salt SSH

Salt SSH takes its configuration from a master configuration file. Normally, this file is in `/etc/salt/master`. If one wishes to use a customized configuration file, the `-c` option to Salt SSH facilitates passing in a directory to look inside for a configuration file named `master`.

20.5.1 Minion Config

New in version 2015.5.1.

Minion config options can be defined globally using the master configuration option `ssh_minion_opts`. It can also be defined on a per-minion basis with the `minion_opts` entry in the roster.

20.6 Running Salt SSH as non-root user

By default, Salt read all the configuration from /etc/salt/. If you are running Salt SSH with a regular user you have to modify some paths or you will get ``Permission denied'' messages. You have to modify two parameters: `pki_dir` and `cachedir`. Those should point to a full path writable for the user.

It's recommed not to modify /etc/salt for this purpose. Create a private copy of /etc/salt for the user and run the command with `-c /new/config/path`.

20.7 Define CLI Options with Saltfile

If you are commonly passing in CLI options to `salt-ssh`, you can create a `Saltfile` to automatically use these options. This is common if you're managing several different salt projects on the same server.

So if you `cd` into a directory with a `Saltfile` with the following YAML contents:

```
salt-ssh:
  config_dir: path/to/config/dir
  max_prox: 30
  wipe_ssh: true
```

Instead of having to call `salt-ssh --config-dir=path/to/config/dir --max-procs=30 --wipe \* test.ping` you can call `salt-ssh \* test.ping`.

Boolean-style options should be specified in their YAML representation.

Note: The option keys specified must match the destination attributes for the options specified in the parser `salt.utils.parsers.SaltSSHOptionParser`. For example, in the case of the `--wipe` command line option, its `dest` is configured to be `wipe_ssh` and thus this is what should be configured in the `Saltfile`. Using the names of flags for this option, being `wipe: true` or `w: true`, will not work.

CHAPTER 21

Salt Rosters

Salt rosters are pluggable systems added in Salt 0.17.0 to facilitate the `salt-ssh` system. The roster system was created because `salt-ssh` needs a means to identify which systems need to be targeted for execution.

See also:

Full list of builtin roster modules

Note: The Roster System is not needed or used in standard Salt because the master does not need to be initially aware of target systems, since the Salt Minion checks itself into the master.

Since the roster system is pluggable, it can be easily augmented to attach to any existing systems to gather information about what servers are presently available and should be attached to by `salt-ssh`. By default the roster file is located at /etc/salt/roster.

21.1 How Rosters Work

The roster system compiles a data structure internally referred to as *targets*. The *targets* is a list of target systems and attributes about how to connect to said systems. The only requirement for a roster module in Salt is to return the *targets* data structure.

21.1.1 Targets Data

The information which can be stored in a roster *target* is the following:

```
<Salt ID>:       # The id to reference the target system with
    host:        # The IP address or DNS name of the remote host
    user:        # The user to log in as
    passwd:      # The password to log in with

    # Optional parameters
    port:        # The target system's ssh port number
    sudo:        # Boolean to run command via sudo
    priv:        # File path to ssh private key, defaults to salt-ssh.rsa
    timeout:     # Number of seconds to wait for response when establishing
                 # an SSH connection
    timeout:     # Number of seconds to wait for response
    minion_opts: # Dictionary of minion opts
```

CHAPTER 22

Reference

22.1 Full list of builtin auth modules

auto	An ``Always Approved'' eauth interface to test against, not intended for
django	Provide authentication using Django Web Framework
keystone	Provide authentication using OpenStack Keystone
ldap	Provide authentication using simple LDAP binds
mysql	Provide authentication using MySQL.
pam	Authenticate against PAM
pki	Authenticate via a PKI certificate.
stormpath_mod	Salt Stormpath Authentication
yubico	Provide authentication using YubiKey.

22.1.1 salt.auth.auto

An ``Always Approved'' eauth interface to test against, not intended for production use

`salt.auth.auto.`**`auth`**(*username*, *password*)
Authenticate!

22.1.2 salt.auth.django

Provide authentication using Django Web Framework

depends

- Django Web Framework

Django authentication depends on the presence of the django framework in the `PYTHONPATH`, the Django project's `settings.py` file being in the `PYTHONPATH` and accessible via the `DJANGO_SETTINGS_MODULE` environment variable.

Django auth can be defined like any other eauth module:

```
external_auth:
  django:
    fred:
      - .*
      - '@runner'
```

This will authenticate Fred via Django and allow him to run any execution module and all runners.

The authorization details can optionally be located inside the Django database. The relevant entry in the `models.py` file would look like this:

```
class SaltExternalAuthModel(models.Model):
    user_fk = models.ForeignKey(auth.User)
    minion_matcher = models.CharField()
    minion_fn = models.CharField()
```

The `external_auth` clause in the master config would then look like this:

```
external_auth:
  django:
    ^model: <fully-qualified reference to model class>
```

When a user attempts to authenticate via Django, Salt will import the package indicated via the keyword `^model`. That model must have the fields indicated above, though the model DOES NOT have to be named `SaltExternalAuthModel'.

`salt.auth.django.`**`auth`**`(`*username*, *password*`)`
Simple Django auth

`salt.auth.django.`**`django_auth_setup`**`()`

`salt.auth.django.`**`retrieve_auth_entries`**`(`*u=None*`)`

Parameters

- **`django_auth_class`** -- Reference to the django model class for auth
- **`u`** -- Username to filter for

Returns Dictionary that can be slotted into the `__opts__` structure for eauth that designates the user associated ACL

Database records such as:

username	minion_or_fn_matcher	minion_fn
fred		test.ping
fred	server1	network.interfaces
fred	server1	raid.list
fred	server2	.*
guru	.*	
smartadmin	server1	.*

Should result in an eauth config such as:

```
    fred:
      - test.ping
      - server1:
          - network.interfaces
          - raid.list
      - server2:
          - .*
    guru:
      - .*
    smartadmin:
      - server1:
        - .*
```

22.1.3 salt.auth.keystone

Provide authentication using OpenStack Keystone

depends

- keystoneclient Python module

`salt.auth.keystone.`**`auth`**(*username*, *password*)
Try and authenticate

`salt.auth.keystone.`**`get_auth_url`**()
Try and get the URL from the config, else return localhost

22.1.4 salt.auth.ldap

Provide authentication using simple LDAP binds

depends

- ldap Python module

`salt.auth.ldap.`**`auth`**(*username*, *password*)
Simple LDAP auth

`salt.auth.ldap.`**`groups`**(*username*, ***kwargs*)
Authenticate against an LDAP group

Behavior is highly dependent on if Active Directory is in use.

AD handles group membership very differently than OpenLDAP. See the *External Authentication* documentation for a thorough discussion of available parameters for customizing the search.

OpenLDAP allows you to search for all groups in the directory and returns members of those groups. Then we check against the username entered.

22.1.5 salt.auth.mysql

Provide authentication using MySQL.

When using MySQL as an authentication backend, you will need to create or use an existing table that has a username and a password column.

To get started, create a simple table that holds just a username and a password. The password field will hold a SHA256 checksum.

```
CREATE TABLE users (
    id int(11) NOT NULL AUTO_INCREMENT,
    username varchar(25) DEFAULT NULL,
    password varchar(70) DEFAULT NULL,
    PRIMARY KEY (id)
) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=latin1;
```

To create a user within MySQL, execute the following statement.

```
INSERT INTO users VALUES (NULL, `diana', SHA2(`secret', 256))
```

mysql_auth:
    hostname: localhost
    database: SaltStack
    username: root
    password: letmein
    auth_sql: `SELECT username FROM users WHERE username = ``{0}'' AND password = SHA2(``{1}'', 256)'

The *auth_sql* contains the SQL that will validate a user to ensure they are correctly authenticated. This is where you can specify other SQL queries to authenticate users.

Enable MySQL authentication.

```
external_auth:
  mysql:
    damian:
      - test.*
```

depends

- MySQL-python Python module

salt.auth.mysql.**auth**(*username*, *password*)
Authenticate using a MySQL user table

22.1.6 salt.auth.pam

Authenticate against PAM

Provides an authenticate function that will allow the caller to authenticate a user against the Pluggable Authentication Modules (PAM) on the system.

Implemented using ctypes, so no compilation is necessary.

Note: PAM authentication will not work for the `root` user.

The Python interface to PAM does not support authenticating as `root`.

---

class salt.auth.pam.**PamConv**
Wrapper class for pam_conv structure

appdata_ptr
Structure/Union member

conv
Structure/Union member

class salt.auth.pam.**PamHandle**
Wrapper class for pam_handle_t

handle
Structure/Union member

class salt.auth.pam.**PamMessage**
Wrapper class for pam_message structure

msg
Structure/Union member

msg_style
Structure/Union member

class salt.auth.pam.**PamResponse**
Wrapper class for pam_response structure

resp
Structure/Union member

resp_retcode
Structure/Union member

salt.auth.pam.**auth**(*username*, *password*, ***kwargs*)
Authenticate via pam

salt.auth.pam.**authenticate**(*username*, *password*, *service='login'*)
Returns True if the given username and password authenticate for the given service. Returns False otherwise

`username`: the username to authenticate

`password`: the password in plain text

`service`: the PAM service to authenticate against. Defaults to `login'

salt.auth.pam.**groups**(*username*, **args*, ***kwargs*)
Retrieve groups for a given user for this auth provider

Uses system groups

22.1.7 salt.auth.pki

Authenticate via a PKI certificate.

Note: This module is Experimental and should be used with caution

Provides an authenticate function that will allow the caller to authenticate a user via their public cert against a pre-defined Certificate Authority.

TODO: Add a `ca_dir' option to configure a directory of CA files, a la Apache.

depends

- pyOpenSSL module

salt.auth.pki.**auth**(*pem*, ***kwargs*)
Returns True if the given user cert was issued by the CA. Returns False otherwise.

`pem`: a pem-encoded user public key (certificate)

Configure the CA cert in the master config file:

```
external_auth:
  pki:
    ca_file: /etc/pki/tls/ca_certs/trusted-ca.crt
```

22.1.8 salt.auth.stormpath_mod

Salt Stormpath Authentication

Module to provide authentication using Stormpath as the backend.

depends

- stormpath-sdk Python module

configuration This module requires the development branch of the stormpath-sdk which can be found here: https://github.com/stormpath/stormpath-sdk-python

The following config items are required in the master config:

```
stormpath.api_key_file: <path/to/apiKey.properties>
stormpath.app_url: <Rest url of your Stormpath application>
```

Ensure that your apiKey.properties is readable by the user the Salt Master is running as, but not readable by other system users.

`salt.auth.stormpath_mod.auth`(*username*, *password*)
Try and authenticate

22.1.9 salt.auth.yubico

Provide authentication using YubiKey.

New in version 2015.5.0.

depends yubico-client Python module

To get your YubiKey API key you will need to visit the website below.

https://upgrade.yubico.com/getapikey/

The resulting page will show the generated Client ID (aka AuthID or API ID) and the generated API key (Secret Key). Make a note of both and use these two values in your /etc/salt/master configuration.

/etc/salt/master

```
yubico_users:
  damian:
    id: 12345
    key: ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

```
external_auth:
  yubico:
    damian:
      - test.*
```

Please wait five to ten minutes after generating the key before testing so that the API key will be updated on all the YubiCloud servers.

`salt.auth.yubico.auth`(*username*, *password*)
Authentcate against yubico server

22.2 Command Line Reference

Salt can be controlled by a command line client by the root user on the Salt master. The Salt command line client uses the Salt client API to communicate with the Salt master server. The Salt client is straightforward and simple to use.

Using the Salt client commands can be easily sent to the minions.

Each of these commands accepts an explicit *--config* option to point to either the master or minion configuration file. If this option is not provided and the default configuration file does not exist then Salt falls back to use the environment variables `SALT_MASTER_CONFIG` and `SALT_MINION_CONFIG`.

See also:

Configuration

22.2.1 Using the Salt Command

The Salt command needs a few components to send information to the Salt minions. The target minions need to be defined, the function to call and any arguments the function requires.

Defining the Target Minions

The first argument passed to salt, defines the target minions, the target minions are accessed via their hostname. The default target type is a bash glob:

```
salt '*foo.com' sys.doc
```

Salt can also define the target minions with regular expressions:

```
salt -E '.*' cmd.run 'ls -l | grep foo'
```

Or to explicitly list hosts, salt can take a list:

```
salt -L foo.bar.baz,quo.qux cmd.run 'ps aux | grep foo'
```

More Powerful Targets

The simple target specifications, glob, regex, and list will cover many use cases, and for some will cover all use cases, but more powerful options exist.

Targeting with Grains

The Grains interface was built into Salt to allow minions to be targeted by system properties. So minions running on a particular operating system can be called to execute a function, or a specific kernel.

Calling via a grain is done by passing the -G option to salt, specifying a grain and a glob expression to match the value of the grain. The syntax for the target is the grain key followed by a globexpression: ``os:Arch*'".

```
salt -G 'os:Fedora' test.ping
```

Will return True from all of the minions running Fedora.

To discover what grains are available and what the values are, execute the grains.item salt function:

```
salt '*' grains.items
```

more info on using targeting with grains can be found *here*.

Targeting with Executions

As of 0.8.8 targeting with executions is still under heavy development and this documentation is written to reference the behavior of execution matching in the future.

Execution matching allows for a primary function to be executed, and then based on the return of the primary function the main function is executed.

Execution matching allows for matching minions based on any arbitrary running data on the minions.

Compound Targeting

New in version 0.9.5.

Multiple target interfaces can be used in conjunction to determine the command targets. These targets can then be combined using and or or statements. This is well defined with an example:

```
salt -C 'G@os:Debian and webser* or E@db.*' test.ping
```

In this example any minion who's id starts with `webser` and is running Debian, or any minion who's id starts with db will be matched.

The type of matcher defaults to glob, but can be specified with the corresponding letter followed by the `@` symbol. In the above example a grain is used with `G@` as well as a regular expression with `E@`. The `webser*` target does not need to be prefaced with a target type specifier because it is a glob.

more info on using compound targeting can be found *here*.

Node Group Targeting

New in version 0.9.5.

For certain cases, it can be convenient to have a predefined group of minions on which to execute commands. This can be accomplished using what are called *nodegroups*. Nodegroups allow for predefined compound targets to be declared in the master configuration file, as a sort of shorthand for having to type out complicated compound expressions.

```
nodegroups:
  group1: 'L@foo.domain.com,bar.domain.com,baz.domain.com and bl*.domain.com'
  group2: 'G@os:Debian and foo.domain.com'
  group3: 'G@os:Debian and N@group1'
```

Calling the Function

The function to call on the specified target is placed after the target specification.

New in version 0.9.8.

Functions may also accept arguments, space-delimited:

```
salt '*' cmd.exec_code python 'import sys; print sys.version'
```

Optional, keyword arguments are also supported:

```
salt '*' pip.install salt timeout=5 upgrade=True
```

They are always in the form of `kwarg=argument`.

Arguments are formatted as YAML:

```
salt '*' cmd.run 'echo "Hello: $FIRST_NAME"' env='{FIRST_NAME: "Joe"}'
```

Note: dictionaries must have curly braces around them (like the `env` keyword argument above). This was changed in 0.15.1: in the above example, the first argument used to be parsed as the dictionary `{'echo "Hello': '$FIRST_NAME"'}`. This was generally not the expected behavior.

If you want to test what parameters are actually passed to a module, use the `test.arg_repr` command:

```
salt '*' test.arg_repr 'echo "Hello: $FIRST_NAME"' env='{FIRST_NAME: "Joe"}'
```

Finding available minion functions

The Salt functions are self documenting, all of the function documentation can be retried from the minions via the `sys.doc()` function:

```
salt '*' sys.doc
```

Compound Command Execution

If a series of commands needs to be sent to a single target specification then the commands can be sent in a single publish. This can make gathering groups of information faster, and lowers the stress on the network for repeated commands.

Compound command execution works by sending a list of functions and arguments instead of sending a single function and argument. The functions are executed on the minion in the order they are defined on the command line, and then the data from all of the commands are returned in a dictionary. This means that the set of commands are called in a predictable way, and the returned data can be easily interpreted.

Executing compound commands if done by passing a comma delimited list of functions, followed by a comma delimited list of arguments:

```
salt '*' cmd.run,test.ping,test.echo 'cat /proc/cpuinfo',,foo
```

The trick to look out for here, is that if a function is being passed no arguments, then there needs to be a placeholder for the absent arguments. This is why in the above example, there are two commas right next to each other. `test.ping` takes no arguments, so we need to add another comma, otherwise Salt would attempt to pass ``foo'' to `test.ping`.

If you need to pass arguments that include commas, then make sure you add spaces around the commas that separate arguments. For example:

```
salt '*' cmd.run,test.ping,test.echo 'echo "1,2,3"' , , foo
```

You may change the arguments separator using the `--args-separator` option:

```
salt --args-separator=:: '*' some.fun,test.echo params with , comma :: foo
```

22.2.2 salt-call

salt-call

Synopsis

```
salt-call [options]
```

Description

The salt-call command is used to run module functions locally on a minion instead of executing them from the master.

Options

--version
Print the version of Salt that is running.

--versions-report
Show program's dependencies and version number, and then exit

-h, --help
Show the help message and exit

-c CONFIG_DIR, **--config-dir**=CONFIG_dir
The location of the Salt configuration directory. This directory contains the configuration files for Salt master and minions. The default location on most systems is `/etc/salt`.

--hard-crash
Raise any original exception rather than exiting gracefully Default: False

-g, --grains
Return the information generated by the Salt grains

-m MODULE_DIRS, **--module-dirs**=MODULE_DIRS
Specify an additional directory to pull modules from. Multiple directories can be provided by passing -m /--module-dirs multiple times.

-d, --doc, --documentation
Return the documentation for the specified module or for all modules if none are specified

--master=MASTER
Specify the master to use. The minion must be authenticated with the master. If this option is omitted, the master options from the minion config will be used. If multi masters are set up the first listed master that responds will be used.

--return RETURNER
Set salt-call to pass the return data to one or many returner interfaces. To use many returner interfaces specify a comma delimited list of returners.

--local
Run salt-call locally, as if there was no master running.

--file-root=FILE_ROOT
Set this directory as the base file root.

--pillar-root=PILLAR_ROOT
Set this directory as the base pillar root.

--retcode-passthrough
Exit with the salt call retcode and not the salt binary retcode

--metadata
Print out the execution metadata as well as the return. This will print out the outputter data, the return code, etc.

--id=ID
Specify the minion id to use. If this option is omitted, the id option from the minion config will be used.

--skip-grains
Do not load grains.

--refresh-grains-cache
Force a refresh of the grains cache

Logging Options Logging options which override any settings defined on the configuration files.

-l LOG_LEVEL, **--log-level**=LOG_LEVEL
Console logging log level. One of `all`, `garbage`, `trace`, `debug`, `info`, `warning`, `error`, `quiet`. Default: `info`.

--log-file=LOG_FILE
Log file path. Default: /var/log/salt/minion.

--log-file-level=LOG_LEVEL_LOGFILE
Logfile logging log level. One of `all`, `garbage`, `trace`, `debug`, `info`, `warning`, `error`, `quiet`. Default: `info`.

Output Options

--out
Pass in an alternative outputter to display the return of data. This outputter can be any of the available outputters:

`grains`, `highstate`, `json`, `key`, `overstatestage`, `pprint`, `raw`, `txt`, `yaml`

Some outputters are formatted only for data returned from specific functions; for instance, the `grains` outputter will not work for non-grains data.

If an outputter is used that does not support the data passed into it, then Salt will fall back on the `pprint` outputter and display the return data using the Python `pprint` standard library module.

Note: If using `--out=json`, you will probably want `--static` as well. Without the static option, you will get a JSON string for each minion. This is due to using an iterative outputter. So if you want to feed it to a JSON parser, use `--static` as well.

--out-indent OUTPUT_INDENT, **--output-indent** OUTPUT_INDENT
Print the output indented by the provided value in spaces. Negative values disable indentation. Only applicable in outputters that support indentation.

--out-file=OUTPUT_FILE, **--output-file**=OUTPUT_FILE
Write the output to the specified file.

--no-color
Disable all colored output

--force-color
Force colored output

Note: When using colored output the color codes are as follows:

`green` denotes success, `red` denotes failure, `blue` denotes changes and success and `yellow` denotes a expected future change in configuration.

See also

`salt(1)` `salt-master(1)` `salt-minion(1)`

22.2.3 salt

salt

Synopsis

salt `*' [options] sys.doc

salt -E `.*' [options] sys.doc cmd

salt -G `os:Arch.*' [options] test.ping

salt -C `G@os:Arch.* and webserv* or G@kernel:FreeBSD' [options] test.ping

Description

Salt allows for commands to be executed across a swath of remote systems in parallel. This means that remote systems can be both controlled and queried with ease.

Options

--version
Print the version of Salt that is running.

--versions-report
Show program's dependencies and version number, and then exit

-h, --help
Show the help message and exit

-c CONFIG_DIR, **--config-dir**=CONFIG_dir
The location of the Salt configuration directory. This directory contains the configuration files for Salt master and minions. The default location on most systems is `/etc/salt`.

-t TIMEOUT, **--timeout**=TIMEOUT
The timeout in seconds to wait for replies from the Salt minions. The timeout number specifies how long the command line client will wait to query the minions and check on running jobs. Default: 5

-s, --static
By default as of version 0.9.8 the salt command returns data to the console as it is received from minions, but previous releases would return data only after all data was received. To only return the data with a hard timeout and after all minions have returned then use the static option.

--async
Instead of waiting for the job to run on minions only print the job id of the started execution and complete.

--state-output=STATE_OUTPUT
New in version 0.17.

Override the configured `state_output` value for minion output. One of `full`, `terse`, `mixed`, `changes` or `filter`. Default: `full`.

--subset=SUBSET
Execute the routine on a random subset of the targeted minions. The minions will be verified that they have the named function before executing.

-v VERBOSE, **--verbose**
Turn on verbosity for the salt call, this will cause the salt command to print out extra data like the job id.

--hide-timeout
Instead of showing the return data for all minions. This option prints only the online minions which could be reached.

-b BATCH, **--batch-size**=BATCH
Instead of executing on all targeted minions at once, execute on a progressive set of minions. This option takes an argument in the form of an explicit number of minions to execute at once, or a percentage of minions to execute on.

-a EAUTH, **--auth**=EAUTH
Pass in an external authentication medium to validate against. The credentials will be prompted for. The options are *auto*, *keystone*, *ldap*, *pam*, and *stormpath*. Can be used with the -T option.

-T, --make-token
Used in conjunction with the -a option. This creates a token that allows for the authenticated user to send commands without needing to re-authenticate.

--return=RETURNER
Choose an alternative returner to call on the minion, if an alternative returner is used then the return will not come back to the command line but will be sent to the specified return system. The options are *carbon*, *cassandra*, *couchbase*, *couchdb*, *elasticsearch*, *etcd*, *hipchat*, *local*, *local_cache*, *memcache*, *mongo*, *mysql*, *odbc*, *postgres*, *redis*, *sentry*, *slack*, *sms*, *smtp*, *sqlite3*, *syslog*, and *xmpp*.

-d, --doc, --documentation
Return the documentation for the module functions available on the minions

--args-separator=ARGS_SEPARATOR
Set the special argument used as a delimiter between command arguments of compound commands. This is useful when one wants to pass commas as arguments to some of the commands in a compound command.

Logging Options Logging options which override any settings defined on the configuration files.

-l LOG_LEVEL, **--log-level**=LOG_LEVEL
Console logging log level. One of `all`, `garbage`, `trace`, `debug`, `info`, `warning`, `error`, `quiet`. Default: `warning`.

--log-file=LOG_FILE
Log file path. Default: /var/log/salt/master.

--log-file-level=LOG_LEVEL_LOGFILE
Logfile logging log level. One of `all`, `garbage`, `trace`, `debug`, `info`, `warning`, `error`, `quiet`. Default: `warning`.

Target Selection

-E, --pcre
The target expression will be interpreted as a PCRE regular expression rather than a shell glob.

-L, --list
The target expression will be interpreted as a comma-delimited list; example: server1.foo.bar,server2.foo.bar,example7.quo.qux

-G, --grain
The target expression matches values returned by the Salt grains system on the minions. The target expression is in the format of `<grain value>:<glob expression>'; example: `os:Arch*'

This was changed in version 0.9.8 to accept glob expressions instead of regular expression. To use regular expression matching with grains, use the --grain-pcre option.

--grain-pcre
The target expression matches values returned by the Salt grains system on the minions. The target expression is in the format of `<grain value>:< regular expression>'; example: `os:Arch.*'

-N, --nodegroup
Use a predefined compound target defined in the Salt master configuration file.

-R, --range
Instead of using shell globs to evaluate the target, use a range expression to identify targets. Range expressions look like %cluster.

Using the Range option requires that a range server is set up and the location of the range server is referenced in the master configuration file.

-C, --compound
Utilize many target definitions to make the call very granular. This option takes a group of targets separated by `and` or `or`. The default matcher is a glob as usual. If something other than a glob is used, preface it with the letter denoting the type; example: `webserv* and G@os:Debian or E@db*` Make sure that the compound target is encapsulated in quotes.

-I, --pillar
Instead of using shell globs to evaluate the target, use a pillar value to identify targets. The syntax for the target is the pillar key followed by a glob expression: ``role:production*'

-S, --ipcidr
Match based on Subnet (CIDR notation) or IPv4 address.

Output Options

--out
Pass in an alternative outputter to display the return of data. This outputter can be any of the available outputters:

`grains`, `highstate`, `json`, `key`, `overstatestage`, `pprint`, `raw`, `txt`, `yaml`

Some outputters are formatted only for data returned from specific functions; for instance, the `grains` outputter will not work for non-grains data.

If an outputter is used that does not support the data passed into it, then Salt will fall back on the `pprint` outputter and display the return data using the Python `pprint` standard library module.

Note: If using `--out=json`, you will probably want `--static` as well. Without the static option, you will get a JSON string for each minion. This is due to using an iterative outputter. So if you want to feed it to a JSON parser, use `--static` as well.

--out-indent OUTPUT_INDENT, **--output-indent** OUTPUT_INDENT
Print the output indented by the provided value in spaces. Negative values disable indentation. Only applicable in outputters that support indentation.

--out-file=OUTPUT_FILE, **--output-file**=OUTPUT_FILE
Write the output to the specified file.

--no-color
Disable all colored output

--force-color
Force colored output

Note: When using colored output the color codes are as follows:

`green` denotes success, `red` denotes failure, `blue` denotes changes and success and `yellow` denotes a expected future change in configuration.

See also

`salt(7)` `salt-master(1)` `salt-minion(1)`

22.2.4 salt-cloud

salt-cloud

Provision virtual machines in the cloud with Salt

Synopsis

```
salt-cloud -m /etc/salt/cloud.map

salt-cloud -m /etc/salt/cloud.map NAME

salt-cloud -m /etc/salt/cloud.map NAME1 NAME2

salt-cloud -p PROFILE NAME

salt-cloud -p PROFILE NAME1 NAME2 NAME3 NAME4 NAME5 NAME6
```

Description

Salt Cloud is the system used to provision virtual machines on various public clouds via a cleanly controlled profile and mapping system.

Options

--version
Print the version of Salt that is running.

--versions-report
Show program's dependencies and version number, and then exit

-h, --help
Show the help message and exit

-c CONFIG_DIR, **--config-dir**=CONFIG_dir
The location of the Salt configuration directory. This directory contains the configuration files for Salt master and minions. The default location on most systems is `/etc/salt`.

Execution Options
-L LOCATION, **--location**=LOCATION
Specify which region to connect to.

-a ACTION, **--action**=ACTION
: Perform an action that may be specific to this cloud provider. This argument requires one or more instance names to be specified.

-f <FUNC-NAME> <PROVIDER>, **--function**=<FUNC-NAME> <PROVIDER>
: Perform an function that may be specific to this cloud provider, that does not apply to an instance. This argument requires a provider to be specified (i.e.: nova).

-p PROFILE, **--profile**=PROFILE
: Select a single profile to build the named cloud VMs from. The profile must be defined in the specified profiles file.

-m MAP, **--map**=MAP
: Specify a map file to use. If used without any other options, this option will ensure that all of the mapped VMs are created. If the named VM already exists then it will be skipped.

-H, --hard
: When specifying a map file, the default behavior is to ensure that all of the VMs specified in the map file are created. If the --hard option is set, then any VMs that exist on configured cloud providers that are not specified in the map file will be destroyed. Be advised that this can be a destructive operation and should be used with care.

-d, --destroy
: Pass in the name(s) of VMs to destroy, salt-cloud will search the configured cloud providers for the specified names and destroy the VMs. Be advised that this is a destructive operation and should be used with care. Can be used in conjunction with the -m option to specify a map of VMs to be deleted.

-P, --parallel
: Normally when building many cloud VMs they are executed serially. The -P option will run each cloud vm build in a separate process allowing for large groups of VMs to be build at once.

  Be advised that some cloud provider's systems don't seem to be well suited for this influx of vm creation. When creating large groups of VMs watch the cloud provider carefully.

-Q, --query
: Execute a query and print out information about all cloud VMs. Can be used in conjunction with -m to display only information about the specified map.

-u, --update-bootstrap
: Update salt-bootstrap to the latest develop version on GitHub.

-y, --assume-yes
: Default yes in answer to all confirmation questions.

-k, --keep-tmp
: Do not remove files from /tmp/ after deploy.sh finishes.

--show-deploy-args
: Include the options used to deploy the minion in the data returned.

--script-args=SCRIPT_ARGS
: Script arguments to be fed to the bootstrap script when deploying the VM.

Query Options

-Q, --query
: Execute a query and return some information about the nodes running on configured cloud providers

-F, --full-query
: Execute a query and print out all available information about all cloud VMs. Can be used in conjunction with -m to display only information about the specified map.

-S, --select-query
Execute a query and print out selected information about all cloud VMs. Can be used in conjunction with -m to display only information about the specified map.

--list-providers
Display a list of configured providers.

--list-profiles
New in version 2014.7.0.

Display a list of configured profiles. Pass in a cloud provider to view the provider's associated profiles, such as `digital_ocean`, or pass in `all` to list all the configured profiles.

Cloud Providers Listings

--list-locations=LIST_LOCATIONS
Display a list of locations available in configured cloud providers. Pass the cloud provider that available locations are desired on, aka ``linode'', or pass ``all'' to list locations for all configured cloud providers

--list-images=LIST_IMAGES
Display a list of images available in configured cloud providers. Pass the cloud provider that available images are desired on, aka ``linode'', or pass ``all'' to list images for all configured cloud providers

--list-sizes=LIST_SIZES
Display a list of sizes available in configured cloud providers. Pass the cloud provider that available sizes are desired on, aka ``AWS'', or pass ``all'' to list sizes for all configured cloud providers

Cloud Credentials

--set-password=<USERNAME> <PROVIDER>
Configure password for a cloud provider and save it to the keyring. PROVIDER can be specified with or without a driver, for example: ``--set-password bob rackspace'' or more specific ``--set-password bob rackspace:openstack'' DEPRECATED!

Output Options

--out
Pass in an alternative outputter to display the return of data. This outputter can be any of the available outputters:

`grains`, `highstate`, `json`, `key`, `overstatestage`, `pprint`, `raw`, `txt`, `yaml`

Some outputters are formatted only for data returned from specific functions; for instance, the `grains` outputter will not work for non-grains data.

If an outputter is used that does not support the data passed into it, then Salt will fall back on the `pprint` outputter and display the return data using the Python `pprint` standard library module.

Note: If using `--out=json`, you will probably want `--static` as well. Without the static option, you will get a JSON string for each minion. This is due to using an iterative outputter. So if you want to feed it to a JSON parser, use `--static` as well.

--out-indent OUTPUT_INDENT, **--output-indent** OUTPUT_INDENT
Print the output indented by the provided value in spaces. Negative values disable indentation. Only applicable in outputters that support indentation.

--out-file=OUTPUT_FILE, **--output-file**=OUTPUT_FILE
Write the output to the specified file.

--no-color
Disable all colored output

--force-color
Force colored output

Note: When using colored output the color codes are as follows:

`green` denotes success, `red` denotes failure, `blue` denotes changes and success and `yellow` denotes a expected future change in configuration.

Examples

To create 4 VMs named web1, web2, db1, and db2 from specified profiles:

```
salt-cloud -p fedora_rackspace web1 web2 db1 db2
```

To read in a map file and create all VMs specified therein:

```
salt-cloud -m /path/to/cloud.map
```

To read in a map file and create all VMs specified therein in parallel:

```
salt-cloud -m /path/to/cloud.map -P
```

To delete any VMs specified in the map file:

```
salt-cloud -m /path/to/cloud.map -d
```

To delete any VMs NOT specified in the map file:

```
salt-cloud -m /path/to/cloud.map -H
```

To display the status of all VMs specified in the map file:

```
salt-cloud -m /path/to/cloud.map -Q
```

See also

`salt-cloud(7)` `salt(7)` `salt-master(1)` `salt-minion(1)`

22.2.5 salt-cp

salt-cp

Copy a file to a set of systems

Synopsis

```
salt-cp '*' [ options ] SOURCE DEST

salt-cp -E '.*' [ options ] SOURCE DEST

salt-cp -G 'os:Arch.*' [ options ] SOURCE DEST
```

Description

Salt copy copies a local file out to all of the Salt minions matched by the given target.

Options

--version
Print the version of Salt that is running.

--versions-report
Show program's dependencies and version number, and then exit

-h, --help
Show the help message and exit

-c CONFIG_DIR, **--config-dir**=CONFIG_dir
The location of the Salt configuration directory. This directory contains the configuration files for Salt master and minions. The default location on most systems is `/etc/salt`.

-t TIMEOUT, **--timeout**=TIMEOUT
The timeout in seconds to wait for replies from the Salt minions. The timeout number specifies how long the command line client will wait to query the minions and check on running jobs. Default: 5

Logging Options Logging options which override any settings defined on the configuration files.

-l LOG_LEVEL, **--log-level**=LOG_LEVEL
Console logging log level. One of `all`, `garbage`, `trace`, `debug`, `info`, `warning`, `error`, `quiet`. Default: `warning`.

--log-file=LOG_FILE
Log file path. Default: /var/log/salt/master.

--log-file-level=LOG_LEVEL_LOGFILE
Logfile logging log level. One of `all`, `garbage`, `trace`, `debug`, `info`, `warning`, `error`, `quiet`. Default: `warning`.

Target Selection

-E, --pcre
The target expression will be interpreted as a PCRE regular expression rather than a shell glob.

-L, --list
The target expression will be interpreted as a comma-delimited list; example: server1.foo.bar,server2.foo.bar,example7.quo.qux

-G, --grain
The target expression matches values returned by the Salt grains system on the minions. The target expression is in the format of `<grain value>:<glob expression>'; example: `os:Arch*'

This was changed in version 0.9.8 to accept glob expressions instead of regular expression. To use regular expression matching with grains, use the --grain-pcre option.

--grain-pcre
The target expression matches values returned by the Salt grains system on the minions. The target expression is in the format of `<grain value>:< regular expression>'; example: `os:Arch.*'

-N, --nodegroup
Use a predefined compound target defined in the Salt master configuration file.

-R, --range
Instead of using shell globs to evaluate the target, use a range expression to identify targets. Range expressions look like %cluster.

Using the Range option requires that a range server is set up and the location of the range server is referenced in the master configuration file.

See also

`salt(1)` `salt-master(1)` `salt-minion(1)`

22.2.6 salt-key

salt-key

Synopsis

```
salt-key [ options ]
```

Description

Salt-key executes simple management of Salt server public keys used for authentication.

Options

--version
Print the version of Salt that is running.

--versions-report
Show program's dependencies and version number, and then exit

-h, --help
Show the help message and exit

-c CONFIG_DIR, **--config-dir**=CONFIG_dir
The location of the Salt configuration directory. This directory contains the configuration files for Salt master and minions. The default location on most systems is `/etc/salt`.

-u USER, **--user**=USER
Specify user to run salt-key

--hard-crash
Raise any original exception rather than exiting gracefully. Default is False.

-q, --quiet
Suppress output

-y, --yes
Answer `Yes' to all questions presented, defaults to False

--rotate-aes-key=ROTATE_AES_KEY
Setting this to False prevents the master from refreshing the key session when keys are deleted or rejected, this lowers the security of the key deletion/rejection operation. Default is True.

Logging Options Logging options which override any settings defined on the configuration files.

--log-file=LOG_FILE
Log file path. Default: /var/log/salt/minion.

--log-file-level=LOG_LEVEL_LOGFILE
Logfile logging log level. One of `all`, `garbage`, `trace`, `debug`, `info`, `warning`, `error`, `quiet`. Default: `warning`.

Output Options

--out
Pass in an alternative outputter to display the return of data. This outputter can be any of the available outputters:

`grains`, `highstate`, `json`, `key`, `overstatestage`, `pprint`, `raw`, `txt`, `yaml`

Some outputters are formatted only for data returned from specific functions; for instance, the `grains` outputter will not work for non-grains data.

If an outputter is used that does not support the data passed into it, then Salt will fall back on the `pprint` outputter and display the return data using the Python `pprint` standard library module.

Note: If using `--out=json`, you will probably want `--static` as well. Without the static option, you will get a JSON string for each minion. This is due to using an iterative outputter. So if you want to feed it to a JSON parser, use `--static` as well.

--out-indent OUTPUT_INDENT, **--output-indent** OUTPUT_INDENT
Print the output indented by the provided value in spaces. Negative values disable indentation. Only applicable in outputters that support indentation.

--out-file=OUTPUT_FILE, **--output-file**=OUTPUT_FILE
Write the output to the specified file.

--no-color
Disable all colored output

--force-color
Force colored output

Note: When using colored output the color codes are as follows:

`green` denotes success, `red` denotes failure, `blue` denotes changes and success and `yellow` denotes a expected future change in configuration.

Actions

-l ARG, **--list**=ARG
List the public keys. The args `pre`, `un`, and `unaccepted` will list unaccepted/unsigned keys. `acc` or `accepted` will list accepted/signed keys. `rej` or `rejected` will list rejected keys. Finally, `all` will list all keys.

-L, **--list-all**
List all public keys. (Deprecated: use `--list all`)

-a ACCEPT, **--accept**=ACCEPT
Accept the specified public key (use --include-all to match rejected keys in addition to pending keys). Globs are supported.

-A, **--accept-all**
Accepts all pending keys.

-r REJECT, **--reject**=REJECT
Reject the specified public key (use --include-all to match accepted keys in addition to pending keys). Globs are supported.

-R, --reject-all
Rejects all pending keys.

--include-all
Include non-pending keys when accepting/rejecting.

-p PRINT, **--print**=PRINT
Print the specified public key.

-P, --print-all
Print all public keys

-d DELETE, **--delete**=DELETE
Delete the specified key. Globs are supported.

-D, --delete-all
Delete all keys.

-f FINGER, **--finger**=FINGER
Print the specified key's fingerprint.

-F, --finger-all
Print all keys' fingerprints.

Key Generation Options

--gen-keys=GEN_KEYS
Set a name to generate a keypair for use with salt

--gen-keys-dir=GEN_KEYS_DIR
Set the directory to save the generated keypair. Only works with `gen_keys_dir' option; default is the current directory.

--keysize=KEYSIZE
Set the keysize for the generated key, only works with the `--gen-keys' option, the key size must be 2048 or higher, otherwise it will be rounded up to 2048. The default is 2048.

--gen-signature
Create a signature file of the masters public-key named master_pubkey_signature. The signature can be send to a minion in the masters auth-reply and enables the minion to verify the masters public-key cryptographically. This requires a new signing-key- pair which can be auto-created with the --auto-create parameter.

--priv=PRIV
The private-key file to create a signature with

--signature-path=SIGNATURE_PATH
The path where the signature file should be written

--pub=PUB
The public-key file to create a signature for

--auto-create
Auto-create a signing key-pair if it does not yet exist

See also

salt(7) salt-master(1) salt-minion(1)

22.2.7 salt-master

salt-master

The Salt master daemon, used to control the Salt minions

Synopsis

```
salt-master [ options ]
```

Description

The master daemon controls the Salt minions

Options

--version
Print the version of Salt that is running.

--versions-report
Show program's dependencies and version number, and then exit

-h, --help
Show the help message and exit

-c CONFIG_DIR, **--config-dir**=CONFIG_dir
The location of the Salt configuration directory. This directory contains the configuration files for Salt master and minions. The default location on most systems is `/etc/salt`.

-u USER, **--user**=USER
Specify user to run salt-master

-d, --daemon
Run salt-master as a daemon

--pid-file PIDFILE
Specify the location of the pidfile. Default: /var/run/salt-master.pid

Logging Options Logging options which override any settings defined on the configuration files.

-l LOG_LEVEL, **--log-level**=LOG_LEVEL
Console logging log level. One of `all`, `garbage`, `trace`, `debug`, `info`, `warning`, `error`, `quiet`. Default: `warning`.

--log-file=LOG_FILE
Log file path. Default: /var/log/salt/master.

--log-file-level=LOG_LEVEL_LOGFILE
Logfile logging log level. One of `all`, `garbage`, `trace`, `debug`, `info`, `warning`, `error`, `quiet`. Default: `warning`.

See also

`salt(1)` `salt(7)` `salt-minion(1)`

22.2.8 salt-minion

salt-minion

The Salt minion daemon, receives commands from a remote Salt master.

Synopsis

```
salt-minion [ options ]
```

Description

The Salt minion receives commands from the central Salt master and replies with the results of said commands.

Options

--version
Print the version of Salt that is running.

--versions-report
Show program's dependencies and version number, and then exit

-h, --help
Show the help message and exit

-c CONFIG_DIR, **--config-dir**=CONFIG_dir
The location of the Salt configuration directory. This directory contains the configuration files for Salt master and minions. The default location on most systems is `/etc/salt`.

-u USER, **--user**=USER
Specify user to run salt-minion

-d, --daemon
Run salt-minion as a daemon

--pid-file PIDFILE
Specify the location of the pidfile. Default: /var/run/salt-minion.pid

Logging Options Logging options which override any settings defined on the configuration files.

-l LOG_LEVEL, **--log-level**=LOG_LEVEL
Console logging log level. One of `all`, `garbage`, `trace`, `debug`, `info`, `warning`, `error`, `quiet`. Default: `warning`.

--log-file=LOG_FILE
Log file path. Default: /var/log/salt/minion.

--log-file-level=LOG_LEVEL_LOGFILE
Logfile logging log level. One of `all`, `garbage`, `trace`, `debug`, `info`, `warning`, `error`, `quiet`. Default: `warning`.

See also

`salt(1)` `salt(7)` `salt-master(1)`

22.2.9 salt-run

salt-run

Execute a Salt runner

Synopsis

```
salt-run RUNNER
```

Description

salt-run is the frontend command for executing `Salt Runners`. Salt runners are simple modules used to execute convenience functions on the master

Options

--version
Print the version of Salt that is running.

--versions-report
Show program's dependencies and version number, and then exit

-h, --help
Show the help message and exit

-c CONFIG_DIR, **--config-dir**=CONFIG_dir
The location of the Salt configuration directory. This directory contains the configuration files for Salt master and minions. The default location on most systems is `/etc/salt`.

-t TIMEOUT, **--timeout**=TIMEOUT
The timeout in seconds to wait for replies from the Salt minions. The timeout number specifies how long the command line client will wait to query the minions and check on running jobs. Default: 1

--hard-crash
Raise any original exception rather than exiting gracefully. Default is False.

-d, --doc, --documentation
Display documentation for runners, pass a module or a runner to see documentation on only that module/runner.

Logging Options Logging options which override any settings defined on the configuration files.

-l LOG_LEVEL, **--log-level**=LOG_LEVEL
Console logging log level. One of `all`, `garbage`, `trace`, `debug`, `info`, `warning`, `error`, `quiet`. Default: `warning`.

--log-file=LOG_FILE
Log file path. Default: /var/log/salt/master.

--log-file-level=LOG_LEVEL_LOGFILE
Logfile logging log level. One of `all`, `garbage`, `trace`, `debug`, `info`, `warning`, `error`, `quiet`. Default: `warning`.

See also

`salt(1)` `salt-master(1)` `salt-minion(1)`

22.2.10 salt-ssh

salt-ssh

Synopsis

```
salt-ssh '*' [ options ] sys.doc

salt-ssh -E '.*' [ options ] sys.doc cmd
```

Description

Salt SSH allows for salt routines to be executed using only SSH for transport

Options

-r, --raw, --raw-shell
Execute a raw shell command.

--priv
Specify the SSH private key file to be used for authentication.

--roster
Define which roster system to use, this defines if a database backend, scanner, or custom roster system is used. Default is the flat file roster.

--roster-file
Define an alternative location for the default roster file location. The default roster file is called `roster` and is found in the same directory as the master config file.

New in version 2014.1.0.

--refresh, --refresh-cache
Force a refresh of the master side data cache of the target's data. This is needed if a target's grains have been changed and the auto refresh timeframe has not been reached.

--max-procs
Set the number of concurrent minions to communicate with. This value defines how many processes are opened up at a time to manage connections, the more running process the faster communication should be, default is 25.

-i, --ignore-host-keys
Ignore the ssh host keys which by default are honored and connections would ask for approval.

--passwd
Set the default password to attempt to use when authenticating.

--key-deploy
Set this flag to attempt to deploy the authorized ssh key with all minions. This combined with --passwd can make initial deployment of keys very fast and easy.

--version
Print the version of Salt that is running.

--versions-report
Show program's dependencies and version number, and then exit

-h, --help
Show the help message and exit

-c CONFIG_DIR, **--config-dir**=CONFIG_dir
The location of the Salt configuration directory. This directory contains the configuration files for Salt master and minions. The default location on most systems is `/etc/salt`.

Target Selection
-E, --pcre
The target expression will be interpreted as a PCRE regular expression rather than a shell glob.
-L, --list
The target expression will be interpreted as a comma-delimited list; example: server1.foo.bar,server2.foo.bar,example7.quo.qux

-G, --grain
The target expression matches values returned by the Salt grains system on the minions. The target expression is in the format of `<grain value>:<glob expression>'; example: `os:Arch*'

This was changed in version 0.9.8 to accept glob expressions instead of regular expression. To use regular expression matching with grains, use the --grain-pcre option.

--grain-pcre
The target expression matches values returned by the Salt grains system on the minions. The target expression is in the format of `<grain value>:< regular expression>'; example: `os:Arch.*'

-N, --nodegroup
Use a predefined compound target defined in the Salt master configuration file.

-R, --range
Instead of using shell globs to evaluate the target, use a range expression to identify targets. Range expressions look like %cluster.

Using the Range option requires that a range server is set up and the location of the range server is referenced in the master configuration file.

Logging Options Logging options which override any settings defined on the configuration files.

-l LOG_LEVEL, **--log-level**=LOG_LEVEL
Console logging log level. One of `all`, `garbage`, `trace`, `debug`, `info`, `warning`, `error`, `quiet`. Default: `warning`.

--log-file=LOG_FILE
Log file path. Default: /var/log/salt/ssh.

--log-file-level=LOG_LEVEL_LOGFILE
Logfile logging log level. One of `all`, `garbage`, `trace`, `debug`, `info`, `warning`, `error`, `quiet`. Default: `warning`.

Output Options
--out
Pass in an alternative outputter to display the return of data. This outputter can be any of the available outputters:

`grains`, `highstate`, `json`, `key`, `overstatestage`, `pprint`, `raw`, `txt`, `yaml`

Some outputters are formatted only for data returned from specific functions; for instance, the `grains` outputter will not work for non-grains data.

If an outputter is used that does not support the data passed into it, then Salt will fall back on the `pprint` outputter and display the return data using the Python `pprint` standard library module.

Note: If using `--out=json`, you will probably want `--static` as well. Without the static option, you will get a JSON string for each minion. This is due to using an iterative outputter. So if you want to feed it to a JSON parser, use `--static` as well.

`--out-indent` OUTPUT_INDENT, **`--output-indent`** OUTPUT_INDENT
Print the output indented by the provided value in spaces. Negative values disable indentation. Only applicable in outputters that support indentation.

`--out-file`=OUTPUT_FILE, **`--output-file`**=OUTPUT_FILE
Write the output to the specified file.

`--no-color`
Disable all colored output

`--force-color`
Force colored output

Note: When using colored output the color codes are as follows:

`green` denotes success, `red` denotes failure, `blue` denotes changes and success and `yellow` denotes a expected future change in configuration.

See also

`salt(7)` `salt-master(1)` `salt-minion(1)`

22.2.11 salt-syndic

salt-syndic

The Salt syndic daemon, a special minion that passes through commands from a higher master

Synopsis

```
salt-syndic [ options ]
```

Description

The Salt syndic daemon, a special minion that passes through commands from a higher master.

Options

`--version`
Print the version of Salt that is running.

--versions-report
Show program's dependencies and version number, and then exit

-h, --help
Show the help message and exit

-c CONFIG_DIR, **--config-dir**=CONFIG_dir
The location of the Salt configuration directory. This directory contains the configuration files for Salt master and minions. The default location on most systems is `/etc/salt`.

-u USER, **--user**=USER
Specify user to run salt-syndic

-d, --daemon
Run salt-syndic as a daemon

--pid-file PIDFILE
Specify the location of the pidfile. Default: /var/run/salt-syndic.pid

Logging Options Logging options which override any settings defined on the configuration files.

-l LOG_LEVEL, **--log-level**=LOG_LEVEL
Console logging log level. One of `all`, `garbage`, `trace`, `debug`, `info`, `warning`, `error`, `quiet`. Default: `warning`.

--log-file=LOG_FILE
Log file path. Default: /var/log/salt/master.

--log-file-level=LOG_LEVEL_LOGFILE
Logfile logging log level. One of `all`, `garbage`, `trace`, `debug`, `info`, `warning`, `error`, `quiet`. Default: `warning`.

See also

`salt(1)` `salt-master(1)` `salt-minion(1)`

22.2.12 salt-api

salt-api

Start interfaces used to remotely connect to the salt master

Synopsis

```
salt-api
```

Description

The Salt API system manages network api connectors for the Salt Master

Options

--version
Print the version of Salt that is running.

--versions-report
Show program's dependencies and version number, and then exit

-h, --help
Show the help message and exit

-c CONFIG_DIR, **--config-dir**=CONFIG_dir
The location of the Salt configuration directory. This directory contains the configuration files for Salt master and minions. The default location on most systems is `/etc/salt`.

-d, --daemon
Run the salt-api as a daemon

--pid-file=PIDFILE
Specify the location of the pidfile. Default: /var/run/salt-api.pid

Logging Options Logging options which override any settings defined on the configuration files.

-l LOG_LEVEL, **--log-level**=LOG_LEVEL
Console logging log level. One of `all`, `garbage`, `trace`, `debug`, `info`, `warning`, `error`, `quiet`. Default: `warning`.

--log-file=LOG_FILE
Log file path. Default: /var/log/salt/api.

--log-file-level=LOG_LEVEL_LOGFILE
Logfile logging log level. One of `all`, `garbage`, `trace`, `debug`, `info`, `warning`, `error`, `quiet`. Default: `warning`.

See also

`salt-api(7)` `salt(7)` `salt-master(1)`

22.3 Client ACL system

The salt client ACL system is a means to allow system users other than root to have access to execute select salt commands on minions from the master.

The client ACL system is configured in the master configuration file via the `client_acl` configuration option. Under the `client_acl` configuration option the users open to send commands are specified and then a list of regular expressions which specify the minion functions which will be made available to specified user. This configuration is much like the `peer` configuration:

```
# Allow thatch to execute anything and allow fred to use ping and pkg
client_acl:
  thatch:
    - .*
  fred:
    - test.*
    - pkg.*
```

22.3.1 Permission Issues

Directories required for `client_acl` must be modified to be readable by the users specified:

```
chmod 755 /var/cache/salt /var/cache/salt/master /var/cache/salt/master/jobs /var/run/salt /var/run/s
```

Note: In addition to the changes above you will also need to modify the permissions of /var/log/salt and the existing log file to be writable by the user(s) which will be running the commands. If you do not wish to do this then you must disable logging or Salt will generate errors as it cannot write to the logs as the system users.

If you are upgrading from earlier versions of salt you must also remove any existing user keys and re-start the Salt master:

```
rm /var/cache/salt/.*key
service salt-master restart
```

22.4 Python client API

Salt provides several entry points for interfacing with Python applications. These entry points are often referred to as `*Client()` APIs. Each client accesses different parts of Salt, either from the master or from a minion. Each client is detailed below.

See also:

There are many ways to access Salt programmatically.

Salt can be used from CLI scripts as well as via a REST interface.

See Salt's *outputter system* to retrieve structured data from Salt as JSON, or as shell-friendly text, or many other formats.

See the `state.event` runner to utilize Salt's event bus from shell scripts.

Salt's netapi module provides access to Salt externally via a REST interface. Review the netapi module documentation for more information.

22.4.1 Salt's `opts` dictionary

Some clients require access to Salt's `opts` dictionary. (The dictionary representation of the *master* or *minion* config files.)

A common pattern for fetching the `opts` dictionary is to defer to environment variables if they exist or otherwise fetch the config from the default location.

`salt.config.client_config`(*path*, *env_var='SALT_CLIENT_CONFIG'*, *defaults=None*)

Load Master configuration data

Usage:

```
import salt.config
master_opts = salt.config.client_config('/etc/salt/master')
```

Returns a dictionary of the Salt Master configuration file with necessary options needed to communicate with a locally-running Salt Master daemon. This function searches for client specific configurations and adds them to the data from the master configuration.

This is useful for master-side operations like `LocalClient`.

salt.config.**minion_config**(*path*, *env_var='SALT_MINION_CONFIG'*, *defaults=None*, *cache_minion_id=False*)

Reads in the minion configuration file and sets up special options

This is useful for Minion-side operations, such as the `Caller` class, and manually running the loader interface.

```
import salt.client
minion_opts = salt.config.minion_config('/etc/salt/minion')
```

22.4.2 Salt's Loader Interface

Modules in the Salt ecosystem are loaded into memory using a custom loader system. This allows modules to have conditional requirements (OS, OS version, installed libraries, etc) and allows Salt to inject special variables (`__salt__`, `__opts`, etc).

Most modules can be manually loaded. This is often useful in third-party Python apps or when writing tests. However some modules require and expect a full, running Salt system underneath. Notably modules that facilitate master-to-minion communication such as the `mine`, `publish`, and `peer` execution modules. The error `KeyError: 'master_uri'` is a likely indicator for this situation. In those instances use the `Caller` class to execute those modules instead.

Each module type has a corresponding loader function.

salt.loader.**minion_mods**(*opts*, *context=None*, *whitelist=None*, *include_errors=False*, *initial_load=False*, *loaded_base_name=None*)

Load execution modules

Returns a dictionary of execution modules appropriate for the current system by evaluating the __virtual__() function in each module.

```
import salt.config
import salt.loader

__opts__ = salt.config.minion_config('/etc/salt/minion')
__grains__ = salt.loader.grains(__opts__)
__opts__['grains'] = __grains__
__salt__ = salt.loader.minion_mods(__opts__)
__salt__['test.ping']()
```

salt.loader.**raw_mod**(*opts*, *name*, *functions*, *mod='modules'*)

Returns a single module loaded raw and bypassing the __virtual__ function

```
import salt.config
import salt.loader

__opts__ = salt.config.minion_config('/etc/salt/minion')
testmod = salt.loader.raw_mod(__opts__, 'test', None)
testmod['test.ping']()
```

salt.loader.**states**(*opts*, *functions*, *whitelist=None*)

Returns the state modules

```
import salt.config
import salt.loader

__opts__ = salt.config.minion_config('/etc/salt/minion')
statemods = salt.loader.states(__opts__, None)
```

salt.loader.**grains**(*opts*, *force_refresh=False*)
Return the functions for the dynamic grains and the values for the static grains.

```
import salt.config
import salt.loader

__opts__ = salt.config.minion_config('/etc/salt/minion')
__grains__ = salt.loader.grains(__opts__)
print __grains__['id']
```

22.4.3 Salt's Client Interfaces

LocalClient

class salt.client.**LocalClient**(*c_path='/etc/salt/master'*, *mopts=None*, *skip_perm_errors=False*)
The interface used by the **salt** CLI tool on the Salt Master

`LocalClient` is used to send a command to Salt minions to execute *execution modules* and return the results to the Salt Master.

Importing and using `LocalClient` must be done on the same machine as the Salt Master and it must be done using the same user that the Salt Master is running as. (Unless `external_auth` is configured and authentication credentials are included in the execution).

```
import salt.client

local = salt.client.LocalClient()
local.cmd('*', 'test.fib', [10])
```

cmd(*tgt*, *fun*, *arg=()*, *timeout=None*, *expr_form='glob'*, *ret=''*, *jid=''*, *kwarg=None*, ***kwargs*)
Synchronously execute a command on targeted minions

The cmd method will execute and wait for the timeout period for all minions to reply, then it will return all minion data at once.

```
>>> import salt.client
>>> local = salt.client.LocalClient()
>>> local.cmd('*', 'cmd.run', ['whoami'])
{'jerry': 'root'}
```

With extra keyword arguments for the command function to be run:

```
local.cmd('*', 'test.arg', ['arg1', 'arg2'], kwarg={'foo': 'bar'})
```

Compound commands can be used for multiple executions in a single publish. Function names and function arguments are provided in separate lists but the index values must correlate and an empty list must be used if no arguments are required.

```
>>> local.cmd('*', [
        'grains.items',
        'sys.doc',
        'cmd.run',
    ],
    [
        [],
        [],
        ['uptime'],
    ])
```

Parameters

- **tgt** (*string or list*) -- Which minions to target for the execution. Default is shell glob. Modified by the `expr_form` option.
- **fun** (*string or list of strings*) -- The module and function to call on the specified minions of the form `module.function`. For example `test.ping` or `grains.items`.

  Compound commands Multiple functions may be called in a single publish by passing a list of commands. This can dramatically lower overhead and speed up the application communicating with Salt.

  This requires that the `arg` param is a list of lists. The `fun` list and the `arg` list must correlate by index meaning a function that does not take arguments must still have a corresponding empty list at the expected index.
- **arg** (*list or list-of-lists*) -- A list of arguments to pass to the remote function. If the function takes no arguments `arg` may be omitted except when executing a compound command.
- **timeout** -- Seconds to wait after the last minion returns but before all minions return.
- **expr_form** -- The type of `tgt`. Allowed values:
  - `glob` - Bash glob completion - Default
  - `pcre` - Perl style regular expression
  - `list` - Python list of hosts
  - `grain` - Match based on a grain comparison
  - `grain_pcre` - Grain comparison with a regex
  - `pillar` - Pillar data comparison
  - `pillar_pcre` - Pillar data comparison with a regex
  - `nodegroup` - Match on nodegroup
  - `range` - Use a Range server for matching
  - `compound` - Pass a compound match string
- **ret** -- The returner to use. The value passed can be single returner, or a comma delimited list of returners to call in order on the minions
- **kwarg** -- A dictionary with keyword arguments for the function.
- **kwargs** -- Optional keyword arguments. Authentication credentials may be passed when using `external_auth`.

  For example: `local.cmd('*', 'test.ping', username='saltdev', password='saltdev', eauth='pam')`. Or: `local.cmd('*', 'test.ping', token='5871821ea51754fdcea8153c1c745433')`

Returns A dictionary with the result of the execution, keyed by minion ID. A compound command will return a sub-dictionary keyed by function name.

cmd_async(*tgt*, *fun*, *arg=()*, *expr_form='glob'*, *ret=''*, *jid=''*, *kwarg=None*, ***kwargs*)

Asynchronously send a command to connected minions

The function signature is the same as `cmd()` with the following exceptions.

Returns A job ID or 0 on failure.

```
>>> local.cmd_async('*', 'test.sleep', [300])
'20131219215921857715'
```

cmd_batch(*tgt*, *fun*, *arg=()*, *expr_form='glob'*, *ret=''*, *kwarg=None*, *batch='10%'*, ***kwargs*)
Iteratively execute a command on subsets of minions at a time

The function signature is the same as `cmd()` with the following exceptions.

Parameters **batch** -- The batch identifier of systems to execute on

Returns A generator of minion returns

```
>>> returns = local.cmd_batch('*', 'state.highstate', bat='10%')
>>> for return in returns:
...     print return
{'jerry': {...}}
{'dave': {...}}
{'stewart': {...}}
```

cmd_iter(*tgt*, *fun*, *arg=()*, *timeout=None*, *expr_form='glob'*, *ret=''*, *kwarg=None*, ***kwargs*)
Yields the individual minion returns as they come in

The function signature is the same as `cmd()` with the following exceptions.

Returns A generator

```
>>> ret = local.cmd_iter('*', 'test.ping')
>>> for i in ret:
...     print i
{'jerry': {'ret': True}}
{'dave': {'ret': True}}
{'stewart': {'ret': True}}
```

cmd_iter_no_block(*tgt*, *fun*, *arg=()*, *timeout=None*, *expr_form='glob'*, *ret=''*, *kwarg=None*, ***kwargs*)
Blocks while waiting for individual minions to return.

The function signature is the same as `cmd()` with the following exceptions.

Returns None until the next minion returns. This allows for actions to be injected in between minion returns.

```
>>> ret = local.cmd_iter('*', 'test.ping')
>>> for i in ret:
...     print i
None
{'jerry': {'ret': True}}
{'dave': {'ret': True}}
None
{'stewart': {'ret': True}}
```

cmd_subset(*tgt*, *fun*, *arg=()*, *expr_form='glob'*, *ret=''*, *kwarg=None*, *sub=3*, *cli=False*, *progress=False*, ***kwargs*)
Execute a command on a random subset of the targeted systems

The function signature is the same as `cmd()` with the following exceptions.

Parameters **sub** -- The number of systems to execute on

```
>>> SLC.cmd_subset('*', 'test.ping', sub=1)
{'jerry': True}
```

get_cli_returns(*jid*, *minions*, *timeout=None*, *tgt='**', *tgt_type='glob'*, *verbose=False*, *show_jid=False*, ***kwargs*)
Starts a watcher looking at the return data for a specified JID

Returns all of the information for the JID

get_event_iter_returns(*jid*, *minions*, *timeout=None*)
Gather the return data from the event system, break hard when timeout is reached.

run_job(*tgt*, *fun*, *arg=()*, *expr_form='glob'*, *ret=''*, *timeout=None*, *jid=''*, *kwarg=None*, ***kwargs*)
Asynchronously send a command to connected minions

Prep the job directory and publish a command to any targeted minions.

Returns A dictionary of (validated) `pub_data` or an empty dictionary on failure. The `pub_data` contains the job ID and a list of all minions that are expected to return data.

```
>>> local.run_job('*', 'test.sleep', [300])
{'jid': '20131219215650131543', 'minions': ['jerry']}
```

Salt Caller

class `salt.client.`**Caller**(*c_path='/etc/salt/minion'*, *mopts=None*)
`Caller` is the same interface used by the **salt-call** command-line tool on the Salt Minion.

Importing and using `Caller` must be done on the same machine as a Salt Minion and it must be done using the same user that the Salt Minion is running as.

Usage:

```
import salt.client
caller = salt.client.Caller()
caller.function('test.ping')

# Or call objects directly
caller.sminion.functions['cmd.run']('ls -l')
```

Note, a running master or minion daemon is not required to use this class. Running `salt-call --local` simply sets `file_client` to `'local'`. The same can be achieved at the Python level by including that setting in a minion config file.

Instantiate a new Caller() instance using a file system path to the minion config file:

```
caller = salt.client.Caller('/path/to/custom/minion_config')
caller.sminion.functions['grains.items']()
```

Instantiate a new Caller() instance using a dictionary of the minion config:

New in version 2014.7.0: Pass the minion config as a dictionary.

```
import salt.client
import salt.config

opts = salt.config.minion_config('/etc/salt/minion')
opts['file_client'] = 'local'
caller = salt.client.Caller(mopts=opts)
caller.sminion.functions['grains.items']()
```

function(*fun*, **args*, ***kwargs*)
Call a single salt function

RunnerClient

class salt.runner.**RunnerClient**(*opts*)

The interface used by the **salt-run** CLI tool on the Salt Master

It executes *runner modules* which run on the Salt Master.

Importing and using `RunnerClient` must be done on the same machine as the Salt Master and it must be done using the same user that the Salt Master is running as.

Salt's `external_auth` can be used to authenticate calls. The eauth user must be authorized to execute runner modules: (`@runner`). Only the `master_call()` below supports eauth.

async(*fun*, *low*, *user='UNKNOWN'*)

Execute the function in a multiprocess and return the event tag to use to watch for the return

cmd(*fun*, *arg=None*, *pub_data=None*, *kwarg=None*)

Execute a function

```
>>> opts = salt.config.master_config('/etc/salt/master')
>>> runner = salt.runner.RunnerClient(opts)
>>> runner.cmd('jobs.list_jobs', [])
{
    '20131219215650131543': {
        'Arguments': [300],
        'Function': 'test.sleep',
        'StartTime': '2013, Dec 19 21:56:50.131543',
        'Target': '*',
        'Target-type': 'glob',
        'User': 'saltdev'
    },
    '20131219215921857715': {
        'Arguments': [300],
        'Function': 'test.sleep',
        'StartTime': '2013, Dec 19 21:59:21.857715',
        'Target': '*',
        'Target-type': 'glob',
        'User': 'saltdev'
    },
}
```

cmd_async(*low*)

Execute a runner function asynchronously; eauth is respected

This function requires that `external_auth` is configured and the user is authorized to execute runner functions: (`@runner`).

```
runner.eauth_async({
    'fun': 'jobs.list_jobs',
    'username': 'saltdev',
    'password': 'saltdev',
    'eauth': 'pam',
})
```

cmd_sync(*low*, *timeout=None*)

Execute a runner function synchronously; eauth is respected

This function requires that `external_auth` is configured and the user is authorized to execute runner functions: (`@runner`).

```
        runner.eauth_sync({
            'fun': 'jobs.list_jobs',
            'username': 'saltdev',
            'password': 'saltdev',
            'eauth': 'pam',
        })
```

WheelClient

class salt.wheel.**WheelClient**(*opts=None*)

An interface to Salt's wheel modules

Wheel modules interact with various parts of the Salt Master.

Importing and using `WheelClient` must be done on the same machine as the Salt Master and it must be done using the same user that the Salt Master is running as. Unless `external_auth` is configured and the user is authorized to execute wheel functions: (`@wheel`).

async(*fun*, *low*, *user='UNKNOWN'*)

Execute the function in a multiprocess and return the event tag to use to watch for the return

cmd(*fun*, *arg=None*, *pub_data=None*, *kwarg=None*)

Execute a function

```
        >>> opts = salt.config.master_config('/etc/salt/master')
        >>> runner = salt.runner.RunnerClient(opts)
        >>> runner.cmd('jobs.list_jobs', [])
        {
            '20131219215650131543': {
                'Arguments': [300],
                'Function': 'test.sleep',
                'StartTime': '2013, Dec 19 21:56:50.131543',
                'Target': '*',
                'Target-type': 'glob',
                'User': 'saltdev'
            },
            '20131219215921857715': {
                'Arguments': [300],
                'Function': 'test.sleep',
                'StartTime': '2013, Dec 19 21:59:21.857715',
                'Target': '*',
                'Target-type': 'glob',
                'User': 'saltdev'
            },
        }
```

cmd_async(*low*)

Execute a function asynchronously; eauth is respected

This function requires that `external_auth` is configured and the user is authorized

```
        >>> wheel.cmd_async({
            'fun': 'key.finger',
            'match': 'jerry',
            'eauth': 'auto',
            'username': 'saltdev',
            'password': 'saltdev',
```

```
        })
        {'jid': '20131219224744416681', 'tag': 'salt/wheel/20131219224744416681'}
```

cmd_sync(*low*, *timeout=None*)
Execute a wheel function synchronously; eauth is respected

This function requires that `external_auth` is configured and the user is authorized to execute runner functions: (`@wheel`).

```

```

```
        >>> wheel.cmd_sync({
        'fun': 'key.finger',
        'match': 'jerry',
        'eauth': 'auto',
        'username': 'saltdev',
        'password': 'saltdev',
        })
        {'minions': {'jerry': '5d:f6:79:43:5e:d4:42:3f:57:b8:45:a8:7e:a4:6e:ca'}}
```

CloudClient

class `salt.cloud.`**CloudClient**(*path=None*, *opts=None*, *config_dir=None*, *pillars=None*)
The client class to wrap cloud interactions

action(*fun=None*, *cloudmap=None*, *names=None*, *provider=None*, *instance=None*, *kwargs=None*)
Execute a single action via the cloud plugin backend

Examples:

```
        client.action(fun='show_instance', names=['myinstance'])
        client.action(fun='show_image', provider='my-ec2-config',
            kwargs={'image': 'ami-10314d79'}
        )
```

create(*provider*, *names*, ***kwargs*)
Create the named VMs, without using a profile

Example:

```
        client.create(names=['myinstance'], provider='my-ec2-config',
            kwargs={'image': 'ami-1624987f', 'size': 't1.micro',
                    'ssh_username': 'ec2-user', 'securitygroup': 'default',
                    'delvol_on_destroy': True})
```

destroy(*names*)
Destroy the named VMs

extra_action(*names*, *provider*, *action*, ***kwargs*)
Perform actions with block storage devices

Example:

```
        client.extra_action(names=['myblock'], action='volume_create',
            provider='my-nova', kwargs={'voltype': 'SSD', 'size': 1000}
        )
        client.extra_action(names=['salt-net'], action='network_create',
            provider='my-nova', kwargs={'cidr': '192.168.100.0/24'}
        )
```

full_query(*query_type='list_nodes_full'*)
Query all instance information

list_images(*provider=None*)
List all available images in configured cloud systems

list_locations(*provider=None*)
List all available locations in configured cloud systems

list_sizes(*provider=None*)
List all available sizes in configured cloud systems

low(*fun*, *low*)
Pass the cloud function and low data structure to run

map_run(*path*, ***kwargs*)
Pass in a location for a map to execute

min_query(*query_type='list_nodes_min'*)
Query select instance information

profile(*profile*, *names*, *vm_overrides=None*, ***kwargs*)
Pass in a profile to create, names is a list of vm names to allocate

vm_overrides is a special dict that will be per node options overrides

Example:

```
>>> client= salt.cloud.CloudClient(path='/etc/salt/cloud')
>>> client.profile('do_512_git', names=['minion01',])
{'minion01': {u'backups_active': 'False',
        u'created_at': '2014-09-04T18:10:15Z',
        u'droplet': {u'event_id': 31000502,
                     u'id': 2530006,
                     u'image_id': 5140006,
                     u'name': u'minion01',
                     u'size_id': 66},
        u'id': '2530006',
        u'image_id': '5140006',
        u'ip_address': '107.XXX.XXX.XXX',
        u'locked': 'True',
        u'name': 'minion01',
        u'private_ip_address': None,
        u'region_id': '4',
        u'size_id': '66',
        u'status': 'new'}}
```

query(*query_type='list_nodes'*)
Query basic instance information

select_query(*query_type='list_nodes_select'*)
Query select instance information

SSHClient

class salt.client.ssh.client.**SSHClient**(*c_path='/etc/salt/master'*, *mopts=None*)
Create a client object for executing routines via the salt-ssh backend

New in version 2015.2.

cmd(*tgt*, *fun*, *arg=()*, *timeout=None*, *expr_form='glob'*, *kwarg=None*, ***kwargs*)
Execute a single command via the salt-ssh subsystem and return all routines at once

New in version 2015.2.

cmd_iter(*tgt*, *fun*, *arg=()*, *timeout=None*, *expr_form='glob'*, *ret=''*, *kwarg=None*, ***kwargs*)
Execute a single command via the salt-ssh subsystem and return a generator

New in version 2015.2.

22.5 Full list of Salt Cloud modules

aliyun	AliYun ECS Cloud Module
botocore_aws	The AWS Cloud Module
cloudstack	CloudStack Cloud Module
digital_ocean	DigitalOcean Cloud Module
digital_ocean_v2	DigitalOcean Cloud Module v2
ec2	The EC2 Cloud Module
gce	Copyright 2013 Google Inc.
gogrid	GoGrid Cloud Module
joyent	
libcloud_aws	The AWS Cloud Module
linode	Linode Cloud Module using Apache Libcloud OR linode-python bindings
linodepy	
lxc	Install Salt on an LXC Container
msazure	Azure Cloud Module
nova	OpenStack Nova Cloud Module
opennebula	OpenNebula Cloud Module
openstack	OpenStack Cloud Module
parallels	Parallels Cloud Module
proxmox	Proxmox Cloud Module
pyrax	Pyrax Cloud Module
rackspace	Rackspace Cloud Module
saltify	Saltify Module ========== The Saltify module is designed to install Salt on a remote machine, virt
softlayer	SoftLayer Cloud Module
softlayer_hw	SoftLayer HW Cloud Module
vsphere	vSphere Cloud Module

22.5.1 salt.cloud.clouds.aliyun

AliYun ECS Cloud Module

New in version 2014.7.0.

The Aliyun cloud module is used to control access to the aliyun ECS. http://www.aliyun.com/

Use of this module requires the `id` and `key` parameter to be set. Set up the cloud configuration at `/etc/salt/cloud.providers` or `/etc/salt/cloud.providers.d/aliyun.conf`:

```
my-aliyun-config:
  # aliyun Access Key ID
  id: wFGEwgregeqw3435gDger
  # aliyun Access Key Secret
```

```
  key: GDE43t43REGTrkilg43934t34qT43t4dgegerGEgg
  location: cn-qingdao
  provider: aliyun
```

depends requests

salt.cloud.clouds.aliyun.**avail_images**(*kwargs=None*, *call=None*)
Return a list of the images that are on the provider

salt.cloud.clouds.aliyun.**avail_locations**(*call=None*)
Return a dict of all available VM locations on the cloud provider with relevant data

salt.cloud.clouds.aliyun.**avail_sizes**(*call=None*)
Return a list of the image sizes that are on the provider

salt.cloud.clouds.aliyun.**create**(*vm_*)
Create a single VM from a data dict

salt.cloud.clouds.aliyun.**create_node**(*kwargs*)
Convenience function to make the rest api call for node creation.

salt.cloud.clouds.aliyun.**destroy**(*name*, *call=None*)
Destroy a node.

CLI Example:

```
    salt-cloud -a destroy myinstance
    salt-cloud -d myinstance
```

salt.cloud.clouds.aliyun.**get_configured_provider**()
Return the first configured instance.

salt.cloud.clouds.aliyun.**get_image**(*vm_*)
Return the image object to use

salt.cloud.clouds.aliyun.**get_location**(*vm_=None*)

Return the aliyun region to use, in this order:

- CLI parameter
- VM parameter
- Cloud profile setting

salt.cloud.clouds.aliyun.**get_securitygroup**(*vm_*)
Return the security group

salt.cloud.clouds.aliyun.**get_size**(*vm_*)
Return the VM's size. Used by create_node().

salt.cloud.clouds.aliyun.**list_availability_zones**(*call=None*)
List all availability zones in the current region

salt.cloud.clouds.aliyun.**list_monitor_data**(*kwargs=None*, *call=None*)
Get monitor data of the instance. If instance name is missing, will show all the instance monitor data on the region.

CLI Examples:

```
    salt-cloud -f list_monitor_data aliyun
    salt-cloud -f list_monitor_data aliyun name=AY14051311071990225bd
```

salt.cloud.clouds.aliyun.**list_nodes**(*call=None*)
Return a list of the VMs that are on the provider

salt.cloud.clouds.aliyun.**list_nodes_full**(*call=None*)
Return a list of the VMs that are on the provider

salt.cloud.clouds.aliyun.**list_nodes_min**(*call=None*)
Return a list of the VMs that are on the provider. Only a list of VM names, and their state, is returned. This is the minimum amount of information needed to check for existing VMs.

salt.cloud.clouds.aliyun.**list_nodes_select**(*call=None*)
Return a list of the VMs that are on the provider, with select fields

salt.cloud.clouds.aliyun.**list_securitygroup**(*call=None*)
Return a list of security group

salt.cloud.clouds.aliyun.**query**(*params=None*)
Make a web call to aliyun ECS REST API

salt.cloud.clouds.aliyun.**reboot**(*name*, *call=None*)
Reboot a node

CLI Examples:

```
salt-cloud -a reboot myinstance
```

salt.cloud.clouds.aliyun.**script**(*vm_*)
Return the script deployment object

salt.cloud.clouds.aliyun.**show_disk**(*name*, *call=None*)
Show the disk details of the instance

CLI Examples:

```
salt-cloud -a show_disk aliyun myinstance
```

salt.cloud.clouds.aliyun.**show_image**(*kwargs*, *call=None*)
Show the details from aliyun image

salt.cloud.clouds.aliyun.**show_instance**(*name*, *call=None*)
Show the details from aliyun instance

salt.cloud.clouds.aliyun.**start**(*name*, *call=None*)
Start a node

CLI Examples:

```
salt-cloud -a start myinstance
```

salt.cloud.clouds.aliyun.**stop**(*name*, *force=False*, *call=None*)
Stop a node

CLI Examples:

```
salt-cloud -a stop myinstance
salt-cloud -a stop myinstance force=True
```

22.5.2 salt.cloud.clouds.botocore_aws

The AWS Cloud Module

The AWS cloud module is used to interact with the Amazon Web Services system.

This module has been replaced by the EC2 cloud module, and is no longer supported. The documentation shown here is for reference only; it is highly recommended to change all usages of this driver over to the EC2 driver.

If this driver is still needed, set up the cloud configuration at `/etc/salt/cloud.providers` or `/etc/salt/cloud.providers.d/aws.conf`:

```
my-aws-botocore-config:
  # The AWS API authentication id
  id: GKTADJGHEIQSXMKKRBJ08H
  # The AWS API authentication key
  key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs
  # The ssh keyname to use
  keyname: default
  # The amazon security group
  securitygroup: ssh_open
  # The location of the private key which corresponds to the keyname
  private_key: /root/default.pem
  provider: aws
```

`salt.cloud.clouds.botocore_aws.`**`disable_term_protect`**`(`*name*, *call=None*`)`

Disable termination protection on a node

CLI Example:

```
salt-cloud -a disable_term_protect mymachine
```

`salt.cloud.clouds.botocore_aws.`**`enable_term_protect`**`(`*name*, *call=None*`)`

Enable termination protection on a node

CLI Example:

```
salt-cloud -a enable_term_protect mymachine
```

`salt.cloud.clouds.botocore_aws.`**`get_configured_provider`**`()`

Return the first configured instance.

22.5.3 salt.cloud.clouds.cloudstack

CloudStack Cloud Module

The CloudStack cloud module is used to control access to a CloudStack based Public Cloud.

depends libcloud

Use of this module requires the `apikey`, `secretkey`, `host` and `path` parameters.

```
my-cloudstack-cloud-config:
  apikey: <your api key >
  secretkey: <your secret key >
  host: localhost
  path: /client/api
  provider: cloudstack
```

salt.cloud.clouds.cloudstack.**avail_images**(*conn=None*, *call=None*)
Return a dict of all available VM images on the cloud provider with relevant data

salt.cloud.clouds.cloudstack.**avail_locations**(*conn=None*, *call=None*)
Return a dict of all available VM locations on the cloud provider with relevant data

salt.cloud.clouds.cloudstack.**avail_sizes**(*conn=None*, *call=None*)
Return a dict of all available VM images on the cloud provider with relevant data

salt.cloud.clouds.cloudstack.**block_device_mappings**(*vm_*)
Return the block device mapping:

```
[{'DeviceName': '/dev/sdb', 'VirtualName': 'ephemeral0'},
  {'DeviceName': '/dev/sdc', 'VirtualName': 'ephemeral1'}]
```

salt.cloud.clouds.cloudstack.**create**(*vm_*)
Create a single VM from a data dict

salt.cloud.clouds.cloudstack.**destroy**(*name*, *conn=None*, *call=None*)
Delete a single VM, and all of its volumes

salt.cloud.clouds.cloudstack.**get_configured_provider**()
Return the first configured instance.

salt.cloud.clouds.cloudstack.**get_conn**()
Return a conn object for the passed VM data

salt.cloud.clouds.cloudstack.**get_image**(*conn*, *vm_*)
Return the image object to use

salt.cloud.clouds.cloudstack.**get_ip**(*data*)
Return the IP address of the VM If the VM has public IP as defined by libcloud module then use it Otherwise try to extract the private IP and use that one.

salt.cloud.clouds.cloudstack.**get_key**()
Returns the ssh private key for VM access

salt.cloud.clouds.cloudstack.**get_keypair**(*vm_*)
Return the keypair to use

salt.cloud.clouds.cloudstack.**get_location**(*conn*, *vm_*)
Return the node location to use

salt.cloud.clouds.cloudstack.**get_networkid**(*vm_*)
Return the networkid to use, only valid for Advanced Zone

salt.cloud.clouds.cloudstack.**get_node**(*conn*, *name*)
Return a libcloud node for the named VM

salt.cloud.clouds.cloudstack.**get_password**(*vm_*)
Return the password to use

salt.cloud.clouds.cloudstack.**get_project**(*conn*, *vm_*)
Return the project to use.

salt.cloud.clouds.cloudstack.**get_size**(*conn*, *vm_*)
Return the VM's size object

salt.cloud.clouds.cloudstack.**list_nodes**(*conn=None*, *call=None*)
Return a list of the VMs that are on the provider

salt.cloud.clouds.cloudstack.**list_nodes_full**(*conn=None*, *call=None*)
Return a list of the VMs that are on the provider, with all fields

salt.cloud.clouds.cloudstack.**list_nodes_select**(*conn=None*, *call=None*)
Return a list of the VMs that are on the provider, with select fields

salt.cloud.clouds.cloudstack.**script**(*vm_*)
Return the script deployment object

salt.cloud.clouds.cloudstack.**show_instance**(*name*, *call=None*)
Show the details from the provider concerning an instance

22.5.4 salt.cloud.clouds.digital_ocean

DigitalOcean Cloud Module

The DigitalOcean cloud module is used to control access to the DigitalOcean VPS system.

Use of this module only requires the `api_key` parameter to be set. Set up the cloud configuration at `/etc/salt/cloud.providers` or `/etc/salt/cloud.providers.d/digital_ocean.conf`:

```
my-digital-ocean-config:
  # DigitalOcean account keys
  client_key: wFGEwgregeqw3435gDger
  api_key: GDE43t43REGTrkilg43934t34qT43t4dgegerGEgg
  provider: digital_ocean
```

depends requests

salt.cloud.clouds.digital_ocean.**avail_images**(*call=None*)
Return a list of the images that are on the provider

salt.cloud.clouds.digital_ocean.**avail_locations**(*call=None*)
Return a dict of all available VM locations on the cloud provider with relevant data

salt.cloud.clouds.digital_ocean.**avail_sizes**(*call=None*)
Return a list of the image sizes that are on the provider

salt.cloud.clouds.digital_ocean.**create**(*vm_*)
Create a single VM from a data dict

salt.cloud.clouds.digital_ocean.**create_node**(*args*)
Create a node

salt.cloud.clouds.digital_ocean.**destroy**(*name*, *call=None*)
Destroy a node. Will check termination protection and warn if enabled.

CLI Example:

```
salt-cloud --destroy mymachine
```

salt.cloud.clouds.digital_ocean.**get_configured_provider**()
Return the first configured instance.

salt.cloud.clouds.digital_ocean.**get_image**(*vm_*)
Return the image object to use

salt.cloud.clouds.digital_ocean.**get_keyid**(*keyname*)
Return the ID of the keyname

salt.cloud.clouds.digital_ocean.**get_location**(*vm_*)
Return the VM's location

salt.cloud.clouds.digital_ocean.**get_size**(*vm_*)
Return the VM's size. Used by create_node().

salt.cloud.clouds.digital_ocean.**list_keypairs**(*call=None*)
Return a dict of all available VM locations on the cloud provider with relevant data

salt.cloud.clouds.digital_ocean.**list_nodes**(*call=None*)
Return a list of the VMs that are on the provider

salt.cloud.clouds.digital_ocean.**list_nodes_full**(*call=None*)
Return a list of the VMs that are on the provider

salt.cloud.clouds.digital_ocean.**list_nodes_select**(*call=None*)
Return a list of the VMs that are on the provider, with select fields

salt.cloud.clouds.digital_ocean.**query**(*method='droplets'*, *droplet_id=None*, *command=None*, *args=None*)
Make a web call to DigitalOcean

salt.cloud.clouds.digital_ocean.**script**(*vm_*)
Return the script deployment object

salt.cloud.clouds.digital_ocean.**show_instance**(*name*, *call=None*)
Show the details from DigitalOcean concerning a droplet

salt.cloud.clouds.digital_ocean.**show_keypair**(*kwargs=None*, *call=None*)
Show the details of an SSH keypair

22.5.5 salt.cloud.clouds.digital_ocean_v2

DigitalOcean Cloud Module v2

The DigitalOcean cloud module is used to control access to the DigitalOcean VPS system.

Use of this module only requires the `personal_access_token` parameter to be set. Set up the cloud configuration at `/etc/salt/cloud.providers` or `/etc/salt/cloud.providers.d/digital_ocean.conf`:

```
my-digital-ocean-config:
  personal_access_token: xxx
  provider: digital_ocean
```

depends requests

salt.cloud.clouds.digital_ocean_v2.**avail_images**(*call=None*)
Return a list of the images that are on the provider

salt.cloud.clouds.digital_ocean_v2.**avail_locations**(*call=None*)
Return a dict of all available VM locations on the cloud provider with relevant data

salt.cloud.clouds.digital_ocean_v2.**avail_sizes**(*call=None*)
Return a list of the image sizes that are on the provider

salt.cloud.clouds.digital_ocean_v2.**create**(*vm_*)
Create a single VM from a data dict

salt.cloud.clouds.digital_ocean_v2.**create_dns_record**(*hostname*, *ip_address*)
Creates a DNS record for the given hostname if the domain is managed with DO.

salt.cloud.clouds.digital_ocean_v2.**create_node**(*args*)
Create a node

salt.cloud.clouds.digital_ocean_v2.**delete_dns_record**(*hostname*)
Deletes a DNS for the given hostname if the domain is managed with DO.

salt.cloud.clouds.digital_ocean_v2.**destroy**(*name*, *call=None*)
Destroy a node. Will check termination protection and warn if enabled.

CLI Example:

```
salt-cloud --destroy mymachine
```

salt.cloud.clouds.digital_ocean_v2.**get_configured_provider**()
Return the first configured instance.

salt.cloud.clouds.digital_ocean_v2.**get_image**(*vm_*)
Return the image object to use

salt.cloud.clouds.digital_ocean_v2.**get_keyid**(*keyname*)
Return the ID of the keyname

salt.cloud.clouds.digital_ocean_v2.**get_location**(*vm_*)
Return the VM's location

salt.cloud.clouds.digital_ocean_v2.**get_size**(*vm_*)
Return the VM's size. Used by create_node().

salt.cloud.clouds.digital_ocean_v2.**list_keypairs**(*call=None*)
Return a dict of all available VM locations on the cloud provider with relevant data

salt.cloud.clouds.digital_ocean_v2.**list_nodes**(*call=None*)
Return a list of the VMs that are on the provider

salt.cloud.clouds.digital_ocean_v2.**list_nodes_full**(*call=None*, *forOutput=True*)
Return a list of the VMs that are on the provider

salt.cloud.clouds.digital_ocean_v2.**list_nodes_select**(*call=None*)
Return a list of the VMs that are on the provider, with select fields

salt.cloud.clouds.digital_ocean_v2.**query**(*method='droplets'*, *droplet_id=None*, *command=None*, *args=None*, *http_method='get'*)
Make a web call to DigitalOcean

salt.cloud.clouds.digital_ocean_v2.**script**(*vm_*)
Return the script deployment object

salt.cloud.clouds.digital_ocean_v2.**show_instance**(*name*, *call=None*)
Show the details from DigitalOcean concerning a droplet

salt.cloud.clouds.digital_ocean_v2.**show_keypair**(*kwargs=None*, *call=None*)
Show the details of an SSH keypair

22.5.6 salt.cloud.clouds.ec2

The EC2 Cloud Module

The EC2 cloud module is used to interact with the Amazon Elastic Cloud Computing.

To use the EC2 cloud module, set up the cloud configuration at `/etc/salt/cloud.providers` or `/etc/salt/cloud.providers.d/ec2.conf`:

```
my-ec2-config:
  # The EC2 API authentication id, set this and/or key to
  # 'use-instance-role-credentials' to use the instance role credentials
  # from the meta-data if running on an AWS instance
  id: GKTADJGHEIQSXMKKRBJ08H
  # The EC2 API authentication key, set this and/or id to
  # 'use-instance-role-credentials' to use the instance role credentials
  # from the meta-data if running on an AWS instance
  key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs
  # The ssh keyname to use
  keyname: default
  # The amazon security group
  securitygroup: ssh_open
  # The location of the private key which corresponds to the keyname
  private_key: /root/default.pem

  # Be default, service_url is set to amazonaws.com. If you are using this
  # driver for something other than Amazon EC2, change it here:
  service_url: amazonaws.com

  # The endpoint that is ultimately used is usually formed using the region
  # and the service_url. If you would like to override that entirely, you
  # can explicitly define the endpoint:
  endpoint: myendpoint.example.com:1138/services/Cloud

  # SSH Gateways can be used with this provider. Gateways can be used
  # when a salt-master is not on the same private network as the instance
  # that is being deployed.

  # Defaults to None
  # Required
  ssh_gateway: gateway.example.com

  # Defaults to port 22
  # Optional
  ssh_gateway_port: 22

  # Defaults to root
  # Optional
  ssh_gateway_username: root

  # One authentication method is required. If both
  # are specified, Private key wins.

  # Private key defaults to None
  ssh_gateway_private_key: /path/to/key.pem

  # Password defaults to None
  ssh_gateway_password: ExamplePasswordHere

  provider: ec2
```

depends requests

`salt.cloud.clouds.ec2.`**`attach_volume`**(*name=None, kwargs=None, instance_id=None, call=None*)

Attach a volume to an instance

`salt.cloud.clouds.ec2.`**`avail_images`**(*kwargs=None, call=None*)

Return a dict of all available VM images on the cloud provider.

salt.cloud.clouds.ec2.**avail_locations**(*call=None*)
List all available locations

salt.cloud.clouds.ec2.**avail_sizes**(*call=None*)
Return a dict of all available VM sizes on the cloud provider with relevant data. Latest version can be found at:

http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/instance-types.html

salt.cloud.clouds.ec2.**block_device_mappings**(*vm_*)
Return the block device mapping:

```
[{'DeviceName': '/dev/sdb', 'VirtualName': 'ephemeral0'},
  {'DeviceName': '/dev/sdc', 'VirtualName': 'ephemeral1'}]
```

salt.cloud.clouds.ec2.**copy_snapshot**(*kwargs=None*, *call=None*)
Copy a snapshot

salt.cloud.clouds.ec2.**create**(*vm_=None*, *call=None*)
Create a single VM from a data dict

salt.cloud.clouds.ec2.**create_attach_volumes**(*name*, *kwargs*, *call=None*, *wait_to_finish=True*)
Create and attach volumes to created node

salt.cloud.clouds.ec2.**create_keypair**(*kwargs=None*, *call=None*)
Create an SSH keypair

salt.cloud.clouds.ec2.**create_snapshot**(*kwargs=None*, *call=None*, *wait_to_finish=False*)
Create a snapshot

salt.cloud.clouds.ec2.**create_volume**(*kwargs=None*, *call=None*, *wait_to_finish=False*)
Create a volume

CLI Examples:

```
salt-cloud -f create_volume my-ec2-config zone=us-east-1b
salt-cloud -f create_volume my-ec2-config zone=us-east-1b tags='{"tag1": "val1", "tag2", "val2"}
```

salt.cloud.clouds.ec2.**del_tags**(*name=None*, *kwargs=None*, *call=None*, *instance_id=None*, *resource_id=None*)
Delete tags for a resource. Normally a VM name or instance_id is passed in, but a resource_id may be passed instead. If both are passed in, the instance_id will be used.

CLI Examples:

```
salt-cloud -a del_tags mymachine tags=mytag,
salt-cloud -a del_tags mymachine tags=tag1,tag2,tag3
salt-cloud -a del_tags resource_id=vol-3267ab32 tags=tag1,tag2,tag3
```

salt.cloud.clouds.ec2.**delete_keypair**(*kwargs=None*, *call=None*)
Delete an SSH keypair

salt.cloud.clouds.ec2.**delete_snapshot**(*kwargs=None*, *call=None*)
Delete a snapshot

salt.cloud.clouds.ec2.**delete_volume**(*name=None*, *kwargs=None*, *instance_id=None*, *call=None*)
Delete a volume

salt.cloud.clouds.ec2.**delvol_on_destroy**(*name*, *kwargs=None*, *call=None*)
Delete all/specified EBS volumes upon instance termination

CLI Example:

```
salt-cloud -a delvol_on_destroy mymachine
```

salt.cloud.clouds.ec2.**describe_snapshots**(*kwargs=None*, *call=None*)
Describe a snapshot (or snapshots)

snapshot_id One or more snapshot IDs. Multiple IDs must be separated by ",".

owner Return the snapshots owned by the specified owner. Valid values include: self, amazon, <AWS Account ID>. Multiple values must be separated by ",".

restorable_by One or more AWS accounts IDs that can create volumes from the snapshot. Multiple aws account IDs must be separated by ",".

TODO: Add all of the filters.

salt.cloud.clouds.ec2.**describe_volumes**(*kwargs=None*, *call=None*)
Describe a volume (or volumes)

volume_id One or more volume IDs. Multiple IDs must be separated by ",".

TODO: Add all of the filters.

salt.cloud.clouds.ec2.**destroy**(*name*, *call=None*)
Destroy a node. Will check termination protection and warn if enabled.

CLI Example:

```
salt-cloud --destroy mymachine
```

salt.cloud.clouds.ec2.**detach_volume**(*name=None*, *kwargs=None*, *instance_id=None*, *call=None*)
Detach a volume from an instance

salt.cloud.clouds.ec2.**disable_term_protect**(*name*, *call=None*)
Disable termination protection on a node

CLI Example:

```
salt-cloud -a disable_term_protect mymachine
```

salt.cloud.clouds.ec2.**enable_term_protect**(*name*, *call=None*)
Enable termination protection on a node

CLI Example:

```
salt-cloud -a enable_term_protect mymachine
```

salt.cloud.clouds.ec2.**get_availability_zone**(*vm_*)
Return the availability zone to use

salt.cloud.clouds.ec2.**get_configured_provider**()
Return the first configured instance.

salt.cloud.clouds.ec2.**get_console_output**(*name=None*, *instance_id=None*, *call=None*, *kwargs=None*)
Show the console output from the instance.

By default, returns decoded data, not the Base64-encoded data that is actually returned from the EC2 API.

salt.cloud.clouds.ec2.**get_location**(*vm_=None*)

Return the EC2 region to use, in this order:

- CLI parameter
- VM parameter
- Cloud profile setting

salt.cloud.clouds.ec2.**get_password_data**(*name=None*, *kwargs=None*, *instance_id=None*, *call=None*)

Return password data for a Windows instance.

By default only the encrypted password data will be returned. However, if a key_file is passed in, then a decrypted password will also be returned.

Note that the key_file references the private key that was used to generate the keypair associated with this instance. This private key will _not_ be transmitted to Amazon; it is only used internally inside of Salt Cloud to decrypt data _after_ it has been received from Amazon.

CLI Examples:

```
salt-cloud -a get_password_data mymachine
salt-cloud -a get_password_data mymachine key_file=/root/ec2key.pem
```

Note: PKCS1_v1_5 was added in PyCrypto 2.5

salt.cloud.clouds.ec2.**get_placementgroup**(*vm_*)

Returns the PlacementGroup to use

salt.cloud.clouds.ec2.**get_provider**(*vm_=None*)

Extract the provider name from vm

salt.cloud.clouds.ec2.**get_salt_interface**(*vm_*)

Return the salt_interface type to connect to. Either `public_ips' (default) or `private_ips'.

salt.cloud.clouds.ec2.**get_spot_config**(*vm_*)

Returns the spot instance configuration for the provided vm

salt.cloud.clouds.ec2.**get_ssh_gateway_config**(*vm_*)

Return the ssh_gateway configuration.

salt.cloud.clouds.ec2.**get_subnetid**(*vm_*)

Returns the SubnetId to use

salt.cloud.clouds.ec2.**get_tags**(*name=None*, *instance_id=None*, *call=None*, *location=None*, *kwargs=None*, *resource_id=None*)

Retrieve tags for a resource. Normally a VM name or instance_id is passed in, but a resource_id may be passed instead. If both are passed in, the instance_id will be used.

CLI Examples:

```
salt-cloud -a get_tags mymachine
salt-cloud -a get_tags resource_id=vol-3267ab32
```

salt.cloud.clouds.ec2.**get_tenancy**(*vm_*)

Returns the Tenancy to use.

Can be ``dedicated'' or ``default''. Cannot be present for spot instances.

salt.cloud.clouds.ec2.**iam_profile**(*vm_*)

Return the IAM profile.

The IAM instance profile to associate with the instances. This is either the Amazon Resource Name (ARN) of the instance profile or the name of the role.

Type: String

Default: None

Required: No

Example: arn:aws:iam::111111111111:instance-profile/s3access

Example: s3access

salt.cloud.clouds.ec2.**keepvol_on_destroy**(*name*, *kwargs=None*, *call=None*)
Do not delete all/specified EBS volumes upon instance termination

CLI Example:

```
salt-cloud -a keepvol_on_destroy mymachine
```

salt.cloud.clouds.ec2.**keyname**(*vm_*)
Return the keyname

salt.cloud.clouds.ec2.**list_nodes**(*call=None*)
Return a list of the VMs that are on the provider

salt.cloud.clouds.ec2.**list_nodes_full**(*location=None*, *call=None*)
Return a list of the VMs that are on the provider

salt.cloud.clouds.ec2.**list_nodes_min**(*location=None*, *call=None*)
Return a list of the VMs that are on the provider. Only a list of VM names, and their state, is returned. This is the minimum amount of information needed to check for existing VMs.

salt.cloud.clouds.ec2.**list_nodes_select**(*call=None*)
Return a list of the VMs that are on the provider, with select fields

salt.cloud.clouds.ec2.**optimize_providers**(*providers*)
Return an optimized list of providers.

We want to reduce the duplication of querying the same region.

If a provider is using the same credentials for the same region the same data will be returned for each provider, thus causing un-wanted duplicate data and API calls to EC2.

salt.cloud.clouds.ec2.**query**(*params=None*, *setname=None*, *requesturl=None*, *location=None*, *return_url=False*, *return_root=False*)

salt.cloud.clouds.ec2.**query_instance**(*vm_=None*, *call=None*)
Query an instance upon creation from the EC2 API

salt.cloud.clouds.ec2.**queue_instances**(*instances*)
Queue a set of instances to be provisioned later. Expects a list.

Currently this only queries node data, and then places it in the cloud cache (if configured). If the salt-cloud-reactor is being used, these instances will be automatically provisioned using that.

For more information about the salt-cloud-reactor, see:

https://github.com/saltstack-formulas/salt-cloud-reactor

salt.cloud.clouds.ec2.**reboot**(*name*, *call=None*)
Reboot a node.

CLI Example:

```
salt-cloud -a reboot mymachine
```

salt.cloud.clouds.ec2.**rename**(*name*, *kwargs*, *call=None*)
Properly rename a node. Pass in the new name as ``new name''.

CLI Example:

```
salt-cloud -a rename mymachine newname=yourmachine
```

salt.cloud.clouds.ec2.**request_instance**(*vm_=None*, *call=None*)
Put together all of the information necessary to request an instance on EC2, and then fire off the request the instance.

Returns data about the instance

salt.cloud.clouds.ec2.**script**(*vm_*)
Return the script deployment object

salt.cloud.clouds.ec2.**securitygroup**(*vm_*)
Return the security group

salt.cloud.clouds.ec2.**securitygroupid**(*vm_*)
Returns the SecurityGroupId

salt.cloud.clouds.ec2.**set_tags**(*name=None*, *tags=None*, *call=None*, *location=None*, *instance_id=None*, *resource_id=None*, *kwargs=None*)
Set tags for a resource. Normally a VM name or instance_id is passed in, but a resource_id may be passed instead. If both are passed in, the instance_id will be used.

CLI Examples:

```
salt-cloud -a set_tags mymachine tag1=somestuff tag2='Other stuff'
salt-cloud -a set_tags resource_id=vol-3267ab32 tag=somestuff
```

salt.cloud.clouds.ec2.**show_delvol_on_destroy**(*name*, *kwargs=None*, *call=None*)
Do not delete all/specified EBS volumes upon instance termination

CLI Example:

```
salt-cloud -a show_delvol_on_destroy mymachine
```

salt.cloud.clouds.ec2.**show_image**(*kwargs*, *call=None*)
Show the details from EC2 concerning an AMI

salt.cloud.clouds.ec2.**show_instance**(*name=None*, *instance_id=None*, *call=None*, *kwargs=None*)
Show the details from EC2 concerning an AMI.

Can be called as an action (which requires a name):

```
salt-cloud -a show_instance myinstance
```

...or as a function (which requires either a name or instance_id):

```
salt-cloud -f show_instance my-ec2 name=myinstance
salt-cloud -f show_instance my-ec2 instance_id=i-d34db33f
```

salt.cloud.clouds.ec2.**show_keypair**(*kwargs=None*, *call=None*)
Show the details of an SSH keypair

salt.cloud.clouds.ec2.**show_term_protect**(*name=None*, *instance_id=None*, *call=None*, *quiet=False*)
Show the details from EC2 concerning an AMI

salt.cloud.clouds.ec2.**show_volume**(*kwargs=None*, *call=None*)
Wrapper around describe_volumes. Here just to keep functionality. Might be depreciated later.

salt.cloud.clouds.ec2.**ssh_interface**(*vm_*)
Return the ssh_interface type to connect to. Either `public_ips' (default) or `private_ips'.

salt.cloud.clouds.ec2.**start**(*name*, *call=None*)
Start a node

salt.cloud.clouds.ec2.**stop**(*name*, *call=None*)
Stop a node

salt.cloud.clouds.ec2.**wait_for_instance**(*vm_=None*, *data=None*, *ip_address=None*, *display_ssh_output=True*, *call=None*)
Wait for an instance upon creation from the EC2 API, to become available

22.5.7 salt.cloud.clouds.gce

Copyright 2013 Google Inc. All Rights Reserved.

Licensed under the Apache License, Version 2.0 (the ``License''); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an ``AS IS'' BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Google Compute Engine Module

The Google Compute Engine module. This module interfaces with Google Compute Engine. To authenticate to GCE, you will need to create a Service Account.

Setting up Service Account Authentication:

- Go to the Cloud Console at: https://cloud.google.com/console.
- Create or navigate to your desired Project.
- Make sure Google Compute Engine service is enabled under the Services section.
- Go to ``APIs and auth'' and then the ``Registered apps'' section.
- Click the ``REGISTER APP'' button and give it a meaningful name.
- Select ``Web Application'' and click ``Register''.
- Select Certificate, then ``Generate Certificate''
- Copy the Email Address for inclusion in your /etc/salt/cloud file in the `service_account_email_address' setting.
- Download the Private Key
- The key that you download is a PKCS12 key. It needs to be converted to the PEM format.
- Convert the key using OpenSSL (the default password is `notasecret'): C{openssl pkcs12 -in PRIVKEY.p12 -passin pass:notasecret -nodes -nocerts | openssl rsa -out ~/PRIVKEY.pem}
- Add the full path name of the converted private key to your /etc/salt/cloud file as `service_account_private_key' setting.

- Consider using a more secure location for your private key.

```
my-gce-config:
  # The Google Cloud Platform Project ID
  project: google.com:erjohnso
  # The Service ACcount client ID
  service_account_email_address: 1234567890@developer.gserviceaccount.com
  # The location of the private key (PEM format)
  service_account_private_key: /home/erjohnso/PRIVKEY.pem
  provider: gce
  # Specify whether to use public or private IP for deploy script.
  # Valid options are:
  #     private_ips - The salt-master is also hosted with GCE
  #     public_ips - The salt-master is hosted outside of GCE
  ssh_interface: public_ips
```

maintainer Eric Johnson <erjohnso@google.com>

maturity new

depends libcloud >= 0.14.1

depends pycrypto >= 2.1

`salt.cloud.clouds.gce.`**`attach_disk`**(*name=None*, *kwargs=None*, *call=None*)
Attach an existing disk to an existing instance.

CLI Example:

```
salt-cloud -a attach_disk myinstance disk_name=mydisk mode=READ_WRITE
```

`salt.cloud.clouds.gce.`**`attach_lb`**(*kwargs=None*, *call=None*)
Add an existing node/member to an existing load-balancer configuration.

CLI Example:

```
salt-cloud -f attach_lb gce name=lb member=myinstance
```

`salt.cloud.clouds.gce.`**`avail_images`**(*conn=None*)
Return a dict of all available VM images on the cloud provider with relevant data

Note that for GCE, there are custom images within the project, but the generic images are in other projects. This returns a dict of images in the project plus images in `debian-cloud' and `centos-cloud' (If there is overlap in names, the one in the current project is used.)

`salt.cloud.clouds.gce.`**`avail_locations`**(*conn=None*, *call=None*)
Return a dict of all available VM locations on the cloud provider with relevant data

`salt.cloud.clouds.gce.`**`avail_sizes`**(*conn=None*)
Return a dict of available instances sizes (a.k.a machine types) and convert them to something more serializable.

`salt.cloud.clouds.gce.`**`create`**(*vm_=None*, *call=None*)
Create a single GCE instance from a data dict.

`salt.cloud.clouds.gce.`**`create_address`**(*kwargs=None*, *call=None*)
Create a static address in a region.

CLI Example:

```
salt-cloud -f create_address gce name=my-ip region=us-central1 address=IP
```

salt.cloud.clouds.gce.**create_disk**(*kwargs=None*, *call=None*)
Create a new persistent disk. Must specify *disk_name* and *location*. Can also specify an *image* or *snapshot* but if neither of those are specified, a *size* (in GB) is required.

CLI Example:

```
salt-cloud -f create_disk gce disk_name=pd size=300 location=us-central1-b
```

salt.cloud.clouds.gce.**create_fwrule**(*kwargs=None*, *call=None*)
Create a GCE firewall rule. The `default' network is used if not specified.

CLI Example:

```
salt-cloud -f create_fwrule gce name=allow-http allow=tcp:80
```

salt.cloud.clouds.gce.**create_hc**(*kwargs=None*, *call=None*)
Create an HTTP health check configuration.

CLI Example:

```
salt-cloud -f create_hc gce name=hc path=/healthy port=80
```

salt.cloud.clouds.gce.**create_lb**(*kwargs=None*, *call=None*)
Create a load-balancer configuration.

CLI Example:

```
salt-cloud -f create_lb gce name=lb region=us-central1 ports=80
```

salt.cloud.clouds.gce.**create_network**(*kwargs=None*, *call=None*)
Create a GCE network.

CLI Example:

```
salt-cloud -f create_network gce name=mynet cidr=10.10.10.0/24
```

salt.cloud.clouds.gce.**create_snapshot**(*kwargs=None*, *call=None*)
Create a new disk snapshot. Must specify *name* and *disk_name*.

CLI Example:

```
salt-cloud -f create_snapshot gce name=snap1 disk_name=pd
```

salt.cloud.clouds.gce.**delete_address**(*kwargs=None*, *call=None*)
Permanently delete a static address.

CLI Example:

```
salt-cloud -f delete_address gce name=my-ip
```

salt.cloud.clouds.gce.**delete_disk**(*kwargs=None*, *call=None*)
Permanently delete a persistent disk.

CLI Example:

```
salt-cloud -f delete_disk gce disk_name=pd
```

salt.cloud.clouds.gce.**delete_fwrule**(*kwargs=None*, *call=None*)
Permanently delete a firewall rule.

CLI Example:

```
salt-cloud -f delete_fwrule gce name=allow-http
```

salt.cloud.clouds.gce.**delete_hc**(*kwargs=None*, *call=None*)
Permanently delete a health check.

CLI Example:

```
salt-cloud -f delete_hc gce name=hc
```

salt.cloud.clouds.gce.**delete_lb**(*kwargs=None*, *call=None*)
Permanently delete a load-balancer.

CLI Example:

```
salt-cloud -f delete_lb gce name=lb
```

salt.cloud.clouds.gce.**delete_network**(*kwargs=None*, *call=None*)
Permanently delete a network.

CLI Example:

```
salt-cloud -f delete_network gce name=mynet
```

salt.cloud.clouds.gce.**delete_snapshot**(*kwargs=None*, *call=None*)
Permanently delete a disk snapshot.

CLI Example:

```
salt-cloud -f delete_snapshot gce name=disk-snap-1
```

salt.cloud.clouds.gce.**destroy**(*vm_name*, *call=None*)
Call `destroy' on the instance. Can be called with ``-a destroy'' or -d

CLI Example:

```
salt-cloud -a destroy myinstance1 myinstance2 ...
salt-cloud -d myinstance1 myinstance2 ...
```

salt.cloud.clouds.gce.**detach_disk**(*name=None*, *kwargs=None*, *call=None*)
Detach a disk from an instance.

CLI Example:

```
salt-cloud -a detach_disk myinstance disk_name=mydisk
```

salt.cloud.clouds.gce.**detach_lb**(*kwargs=None*, *call=None*)
Remove an existing node/member from an existing load-balancer configuration.

CLI Example:

```
salt-cloud -f detach_lb gce name=lb member=myinstance
```

salt.cloud.clouds.gce.**get_configured_provider**()
Return the first configured instance.

salt.cloud.clouds.gce.**get_conn**()
Return a conn object for the passed VM data

salt.cloud.clouds.gce.**get_lb_conn**(*gce_driver=None*)
Return a load-balancer conn object

salt.cloud.clouds.gce.**list_nodes**(*conn=None*, *call=None*)
Return a list of the VMs that are on the provider

salt.cloud.clouds.gce.**list_nodes_full**(*conn=None*, *call=None*)
Return a list of the VMs that are on the provider, with all fields

salt.cloud.clouds.gce.**list_nodes_select**(*conn=None*, *call=None*)
Return a list of the VMs that are on the provider, with select fields

salt.cloud.clouds.gce.**reboot**(*vm_name*, *call=None*)
Call GCE `reset' on the instance.

CLI Example:

```
salt-cloud -a reboot myinstance
```

salt.cloud.clouds.gce.**script**(*vm_*)
Return the script deployment object

salt.cloud.clouds.gce.**show_address**(*kwargs=None*, *call=None*)
Show the details of an existing static address.

CLI Example:

```
salt-cloud -f show_address gce name=mysnapshot region=us-central1
```

salt.cloud.clouds.gce.**show_disk**(*name=None*, *kwargs=None*, *call=None*)
Show the details of an existing disk.

CLI Example:

```
salt-cloud -a show_disk myinstance disk_name=mydisk
salt-cloud -f show_disk gce disk_name=mydisk
```

salt.cloud.clouds.gce.**show_fwrule**(*kwargs=None*, *call=None*)
Show the details of an existing firewall rule.

CLI Example:

```
salt-cloud -f show_fwrule gce name=allow-http
```

salt.cloud.clouds.gce.**show_hc**(*kwargs=None*, *call=None*)
Show the details of an existing health check.

CLI Example:

```
salt-cloud -f show_hc gce name=hc
```

salt.cloud.clouds.gce.**show_instance**(*vm_name*, *call=None*)
Show the details of the existing instance.

salt.cloud.clouds.gce.**show_lb**(*kwargs=None*, *call=None*)
Show the details of an existing load-balancer.

CLI Example:

```
salt-cloud -f show_lb gce name=lb
```

salt.cloud.clouds.gce.**show_network**(*kwargs=None*, *call=None*)
Show the details of an existing network.

CLI Example:

```
    salt-cloud -f show_network gce name=mynet
```

salt.cloud.clouds.gce.**show_snapshot**(*kwargs=None*, *call=None*)
Show the details of an existing snapshot.

CLI Example:

```
    salt-cloud -f show_snapshot gce name=mysnapshot
```

22.5.8 salt.cloud.clouds.gogrid

GoGrid Cloud Module

The GoGrid cloud module. This module interfaces with the gogrid public cloud service. To use Salt Cloud with GoGrid log into the GoGrid web interface and create an api key. Do this by clicking on ``My Account'' and then going to the API Keys tab.

depends libcloud >= 0.13.2

Set up the cloud configuration at `/etc/salt/cloud.providers` or `/etc/salt/cloud.providers.d/gogrid.conf`:

```
my-gogrid-config:
  # The generated api key to use
  apikey: asdff7896asdh789
  # The apikey's shared secret
  sharedsecret: saltybacon

  provider: gogrid
```

salt.cloud.clouds.gogrid.**avail_images**(*conn=None*, *call=None*)
Return a dict of all available VM images on the cloud provider with relevant data

salt.cloud.clouds.gogrid.**avail_sizes**(*conn=None*, *call=None*)
Return a dict of all available VM images on the cloud provider with relevant data

salt.cloud.clouds.gogrid.**create**(*vm_*)
Create a single VM from a data dict

salt.cloud.clouds.gogrid.**destroy**(*name*, *conn=None*, *call=None*)
Delete a single VM

salt.cloud.clouds.gogrid.**get_configured_provider**()
Return the first configured instance.

salt.cloud.clouds.gogrid.**get_conn**()
Return a conn object for the passed VM data

salt.cloud.clouds.gogrid.**get_image**(*conn*, *vm_*)
Return the image object to use

salt.cloud.clouds.gogrid.**get_size**(*conn*, *vm_*)
Return the VM's size object

salt.cloud.clouds.gogrid.**list_nodes**(*conn=None*, *call=None*)
Return a list of the VMs that are on the provider

salt.cloud.clouds.gogrid.**list_nodes_full**(*conn=None*, *call=None*)
Return a list of the VMs that are on the provider, with all fields

salt.cloud.clouds.gogrid.**list_nodes_select**(*conn=None*, *call=None*)
Return a list of the VMs that are on the provider, with select fields

salt.cloud.clouds.gogrid.**script**(*vm_*)
Return the script deployment object

salt.cloud.clouds.gogrid.**show_instance**(*name*, *call=None*)
Show the details from the provider concerning an instance

22.5.9 salt.cloud.clouds.libcloud_aws

The AWS Cloud Module

The AWS cloud module is used to interact with the Amazon Web Services system.

This module has been replaced by the EC2 cloud module, and is no longer supported. The documentation shown here is for reference only; it is highly recommended to change all usages of this driver over to the EC2 driver.

If this driver is still needed, set up the cloud configuration at `/etc/salt/cloud.providers` or `/etc/salt/cloud.providers.d/aws.conf`:

```
my-aws-config:
  # The AWS API authentication id
  id: GKTADJGHEIQSXMKKRBJ08H
  # The AWS API authentication key
  key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs
  # The ssh keyname to use
  keyname: default
  # The amazon security group
  securitygroup: ssh_open
  # The location of the private key which corresponds to the keyname
  private_key: /root/default.pem

  provider: aws
```

salt.cloud.clouds.libcloud_aws.**block_device_mappings**(*vm_*)
Return the block device mapping:

```
[{'DeviceName': '/dev/sdb', 'VirtualName': 'ephemeral0'},
  {'DeviceName': '/dev/sdc', 'VirtualName': 'ephemeral1'}]
```

salt.cloud.clouds.libcloud_aws.**create**(*vm_*)
Create a single VM from a data dict

salt.cloud.clouds.libcloud_aws.**create_attach_volumes**(*volumes*, *location*, *data*)
Create and attach volumes to created node

salt.cloud.clouds.libcloud_aws.**del_tags**(*name*, *kwargs*, *call=None*)
Delete tags for a node

CLI Example:

```
salt-cloud -a del_tags mymachine tag1,tag2,tag3
```

salt.cloud.clouds.libcloud_aws.**destroy**(*name*)
Wrap core libcloudfuncs destroy method, adding check for termination protection

salt.cloud.clouds.libcloud_aws.**get_availability_zone**(*conn*, *vm_*)
Return the availability zone to use

salt.cloud.clouds.libcloud_aws.**get_configured_provider**()
Return the first configured instance.

salt.cloud.clouds.libcloud_aws.**get_conn**(***kwargs*)
Return a conn object for the passed VM data

salt.cloud.clouds.libcloud_aws.**get_location**(*vm_=None*)

Return the AWS region to use, in this order:

- CLI parameter
- Cloud profile setting
- Global salt-cloud config

salt.cloud.clouds.libcloud_aws.**get_salt_interface**(*vm_*)
Return the salt_interface type to connect to. Either `public_ips' (default) or `private_ips'.

salt.cloud.clouds.libcloud_aws.**get_tags**(*name*, *call=None*)
Retrieve tags for a node

salt.cloud.clouds.libcloud_aws.**iam_profile**(*vm_*)
Return the IAM role

salt.cloud.clouds.libcloud_aws.**keyname**(*vm_*)
Return the keyname

salt.cloud.clouds.libcloud_aws.**rename**(*name*, *kwargs*, *call=None*)
Properly rename a node. Pass in the new name as ``new name''.

CLI Example:

```
salt-cloud -a rename mymachine newname=yourmachine
```

salt.cloud.clouds.libcloud_aws.**securitygroup**(*vm_*)
Return the security group

salt.cloud.clouds.libcloud_aws.**set_tags**(*name*, *tags*, *call=None*)
Set tags for a node

CLI Example:

```
salt-cloud -a set_tags mymachine tag1=somestuff tag2='Other stuff'
```

salt.cloud.clouds.libcloud_aws.**ssh_interface**(*vm_*)
Return the ssh_interface type to connect to. Either `public_ips' (default) or `private_ips'.

salt.cloud.clouds.libcloud_aws.**ssh_username**(*vm_*)
Return the ssh_username. Defaults to `ec2-user'.

salt.cloud.clouds.libcloud_aws.**start**(*name*, *call=None*)
Start a node

salt.cloud.clouds.libcloud_aws.**stop**(*name*, *call=None*)
Stop a node

22.5.10 salt.cloud.clouds.linode

Linode Cloud Module using Apache Libcloud OR linode-python bindings

The Linode cloud module is used to control access to the Linode VPS system

Use of this module only requires the `apikey` parameter.

depends linode-python >= 1.1.1

OR :depends: apache-libcloud >= 0.13.2

Note: The linode-python driver will work with earlier versions of linode-python, but it is highly recommended to use a minimum version of 1.1.1. Earlier versions leak sensitive information into the debug logs.

Set up the cloud configuration at `/etc/salt/cloud.providers` or `/etc/salt/cloud.providers.d/linode.conf`:

my-linode-config: # Linode account api key apikey: JVkbSJDGHSDKUKSDJfhsdklfjgsjdkflhjlsdfffhgdgjkenrtuinv provider: linode

When used with linode-python, this provider supports cloning existing Linodes. To clone, add a profile with a `clonefrom` key, and a `script_args: -C`.

`Clonefrom` should be the name of the that is the source for the clone. `script_args: -C` passes a -C to the bootstrap script, which only configures the minion and doesn't try to install a new copy of salt-minion. This way the minion gets new keys and the keys get pre-seeded on the master, and the /etc/salt/minion file has the right `id:' declaration.

Cloning requires a post 2015-02-01 salt-bootstrap.

`salt.cloud.clouds.linode.`**`avail_images`**`(`*conn=None*, *call=None*`)`
Return a dict of all available VM images on the cloud provider with relevant data

`salt.cloud.clouds.linode.`**`avail_locations`**`(`*conn=None*, *call=None*`)`
Return a dict of all available VM locations on the cloud provider with relevant data

`salt.cloud.clouds.linode.`**`avail_sizes`**`(`*conn=None*, *call=None*`)`
Return a dict of all available VM images on the cloud provider with relevant data

`salt.cloud.clouds.linode.`**`boot`**`(`*LinodeID=None*, *configid=None*`)`
Execute a boot sequence on a linode

`salt.cloud.clouds.linode.`**`create`**`(`*vm_*`)`
Create a single VM from a data dict

`salt.cloud.clouds.linode.`**`create_config`**`(`*vm_*, *LinodeID=None*, *root_disk_id=None*, *swap_disk_id=None*`)`
Create a Linode Config

`salt.cloud.clouds.linode.`**`create_disk_from_distro`**`(`*vm_=None*, *LinodeID=None*, *swapsize=None*`)`
Create the disk for the linode

`salt.cloud.clouds.linode.`**`create_swap_disk`**`(`*vm_=None*, *LinodeID=None*, *swapsize=None*`)`
Create the disk for the linode

`salt.cloud.clouds.linode.`**`destroy`**`(`*name*, *conn=None*, *call=None*`)`
Delete a single VM

`salt.cloud.clouds.linode.`**`get_auth`**`(`*vm_*`)`
Return either NodeAuthSSHKey or NodeAuthPassword, preferring NodeAuthSSHKey if both are provided.

`salt.cloud.clouds.linode.`**`get_configured_provider`**`()`
Return the first configured instance.

`salt.cloud.clouds.linode.`**`get_conn`**`()`
Return a conn object for the passed VM data

salt.cloud.clouds.linode.**get_disk_size**(*vm_*, *size*, *swap*)
Return the size of of the root disk in MB

salt.cloud.clouds.linode.**get_image**(*conn*, *vm_*)
Return the image object to use

salt.cloud.clouds.linode.**get_kernels**(*conn=None*)
Get Linode's list of kernels available

salt.cloud.clouds.linode.**get_location**(*conn*, *vm_*)
Return the node location to use

salt.cloud.clouds.linode.**get_node**(*conn*, *name*)
Return a libcloud node for the named VM

salt.cloud.clouds.linode.**get_one_kernel**(*conn=None*, *name=None*)
Return data on one kernel name=None returns latest kernel

salt.cloud.clouds.linode.**get_password**(*vm_*)
Return the password to use

salt.cloud.clouds.linode.**get_private_ip**(*vm_*)
Return True if a private ip address is requested

salt.cloud.clouds.linode.**get_pubkey**(*vm_*)
Return the SSH pubkey to use

salt.cloud.clouds.linode.**get_size**(*conn*, *vm_*)
Return the VM's size object

salt.cloud.clouds.linode.**get_ssh_key_filename**(*vm_*)
Return path to filename if get_auth() returns a NodeAuthSSHKey.

salt.cloud.clouds.linode.**get_swap**(*vm_*)
Return the amount of swap space to use in MB

salt.cloud.clouds.linode.**list_nodes**(*conn=None*, *call=None*)
Return a list of the VMs that are on the provider

salt.cloud.clouds.linode.**list_nodes_full**(*conn=None*, *call=None*)
Return a list of the VMs that are on the provider, with all fields

salt.cloud.clouds.linode.**list_nodes_select**(*conn=None*, *call=None*)
Return a list of the VMs that are on the provider, with select fields

salt.cloud.clouds.linode.**remove_complex_types**(*dictionary*)
Linode-python is now returning some complex types that are not serializable by msgpack. Kill those.

salt.cloud.clouds.linode.**script**(*vm_*)
Return the script deployment object

salt.cloud.clouds.linode.**show_instance**(*name*, *call=None*)
Show the details from the provider concerning an instance

salt.cloud.clouds.linode.**waitfor_job**(*conn=None*, *LinodeID=None*, *JobID=None*, *timeout=300*, *quiet=True*)

salt.cloud.clouds.linode.**waitfor_status**(*conn=None*, *LinodeID=None*, *status=None*, *timeout=300*, *quiet=True*)
Wait for a certain status

22.5.11 salt.cloud.clouds.lxc

Install Salt on an LXC Container

New in version 2014.7.0.

Please read *core config documentation*.

`salt.cloud.clouds.lxc.`**`avail_images`**`()`

`salt.cloud.clouds.lxc.`**`create`**`(`*vm_*, *call=None*`)`
Create an lxc Container. This function is idempotent and will try to either provision or finish the provision of an lxc container.

NOTE: Most of the initialization code has been moved and merged with the lxc runner and lxc.init functions

`salt.cloud.clouds.lxc.`**`destroy`**`(`*vm_*, *call=None*`)`
Destroy a lxc container

`salt.cloud.clouds.lxc.`**`get_configured_provider`**`(`*vm_=None*`)`
Return the contextual provider of None if no configured one can be found.

`salt.cloud.clouds.lxc.`**`get_provider`**`(`*name*`)`

`salt.cloud.clouds.lxc.`**`list_nodes`**`(`*conn=None*, *call=None*`)`

`salt.cloud.clouds.lxc.`**`list_nodes_full`**`(`*conn=None*, *call=None*`)`

`salt.cloud.clouds.lxc.`**`list_nodes_select`**`(`*call=None*`)`
Return a list of the VMs that are on the provider, with select fields

`salt.cloud.clouds.lxc.`**`show_instance`**`(`*name*, *call=None*`)`
Show the details from the provider concerning an instance

22.5.12 salt.cloud.clouds.msazure

Azure Cloud Module

The Azure cloud module is used to control access to Microsoft Azure

depends

- Microsoft Azure SDK for Python

configuration Required provider parameters:

- `apikey`
- `certificate_path`
- `subscription_id`

A Management Certificate (.pem and .crt files) must be created and the .pem file placed on the same machine that salt-cloud is run from. Information on creating the pem file to use, and uploading the associated cer file can be found at:

http://www.windowsazure.com/en-us/develop/python/how-to-guides/service-management/

Example `/etc/salt/cloud.providers` or `/etc/salt/cloud.providers.d/azure.conf` configuration:

```
my-azure-config:
  provider: azure
  subscription_id: 3287abc8-f98a-c678-3bde-326766fd3617
  certificate_path: /etc/salt/azure.pem
  management_host: management.core.windows.net
```

salt.cloud.clouds.msazure.**avail_images**(*conn=None*, *call=None*)
List available images for Azure

salt.cloud.clouds.msazure.**avail_locations**(*conn=None*, *call=None*)
List available locations for Azure

salt.cloud.clouds.msazure.**avail_sizes**(*call=None*)
Because sizes are built into images with Azure, there will be no sizes to return here

salt.cloud.clouds.msazure.**create**(*vm_*)
Create a single VM from a data dict

salt.cloud.clouds.msazure.**create_attach_volumes**(*name*, *kwargs*, *call=None*, *wait_to_finish=True*)
Create and attach volumes to created node

salt.cloud.clouds.msazure.**destroy**(*name*, *conn=None*, *call=None*, *kwargs=None*)
Destroy a VM

CLI Examples:

```
salt-cloud -d myminion
salt-cloud -a destroy myminion service_name=myservice
```

salt.cloud.clouds.msazure.**get_configured_provider**()
Return the first configured instance.

salt.cloud.clouds.msazure.**get_conn**()
Return a conn object for the passed VM data

salt.cloud.clouds.msazure.**list_disks**(*conn=None*, *call=None*)
Destroy a VM

salt.cloud.clouds.msazure.**list_hosted_services**(*conn=None*, *call=None*)
List VMs on this Azure account, with full information

salt.cloud.clouds.msazure.**list_nodes**(*conn=None*, *call=None*)
List VMs on this Azure account

salt.cloud.clouds.msazure.**list_nodes_full**(*conn=None*, *call=None*)
List VMs on this Azure account, with full information

salt.cloud.clouds.msazure.**list_nodes_select**(*conn=None*, *call=None*)
Return a list of the VMs that are on the provider, with select fields

salt.cloud.clouds.msazure.**list_storage_services**(*conn=None*, *call=None*)
List VMs on this Azure account, with full information

salt.cloud.clouds.msazure.**script**(*vm_*)
Return the script deployment object

salt.cloud.clouds.msazure.**show_instance**(*name*, *call=None*)
Show the details from the provider concerning an instance

salt.cloud.clouds.msazure.**show_service**(*kwargs=None*, *conn=None*, *call=None*)
Show the details from the provider concerning an instance

22.5.13 salt.cloud.clouds.nova

OpenStack Nova Cloud Module

PLEASE NOTE: This module is currently in early development, and considered to be experimental and unstable. It is not recommended for production use. Unless you are actively developing code in this module, you should use the OpenStack module instead.

OpenStack is an open source project that is in use by a number a cloud providers, each of which have their own ways of using it.

The OpenStack Nova module for Salt Cloud was bootstrapped from the OpenStack module for Salt Cloud, which uses a libcloud-based connection. The Nova module is designed to use the nova and glance modules already built into Salt.

These modules use the Python novaclient and glanceclient libraries, respectively. In order to use this module, the proper salt configuration must also be in place. This can be specified in the master config, the minion config, a set of grains or a set of pillars.

```
my_openstack_profile:
  keystone.user: admin
  keystone.password: verybadpass
  keystone.tenant: admin
  keystone.auth_url: 'http://127.0.0.1:5000/v2.0/'
```

Note that there is currently a dependency upon netaddr. This can be installed on Debian-based systems by means of the python-netaddr package.

This module currently requires the latest develop branch of Salt to be installed.

This module has been tested to work with HP Cloud and Rackspace. See the documentation for specific options for either of these providers. These examples could be set up in the cloud configuration at `/etc/salt/cloud.providers` or `/etc/salt/cloud.providers.d/openstack.conf`:

```
my-openstack-config:
  # The ID of the minion that will execute the salt nova functions
  auth_minion: myminion
  # The name of the configuration profile to use on said minion
  config_profile: my_openstack_profile

  ssh_key_name: mykey

  provider: nova
  userdata_file: /tmp/userdata.txt
```

For local installations that only use private IP address ranges, the following option may be useful. Using the old syntax:

Note: For api use, you will need an auth plugin. The base novaclient does not support apikeys, but some providers such as rackspace have extended keystone to accept them

```
my-openstack-config:
  # Ignore IP addresses on this network for bootstrap
  ignore_cidr: 192.168.50.0/24

my-nova:
  identity_url: 'https://identity.api.rackspacecloud.com/v2.0/'
  compute_region: IAD
  user: myusername
  password: mypassword
```

```
  tenant: <userid>
  provider: nova

my-api:
  identity_url: 'https://identity.api.rackspacecloud.com/v2.0/'
  compute_region: IAD
  user: myusername
  api_key: <api_key>
  os_auth_plugin: rackspace
  tenant: <userid>
  provider: nova
  networks:
    - net-id: 47a38ff2-fe21-4800-8604-42bd1848e743
    - net-id: 00000000-0000-0000-0000-000000000000
    - net-id: 11111111-1111-1111-1111-111111111111
```

Note: You must include the default net-ids when setting networks or the server will be created without the rest of the interfaces

Note: For rackconnect v3, rackconnectv3 needs to be specified with the rackconnect v3 cloud network as it's variable

salt.cloud.clouds.nova.**avail_images**()
Return a dict of all available VM images on the cloud provider.

salt.cloud.clouds.nova.**avail_locations**(*conn=None*, *call=None*)
Return a list of locations

salt.cloud.clouds.nova.**avail_sizes**()
Return a dict of all available VM sizes on the cloud provider.

salt.cloud.clouds.nova.**create**(*vm_*)
Create a single VM from a data dict

salt.cloud.clouds.nova.**destroy**(*name*, *conn=None*, *call=None*)
Delete a single VM

salt.cloud.clouds.nova.**get_configured_provider**()
Return the first configured instance.

salt.cloud.clouds.nova.**get_conn**()
Return a conn object for the passed VM data

salt.cloud.clouds.nova.**get_image**(*conn*, *vm_*)
Return the image object to use

salt.cloud.clouds.nova.**get_salt_interface**(*vm_*)
Return the salt_interface type to connect to. Either `public_ips' (default) or `private_ips'.

salt.cloud.clouds.nova.**get_size**(*conn*, *vm_*)
Return the VM's size object

salt.cloud.clouds.nova.**ignore_cidr**(*vm_*, *ip*)
Return True if we are to ignore the specified IP. Compatible with IPv4.

salt.cloud.clouds.nova.**list_nodes**(*call=None*, ***kwargs*)
Return a list of the VMs that in this location

salt.cloud.clouds.nova.**list_nodes_full**(*call=None*, ***kwargs*)
Return a list of the VMs that in this location

salt.cloud.clouds.nova.**list_nodes_select**(*call=None*)
Return a list of the VMs that are on the provider, with select fields

salt.cloud.clouds.nova.**managedcloud**(*vm_*)
Determine if we should wait for the managed cloud automation before running. Either `False' (default) or `True'.

salt.cloud.clouds.nova.**network_create**(*name*, ***kwargs*)
Create private networks

salt.cloud.clouds.nova.**network_list**(*call=None*, ***kwargs*)
List private networks

salt.cloud.clouds.nova.**preferred_ip**(*vm_*, *ips*)
Return the preferred Internet protocol. Either `ipv4' (default) or `ipv6'.

salt.cloud.clouds.nova.**rackconnect**(*vm_*)
Determine if we should wait for rackconnect automation before running. Either `False' (default) or `True'.

salt.cloud.clouds.nova.**reboot**(*name*, *conn=None*)
Reboot a single VM

salt.cloud.clouds.nova.**request_instance**(*vm_=None*, *call=None*)
Put together all of the information necessary to request an instance through Novaclient and then fire off the request the instance.

Returns data about the instance

salt.cloud.clouds.nova.**script**(*vm_*)
Return the script deployment object

salt.cloud.clouds.nova.**show_instance**(*name*, *call=None*)
Show the details from the provider concerning an instance

salt.cloud.clouds.nova.**ssh_interface**(*vm_*)
Return the ssh_interface type to connect to. Either `public_ips' (default) or `private_ips'.

salt.cloud.clouds.nova.**virtual_interface_create**(*name*, *net_name*, ***kwargs*)
Create private networks

salt.cloud.clouds.nova.**virtual_interface_list**(*name*, ***kwargs*)
Create private networks

salt.cloud.clouds.nova.**volume_attach**(*name*, *server_name*, *device='/dev/xvdb'*, ***kwargs*)
Attach block volume

salt.cloud.clouds.nova.**volume_create**(*name*, *size=100*, *snapshot=None*, *voltype=None*, ***kwargs*)
Create block storage device

salt.cloud.clouds.nova.**volume_delete**(*name*, ***kwargs*)
Delete block storage device

salt.cloud.clouds.nova.**volume_detach**(*name*, ***kwargs*)
Detach block volume

salt.cloud.clouds.nova.**volume_list**(***kwargs*)
List block devices

22.5.14 salt.cloud.clouds.opennebula

OpenNebula Cloud Module

The OpenNebula cloud module is used to control access to an OpenNebula cloud.

Use of this module requires the `xml_rpc`, `user` and `password` parameter to be set. Set up the cloud configuration at `/etc/salt/cloud.providers` or `/etc/salt/cloud.providers.d/opennebula.conf`:

```
my-opennebula-config:
  xml_rpc: http://localhost:2633/RPC2
  user: oneadmin
  password: JHGhgsayu32jsa
  provider: opennebula
```

`salt.cloud.clouds.opennebula.`**`avail_images`**(*call=None*)
Return a list of the templates that are on the provider

`salt.cloud.clouds.opennebula.`**`avail_locations`**(*call=None*)
Return a dict of all available VM locations on the cloud provider with relevant data

`salt.cloud.clouds.opennebula.`**`avail_sizes`**(*call=None*)
Because sizes are built into templates with OpenNebula, there will be no sizes to return here

`salt.cloud.clouds.opennebula.`**`create`**(*vm_*)
Create a single VM from a data dict

`salt.cloud.clouds.opennebula.`**`destroy`**(*name, call=None*)
Destroy a node. Will check termination protection and warn if enabled.

CLI Example:

```
salt-cloud --destroy mymachine
```

`salt.cloud.clouds.opennebula.`**`get_configured_provider`**()
Return the first configured instance.

`salt.cloud.clouds.opennebula.`**`get_image`**(*vm_*)
Return the image object to use

`salt.cloud.clouds.opennebula.`**`get_location`**(*vm_*)
Return the VM's location

`salt.cloud.clouds.opennebula.`**`list_nodes`**(*call=None*)
Return a list of the VMs that are on the provider

`salt.cloud.clouds.opennebula.`**`list_nodes_full`**(*call=None*)
Return a list of the VMs that are on the provider

`salt.cloud.clouds.opennebula.`**`list_nodes_select`**(*call=None*)
Return a list of the VMs that are on the provider, with select fields

`salt.cloud.clouds.opennebula.`**`script`**(*vm_*)
Return the script deployment object

`salt.cloud.clouds.opennebula.`**`show_instance`**(*name, call=None*)
Show the details from OpenNebula concerning a VM

22.5.15 salt.cloud.clouds.openstack

OpenStack Cloud Module

OpenStack is an open source project that is in use by a number a cloud providers, each of which have their own ways of using it.

depends libcloud >- 0.13.2

OpenStack provides a number of ways to authenticate. This module uses password- based authentication, using auth v2.0. It is likely to start supporting other methods of authentication provided by OpenStack in the future.

Note that there is currently a dependency upon netaddr. This can be installed on Debian-based systems by means of the python-netaddr package.

This module has been tested to work with HP Cloud and Rackspace. See the documentation for specific options for either of these providers. Some examples, using the old cloud configuration syntax, are provided below:

Set up in the cloud configuration at `/etc/salt/cloud.providers` or `/etc/salt/cloud.providers.d/openstack.conf`:

```
my-openstack-config:
  # The OpenStack identity service url
  identity_url: https://region-b.geo-1.identity.hpcloudsvc.com:35357/v2.0/
  # The OpenStack compute region
  compute_region: region-b.geo-1
  # The OpenStack compute service name
  compute_name: Compute
  # The OpenStack tenant name (not tenant ID)
  tenant: myuser-tenant1
  # The OpenStack user name
  user: myuser
  # The OpenStack keypair name
  ssh_key_name: mykey
  # Skip SSL certificate validation
  insecure: false
  # The ssh key file
  ssh_key_file: /path/to/keyfile/test.pem
  # The OpenStack network UUIDs
  networks:
      - fixed:
          - 4402cd51-37ee-435e-a966-8245956dc0e6
      - floating:
          - Ext-Net
  files:
      /path/to/dest.txt:
          /local/path/to/src.txt
  # Skips the service catalog API endpoint, and uses the following
  base_url: http://192.168.1.101:3000/v2/12345
  provider: openstack
  userdata_file: /tmp/userdata.txt
  # config_drive is required for userdata at rackspace
  config_drive: True
```

For in-house Openstack Essex installation, libcloud needs the service_type :

```
my-openstack-config:
  identity_url: 'http://control.openstack.example.org:5000/v2.0/'
  compute_name : Compute Service
  service_type : compute
```

Either a password or an API key must also be specified:

```
my-openstack-password-or-api-config:
  # The OpenStack password
  password: letmein
  # The OpenStack API key
  apikey: 901d3f579h23c8v73q9
```

Optionally, if you don't want to save plain-text password in your configuration file, you can use keyring:

```
my-openstack-keyring-config:
  # The OpenStack password is stored in keyring
  # don't forget to set the password by running something like:
  # salt-cloud --set-password=myuser my-openstack-keyring-config
  password: USE_KEYRING
```

For local installations that only use private IP address ranges, the following option may be useful. Using the old syntax:

```
my-openstack-config:
  # Ignore IP addresses on this network for bootstrap
  ignore_cidr: 192.168.50.0/24
```

It is possible to upload a small set of files (no more than 5, and nothing too large) to the remote server. Generally this should not be needed, as salt itself can upload to the server after it is spun up, with nowhere near the same restrictions.

```
my-openstack-config:
  files:
      /path/to/dest.txt:
          /local/path/to/src.txt
```

Alternatively, one could use the private IP to connect by specifying:

```
my-openstack-config:
  ssh_interface: private_ips
```

salt.cloud.clouds.openstack.**avail_images**(*conn=None*, *call=None*)
Return a dict of all available VM images on the cloud provider with relevant data

salt.cloud.clouds.openstack.**avail_locations**(*conn=None*, *call=None*)
Return a dict of all available VM locations on the cloud provider with relevant data

salt.cloud.clouds.openstack.**avail_sizes**(*conn=None*, *call=None*)
Return a dict of all available VM images on the cloud provider with relevant data

salt.cloud.clouds.openstack.**create**(*vm_*)
Create a single VM from a data dict

salt.cloud.clouds.openstack.**destroy**(*name*, *conn=None*, *call=None*)
Delete a single VM

salt.cloud.clouds.openstack.**get_configured_provider**()
Return the first configured instance.

salt.cloud.clouds.openstack.**get_conn**()
Return a conn object for the passed VM data

salt.cloud.clouds.openstack.**get_image**(*conn*, *vm_*)
Return the image object to use

salt.cloud.clouds.openstack.**get_node**(*conn*, *name*)
Return a libcloud node for the named VM

salt.cloud.clouds.openstack.**get_salt_interface**(*vm_*)
Return the salt_interface type to connect to. Either `public_ips' (default) or `private_ips'.

salt.cloud.clouds.openstack.**get_size**(*conn*, *vm_*)
Return the VM's size object

salt.cloud.clouds.openstack.**ignore_cidr**(*vm_*, *ip*)
Return True if we are to ignore the specified IP. Compatible with IPv4.

salt.cloud.clouds.openstack.**list_nodes**(*conn=None*, *call=None*)
Return a list of the VMs that are on the provider

salt.cloud.clouds.openstack.**list_nodes_full**(*conn=None*, *call=None*)
Return a list of the VMs that are on the provider, with all fields

salt.cloud.clouds.openstack.**list_nodes_select**(*conn=None*, *call=None*)
Return a list of the VMs that are on the provider, with select fields

salt.cloud.clouds.openstack.**managedcloud**(*vm_*)
Determine if we should wait for the managed cloud automation before running. Either `False' (default) or `True'.

salt.cloud.clouds.openstack.**networks**(*vm_*, *kwargs=None*)

salt.cloud.clouds.openstack.**preferred_ip**(*vm_*, *ips*)
Return the preferred Internet protocol. Either `ipv4' (default) or `ipv6'.

salt.cloud.clouds.openstack.**rackconnect**(*vm_*)
Determine if we should wait for rackconnect automation before running. Either `False' (default) or `True'.

salt.cloud.clouds.openstack.**reboot**(*name*, *conn=None*)
Reboot a single VM

salt.cloud.clouds.openstack.**request_instance**(*vm_=None*, *call=None*)
Put together all of the information necessary to request an instance on Openstack and then fire off the request the instance.

Returns data about the instance

salt.cloud.clouds.openstack.**script**(*vm_*)
Return the script deployment object

salt.cloud.clouds.openstack.**show_instance**(*name*, *call=None*)
Show the details from the provider concerning an instance

salt.cloud.clouds.openstack.**ssh_interface**(*vm_*)
Return the ssh_interface type to connect to. Either `public_ips' (default) or `private_ips'.

22.5.16 salt.cloud.clouds.parallels

Parallels Cloud Module

The Parallels cloud module is used to control access to cloud providers using the Parallels VPS system.

Set up the cloud configuration at `/etc/salt/cloud.providers` or `/etc/salt/cloud.providers.d/parallels.c`

```
my-parallels-config:
  # Parallels account information
  user: myuser
  password: mypassword
  url: https://api.cloud.xmission.com:4465/paci/v1.0/
  provider: parallels
```

salt.cloud.clouds.parallels.**avail_images**(*call=None*)
Return a list of the images that are on the provider

salt.cloud.clouds.parallels.**create**(*vm_*)
Create a single VM from a data dict

salt.cloud.clouds.parallels.**create_node**(*vm_*)
Build and submit the XML to create a node

salt.cloud.clouds.parallels.**destroy**(*name*, *call=None*)
Destroy a node.

CLI Example:

```
salt-cloud --destroy mymachine
```

salt.cloud.clouds.parallels.**get_configured_provider**()
Return the first configured instance.

salt.cloud.clouds.parallels.**get_image**(*vm_*)
Return the image object to use

salt.cloud.clouds.parallels.**list_nodes**(*call=None*)
Return a list of the VMs that are on the provider

salt.cloud.clouds.parallels.**list_nodes_full**(*call=None*)
Return a list of the VMs that are on the provider

salt.cloud.clouds.parallels.**list_nodes_select**(*call=None*)
Return a list of the VMs that are on the provider, with select fields

salt.cloud.clouds.parallels.**query**(*action=None*, *command=None*, *args=None*, *method='GET'*, *data=None*)
Make a web call to a Parallels provider

salt.cloud.clouds.parallels.**script**(*vm_*)
Return the script deployment object

salt.cloud.clouds.parallels.**show_image**(*kwargs*, *call=None*)
Show the details from Parallels concerning an image

salt.cloud.clouds.parallels.**show_instance**(*name*, *call=None*)
Show the details from Parallels concerning an instance

salt.cloud.clouds.parallels.**start**(*name*, *call=None*)
Start a node.

CLI Example:

```
salt-cloud -a start mymachine
```

salt.cloud.clouds.parallels.**stop**(*name*, *call=None*)
Stop a node.

CLI Example:

```
salt-cloud -a stop mymachine
```

salt.cloud.clouds.parallels.**wait_until**(*name*, *state*, *timeout=300*)
Wait until a specific state has been reached on a node

22.5.17 salt.cloud.clouds.proxmox

Proxmox Cloud Module

New in version 2014.7.0.

The Proxmox cloud module is used to control access to cloud providers using the Proxmox system (KVM / OpenVZ).

Set up the cloud configuration at /etc/salt/cloud.providers or /etc/salt/cloud.providers.d/proxmox.con

```
my-proxmox-config:
  # Proxmox account information
  user: myuser@pam or myuser@pve
  password: mypassword
  url: hypervisor.domain.tld
  provider: proxmox
```

maintainer Frank Klaassen <frank@cloudright.nl>

maturity new

depends requests >= 2.2.1

depends IPy >= 0.81

salt.cloud.clouds.proxmox.**avail_images**(*call=None*, *location='local'*)
Return a list of the images that are on the provider

CLI Example:

```
salt-cloud --list-images my-proxmox-config
```

salt.cloud.clouds.proxmox.**avail_locations**(*call=None*)
Return a list of the hypervisors (nodes) which this Proxmox PVE machine manages

CLI Example:

```
salt-cloud --list-locations my-proxmox-config
```

salt.cloud.clouds.proxmox.**create**(*vm_*)
Create a single VM from a data dict

CLI Example:

```
salt-cloud -p proxmox-ubuntu vmhostname
```

salt.cloud.clouds.proxmox.**create_node**(*vm_*)
Build and submit the requestdata to create a new node

salt.cloud.clouds.proxmox.**destroy**(*name*, *call=None*)
Destroy a node.

CLI Example:

```
salt-cloud --destroy mymachine
```

salt.cloud.clouds.proxmox.**get_configured_provider**()
Return the first configured instance.

salt.cloud.clouds.proxmox.**get_resources_nodes**(*call=None*, *resFilter=None*)
Retrieve all hypervisors (nodes) available on this environment CLI Example:

```
salt-cloud -f get_resources_nodes my-proxmox-config
```

salt.cloud.clouds.proxmox.**get_resources_vms**(*call=None*, *resFilter=None*, *includeConfig=True*)
Retrieve all VMs available on this environment

CLI Example:

```
salt-cloud -f get_resources_vms my-proxmox-config
```

salt.cloud.clouds.proxmox.**get_vm_status**(*vmid=None*, *name=None*)
Get the status for a VM, either via the ID or the hostname

salt.cloud.clouds.proxmox.**get_vmconfig**(*vmid*, *node=None*, *node_type='openvz'*)
Get VM configuration

salt.cloud.clouds.proxmox.**list_nodes**(*call=None*)
Return a list of the VMs that are managed by the provider

CLI Example:

```
salt-cloud -Q my-proxmox-config
```

salt.cloud.clouds.proxmox.**list_nodes_full**(*call=None*)
Return a list of the VMs that are on the provider

CLI Example:

```
salt-cloud -F my-proxmox-config
```

salt.cloud.clouds.proxmox.**list_nodes_select**(*call=None*)
Return a list of the VMs that are on the provider, with select fields

CLI Example:

```
salt-cloud -S my-proxmox-config
```

salt.cloud.clouds.proxmox.**query**(*conn_type*, *option*, *post_data=None*)
Execute the HTTP request to the API

salt.cloud.clouds.proxmox.**script**(*vm_*)
Return the script deployment object

salt.cloud.clouds.proxmox.**set_vm_status**(*status*, *name=None*, *vmid=None*)
Convenience function for setting VM status

salt.cloud.clouds.proxmox.**show_instance**(*name*, *call=None*)
Show the details from Proxmox concerning an instance

salt.cloud.clouds.proxmox.**shutdown**(*name=None*, *vmid=None*, *call=None*)
Shutdown a node via ACPI.

CLI Example:

```
salt-cloud -a shutdown mymachine
```

salt.cloud.clouds.proxmox.**start**(*name*, *vmid=None*, *call=None*)
Start a node.

CLI Example:

```
salt-cloud -a start mymachine
```

salt.cloud.clouds.proxmox.**stop**(*name*, *vmid=None*, *call=None*)
Stop a node (``pulling the plug'').

CLI Example:

```
salt-cloud -a stop mymachine
```

salt.cloud.clouds.proxmox.**wait_for_created**(*upid*, *timeout=300*)
Wait until a the vm has been created successfully

salt.cloud.clouds.proxmox.**wait_for_state**(*vmid*, *state*, *timeout=300*)
Wait until a specific state has been reached on a node

22.5.18 salt.cloud.clouds.pyrax

Pyrax Cloud Module

PLEASE NOTE: This module is currently in early development, and considered to be experimental and unstable. It is not recommended for production use. Unless you are actively developing code in this module, you should use the OpenStack module instead.

salt.cloud.clouds.pyrax.**get_configured_provider**()
Return the first configured instance.

salt.cloud.clouds.pyrax.**get_conn**(*conn_type*)
Return a conn object for the passed VM data

salt.cloud.clouds.pyrax.**queues_create**(*call*, *kwargs*)

salt.cloud.clouds.pyrax.**queues_delete**(*call*, *kwargs*)

salt.cloud.clouds.pyrax.**queues_exists**(*call*, *kwargs*)

salt.cloud.clouds.pyrax.**queues_show**(*call*, *kwargs*)

22.5.19 salt.cloud.clouds.rackspace

Rackspace Cloud Module

The Rackspace cloud module. This module uses the preferred means to set up a libcloud based cloud module and should be used as the general template for setting up additional libcloud based modules.

depends libcloud >= 0.13.2

Please note that the *rackspace* driver is only intended for 1st gen instances, aka, ``the old cloud'' at Rackspace. It is required for 1st gen instances, but will *not* work with OpenStack-based instances. Unless you explicitly have a reason to use it, it is highly recommended that you use the *openstack* driver instead.

The rackspace cloud module interfaces with the Rackspace public cloud service and requires that two configuration parameters be set for use, `user` and `apikey`.

Set up the cloud configuration at `/etc/salt/cloud.providers` or `/etc/salt/cloud.providers.d/rackspace.conf`

```
my-rackspace-config:
  provider: rackspace
  # The Rackspace login user
  user: fred
  # The Rackspace user's apikey
  apikey: 901d3f579h23c8v73q9
```

salt.cloud.clouds.rackspace.**avail_images**(*conn=None*, *call=None*)
Return a dict of all available VM images on the cloud provider with relevant data

salt.cloud.clouds.rackspace.**avail_locations**(*conn=None*, *call=None*)
Return a dict of all available VM locations on the cloud provider with relevant data

salt.cloud.clouds.rackspace.**avail_sizes**(*conn=None*, *call=None*)
Return a dict of all available VM images on the cloud provider with relevant data

salt.cloud.clouds.rackspace.**create**(*vm_*)
Create a single VM from a data dict

salt.cloud.clouds.rackspace.**destroy**(*name*, *conn=None*, *call=None*)
Delete a single VM

salt.cloud.clouds.rackspace.**get_configured_provider**()
Return the first configured instance.

salt.cloud.clouds.rackspace.**get_conn**()
Return a conn object for the passed VM data

salt.cloud.clouds.rackspace.**get_image**(*conn*, *vm_*)
Return the image object to use

salt.cloud.clouds.rackspace.**get_salt_interface**(*vm_*)
Return the salt_interface type to connect to. Either `public_ips' (default) or `private_ips'.

salt.cloud.clouds.rackspace.**get_size**(*conn*, *vm_*)
Return the VM's size object

salt.cloud.clouds.rackspace.**list_nodes**(*conn=None*, *call=None*)
Return a list of the VMs that are on the provider

salt.cloud.clouds.rackspace.**list_nodes_full**(*conn=None*, *call=None*)
Return a list of the VMs that are on the provider, with all fields

salt.cloud.clouds.rackspace.**list_nodes_select**(*conn=None*, *call=None*)
Return a list of the VMs that are on the provider, with select fields

salt.cloud.clouds.rackspace.**preferred_ip**(*vm_*, *ips*)
Return the preferred Internet protocol. Either `ipv4' (default) or `ipv6'.

salt.cloud.clouds.rackspace.**script**(*vm_*)
Return the script deployment object

salt.cloud.clouds.rackspace.**show_instance**(*name*, *call=None*)
Show the details from the provider concerning an instance

salt.cloud.clouds.rackspace.**ssh_interface**(*vm_*)
Return the ssh_interface type to connect to. Either `public_ips' (default) or `private_ips'.

22.5.20 salt.cloud.clouds.saltify

Saltify Module

The Saltify module is designed to install Salt on a remote machine, virtual or bare metal, using SSH. This module is useful for provisioning machines which are already installed, but not Salted.

Use of this module requires no configuration in the main cloud configuration file. However, profiles must still be configured, as described in the *core config documentation*.

`salt.cloud.clouds.saltify.`**`create`**`(vm_)`
Provision a single machine

`salt.cloud.clouds.saltify.`**`get_configured_provider`**`()`
Return the first configured instance.

`salt.cloud.clouds.saltify.`**`list_nodes`**`()`
Because this module is not specific to any cloud providers, there will be no nodes to list.

`salt.cloud.clouds.saltify.`**`list_nodes_full`**`()`
Because this module is not specific to any cloud providers, there will be no nodes to list.

`salt.cloud.clouds.saltify.`**`list_nodes_select`**`()`
Because this module is not specific to any cloud providers, there will be no nodes to list.

`salt.cloud.clouds.saltify.`**`script`**`(vm_)`
Return the script deployment object

22.5.21 salt.cloud.clouds.softlayer

SoftLayer Cloud Module

The SoftLayer cloud module is used to control access to the SoftLayer VPS system.

Use of this module only requires the `apikey` parameter. Set up the cloud configuration at:

`/etc/salt/cloud.providers` or `/etc/salt/cloud.providers.d/softlayer.conf`:

```
my-softlayer-config:
  # SoftLayer account api key
  user: MYLOGIN
  apikey: JVkbSJDGHSDKUKSDJfhsdklfjgsjdkflhjlsdfffhgdgjkenrtuinv
  provider: softlayer
```

The SoftLayer Python Library needs to be installed in order to use the SoftLayer salt.cloud modules. See: https://pypi.python.org/pypi/SoftLayer

depends softlayer

`salt.cloud.clouds.softlayer.`**`avail_images`**`(call=None)`
Return a dict of all available VM images on the cloud provider.

`salt.cloud.clouds.softlayer.`**`avail_locations`**`(call=None)`
List all available locations

`salt.cloud.clouds.softlayer.`**`avail_sizes`**`(call=None)`
Return a dict of all available VM sizes on the cloud provider with relevant data. This data is provided in three dicts.

`salt.cloud.clouds.softlayer.`**`create`**`(vm_)`
Create a single VM from a data dict

salt.cloud.clouds.softlayer.**destroy**(*name*, *call=None*)
Destroy a node.

CLI Example:

```
salt-cloud --destroy mymachine
```

salt.cloud.clouds.softlayer.**get_configured_provider**()
Return the first configured instance.

salt.cloud.clouds.softlayer.**get_conn**(*service='SoftLayer_Virtual_Guest'*)
Return a conn object for the passed VM data

salt.cloud.clouds.softlayer.**get_location**(*vm_=None*)

Return the location to use, in this order:

- CLI parameter
- VM parameter
- Cloud profile setting

salt.cloud.clouds.softlayer.**list_custom_images**(*call=None*)
Return a dict of all custom VM images on the cloud provider.

salt.cloud.clouds.softlayer.**list_nodes**(*call=None*)
Return a list of the VMs that are on the provider

salt.cloud.clouds.softlayer.**list_nodes_full**(*mask='mask[id]'*, *call=None*)
Return a list of the VMs that are on the provider

salt.cloud.clouds.softlayer.**list_nodes_select**(*call=None*)
Return a list of the VMs that are on the provider, with select fields

salt.cloud.clouds.softlayer.**list_vlans**(*call=None*)
List all VLANs associated with the account

salt.cloud.clouds.softlayer.**script**(*vm_*)
Return the script deployment object

salt.cloud.clouds.softlayer.**show_instance**(*name*, *call=None*)
Show the details from SoftLayer concerning a guest

22.5.22 salt.cloud.clouds.softlayer_hw

SoftLayer HW Cloud Module

The SoftLayer HW cloud module is used to control access to the SoftLayer hardware cloud system

Use of this module only requires the `apikey` parameter. Set up the cloud configuration at:

`/etc/salt/cloud.providers` or `/etc/salt/cloud.providers.d/softlayer.conf`:

```
my-softlayer-config:
  # SoftLayer account api key
  user: MYLOGIN
  apikey: JVkbSJDGHSDKUKSDJfhsdklfjgsjdkflhjlsdfffhgdgjkenrtuinv
  provider: softlayer_hw
```

The SoftLayer Python Library needs to be installed in order to use the SoftLayer salt.cloud modules. See: https://pypi.python.org/pypi/SoftLayer

depends softlayer

salt.cloud.clouds.softlayer_hw.**avail_images**(*call=None*)
Return a dict of all available VM images on the cloud provider.

salt.cloud.clouds.softlayer_hw.**avail_locations**(*call=None*)
List all available locations

salt.cloud.clouds.softlayer_hw.**avail_sizes**(*call=None*)
Return a dict of all available VM sizes on the cloud provider with relevant data. This data is provided in three dicts.

salt.cloud.clouds.softlayer_hw.**create**(*vm_*)
Create a single VM from a data dict

salt.cloud.clouds.softlayer_hw.**destroy**(*name*, *call=None*)
Destroy a node.

CLI Example:

```
salt-cloud --destroy mymachine
```

salt.cloud.clouds.softlayer_hw.**get_configured_provider**()
Return the first configured instance.

salt.cloud.clouds.softlayer_hw.**get_conn**(*service='SoftLayer_Hardware'*)
Return a conn object for the passed VM data

salt.cloud.clouds.softlayer_hw.**get_location**(*vm_=None*)

Return the location to use, in this order:

- CLI parameter
- VM parameter
- Cloud profile setting

salt.cloud.clouds.softlayer_hw.**list_nodes**(*call=None*)
Return a list of the VMs that are on the provider

salt.cloud.clouds.softlayer_hw.**list_nodes_full**(*mask='mask[id, hostname, primaryIpAddress, primaryBackendIpAddress, processorPhysicalCoreAmount, memoryCount]'*, *call=None*)
Return a list of the VMs that are on the provider

salt.cloud.clouds.softlayer_hw.**list_nodes_select**(*call=None*)
Return a list of the VMs that are on the provider, with select fields

salt.cloud.clouds.softlayer_hw.**list_vlans**(*call=None*)
List all VLANs associated with the account

salt.cloud.clouds.softlayer_hw.**script**(*vm_*)
Return the script deployment object

salt.cloud.clouds.softlayer_hw.**show_instance**(*name*, *call=None*)
Show the details from SoftLayer concerning a guest

22.5.23 salt.cloud.clouds.vsphere

vSphere Cloud Module

New in version 2014.7.0.

The vSphere cloud module is used to control access to VMWare vSphere.

depends

- PySphere Python module >= 0.1.8

Note: Ensure python pysphere module is installed by running following one-liner check. The output should be 0.

```
python -c "import pysphere" ; echo $?
# if this fails install using
pip install https://pysphere.googlecode.com/files/pysphere-0.1.8.zip
```

Use of this module only requires a URL, username and password. Set up the cloud configuration at:

`/etc/salt/cloud.providers` or `/etc/salt/cloud.providers.d/vsphere.conf`:

```
my-vsphere-config:
  provider: vsphere
  user: myuser
  password: verybadpass
  url: 'https://10.1.1.1:443'
```

Note: Your URL may or may not look like any of the following, depending on how your VMWare installation is configured:

```
10.1.1.1
10.1.1.1:443
https://10.1.1.1:443
https://10.1.1.1:443/sdk
10.1.1.1:443/sdk
```

folder: **Name of the folder that will contain the new VM. If not set, the VM will** be added to the folder the original VM belongs to.

resourcepool: **MOR of the resourcepool to be used for the new vm. If not set, it** uses the same resourcepool than the original vm.

datastore: **MOR of the datastore where the virtual machine should be located. If** not specified, the current datastore is used.

host: **MOR of the host where the virtual machine should be registered.**

IF not specified:

- if resourcepool is not specified, current host is used.
- if resourcepool is specified, and the target pool represents a stand-alone host, the host is used.
- if resourcepool is specified, and the target pool represents a DRS-enabled cluster, a host selected by DRS is used.
- if resourcepool is specified and the target pool represents a cluster without DRS enabled, an InvalidArgument exception will be thrown.

template: **Specifies whether or not the new virtual machine should be marked as a** template. Default is False.

`salt.cloud.clouds.vsphere.`**`avail_images`**`()`
Return a dict of all available VM images on the cloud provider.

salt.cloud.clouds.vsphere.**avail_locations**()
Return a dict of all available VM locations on the cloud provider with relevant data

salt.cloud.clouds.vsphere.**create**(*vm_*)
Create a single VM from a data dict

salt.cloud.clouds.vsphere.**destroy**(*name*, *call=None*)
Destroy a node.

CLI Example:

```
salt-cloud --destroy mymachine
```

salt.cloud.clouds.vsphere.**get_configured_provider**()
Return the first configured instance.

salt.cloud.clouds.vsphere.**get_conn**()
Return a conn object for the passed VM data

salt.cloud.clouds.vsphere.**list_clusters**(*kwargs=None*, *call=None*)
List the clusters for this VMware environment

salt.cloud.clouds.vsphere.**list_datacenters**(*kwargs=None*, *call=None*)
List the data centers for this VMware environment

salt.cloud.clouds.vsphere.**list_datastores**(*kwargs=None*, *call=None*)
List the datastores for this VMware environment

salt.cloud.clouds.vsphere.**list_folders**(*kwargs=None*, *call=None*)
List the folders for this VMWare environment

salt.cloud.clouds.vsphere.**list_hosts**(*kwargs=None*, *call=None*)
List the hosts for this VMware environment

salt.cloud.clouds.vsphere.**list_nodes**(*kwargs=None*, *call=None*)
Return a list of the VMs that are on the provider, with basic fields

salt.cloud.clouds.vsphere.**list_nodes_full**(*kwargs=None*, *call=None*)
Return a list of the VMs that are on the provider with full details

salt.cloud.clouds.vsphere.**list_nodes_min**(*kwargs=None*, *call=None*)
Return a list of the nodes in the provider, with no details

salt.cloud.clouds.vsphere.**list_nodes_select**()
Return a list of the VMs that are on the provider, with select fields

salt.cloud.clouds.vsphere.**list_resourcepools**(*kwargs=None*, *call=None*)
List the hosts for this VMware environment

salt.cloud.clouds.vsphere.**script**(*vm_*)
Return the script deployment object

salt.cloud.clouds.vsphere.**show_instance**(*name*, *call=None*)
Show the details from vSphere concerning a guest

salt.cloud.clouds.vsphere.**wait_for_ip**(*vm_*)

22.6 Configuration file examples

- Example master configuration file
- Example minion configuration file

22.6.1 Example master configuration file

```
##### Primary configuration settings #####
##########################################
# This configuration file is used to manage the behavior of the Salt Master.
# Values that are commented out but have an empty line after the comment are
# defaults that do not need to be set in the config. If there is no blank line
# after the comment then the value is presented as an example and is not the
# default.

# Per default, the master will automatically include all config files
# from master.d/*.conf (master.d is a directory in the same directory
# as the main master config file).
#default_include: master.d/*.conf

# The address of the interface to bind to:
#interface: 0.0.0.0

# Whether the master should listen for IPv6 connections. If this is set to True,
# the interface option must be adjusted, too. (For example: "interface: '::'")
#ipv6: False

# The tcp port used by the publisher:
#publish_port: 4505

# The user under which the salt master will run. Salt will update all
# permissions to allow the specified user to run the master. The exception is
# the job cache, which must be deleted if this user is changed. If the
# modified files cause conflicts, set verify_env to False.
#user: root

# Max open files
#
# Each minion connecting to the master uses AT LEAST one file descriptor, the
# master subscription connection. If enough minions connect you might start
# seeing on the console (and then salt-master crashes):
#   Too many open files (tcp_listener.cpp:335)
#   Aborted (core dumped)
#
# By default this value will be the one of `ulimit -Hn`, ie, the hard limit for
# max open files.
#
# If you wish to set a different value than the default one, uncomment and
# configure this setting. Remember that this value CANNOT be higher than the
# hard limit. Raising the hard limit depends on your OS and/or distribution,
# a good way to find the limit is to search the internet. For example:
#   raise max open files hard limit debian
#
#max_open_files: 100000

# The number of worker threads to start. These threads are used to manage
# return calls made from minions to the master. If the master seems to be
```

```
# running slowly, increase the number of threads.
#worker_threads: 5

# The port used by the communication interface. The ret (return) port is the
# interface used for the file server, authentication, job returns, etc.
#ret_port: 4506

# Specify the location of the daemon process ID file:
#pidfile: /var/run/salt-master.pid

# The root directory prepended to these options: pki_dir, cachedir,
# sock_dir, log_file, autosign_file, autoreject_file, extension_modules,
# key_logfile, pidfile:
#root_dir: /

# Directory used to store public key data:
#pki_dir: /etc/salt/pki/master

# Directory to store job and cache data:
#cachedir: /var/cache/salt/master

# Directory for custom modules. This directory can contain subdirectories for
# each of Salt's module types such as "runners", "output", "wheel", "modules",
# "states", "returners", etc.
#extension_modules: <no default>

# Directory for custom modules. This directory can contain subdirectories for
# each of Salt's module types such as "runners", "output", "wheel", "modules",
# "states", "returners", etc.
# Like 'extension_modules' but can take an array of paths
#module_dirs: <no default>
#   - /var/cache/salt/minion/extmods

# Verify and set permissions on configuration directories at startup:
#verify_env: True

# Set the number of hours to keep old job information in the job cache:
#keep_jobs: 24

# Set the default timeout for the salt command and api. The default is 5
# seconds.
#timeout: 5

# The loop_interval option controls the seconds for the master's maintenance
# process check cycle. This process updates file server backends, cleans the
# job cache and executes the scheduler.
#loop_interval: 60

# Set the default outputter used by the salt command. The default is "nested".
#output: nested

# Return minions that timeout when running commands like test.ping
#show_timeout: True

# By default, output is colored. To disable colored output, set the color value
# to False.
#color: True
```

```
# Do not strip off the colored output from nested results and state outputs
# (true by default).
# strip_colors: False

# Set the directory used to hold unix sockets:
#sock_dir: /var/run/salt/master

# The master can take a while to start up when lspci and/or dmidecode is used
# to populate the grains for the master. Enable if you want to see GPU hardware
# data for your master.
# enable_gpu_grains: False

# The master maintains a job cache. While this is a great addition, it can be
# a burden on the master for larger deployments (over 5000 minions).
# Disabling the job cache will make previously executed jobs unavailable to
# the jobs system and is not generally recommended.
#job_cache: True

# Cache minion grains and pillar data in the cachedir.
#minion_data_cache: True

# Store all returns in the given returner.
# Setting this option requires that any returner-specific configuration also
# be set. See various returners in salt/returners for details on required
# configuration values. (See also, event_return_queue below.)
#
#event_return: mysql

# On busy systems, enabling event_returns can cause a considerable load on
# the storage system for returners. Events can be queued on the master and
# stored in a batched fashion using a single transaction for multiple events.
# By default, events are not queued.
#event_return_queue: 0

# Only events returns matching tags in a whitelist
# event_return_whitelist:
#   - salt/master/a_tag
#   - salt/master/another_tag

# Store all event returns _except_ the tags in a blacklist
# event_return_blacklist:
#   - salt/master/not_this_tag
#   - salt/master/or_this_one

# Passing very large events can cause the minion to consume large amounts of
# memory. This value tunes the maximum size of a message allowed onto the
# master event bus. The value is expressed in bytes.
#max_event_size: 1048576

# By default, the master AES key rotates every 24 hours. The next command
# following a key rotation will trigger a key refresh from the minion which may
# result in minions which do not respond to the first command after a key refresh.
#
# To tell the master to ping all minions immediately after an AES key refresh, set
# ping_on_rotate to True. This should mitigate the issue where a minion does not
# appear to initially respond after a key is rotated.
#
# Note that ping_on_rotate may cause high load on the master immediately after
```

```
# the key rotation event as minions reconnect. Consider this carefully if this
# salt master is managing a large number of minions.
#
# If disabled, it is recommended to handle this event by listening for the
# 'aes_key_rotate' event with the 'key' tag and acting appropriately.
# ping_on_rotate: False

# By default, the master deletes its cache of minion data when the key for that
# minion is removed. To preserve the cache after key deletion, set
# 'preserve_minion_cache' to True.
#
# WARNING: This may have security implications if compromised minions auth with
# a previous deleted minion ID.
#preserve_minion_cache: False

# If max_minions is used in large installations, the master might experience
# high-load situations because of having to check the number of connected
# minions for every authentication. This cache provides the minion-ids of
# all connected minions to all MWorker-processes and greatly improves the
# performance of max_minions.
# con_cache: False

# The master can include configuration from other files. To enable this,
# pass a list of paths to this option. The paths can be either relative or
# absolute; if relative, they are considered to be relative to the directory
# the main master configuration file lives in (this file). Paths can make use
# of shell-style globbing. If no files are matched by a path passed to this
# option, then the master will log a warning message.
#
# Include a config file from some other path:
# include: /etc/salt/extra_config
#
# Include config from several files and directories:
# include:
#   - /etc/salt/extra_config


#####        Security settings       #####
##########################################
# Enable "open mode", this mode still maintains encryption, but turns off
# authentication, this is only intended for highly secure environments or for
# the situation where your keys end up in a bad state. If you run in open mode
# you do so at your own risk!
#open_mode: False

# Enable auto_accept, this setting will automatically accept all incoming
# public keys from the minions. Note that this is insecure.
#auto_accept: False

# Time in minutes that a incoming public key with a matching name found in
# pki_dir/minion_autosign/keyid is automatically accepted. Expired autosign keys
# are removed when the master checks the minion_autosign directory.
# 0 equals no timeout
# autosign_timeout: 120

# If the autosign_file is specified, incoming keys specified in the
# autosign_file will be automatically accepted. This is insecure.  Regular
# expressions as well as globing lines are supported.
```

```
#autosign_file: /etc/salt/autosign.conf

# Works like autosign_file, but instead allows you to specify minion IDs for
# which keys will automatically be rejected. Will override both membership in
# the autosign_file and the auto_accept setting.
#autoreject_file: /etc/salt/autoreject.conf

# Enable permissive access to the salt keys. This allows you to run the
# master or minion as root, but have a non-root group be given access to
# your pki_dir. To make the access explicit, root must belong to the group
# you've given access to. This is potentially quite insecure. If an autosign_file
# is specified, enabling permissive_pki_access will allow group access to that
# specific file.
#permissive_pki_access: False

# Allow users on the master access to execute specific commands on minions.
# This setting should be treated with care since it opens up execution
# capabilities to non root users. By default this capability is completely
# disabled.
#client_acl:
#  larry:
#    - test.ping
#    - network.*
#
# Blacklist any of the following users or modules
#
# This example would blacklist all non sudo users, including root from
# running any commands. It would also blacklist any use of the "cmd"
# module. This is completely disabled by default.
#
#client_acl_blacklist:
#  users:
#    - root
#    - '^(?!sudo_).*$'   #  all non sudo users
#  modules:
#    - cmd

# Enforce client_acl & client_acl_blacklist when users have sudo
# access to the salt command.
#
#sudo_acl: False

# The external auth system uses the Salt auth modules to authenticate and
# validate users to access areas of the Salt system.
#external_auth:
#  pam:
#    fred:
#      - test.*
#
# Time (in seconds) for a newly generated token to live. Default: 12 hours
#token_expire: 43200

# Allow minions to push files to the master. This is disabled by default, for
# security purposes.
#file_recv: False

# Set a hard-limit on the size of the files that can be pushed to the master.
# It will be interpreted as megabytes. Default: 100
```

```
#file_recv_max_size: 100

# Signature verification on messages published from the master.
# This causes the master to cryptographically sign all messages published to its event
# bus, and minions then verify that signature before acting on the message.
#
# This is False by default.
#
# Note that to facilitate interoperability with masters and minions that are different
# versions, if sign_pub_messages is True but a message is received by a minion with
# no signature, it will still be accepted, and a warning message will be logged.
# Conversely, if sign_pub_messages is False, but a minion receives a signed
# message it will be accepted, the signature will not be checked, and a warning message
# will be logged. This behavior went away in Salt 2014.1.0 and these two situations
# will cause minion to throw an exception and drop the message.
# sign_pub_messages: False

#####     Salt-SSH Configuration     #####
##########################################

# Pass in an alternative location for the salt-ssh roster file
#roster_file: /etc/salt/roster

# Pass in minion option overrides that will be inserted into the SHIM for
# salt-ssh calls. The local minion config is not used for salt-ssh. Can be
# overridden on a per-minion basis in the roster (`minion_opts`)
#ssh_minion_opts:
#  gpg_keydir: /root/gpg

#####    Master Module Management    #####
##########################################
# Manage how master side modules are loaded.

# Add any additional locations to look for master runners:
#runner_dirs: []

# Enable Cython for master side modules:
#cython_enable: False


#####      State System settings     #####
##########################################
# The state system uses a "top" file to tell the minions what environment to
# use and what modules to use. The state_top file is defined relative to the
# root of the base environment as defined in "File Server settings" below.
#state_top: top.sls

# The master_tops option replaces the external_nodes option by creating
# a plugable system for the generation of external top data. The external_nodes
# option is deprecated by the master_tops option.
#
# To gain the capabilities of the classic external_nodes system, use the
# following configuration:
# master_tops:
#   ext_nodes: <Shell command which returns yaml>
#
#master_tops: {}
```

```
# The external_nodes option allows Salt to gather data that would normally be
# placed in a top file. The external_nodes option is the executable that will
# return the ENC data. Remember that Salt will look for external nodes AND top
# files and combine the results if both are enabled!
#external_nodes: None

# The renderer to use on the minions to render the state data
#renderer: yaml_jinja

# The Jinja renderer can strip extra carriage returns and whitespace
# See http://jinja.pocoo.org/docs/api/#high-level-api
#
# If this is set to True the first newline after a Jinja block is removed
# (block, not variable tag!). Defaults to False, corresponds to the Jinja
# environment init variable "trim_blocks".
# jinja_trim_blocks: False
#
# If this is set to True leading spaces and tabs are stripped from the start
# of a line to a block. Defaults to False, corresponds to the Jinja
# environment init variable "lstrip_blocks".
# jinja_lstrip_blocks: False

# The failhard option tells the minions to stop immediately after the first
# failure detected in the state execution, defaults to False
#failhard: False

# The state_verbose and state_output settings can be used to change the way
# state system data is printed to the display. By default all data is printed.
# The state_verbose setting can be set to True or False, when set to False
# all data that has a result of True and no changes will be suppressed.
#state_verbose: True

# The state_output setting changes if the output is the full multi line
# output for each changed state if set to 'full', but if set to 'terse'
# the output will be shortened to a single line.  If set to 'mixed', the output
# will be terse unless a state failed, in which case that output will be full.
# If set to 'changes', the output will be full unless the state didn't change.
#state_output: full

# Automatically aggregate all states that have support for mod_aggregate by
# setting to True. Or pass a list of state module names to automatically
# aggregate just those types.
#
# state_aggregate:
#   - pkg
#
#state_aggregate: False

#####      File Server settings      #####
##########################################
# Salt runs a lightweight file server written in zeromq to deliver files to
# minions. This file server is built into the master daemon and does not
# require a dedicated port.

# The file server works on environments passed to the master, each environment
# can have multiple root directories, the subdirectories in the multiple file
# roots cannot match, otherwise the downloaded files will not be able to be
# reliably ensured. A base environment is required to house the top file.
```

```
# Example:
# file_roots:
#   base:
#     - /srv/salt/
#   dev:
#     - /srv/salt/dev/services
#     - /srv/salt/dev/states
#   prod:
#     - /srv/salt/prod/services
#     - /srv/salt/prod/states
#
#file_roots:
#  base:
#    - /srv/salt

# The hash_type is the hash to use when discovering the hash of a file on
# the master server. The default is md5, but sha1, sha224, sha256, sha384
# and sha512 are also supported.
#
# Prior to changing this value, the master should be stopped and all Salt
# caches should be cleared.
#hash_type: md5

# The buffer size in the file server can be adjusted here:
#file_buffer_size: 1048576

# A regular expression (or a list of expressions) that will be matched
# against the file path before syncing the modules and states to the minions.
# This includes files affected by the file.recurse state.
# For example, if you manage your custom modules and states in subversion
# and don't want all the '.svn' folders and content synced to your minions,
# you could set this to '/\.svn($|/)'. By default nothing is ignored.
#file_ignore_regex:
#  - '/\.svn($|/)'
#  - '/\.git($|/)'

# A file glob (or list of file globs) that will be matched against the file
# path before syncing the modules and states to the minions. This is similar
# to file_ignore_regex above, but works on globs instead of regex. By default
# nothing is ignored.
# file_ignore_glob:
#  - '*.pyc'
#  - '*/somefolder/*.bak'
#  - '*.swp'

# File Server Backend
#
# Salt supports a modular fileserver backend system, this system allows
# the salt master to link directly to third party systems to gather and
# manage the files available to minions. Multiple backends can be
# configured and will be searched for the requested file in the order in which
# they are defined here. The default setting only enables the standard backend
# "roots" which uses the "file_roots" option.
#fileserver_backend:
#  - roots
#
# To use multiple backends list them in the order they are searched:
#fileserver_backend:
```

```
#  - git
#  - roots
#
# Uncomment the line below if you do not want the file_server to follow
# symlinks when walking the filesystem tree. This is set to True
# by default. Currently this only applies to the default roots
# fileserver_backend.
#fileserver_followsymlinks: False
#
# Uncomment the line below if you do not want symlinks to be
# treated as the files they are pointing to. By default this is set to
# False. By uncommenting the line below, any detected symlink while listing
# files on the Master will not be returned to the Minion.
#fileserver_ignoresymlinks: True
#
# By default, the Salt fileserver recurses fully into all defined environments
# to attempt to find files. To limit this behavior so that the fileserver only
# traverses directories with SLS files and special Salt directories like _modules,
# enable the option below. This might be useful for installations where a file root
# has a very large number of files and performance is impacted. Default is False.
# fileserver_limit_traversal: False
#
# The fileserver can fire events off every time the fileserver is updated,
# these are disabled by default, but can be easily turned on by setting this
# flag to True
#fileserver_events: False

# Git File Server Backend Configuration
#
# Gitfs can be provided by one of two python modules: GitPython or pygit2. If
# using pygit2, both libgit2 and git must also be installed.
#gitfs_provider: gitpython
#
# When using the git fileserver backend at least one git remote needs to be
# defined. The user running the salt master will need read access to the repo.
#
# The repos will be searched in order to find the file requested by a client
# and the first repo to have the file will return it.
# When using the git backend branches and tags are translated into salt
# environments.
# Note:  file:// repos will be treated as a remote, so refs you want used must
# exist in that repo as *local* refs.
#gitfs_remotes:
#  - git://github.com/saltstack/salt-states.git
#  - file:///var/git/saltmaster
#
# The gitfs_ssl_verify option specifies whether to ignore ssl certificate
# errors when contacting the gitfs backend. You might want to set this to
# false if you're using a git backend that uses a self-signed certificate but
# keep in mind that setting this flag to anything other than the default of True
# is a security concern, you may want to try using the ssh transport.
#gitfs_ssl_verify: True
#
# The gitfs_root option gives the ability to serve files from a subdirectory
# within the repository. The path is defined relative to the root of the
# repository and defaults to the repository root.
#gitfs_root: somefolder/otherfolder
#
```

```
#
#####         Pillar settings        #####
##########################################
# Salt Pillars allow for the building of global data that can be made selectively
# available to different minions based on minion grain filtering. The Salt
# Pillar is laid out in the same fashion as the file server, with environments,
# a top file and sls files. However, pillar data does not need to be in the
# highstate format, and is generally just key/value pairs.
#pillar_roots:
#  base:
#    - /srv/pillar
#
#ext_pillar:
#  - hiera: /etc/hiera.yaml
#  - cmd_yaml: cat /etc/salt/yaml

# The ext_pillar_first option allows for external pillar sources to populate
# before file system pillar. This allows for targeting file system pillar from
# ext_pillar.
#ext_pillar_first: False

# The pillar_gitfs_ssl_verify option specifies whether to ignore ssl certificate
# errors when contacting the pillar gitfs backend. You might want to set this to
# false if you're using a git backend that uses a self-signed certificate but
# keep in mind that setting this flag to anything other than the default of True
# is a security concern, you may want to try using the ssh transport.
#pillar_gitfs_ssl_verify: True

# The pillar_opts option adds the master configuration file data to a dict in
# the pillar called "master". This is used to set simple configurations in the
# master config file that can then be used on minions.
#pillar_opts: False

# The pillar_safe_render_error option prevents the master from passing piller
# render errors to the minion. This is set on by default because the error could
# contain templating data which would give that minion information it shouldn't
# have, like a password! When set true the error message will only show:
#   Rendering SLS 'my.sls' failed. Please see master log for details.
#pillar_safe_render_error: True

# The pillar_source_merging_strategy option allows you to configure merging strategy
# between different sources. It accepts four values: recurse, aggregate, overwrite,
# or smart. Recurse will merge recursively mapping of data. Aggregate instructs
# aggregation of elements between sources that use the #!yamlex renderer. Overwrite
# will verwrite elements according the order in which they are processed. This is
# behavior of the 2014.1 branch and earlier. Smart guesses the best strategy based
# on the "renderer" setting and is the default value.
#pillar_source_merging_strategy: smart


#####          Syndic settings       #####
##########################################
# The Salt syndic is used to pass commands through a master from a higher
# master. Using the syndic is simple, if this is a master that will have
# syndic servers(s) below it set the "order_masters" setting to True, if this
# is a master that will be running a syndic daemon for passthrough the
# "syndic_master" setting needs to be set to the location of the master server
# to receive commands from.
```

```
# Set the order_masters setting to True if this master will command lower
# masters' syndic interfaces.
#order_masters: False

# If this master will be running a salt syndic daemon, syndic_master tells
# this master where to receive commands from.
#syndic_master: masterofmaster

# This is the 'ret_port' of the MasterOfMaster:
#syndic_master_port: 4506

# PID file of the syndic daemon:
#syndic_pidfile: /var/run/salt-syndic.pid

# LOG file of the syndic daemon:
#syndic_log_file: syndic.log


#####      Peer Publish settings     #####
##########################################
# Salt minions can send commands to other minions, but only if the minion is
# allowed to. By default "Peer Publication" is disabled, and when enabled it
# is enabled for specific minions and specific commands. This allows secure
# compartmentalization of commands based on individual minions.

# The configuration uses regular expressions to match minions and then a list
# of regular expressions to match functions. The following will allow the
# minion authenticated as foo.example.com to execute functions from the test
# and pkg modules.
#peer:
#  foo.example.com:
#    - test.*
#    - pkg.*
#
# This will allow all minions to execute all commands:
#peer:
#  .*:
#    - .*
#
# This is not recommended, since it would allow anyone who gets root on any
# single minion to instantly have root on all of the minions!

# Minions can also be allowed to execute runners from the salt master.
# Since executing a runner from the minion could be considered a security risk,
# it needs to be enabled. This setting functions just like the peer setting
# except that it opens up runners instead of module functions.
#
# All peer runner support is turned off by default and must be enabled before
# using. This will enable all peer runners for all minions:
#peer_run:
#  .*:
#    - .*
#
# To enable just the manage.up runner for the minion foo.example.com:
#peer_run:
#  foo.example.com:
#    - manage.up
#
```

```
#
#####         Mine settings     #####
##########################################
# Restrict mine.get access from minions. By default any minion has a full access
# to get all mine data from master cache. In acl definion below, only pcre matches
# are allowed.
# mine_get:
#   .*:
#     - .*
#
# The example below enables minion foo.example.com to get 'network.interfaces' mine
# data only, minions web* to get all network.* and disk.* mine data and all other
# minions won't get any mine data.
# mine_get:
#   foo.example.com:
#     - network.interfaces
#   web.*:
#     - network.*
#     - disk.*


#####         Logging settings       #####
##########################################
# The location of the master log file
# The master log can be sent to a regular file, local path name, or network
# location. Remote logging works best when configured to use rsyslogd(8) (e.g.:
# ``file:///dev/log``), with rsyslogd(8) configured for network logging. The URI
# format is: <file|udp|tcp>://<host|socketpath>:<port-if-required>/<log-facility>
#log_file: /var/log/salt/master
#log_file: file:///dev/log
#log_file: udp://loghost:10514

#log_file: /var/log/salt/master
#key_logfile: /var/log/salt/key

# The level of messages to send to the console.
# One of 'garbage', 'trace', 'debug', info', 'warning', 'error', 'critical'.
#log_level: warning

# The level of messages to send to the log file.
# One of 'garbage', 'trace', 'debug', info', 'warning', 'error', 'critical'.
# If using 'log_granular_levels' this must be set to the highest desired level.
#log_level_logfile: warning

# The date and time format used in log messages. Allowed date/time formating
# can be seen here: http://docs.python.org/library/time.html#time.strftime
#log_datefmt: '%H:%M:%S'
#log_datefmt_logfile: '%Y-%m-%d %H:%M:%S'

# The format of the console logging messages. Allowed formatting options can
# be seen here: http://docs.python.org/library/logging.html#logrecord-attributes
#log_fmt_console: '[%(levelname)-8s] %(message)s'
#log_fmt_logfile: '%(asctime)s,%(msecs)03.0f [%(name)-17s][%(levelname)-8s] %(message)s'

# This can be used to control logging levels more specificically.  This
# example sets the main salt library at the 'warning' level, but sets
# 'salt.modules' to log at the 'debug' level:
#   log_granular_levels:
```

```
#     'salt': 'warning'
#     'salt.modules': 'debug'
#
#log_granular_levels: {}


#####         Node Groups           #####
##########################################
# Node groups allow for logical groupings of minion nodes. A group consists of a group
# name and a compound target.
#nodegroups:
#  group1: 'L@foo.domain.com,bar.domain.com,baz.domain.com and bl*.domain.com'
#  group2: 'G@os:Debian and foo.domain.com'


#####     Range Cluster settings     #####
##########################################
# The range server (and optional port) that serves your cluster information
# https://github.com/ytoolshed/range/wiki/%22yamlfile%22-module-file-spec
#
#range_server: range:80


#####     Windows Software Repo settings #####
################################################
# Location of the repo on the master:
#win_repo: '/srv/salt/win/repo'
#
# Location of the master's repo cache file:
#win_repo_mastercachefile: '/srv/salt/win/repo/winrepo.p'
#
# List of git repositories to include with the local repo:
#win_gitrepos:
#  - 'https://github.com/saltstack/salt-winrepo.git'

#####      Returner settings          ######
############################################
# Which returner(s) will be used for minion's result:
#return: mysql
```

22.6.2 Example minion configuration file

```
##### Primary configuration settings #####
##########################################
# This configuration file is used to manage the behavior of the Salt Minion.
# With the exception of the location of the Salt Master Server, values that are
# commented out but have an empty line after the comment are defaults that need
# not be set in the config. If there is no blank line after the comment, the
# value is presented as an example and is not the default.

# Per default the minion will automatically include all config files
# from minion.d/*.conf (minion.d is a directory in the same directory
# as the main minion config file).
#default_include: minion.d/*.conf

# Set the location of the salt master server. If the master server cannot be
```

```
# resolved, then the minion will fail to start.
#master: salt

# If multiple masters are specified in the 'master' setting, the default behavior
# is to always try to connect to them in the order they are listed. If random_master is
# set to True, the order will be randomized instead. This can be helpful in distributing
# the load of many minions executing salt-call requests, for example, from a cron job.
# If only one master is listed, this setting is ignored and a warning will be logged.
#random_master: False

# Set whether the minion should connect to the master via IPv6:
#ipv6: False

# Set the number of seconds to wait before attempting to resolve
# the master hostname if name resolution fails. Defaults to 30 seconds.
# Set to zero if the minion should shutdown and not retry.
# retry_dns: 30

# Set the port used by the master reply and authentication server.
#master_port: 4506

# The user to run salt.
#user: root

# Specify the location of the daemon process ID file.
#pidfile: /var/run/salt-minion.pid

# The root directory prepended to these options: pki_dir, cachedir, log_file,
# sock_dir, pidfile.
#root_dir: /

# The directory to store the pki information in
#pki_dir: /etc/salt/pki/minion

# Explicitly declare the id for this minion to use, if left commented the id
# will be the hostname as returned by the python call: socket.getfqdn()
# Since salt uses detached ids it is possible to run multiple minions on the
# same machine but with different ids, this can be useful for salt compute
# clusters.
#id:

# Append a domain to a hostname in the event that it does not exist.  This is
# useful for systems where socket.getfqdn() does not actually result in a
# FQDN (for instance, Solaris).
#append_domain:

# Custom static grains for this minion can be specified here and used in SLS
# files just like all other grains. This example sets 4 custom grains, with
# the 'roles' grain having two values that can be matched against.
#grains:
#  roles:
#    - webserver
#    - memcache
#  deployment: datacenter4
#  cabinet: 13
#  cab_u: 14-15
#
# Where cache data goes.
```

```
#cachedir: /var/cache/salt/minion

# Verify and set permissions on configuration directories at startup.
#verify_env: True

# The minion can locally cache the return data from jobs sent to it, this
# can be a good way to keep track of jobs the minion has executed
# (on the minion side). By default this feature is disabled, to enable, set
# cache_jobs to True.
#cache_jobs: False

# Set the directory used to hold unix sockets.
#sock_dir: /var/run/salt/minion

# Set the default outputter used by the salt-call command. The default is
# "nested".
#output: nested
#
# By default output is colored. To disable colored output, set the color value
# to False.
#color: True

# Do not strip off the colored output from nested results and state outputs
# (true by default).
# strip_colors: False

# Backup files that are replaced by file.managed and file.recurse under
# 'cachedir'/file_backups relative to their original location and appended
# with a timestamp. The only valid setting is "minion". Disabled by default.
#
# Alternatively this can be specified for each file in state files:
# /etc/ssh/sshd_config:
#   file.managed:
#     - source: salt://ssh/sshd_config
#     - backup: minion
#
#backup_mode: minion

# When waiting for a master to accept the minion's public key, salt will
# continuously attempt to reconnect until successful. This is the time, in
# seconds, between those reconnection attempts.
#acceptance_wait_time: 10

# If this is nonzero, the time between reconnection attempts will increase by
# acceptance_wait_time seconds per iteration, up to this maximum. If this is
# set to zero, the time between reconnection attempts will stay constant.
#acceptance_wait_time_max: 0

# If the master rejects the minion's public key, retry instead of exiting.
# Rejected keys will be handled the same as waiting on acceptance.
#rejected_retry: False

# When the master key changes, the minion will try to re-auth itself to receive
# the new master key. In larger environments this can cause a SYN flood on the
# master because all minions try to re-auth immediately. To prevent this and
# have a minion wait for a random amount of time, use this optional parameter.
# The wait-time will be a random number of seconds between 0 and the defined value.
#random_reauth_delay: 60
```

```
# When waiting for a master to accept the minion's public key, salt will
# continuously attempt to reconnect until successful. This is the timeout value,
# in seconds, for each individual attempt. After this timeout expires, the minion
# will wait for acceptance_wait_time seconds before trying again. Unless your master
# is under unusually heavy load, this should be left at the default.
#auth_timeout: 60

# Number of consecutive SaltReqTimeoutError that are acceptable when trying to
# authenticate.
#auth_tries: 7

# If authentication fails due to SaltReqTimeoutError during a ping_interval,
# cause sub minion process to restart.
#auth_safemode: False

# Ping Master to ensure connection is alive (minutes).
#ping_interval: 0

# To auto recover minions if master changes IP address (DDNS)
#    auth_tries: 10
#    auth_safemode: False
#    ping_interval: 90
#
# Minions won't know master is missing until a ping fails. After the ping fail,
# the minion will attempt authentication and likely fails out and cause a restart.
# When the minion restarts it will resolve the masters IP and attempt to reconnect.

# If you don't have any problems with syn-floods, don't bother with the
# three recon_* settings described below, just leave the defaults!
#
# The ZeroMQ pull-socket that binds to the masters publishing interface tries
# to reconnect immediately, if the socket is disconnected (for example if
# the master processes are restarted). In large setups this will have all
# minions reconnect immediately which might flood the master (the ZeroMQ-default
# is usually a 100ms delay). To prevent this, these three recon_* settings
# can be used.
# recon_default: the interval in milliseconds that the socket should wait before
#                trying to reconnect to the master (1000ms = 1 second)
#
# recon_max: the maximum time a socket should wait. each interval the time to wait
#            is calculated by doubling the previous time. if recon_max is reached,
#            it starts again at recon_default. Short example:
#
#            reconnect 1: the socket will wait 'recon_default' milliseconds
#            reconnect 2: 'recon_default' * 2
#            reconnect 3: ('recon_default' * 2) * 2
#            reconnect 4: value from previous interval * 2
#            reconnect 5: value from previous interval * 2
#            reconnect x: if value >= recon_max, it starts again with recon_default
#
# recon_randomize: generate a random wait time on minion start. The wait time will
#                  be a random value between recon_default and recon_default +
#                  recon_max. Having all minions reconnect with the same recon_default
#                  and recon_max value kind of defeats the purpose of being able to
#                  change these settings. If all minions have the same values and your
#                  setup is quite large (several thousand minions), they will still
#                  flood the master. The desired behavior is to have timeframe within
#                  all minions try to reconnect.
```

```
#
# Example on how to use these settings. The goal: have all minions reconnect within a
# 60 second timeframe on a disconnect.
# recon_default: 1000
# recon_max: 59000
# recon_randomize: True
#
# Each minion will have a randomized reconnect value between 'recon_default'
# and 'recon_default + recon_max', which in this example means between 1000ms
# 60000ms (or between 1 and 60 seconds). The generated random-value will be
# doubled after each attempt to reconnect. Lets say the generated random
# value is 11 seconds (or 11000ms).
# reconnect 1: wait 11 seconds
# reconnect 2: wait 22 seconds
# reconnect 3: wait 33 seconds
# reconnect 4: wait 44 seconds
# reconnect 5: wait 55 seconds
# reconnect 6: wait time is bigger than 60 seconds (recon_default + recon_max)
# reconnect 7: wait 11 seconds
# reconnect 8: wait 22 seconds
# reconnect 9: wait 33 seconds
# reconnect x: etc.
#
# In a setup with ~6000 thousand hosts these settings would average the reconnects
# to about 100 per second and all hosts would be reconnected within 60 seconds.
# recon_default: 100
# recon_max: 5000
# recon_randomize: False
#
#
# The loop_interval sets how long in seconds the minion will wait between
# evaluating the scheduler and running cleanup tasks. This defaults to a
# sane 60 seconds, but if the minion scheduler needs to be evaluated more
# often lower this value
#loop_interval: 60

# The grains_refresh_every setting allows for a minion to periodically check
# its grains to see if they have changed and, if so, to inform the master
# of the new grains. This operation is moderately expensive, therefore
# care should be taken not to set this value too low.
#
# Note: This value is expressed in __minutes__!
#
# A value of 10 minutes is a reasonable default.
#
# If the value is set to zero, this check is disabled.
#grains_refresh_every: 1

# Cache grains on the minion. Default is False.
#grains_cache: False

# Grains cache expiration, in seconds. If the cache file is older than this
# number of seconds then the grains cache will be dumped and fully re-populated
# with fresh data. Defaults to 5 minutes. Will have no effect if 'grains_cache'
# is not enabled.
# grains_cache_expiration: 300

# Windows platforms lack posix IPC and must rely on slower TCP based inter-
```

```
# process communications. Set ipc_mode to 'tcp' on such systems
#ipc_mode: ipc

# Overwrite the default tcp ports used by the minion when in tcp mode
#tcp_pub_port: 4510
#tcp_pull_port: 4511

# Passing very large events can cause the minion to consume large amounts of
# memory. This value tunes the maximum size of a message allowed onto the
# minion event bus. The value is expressed in bytes.
#max_event_size: 1048576

# To detect failed master(s) and fire events on connect/disconnect, set
# master_alive_interval to the number of seconds to poll the masters for
# connection events.
#
#master_alive_interval: 30

# The minion can include configuration from other files. To enable this,
# pass a list of paths to this option. The paths can be either relative or
# absolute; if relative, they are considered to be relative to the directory
# the main minion configuration file lives in (this file). Paths can make use
# of shell-style globbing. If no files are matched by a path passed to this
# option then the minion will log a warning message.
#
# Include a config file from some other path:
# include: /etc/salt/extra_config
#
# Include config from several files and directories:
#include:
#  - /etc/salt/extra_config
#  - /etc/roles/webserver
#
#
#
#####   Minion module management     #####
##########################################
# Disable specific modules. This allows the admin to limit the level of
# access the master has to the minion.
#disable_modules: [cmd,test]
#disable_returners: []
#
# Modules can be loaded from arbitrary paths. This enables the easy deployment
# of third party modules. Modules for returners and minions can be loaded.
# Specify a list of extra directories to search for minion modules and
# returners. These paths must be fully qualified!
#module_dirs: []
#returner_dirs: []
#states_dirs: []
#render_dirs: []
#utils_dirs: []
#
# A module provider can be statically overwritten or extended for the minion
# via the providers option, in this case the default module will be
# overwritten by the specified module. In this example the pkg module will
# be provided by the yumpkg5 module instead of the system default.
#providers:
#  pkg: yumpkg5
```

```
#
# Enable Cython modules searching and loading. (Default: False)
#cython_enable: False
#
# Specify a max size (in bytes) for modules on import. This feature is currently
# only supported on *nix operating systems and requires psutil.
# modules_max_memory: -1


#####    State Management Settings    #####
###########################################
# The state management system executes all of the state templates on the minion
# to enable more granular control of system state management. The type of
# template and serialization used for state management needs to be configured
# on the minion, the default renderer is yaml_jinja. This is a yaml file
# rendered from a jinja template, the available options are:
# yaml_jinja
# yaml_mako
# yaml_wempy
# json_jinja
# json_mako
# json_wempy
#
#renderer: yaml_jinja
#
# The failhard option tells the minions to stop immediately after the first
# failure detected in the state execution. Defaults to False.
#failhard: False
#
# autoload_dynamic_modules turns on automatic loading of modules found in the
# environments on the master. This is turned on by default. To turn of
# autoloading modules when states run, set this value to False.
#autoload_dynamic_modules: True
#
# clean_dynamic_modules keeps the dynamic modules on the minion in sync with
# the dynamic modules on the master, this means that if a dynamic module is
# not on the master it will be deleted from the minion. By default, this is
# enabled and can be disabled by changing this value to False.
#clean_dynamic_modules: True
#
# Normally, the minion is not isolated to any single environment on the master
# when running states, but the environment can be isolated on the minion side
# by statically setting it. Remember that the recommended way to manage
# environments is to isolate via the top file.
#environment: None
#
# If using the local file directory, then the state top file name needs to be
# defined, by default this is top.sls.
#state_top: top.sls
#
# Run states when the minion daemon starts. To enable, set startup_states to:
# 'highstate' -- Execute state.highstate
# 'sls' -- Read in the sls_list option and execute the named sls files
# 'top' -- Read top_file option and execute based on that file on the Master
#startup_states: ''
#
# List of states to run when the minion starts up if startup_states is 'sls':
#sls_list:
```

```
#  - edit.vim
#  - hyper
#
# Top file to execute if startup_states is 'top':
#top_file: ''

# Automatically aggregate all states that have support for mod_aggregate by
# setting to True. Or pass a list of state module names to automatically
# aggregate just those types.
#
# state_aggregate:
#   - pkg
#
#state_aggregate: False

#####     File Directory Settings    #####
##############################################
# The Salt Minion can redirect all file server operations to a local directory,
# this allows for the same state tree that is on the master to be used if
# copied completely onto the minion. This is a literal copy of the settings on
# the master but used to reference a local directory on the minion.

# Set the file client. The client defaults to looking on the master server for
# files, but can be directed to look at the local file directory setting
# defined below by setting it to local.
#file_client: remote

# The file directory works on environments passed to the minion, each environment
# can have multiple root directories, the subdirectories in the multiple file
# roots cannot match, otherwise the downloaded files will not be able to be
# reliably ensured. A base environment is required to house the top file.
# Example:
# file_roots:
#   base:
#     - /srv/salt/
#   dev:
#     - /srv/salt/dev/services
#     - /srv/salt/dev/states
#   prod:
#     - /srv/salt/prod/services
#     - /srv/salt/prod/states
#
#file_roots:
#  base:
#    - /srv/salt

# By default, the Salt fileserver recurses fully into all defined environments
# to attempt to find files. To limit this behavior so that the fileserver only
# traverses directories with SLS files and special Salt directories like _modules,
# enable the option below. This might be useful for installations where a file root
# has a very large number of files and performance is negatively impacted. Default
# is False.
#fileserver_limit_traversal: False

# The hash_type is the hash to use when discovering the hash of a file in
# the local fileserver. The default is md5, but sha1, sha224, sha256, sha384
# and sha512 are also supported.
#
```

```
# Warning: Prior to changing this value, the minion should be stopped and all
# Salt caches should be cleared.
#hash_type: md5

# The Salt pillar is searched for locally if file_client is set to local. If
# this is the case, and pillar data is defined, then the pillar_roots need to
# also be configured on the minion:
#pillar_roots:
#  base:
#    - /srv/pillar
#
#
######        Security settings       #####
###########################################
# Enable "open mode", this mode still maintains encryption, but turns off
# authentication, this is only intended for highly secure environments or for
# the situation where your keys end up in a bad state. If you run in open mode
# you do so at your own risk!
#open_mode: False

# Enable permissive access to the salt keys.  This allows you to run the
# master or minion as root, but have a non-root group be given access to
# your pki_dir.  To make the access explicit, root must belong to the group
# you've given access to. This is potentially quite insecure.
#permissive_pki_access: False

# The state_verbose and state_output settings can be used to change the way
# state system data is printed to the display. By default all data is printed.
# The state_verbose setting can be set to True or False, when set to False
# all data that has a result of True and no changes will be suppressed.
#state_verbose: True

# The state_output setting changes if the output is the full multi line
# output for each changed state if set to 'full', but if set to 'terse'
# the output will be shortened to a single line.
#state_output: full

# The state_output_diff setting changes whether or not the output from
# successful states is returned. Useful when even the terse output of these
# states is cluttering the logs. Set it to True to ignore them.
#state_output_diff: False

# Fingerprint of the master public key to double verify the master is valid,
# the master fingerprint can be found by running "salt-key -F master" on the
# salt master.
#master_finger: ''


######         Thread settings        #####
###########################################
# Disable multiprocessing support, by default when a minion receives a
# publication a new process is spawned and the command is executed therein.
#multiprocessing: True


#####         Logging settings       #####
###########################################
# The location of the minion log file
```

```
# The minion log can be sent to a regular file, local path name, or network
# location. Remote logging works best when configured to use rsyslogd(8) (e.g.:
# ``file:///dev/log``), with rsyslogd(8) configured for network logging. The URI
# format is: <file|udp|tcp>://<host|socketpath>:<port-if-required>/<log-facility>
#log_file: /var/log/salt/minion
#log_file: file:///dev/log
#log_file: udp://loghost:10514
#
#log_file: /var/log/salt/minion
#key_logfile: /var/log/salt/key

# The level of messages to send to the console.
# One of 'garbage', 'trace', 'debug', info', 'warning', 'error', 'critical'.
# Default: 'warning'
#log_level: warning

# The level of messages to send to the log file.
# One of 'garbage', 'trace', 'debug', info', 'warning', 'error', 'critical'.
# If using 'log_granular_levels' this must be set to the highest desired level.
# Default: 'warning'
#log_level_logfile:

# The date and time format used in log messages. Allowed date/time formating
# can be seen here: http://docs.python.org/library/time.html#time.strftime
#log_datefmt: '%H:%M:%S'
#log_datefmt_logfile: '%Y-%m-%d %H:%M:%S'

# The format of the console logging messages. Allowed formatting options can
# be seen here: http://docs.python.org/library/logging.html#logrecord-attributes
#log_fmt_console: '[%(levelname)-8s] %(message)s'
#log_fmt_logfile: '%(asctime)s,%(msecs)03.0f [%(name)-17s][%(levelname)-8s] %(message)s'

# This can be used to control logging levels more specificically.  This
# example sets the main salt library at the 'warning' level, but sets
# 'salt.modules' to log at the 'debug' level:
#   log_granular_levels:
#     'salt': 'warning'
#     'salt.modules': 'debug'
#
#log_granular_levels: {}

# To diagnose issues with minions disconnecting or missing returns, ZeroMQ
# supports the use of monitor sockets # to log connection events. This
# feature requires ZeroMQ 4.0 or higher.
#
# To enable ZeroMQ monitor sockets, set 'zmq_monitor' to 'True' and log at a
# debug level or higher.
#
# A sample log event is as follows:
#
# [DEBUG   ] ZeroMQ event: {'endpoint': 'tcp://127.0.0.1:4505', 'event': 512,
# 'value': 27, 'description': 'EVENT_DISCONNECTED'}
#
# All events logged will include the string 'ZeroMQ event'. A connection event
# should be logged on the as the minion starts up and initially connects to the
# master. If not, check for debug log level and that the necessary version of
# ZeroMQ is installed.
#
```

```
#zmq_monitor: False

######      Module configuration      #####
############################################
# Salt allows for modules to be passed arbitrary configuration data, any data
# passed here in valid yaml format will be passed on to the salt minion modules
# for use. It is STRONGLY recommended that a naming convention be used in which
# the module name is followed by a . and then the value. Also, all top level
# data must be applied via the yaml dict construct, some examples:
#
# You can specify that all modules should run in test mode:
#test: True
#
# A simple value for the test module:
#test.foo: foo
#
# A list for the test module:
#test.bar: [baz,quo]
#
# A dict for the test module:
#test.baz: {spam: sausage, cheese: bread}
#
#
######      Update settings          ######
############################################
# Using the features in Esky, a salt minion can both run as a frozen app and
# be updated on the fly. These options control how the update process
# (saltutil.update()) behaves.
#
# The url for finding and downloading updates. Disabled by default.
#update_url: False
#
# The list of services to restart after a successful update. Empty by default.
#update_restart_services: []


######      Keepalive settings        ######
############################################
# ZeroMQ now includes support for configuring SO_KEEPALIVE if supported by
# the OS. If connections between the minion and the master pass through
# a state tracking device such as a firewall or VPN gateway, there is
# the risk that it could tear down the connection the master and minion
# without informing either party that their connection has been taken away.
# Enabling TCP Keepalives prevents this from happening.

# Overall state of TCP Keepalives, enable (1 or True), disable (0 or False)
# or leave to the OS defaults (-1), on Linux, typically disabled. Default True, enabled.
#tcp_keepalive: True

# How long before the first keepalive should be sent in seconds. Default 300
# to send the first keepalive after 5 minutes, OS default (-1) is typically 7200 seconds
# on Linux see /proc/sys/net/ipv4/tcp_keepalive_time.
#tcp_keepalive_idle: 300

# How many lost probes are needed to consider the connection lost. Default -1
# to use OS defaults, typically 9 on Linux, see /proc/sys/net/ipv4/tcp_keepalive_probes.
#tcp_keepalive_cnt: -1
```

```
# How often, in seconds, to send keepalives after the first one. Default -1 to
# use OS defaults, typically 75 seconds on Linux, see
# /proc/sys/net/ipv4/tcp_keepalive_intvl.
#tcp_keepalive_intvl: -1


######      Windows Software settings ######
############################################
# Location of the repository cache file on the master:
#win_repo_cachefile: 'salt://win/repo/winrepo.p'


######      Returner  settings        ######
############################################
# Which returner(s) will be used for minion's result:
#return: mysql
```

22.7 Configuring Salt

Salt configuration is very simple. The default configuration for the *master* will work for most installations and the only requirement for setting up a *minion* is to set the location of the master in the minion configuration file.

The configuration files will be installed to `/etc/salt` and are named after the respective components, `/etc/salt/master`, and `/etc/salt/minion`.

22.7.1 Master Configuration

By default the Salt master listens on ports 4505 and 4506 on all interfaces (0.0.0.0). To bind Salt to a specific IP, redefine the ``interface'' directive in the master configuration file, typically `/etc/salt/master`, as follows:

```
- #interface: 0.0.0.0
+ interface: 10.0.0.1
```

After updating the configuration file, restart the Salt master. See the *master configuration reference* for more details about other configurable options.

22.7.2 Minion Configuration

Although there are many Salt Minion configuration options, configuring a Salt Minion is very simple. By default a Salt Minion will try to connect to the DNS name ``salt''; if the Minion is able to resolve that name correctly, no configuration is needed.

If the DNS name ``salt'' does not resolve to point to the correct location of the Master, redefine the ``master'' directive in the minion configuration file, typically `/etc/salt/minion`, as follows:

```
- #master: salt
+ master: 10.0.0.1
```

After updating the configuration file, restart the Salt minion. See the *minion configuration reference* for more details about other configurable options.

22.7.3 Running Salt

1. Start the master in the foreground (to daemonize the process, pass the `-d flag`):

```
salt-master
```

2. Start the minion in the foreground (to daemonize the process, pass the `-d flag`):

```
salt-minion
```

Having trouble?

The simplest way to troubleshoot Salt is to run the master and minion in the foreground with `log level` set to `debug`:

```
salt-master --log-level=debug
```

For information on salt's logging system please see the *logging document*.

Run as an unprivileged (non-root) user

To run Salt as another user, set the `user` parameter in the master config file.

Additionally, ownership, and permissions need to be set such that the desired user can read from and write to the following directories (and their subdirectories, where applicable):

- /etc/salt
- /var/cache/salt
- /var/log/salt
- /var/run/salt

More information about running salt as a non-privileged user can be found *here*.

There is also a full *troubleshooting guide* available.

22.7.4 Key Management

Salt uses AES encryption for all communication between the Master and the Minion. This ensures that the commands sent to the Minions cannot be tampered with, and that communication between Master and Minion is authenticated through trusted, accepted keys.

Before commands can be sent to a Minion, its key must be accepted on the Master. Run the `salt-key` command to list the keys known to the Salt Master:

```
[root@master ~]# salt-key -L
Unaccepted Keys:
alpha
bravo
charlie
delta
Accepted Keys:
```

This example shows that the Salt Master is aware of four Minions, but none of the keys has been accepted. To accept the keys and allow the Minions to be controlled by the Master, again use the `salt-key` command:

```
[root@master ~]# salt-key -A
[root@master ~]# salt-key -L
Unaccepted Keys:
Accepted Keys:
alpha
bravo
charlie
delta
```

The `salt-key` command allows for signing keys individually or in bulk. The example above, using `-A` bulk-accepts all pending keys. To accept keys individually use the lowercase of the same option, `-a keyname`.

See also:

salt-key manpage

22.7.5 Sending Commands

Communication between the Master and a Minion may be verified by running the `test.ping` command:

```
[root@master ~]# salt alpha test.ping
alpha:
    True
```

Communication between the Master and all Minions may be tested in a similar way:

```
[root@master ~]# salt '*' test.ping
alpha:
    True
bravo:
    True
charlie:
    True
delta:
    True
```

Each of the Minions should send a `True` response as shown above.

22.7.6 What's Next?

Understanding *targeting* is important. From there, depending on the way you wish to use Salt, you should also proceed to learn about *States* and *Execution Modules*.

22.8 Configuring the Salt Master

The Salt system is amazingly simple and easy to configure, the two components of the Salt system each have a respective configuration file. The **`salt-master`** is configured via the master configuration file, and the **`salt-minion`** is configured via the minion configuration file.

See also:

example master configuration file

The configuration file for the salt-master is located at `/etc/salt/master` by default. A notable exception is FreeBSD, where the configuration file is located at `/usr/local/etc/salt`. The available options are as follows:

22.8.1 Primary Master Configuration

interface

Default: `0.0.0.0` (all interfaces)

The local interface to bind to.

```
interface: 192.168.0.1
```

ipv6

Default: `False`

Whether the master should listen for IPv6 connections. If this is set to True, the interface option must be adjusted too (for example: ``interface: `::'")

```
ipv6: True
```

publish_port

Default: `4505`

The network port to set up the publication interface.

```
publish_port: 4505
```

master_id

Default: `None`

The id to be passed in the publish job to minions. This is used for MultiSyndics to return the job to the requesting master.

Note: This must be the same string as the syndic is configured with.

```
master_id: MasterOfMaster
```

user

Default: `root`

The user to run the Salt processes

```
user: root
```

max_open_files

Default: `100000`

Each minion connecting to the master uses AT LEAST one file descriptor, the master subscription connection. If enough minions connect you might start seeing on the console(and then salt-master crashes):

```
Too many open files (tcp_listener.cpp:335)
Aborted (core dumped)
```

```
max_open_files: 100000
```

By default this value will be the one of *ulimit -Hn*, i.e., the hard limit for max open files.

To set a different value than the default one, uncomment, and configure this setting. Remember that this value CANNOT be higher than the hard limit. Raising the hard limit depends on the OS and/or distribution, a good way to find the limit is to search the internet for something like this:

```
raise max open files hard limit debian
```

worker_threads

Default: `5`

The number of threads to start for receiving commands and replies from minions. If minions are stalling on replies because you have many minions, raise the worker_threads value.

Worker threads should not be put below 3 when using the peer system, but can drop down to 1 worker otherwise.

Note: When the master daemon starts, it is expected behaviour to see multiple salt-master processes, even if `worker_threads' is set to `1'. At a minimum, a controlling process will start along with a Publisher, an EventPublisher, and a number of MWorker processes will be started. The number of MWorker processes is tuneable by the `worker_threads' configuration value while the others are not.

```
worker_threads: 5
```

ret_port

Default: `4506`

The port used by the return server, this is the server used by Salt to receive execution returns and command executions.

```
ret_port: 4506
```

pidfile

Default: `/var/run/salt-master.pid`

Specify the location of the master pidfile.

```
pidfile: /var/run/salt-master.pid
```

root_dir

Default: `/`

The system root directory to operate from, change this to make Salt run from an alternative root.

```
root_dir: /
```

Note: This directory is prepended to the following options: pki_dir, cachedir, sock_dir, log_file, autosign_file, autoreject_file, pidfile.

pki_dir

Default: /etc/salt/pki

The directory to store the pki authentication keys.

```
pki_dir: /etc/salt/pki
```

extension_modules

Directory for custom modules. This directory can contain subdirectories for each of Salt's module types such as ``runners'', ``output'', ``wheel'', ``modules'', ``states'', ``returners'', etc. This path is appended to root_dir.

```
extension_modules: srv/modules
```

module_dirs

Default: []

Like extension_modules, but a list of extra directories to search for Salt modules.

```
module_dirs:
  - /var/cache/salt/minion/extmods
```

cachedir

Default: /var/cache/salt

The location used to store cache information, particularly the job information for executed salt commands.

```
cachedir: /var/cache/salt
```

verify_env

Default: True

Verify and set permissions on configuration directories at startup.

```
verify_env: True
```

keep_jobs

Default: 24

Set the number of hours to keep old job information.

timeout

Default: `5`

Set the default timeout for the salt command and api.

loop_interval

Default: `60`

The loop_interval option controls the seconds for the master's maintenance process check cycle. This process updates file server backends, cleans the job cache and executes the scheduler.

output

Default: `nested`

Set the default outputter used by the salt command.

color

Default: `True`

By default output is colored, to disable colored output set the color value to False.

```
color: False
```

sock_dir

Default: `/var/run/salt/master`

Set the location to use for creating Unix sockets for master process communication.

```
sock_dir: /var/run/salt/master
```

enable_gpu_grains

Default: `False`

The master can take a while to start up when lspci and/or dmidecode is used to populate the grains for the master. Enable if you want to see GPU hardware data for your master.

job_cache

Default: `True`

The master maintains a job cache, while this is a great addition it can be a burden on the master for larger deployments (over 5000 minions). Disabling the job cache will make previously executed jobs unavailable to the jobs system and is not generally recommended. Normally it is wise to make sure the master has access to a faster IO system or a tmpfs is mounted to the jobs dir.

minion_data_cache

Default: `True`

The minion data cache is a cache of information about the minions stored on the master, this information is primarily the pillar and grains data. The data is cached in the Master cachedir under the name of the minion and used to predetermine what minions are expected to reply from executions.

```
minion_data_cache: True
```

ext_job_cache

Default: `''`

Used to specify a default returner for all minions, when this option is set the specified returner needs to be properly configured and the minions will always default to sending returns to this returner. This will also disable the local job cache on the master.

```
ext_job_cache: redis
```

event_return

New in version 2015.5.0.

Default: `''`

Specify the returner to use to log events. A returner may have installation and configuration requirements. Read the returner's documentation.

Note: Not all returners support event returns. Verify that a returner has an `event_return()` function before configuring this option with a returner.

```
event_return: cassandra_cql
```

master_job_cache

New in version 2014.7.

Default: `local_cache'

Specify the returner to use for the job cache. The job cache will only be interacted with from the salt master and therefore does not need to be accessible from the minions.

```
master_job_cache: redis
```

enforce_mine_cache

Default: False

By-default when disabling the minion_data_cache mine will stop working since it is based on cached data, by enabling this option we explicitly enabling only the cache for the mine system.

```
enforce_mine_cache: False
```

max_minions

Default: 0

The number of minions the master should allow to connect. Use this to accommodate the number of minions per master if you have different types of hardware serving your minions. The default of `0` means unlimited connections. Please note, that this can slow down the authentication process a bit in large setups.

```
max_minions: 100
```

con_cache

Default: False

If max_minions is used in large installations, the master might experience high-load situations because of having to check the number of connected minions for every authentication. This cache provides the minion-ids of all connected minions to all MWorker-processes and greatly improves the performance of max_minions.

```
con_cache: True
```

presence_events

Default: False

Causes the master to periodically look for actively connected minions. *Presence events* are fired on the event bus on a regular interval with a list of connected minions, as well as events with lists of newly connected or disconnected minions. This is a master-only operation that does not send executions to minions. Note, this does not detect minions that connect to a master via localhost.

```
presence_events: False
```

22.8.2 Salt-SSH Configuration

roster_file

Default: `/etc/salt/roster'

Pass in an alternative location for the salt-ssh roster file.

```
roster_file: /root/roster
```

ssh_minion_opts

Default: None

Pass in minion option overrides that will be inserted into the SHIM for salt-ssh calls. The local minion config is not used for salt-ssh. Can be overridden on a per-minion basis in the roster (`minion_opts`)

```
minion_opts:
  gpg_keydir: /root/gpg
```

22.8.3 Master Security Settings

open_mode

Default: `False`

Open mode is a dangerous security feature. One problem encountered with pki authentication systems is that keys can become ``mixed up'' and authentication begins to fail. Open mode turns off authentication and tells the master to accept all authentication. This will clean up the pki keys received from the minions. Open mode should not be turned on for general use. Open mode should only be used for a short period of time to clean up pki keys. To turn on open mode set this value to `True`.

```
open_mode: False
```

auto_accept

Default: `False`

Enable auto_accept. This setting will automatically accept all incoming public keys from minions.

```
auto_accept: False
```

autosign_timeout

New in version 2014.7.0.

Default: `120`

Time in minutes that a incoming public key with a matching name found in pki_dir/minion_autosign/keyid is automatically accepted. Expired autosign keys are removed when the master checks the minion_autosign directory. This method to auto accept minions can be safer than an autosign_file because the keyid record can expire and is limited to being an exact name match. This should still be considered a less than secure option, due to the fact that trust is based on just the requesting minion id.

autosign_file

Default: `not defined`

If the `autosign_file` is specified incoming keys specified in the autosign_file will be automatically accepted. Matches will be searched for first by string comparison, then by globbing, then by full-string regex matching. This should still be considered a less than secure option, due to the fact that trust is based on just the requesting minion id.

autoreject_file

New in version 2014.1.0.

Default: `not defined`

Works like `autosign_file`, but instead allows you to specify minion IDs for which keys will automatically be rejected. Will override both membership in the `autosign_file` and the `auto_accept` setting.

client_acl

Default: {}

Enable user accounts on the master to execute specific modules. These modules can be expressed as regular expressions.

```
client_acl:
  fred:
    - test.ping
    - pkg.*
```

client_acl_blacklist

Default: {}

Blacklist users or modules

This example would blacklist all non sudo users, including root from running any commands. It would also blacklist any use of the ``cmd'' module.

This is completely disabled by default.

```
client_acl_blacklist:
  users:
    - root
    - '^(?!sudo_).*$'   #  all non sudo users
  modules:
    - cmd
```

external_auth

Default: {}

The external auth system uses the Salt auth modules to authenticate and validate users to access areas of the Salt system.

```
external_auth:
  pam:
    fred:
      - test.*
```

token_expire

Default: 43200

Time (in seconds) for a newly generated token to live.

Default: 12 hours

```
token_expire: 43200
```

file_recv

Default: False

Allow minions to push files to the master. This is disabled by default, for security purposes.

```
file_recv: False
```

master_sign_pubkey

Default: `False`

Sign the master auth-replies with a cryptographic signature of the masters public key. Please see the tutorial how to use these settings in the Multimaster-PKI with Failover Tutorial

```
master_sign_pubkey: True
```

master_sign_key_name

Default: `master_sign`

The customizable name of the signing-key-pair without suffix.

```
master_sign_key_name: <filename_without_suffix>
```

master_pubkey_signature

Default: `master_pubkey_signature`

The name of the file in the masters pki-directory that holds the pre-calculated signature of the masters public-key.

```
master_pubkey_signature: <filename>
```

master_use_pubkey_signature

Default: `False`

Instead of computing the signature for each auth-reply, use a pre-calculated signature. The `master_pubkey_signature` must also be set for this.

```
master_use_pubkey_signature: True
```

rotate_aes_key

Default: `True`

Rotate the salt-masters AES-key when a minion-public is deleted with salt-key. This is a very important security-setting. Disabling it will enable deleted minions to still listen in on the messages published by the salt-master. Do not disable this unless it is absolutely clear what this does.

```
rotate_aes_key: True
```

22.8.4 Master Module Management

runner_dirs

Default: []

Set additional directories to search for runner modules.

cython_enable

Default: False

Set to true to enable Cython modules (.pyx files) to be compiled on the fly on the Salt master.

```
cython_enable: False
```

22.8.5 Master State System Settings

state_top

Default: top.sls

The state system uses a ``top'' file to tell the minions what environment to use and what modules to use. The state_top file is defined relative to the root of the base environment.

```
state_top: top.sls
```

master_tops

Default: {}

The master_tops option replaces the external_nodes option by creating a pluggable system for the generation of external top data. The external_nodes option is deprecated by the master_tops option. To gain the capabilities of the classic external_nodes system, use the following configuration:

```
master_tops:
  ext_nodes: <Shell command which returns yaml>
```

external_nodes

Default: None

The external_nodes option allows Salt to gather data that would normally be placed in a top file from and external node controller. The external_nodes option is the executable that will return the ENC data. Remember that Salt will look for external nodes AND top files and combine the results if both are enabled and available!

```
external_nodes: cobbler-ext-nodes
```

renderer

Default: `yaml_jinja`

The renderer to use on the minions to render the state data.

```
renderer: yaml_jinja
```

failhard

Default: `False`

Set the global failhard flag, this informs all states to stop running states at the moment a single state fails.

```
failhard: False
```

state_verbose

Default: `True`

Controls the verbosity of state runs. By default, the results of all states are returned, but setting this value to `False` will cause salt to only display output for states which either failed, or succeeded without making any changes to the minion.

```
state_verbose: False
```

state_output

Default: `full`

The state_output setting changes if the output is the full multi line output for each changed state if set to `full', but if set to `terse' the output will be shortened to a single line. If set to `mixed', the output will be terse unless a state failed, in which case that output will be full. If set to `changes', the output will be full unless the state didn't change.

```
state_output: full
```

yaml_utf8

Default: `False`

Enable extra routines for YAML renderer used states containing UTF characters.

```
yaml_utf8: False
```

test

Default: `False`

Set all state calls to only test if they are going to actually make changes or just post what changes are going to be made.

```
test: False
```

22.8.6 Master File Server Settings

fileserver_backend

Default: ['roots']

Salt supports a modular fileserver backend system, this system allows the salt master to link directly to third party systems to gather and manage the files available to minions. Multiple backends can be configured and will be searched for the requested file in the order in which they are defined here. The default setting only enables the standard backend roots, which is configured using the file_roots option.

Example:

```
fileserver_backend:
  - roots
  - git
```

hash_type

Default: md5

The hash_type is the hash to use when discovering the hash of a file on the master server. The default is md5, but sha1, sha224, sha256, sha384, and sha512 are also supported.

```
hash_type: md5
```

file_buffer_size

Default: 1048576

The buffer size in the file server in bytes.

```
file_buffer_size: 1048576
```

file_ignore_regex

Default: ''

A regular expression (or a list of expressions) that will be matched against the file path before syncing the modules and states to the minions. This includes files affected by the file.recurse state. For example, if you manage your custom modules and states in subversion and don't want all the `.svn' folders and content synced to your minions, you could set this to `/.svn($|/)'. By default nothing is ignored.

```
file_ignore_regex:
  - '/\.svn($|/)'
  - '/\.git($|/)'
```

file_ignore_glob

Default ''

A file glob (or list of file globs) that will be matched against the file path before syncing the modules and states to the minions. This is similar to file_ignore_regex above, but works on globs instead of regex. By default nothing is ignored.

```
file_ignore_glob:
  - '\*.pyc'
  - '\*/somefolder/\*.bak'
  - '\*.swp'
```

roots: Master's Local File Server

file_roots

Default:

```
base:
  - /srv/salt
```

Salt runs a lightweight file server written in ZeroMQ to deliver files to minions. This file server is built into the master daemon and does not require a dedicated port.

The file server works on environments passed to the master. Each environment can have multiple root directories. The subdirectories in the multiple file roots cannot match, otherwise the downloaded files will not be able to be reliably ensured. A base environment is required to house the top file.

Example:

```
file_roots:
  base:
    - /srv/salt
  dev:
    - /srv/salt/dev/services
    - /srv/salt/dev/states
  prod:
    - /srv/salt/prod/services
    - /srv/salt/prod/states
```

git: Git Remote File Server Backend

gitfs_remotes

Default: []

When using the `git` fileserver backend at least one git remote needs to be defined. The user running the salt master will need read access to the repo.

The repos will be searched in order to find the file requested by a client and the first repo to have the file will return it. Branches and tags are translated into salt environments.

```
gitfs_remotes:
  - git://github.com/saltstack/salt-states.git
  - file:///var/git/saltmaster
```

Note: `file://` repos will be treated as a remote and copied into the master's gitfs cache, so only the *local* refs for those repos will be exposed as fileserver environments.

As of 2014.7.0, it is possible to have per-repo versions of several of the gitfs configuration parameters. For more information, see the *GitFS Walkthrough*.

gitfs_provider

New in version 2014.7.0.

Specify the provider to be used for gitfs. More information can be found in the *GitFS Walkthrough*.

Specify one value among valid values: `gitpython`, `pygit2`, `dulwich`

```
gitfs_provider: dulwich
```

gitfs_ssl_verify

Default: `True`

The `gitfs_ssl_verify` option specifies whether to ignore SSL certificate errors when contacting the gitfs backend. You might want to set this to false if you're using a git backend that uses a self-signed certificate but keep in mind that setting this flag to anything other than the default of `True` is a security concern, you may want to try using the ssh transport.

```
gitfs_ssl_verify: True
```

gitfs_mountpoint

New in version 2014.7.0.

Default: `''`

Specifies a path on the salt fileserver from which gitfs remotes are served. Can be used in conjunction with `gitfs_root`. Can also be configured on a per-remote basis, see *here* for more info.

```
gitfs_mountpoint: salt://foo/bar
```

Note: The `salt://` protocol designation can be left off (in other words, `foo/bar` and `salt://foo/bar` are equivalent).

---

gitfs_root

Default: `''`

Serve files from a subdirectory within the repository, instead of the root. This is useful when there are files in the repository that should not be available to the Salt fileserver. Can be used in conjunction with `gitfs_mountpoint`.

```
gitfs_root: somefolder/otherfolder
```

Changed in version 2014.7.0: Ability to specify gitfs roots on a per-remote basis was added. See *here* for more info.

gitfs_base

Default: `master`

Defines which branch/tag should be used as the `base` environment.

```
gitfs_base: salt
```

Changed in version 2014.7.0: Ability to specify the base on a per-remote basis was added. See *here* for more info.

gitfs_env_whitelist

New in version 2014.7.0.

Default: []

Used to restrict which environments are made available. Can speed up state runs if the repos in `gitfs_remotes` contain many branches/tags. More information can be found in the *GitFS Walkthrough*.

```
gitfs_env_whitelist:
  - base
  - v1.*
  - 'mybranch\d+'
```

gitfs_env_blacklist

New in version 2014.7.0.

Default: []

Used to restrict which environments are made available. Can speed up state runs if the repos in `gitfs_remotes` contain many branches/tags. More information can be found in the *GitFS Walkthrough*.

```
gitfs_env_blacklist:
  - base
  - v1.*
  - 'mybranch\d+'
```

GitFS Authentication Options

These parameters only currently apply to the pygit2 gitfs provider. Examples of how to use these can be found in the *GitFS Walkthrough*.

gitfs_user New in version 2014.7.0.

Default: ''

Along with `gitfs_password`, is used to authenticate to HTTPS remotes.

```
gitfs_user: git
```

gitfs_password New in version 2014.7.0.

Default: ''

Along with `gitfs_user`, is used to authenticate to HTTPS remotes. This parameter is not required if the repository does not use authentication.

```
gitfs_password: mypassword
```

gitfs_insecure_auth New in version 2014.7.0.

Default: `False`

By default, Salt will not authenticate to an HTTP (non-HTTPS) remote. This parameter enables authentication over HTTP. **Enable this at your own risk**.

```
gitfs_insecure_auth: True
```

gitfs_pubkey New in version 2014.7.0.

Default: `''`

Along with `gitfs_privkey` (and optionally `gitfs_passphrase`), is used to authenticate to SSH remotes. This parameter (or its *per-remote counterpart*) is required for SSH remotes.

```
gitfs_pubkey: /path/to/key.pub
```

gitfs_privkey New in version 2014.7.0.

Default: `''`

Along with `gitfs_pubkey` (and optionally `gitfs_passphrase`), is used to authenticate to SSH remotes. This parameter (or its *per-remote counterpart*) is required for SSH remotes.

```
gitfs_privkey: /path/to/key
```

gitfs_passphrase New in version 2014.7.0.

Default: `''`

This parameter is optional, required only when the SSH key being used to authenticate is protected by a passphrase.

```
gitfs_passphrase: mypassphrase
```

hg: Mercurial Remote File Server Backend

hgfs_remotes

New in version 0.17.0.

Default: `[]`

When using the hg fileserver backend at least one mercurial remote needs to be defined. The user running the salt master will need read access to the repo.

The repos will be searched in order to find the file requested by a client and the first repo to have the file will return it. Branches and/or bookmarks are translated into salt environments, as defined by the `hgfs_branch_method` parameter.

```
hgfs_remotes:
  - https://username@bitbucket.org/username/reponame
```

Note: As of 2014.7.0, it is possible to have per-repo versions of the `hgfs_root`, `hgfs_mountpoint`, `hgfs_base`, and `hgfs_branch_method` parameters. For example:

```
hgfs_remotes:
  - https://username@bitbucket.org/username/repo1
    - base: saltstates
  - https://username@bitbucket.org/username/repo2:
    - root: salt
    - mountpoint: salt://foo/bar/baz
  - https://username@bitbucket.org/username/repo3:
    - root: salt/states
    - branch_method: mixed
```

hgfs_branch_method

New in version 0.17.0.

Default: `branches`

Defines the objects that will be used as fileserver environments.

- `branches` - Only branches and tags will be used
- `bookmarks` - Only bookmarks and tags will be used
- `mixed` - Branches, bookmarks, and tags will be used

```
hgfs_branch_method: mixed
```

Note: Starting in version 2014.1.0, the value of the `hgfs_base` parameter defines which branch is used as the `base` environment, allowing for a `base` environment to be used with an `hgfs_branch_method` of `bookmarks`.

Prior to this release, the `default` branch will be used as the `base` environment.

hgfs_mountpoint

New in version 2014.7.0.

Default: `''`

Specifies a path on the salt fileserver from which hgfs remotes are served. Can be used in conjunction with `hgfs_root`. Can also be configured on a per-remote basis, see `here` for more info.

```
hgfs_mountpoint: salt://foo/bar
```

Note: The `salt://` protocol designation can be left off (in other words, `foo/bar` and `salt://foo/bar` are equivalent).

hgfs_root

New in version 0.17.0.

Default: `''`

Serve files from a subdirectory within the repository, instead of the root. This is useful when there are files in the repository that should not be available to the Salt fileserver. Can be used in conjunction with `hgfs_mountpoint`.

```
hgfs_root: somefolder/otherfolder
```

Changed in version 2014.7.0: Ability to specify hgfs roots on a per-remote basis was added. See here for more info.

hgfs_base

New in version 2014.1.0.

Default: `default`

Defines which branch should be used as the `base` environment. Change this if `hgfs_branch_method` is set to `bookmarks` to specify which bookmark should be used as the `base` environment.

```
hgfs_base: salt
```

hgfs_env_whitelist

New in version 2014.7.0.

Default: `[]`

Used to restrict which environments are made available. Can speed up state runs if your hgfs remotes contain many branches/bookmarks/tags. Full names, globs, and regular expressions are supported. If using a regular expression, the expression must match the entire minion ID.

If used, only branches/bookmarks/tags which match one of the specified expressions will be exposed as fileserver environments.

If used in conjunction with `hgfs_env_blacklist`, then the subset of branches/bookmarks/tags which match the whitelist but do *not* match the blacklist will be exposed as fileserver environments.

```
hgfs_env_whitelist:
  - base
  - v1.*
  - 'mybranch\d+'
```

hgfs_env_blacklist

New in version 2014.7.0.

Default: `[]`

Used to restrict which environments are made available. Can speed up state runs if your hgfs remotes contain many branches/bookmarks/tags. Full names, globs, and regular expressions are supported. If using a regular expression, the expression must match the entire minion ID.

If used, branches/bookmarks/tags which match one of the specified expressions will *not* be exposed as fileserver environments.

If used in conjunction with `hgfs_env_whitelist`, then the subset of branches/bookmarks/tags which match the whitelist but do *not* match the blacklist will be exposed as fileserver environments.

```
hgfs_env_blacklist:
  - base
  - v1.*
  - 'mybranch\d+'
```

svn: Subversion Remote File Server Backend

svnfs_remotes

New in version 0.17.0.

Default: []

When using the svn fileserver backend at least one subversion remote needs to be defined. The user running the salt master will need read access to the repo.

The repos will be searched in order to find the file requested by a client and the first repo to have the file will return it. The trunk, branches, and tags become environments, with the trunk being the base environment.

```
svnfs_remotes:
  - svn://foo.com/svn/myproject
```

Note: As of 2014.7.0, it is possible to have per-repo versions of the following configuration parameters:

- svnfs_root
- svnfs_mountpoint
- svnfs_trunk
- svnfs_branches
- svnfs_tags

For example:

```
svnfs_remotes:
  - svn://foo.com/svn/project1
  - svn://foo.com/svn/project2:
    - root: salt
    - mountpoint: salt://foo/bar/baz
  - svn//foo.com/svn/project3:
    - root: salt/states
    - branches: branch
    - tags: tag
```

---

svnfs_mountpoint

New in version 2014.7.0.

Default: ''

Specifies a path on the salt fileserver from which svnfs remotes are served. Can be used in conjunction with svnfs_root. Can also be configured on a per-remote basis, see here for more info.

```
svnfs_mountpoint: salt://foo/bar
```

Note: The salt:// protocol designation can be left off (in other words, foo/bar and salt://foo/bar are equivalent).

---

svnfs_root

New in version 0.17.0.

Default: ''

Serve files from a subdirectory within the repository, instead of the root. This is useful when there are files in the repository that should not be available to the Salt fileserver. Can be used in conjunction with svnfs_mountpoint.

```
svnfs_root: somefolder/otherfolder
```

Changed in version 2014.7.0: Ability to specify svnfs roots on a per-remote basis was added. See here for more info.

svnfs_trunk

New in version 2014.7.0.

Default: trunk

Path relative to the root of the repository where the trunk is located. Can also be configured on a per-remote basis, see here for more info.

```
svnfs_trunk: trunk
```

svnfs_branches

New in version 2014.7.0.

Default: branches

Path relative to the root of the repository where the branches are located. Can also be configured on a per-remote basis, see here for more info.

```
svnfs_branches: branches
```

svnfs_tags

New in version 2014.7.0.

Default: tags

Path relative to the root of the repository where the tags are located. Can also be configured on a per-remote basis, see here for more info.

```
svnfs_tags: tags
```

svnfs_env_whitelist

New in version 2014.7.0.

Default: []

Used to restrict which environments are made available. Can speed up state runs if your svnfs remotes contain many branches/tags. Full names, globs, and regular expressions are supported. If using a regular expression, the expression must match the entire minion ID.

If used, only branches/tags which match one of the specified expressions will be exposed as fileserver environments.

If used in conjunction with `svnfs_env_blacklist`, then the subset of branches/tags which match the whitelist but do *not* match the blacklist will be exposed as fileserver environments.

```
svnfs_env_whitelist:
  - base
  - v1.*
  - 'mybranch\d+'
```

svnfs_env_blacklist

New in version 2014.7.0.

Default: `[]`

Used to restrict which environments are made available. Can speed up state runs if your svnfs remotes contain many branches/tags. Full names, globs, and regular expressions are supported. If using a regular expression, the expression must match the entire minion ID.

If used, branches/tags which match one of the specified expressions will *not* be exposed as fileserver environments.

If used in conjunction with `svnfs_env_whitelist`, then the subset of branches/tags which match the whitelist but do *not* match the blacklist will be exposed as fileserver environments.

```
svnfs_env_blacklist:
  - base
  - v1.*
  - 'mybranch\d+'
```

minion: MinionFS Remote File Server Backend

minionfs_env

New in version 2014.7.0.

Default: `base`

Environment from which MinionFS files are made available.

```
minionfs_env: minionfs
```

minionfs_mountpoint

New in version 2014.7.0.

Default: `''`

Specifies a path on the salt fileserver from which minionfs files are served.

```
minionfs_mountpoint: salt://foo/bar
```

Note: The `salt://` protocol designation can be left off (in other words, `foo/bar` and `salt://foo/bar` are equivalent).

---

minionfs_whitelist

New in version 2014.7.0.

Default: []

Used to restrict which minions' pushed files are exposed via minionfs. If using a regular expression, the expression must match the entire minion ID.

If used, only the pushed files from minions which match one of the specified expressions will be exposed.

If used in conjunction with minionfs_blacklist, then the subset of hosts which match the whitelist but do *not* match the blacklist will be exposed.

```
minionfs_whitelist:
  - base
  - v1.*
  - 'mybranch\d+'
```

minionfs_blacklist

New in version 2014.7.0.

Default: []

Used to restrict which minions' pushed files are exposed via minionfs. If using a regular expression, the expression must match the entire minion ID.

If used, only the pushed files from minions which match one of the specified expressions will *not* be exposed.

If used in conjunction with minionfs_whitelist, then the subset of hosts which match the whitelist but do *not* match the blacklist will be exposed.

```
minionfs_blacklist:
  - base
  - v1.*
  - 'mybranch\d+'
```

22.8.7 Pillar Configuration

pillar_roots

Default:

```
base:
  - /srv/pillar
```

Set the environments and directories used to hold pillar sls data. This configuration is the same as file_roots:

```
pillar_roots:
  base:
    - /srv/pillar
  dev:
    - /srv/pillar/dev
  prod:
    - /srv/pillar/prod
```

ext_pillar

The ext_pillar option allows for any number of external pillar interfaces to be called when populating pillar data. The configuration is based on ext_pillar functions. The available ext_pillar functions can be found herein:

https://github.com/saltstack/salt/blob/develop/salt/pillar

By default, the ext_pillar interface is not configured to run.

Default: `None`

```
ext_pillar:
  - hiera: /etc/hiera.yaml
  - cmd_yaml: cat /etc/salt/yaml
  - reclass:
      inventory_base_uri: /etc/reclass
```

There are additional details at *Pillars*

ext_pillar_first

New in version 2015.5.0.

The ext_pillar_first option allows for external pillar sources to populate before file system pillar. This allows for targeting file system pillar from ext_pillar.

Default: `False`

```
ext_pillar_first: False
```

pillar_source_merging_strategy

New in version 2014.7.0.

Default: `smart`

The pillar_source_merging_strategy option allows you to configure merging strategy between different sources. It accepts 4 values:

- recurse:

  it will merge recursively mapping of data. For example, theses 2 sources:

```
    foo: 42
    bar:
        element1: True
```

```
    bar:
        element2: True
    baz: quux
```

  will be merged as:

```
    foo: 42
    bar:
        element1: True
        element2: True
    baz: quux
```

- aggregate:

  instructs aggregation of elements between sources that use the #!yamlex renderer.

  For example, these two documents:

```
#!yamlex
foo: 42
bar: !aggregate {
  element1: True
}
baz: !aggregate quux
```

```
#!yamlex
bar: !aggregate {
  element2: True
}
baz: !aggregate quux2
```

  will be merged as:

```
foo: 42
bar:
  element1: True
  element2: True
baz:
  - quux
  - quux2
```

- overwrite:

  Will use the behaviour of the 2014.1 branch and earlier.

  Overwrites elements according the order in which they are processed.

  First pillar processed:

```
A:
  first_key: blah
  second_key: blah
```

  Second pillar processed:

```
A:
  third_key: blah
  fourth_key: blah
```

  will be merged as:

```
A:
  third_key: blah
  fourth_key: blah
```

- smart (default):

  Guesses the best strategy based on the ``renderer'' setting.

22.8.8 Syndic Server Settings

A Salt syndic is a Salt master used to pass commands from a higher Salt master to minions below the syndic. Using the syndic is simple. If this is a master that will have syndic servers(s) below it, set the ``order_masters'' setting to

True.

If this is a master that will be running a syndic daemon for passthrough the ``syndic_master'' setting needs to be set to the location of the master server.

Do not not forget that, in other words, it means that it shares with the local minion its ID and PKI_DIR.

order_masters

Default: `False`

Extra data needs to be sent with publications if the master is controlling a lower level master via a syndic minion. If this is the case the order_masters value must be set to True

```
order_masters: False
```

syndic_master

Default: `None`

If this master will be running a salt-syndic to connect to a higher level master, specify the higher level master with this configuration value.

```
syndic_master: masterofmasters
```

syndic_master_port

Default: `4506`

If this master will be running a salt-syndic to connect to a higher level master, specify the higher level master port with this configuration value.

```
syndic_master_port: 4506
```

syndic_pidfile

Default: `salt-syndic.pid`

If this master will be running a salt-syndic to connect to a higher level master, specify the pidfile of the syndic daemon.

```
syndic_pidfile: syndic.pid
```

syndic_log_file

Default: `syndic.log`

If this master will be running a salt-syndic to connect to a higher level master, specify the log_file of the syndic daemon.

```
syndic_log_file: salt-syndic.log
```

22.8.9 Peer Publish Settings

Salt minions can send commands to other minions, but only if the minion is allowed to. By default ``Peer Publication'' is disabled, and when enabled it is enabled for specific minions and specific commands. This allows secure compartmentalization of commands based on individual minions.

peer

Default: {}

The configuration uses regular expressions to match minions and then a list of regular expressions to match functions. The following will allow the minion authenticated as foo.example.com to execute functions from the test and pkg modules.

```
peer:
  foo.example.com:
      - test.*
      - pkg.*
```

This will allow all minions to execute all commands:

```
peer:
  .*:
      - .*
```

This is not recommended, since it would allow anyone who gets root on any single minion to instantly have root on all of the minions!

By adding an additional layer you can limit the target hosts in addition to the accessible commands:

```
peer:
  foo.example.com:
    'db*':
      - test.*
      - pkg.*
```

peer_run

Default: {}

The peer_run option is used to open up runners on the master to access from the minions. The peer_run configuration matches the format of the peer configuration.

The following example would allow foo.example.com to execute the manage.up runner:

```
peer_run:
  foo.example.com:
      - manage.up
```

22.8.10 Master Logging Settings

log_file

Default: /var/log/salt/master

The master log can be sent to a regular file, local path name, or network location. See also log_file.

Examples:

```
log_file: /var/log/salt/master
```

```
log_file: file:///dev/log
```

```
log_file: udp://loghost:10514
```

log_level

Default: `warning`

The level of messages to send to the console. See also `log_level`.

```
log_level: warning
```

log_level_logfile

Default: `warning`

The level of messages to send to the log file. See also `log_level_logfile`.

```
log_level_logfile: warning
```

log_datefmt

Default: `%H:%M:%S`

The date and time format used in console log messages. See also `log_datefmt`.

```
log_datefmt: '%H:%M:%S'
```

log_datefmt_logfile

Default: `%Y-%m-%d %H:%M:%S`

The date and time format used in log file messages. See also `log_datefmt_logfile`.

```
log_datefmt_logfile: '%Y-%m-%d %H:%M:%S'
```

log_fmt_console

Default: `[%(levelname)-8s] %(message)s`

The format of the console logging messages. See also `log_fmt_console`.

```
log_fmt_console: '[%(levelname)-8s] %(message)s'
```

log_fmt_logfile

Default: `%(asctime)s,%(msecs)03.0f [%(name)-17s][%(levelname)-8s] %(message)s`

The format of the log file logging messages. See also `log_fmt_logfile`.

```
log_fmt_logfile: '%(asctime)s,%(msecs)03.0f [%(name)-17s][%(levelname)-8s] %(message)s'
```

log_granular_levels

Default: {}

This can be used to control logging levels more specifically. See also `log_granular_levels`.

22.8.11 Node Groups

Default: {}

Node groups allow for logical groupings of minion nodes. A group consists of a group name and a compound target.

```
nodegroups:
  group1: 'L@foo.domain.com,bar.domain.com,baz.domain.com or bl*.domain.com'
  group2: 'G@os:Debian and foo.domain.com'
  group3: 'G@os:Debian and N@group1'
```

More information on using nodegroups can be found *here*.

22.8.12 Range Cluster Settings

range_server

Default: `''`

The range server (and optional port) that serves your cluster information https://github.com/ytoolshed/range/wiki/%22yamlfile%22-module-file-spec

```
range_server: range:80
```

22.8.13 Include Configuration

default_include

Default: `master.d/*.conf`

The master can include configuration from other files. Per default the master will automatically include all config files from `master.d/*.conf` where `master.d` is relative to the directory of the master configuration file.

include

Default: `not defined`

The master can include configuration from other files. To enable this, pass a list of paths to this option. The paths can be either relative or absolute; if relative, they are considered to be relative to the directory the main minion

configuration file lives in. Paths can make use of shell-style globbing. If no files are matched by a path passed to this option then the master will log a warning message.

```
# Include files from a master.d directory in the same
# directory as the master config file
include: master.d/*

# Include a single extra file into the configuration
include: /etc/roles/webserver

# Include several files and the master.d directory
include:
  - extra_config
  - master.d/*
  - /etc/roles/webserver
```

22.8.14 Windows Software Repo Settings

win_repo

Default: `/srv/salt/win/repo`

Location of the repo on the master

```
win_repo: '/srv/salt/win/repo'
```

win_repo_mastercachefile

Default: `/srv/salt/win/repo/winrepo.p`

```
win_repo_mastercachefile: '/srv/salt/win/repo/winrepo.p'
```

win_gitrepos

Default: `''`

List of git repositories to include with the local repo.

```
win_gitrepos:
  - 'https://github.com/saltstack/salt-winrepo.git'
```

22.9 Configuring the Salt Minion

The Salt system is amazingly simple and easy to configure. The two components of the Salt system each have a respective configuration file. The **salt-master** is configured via the master configuration file, and the **salt-minion** is configured via the minion configuration file.

See also:

example minion configuration file

The Salt Minion configuration is very simple. Typically, the only value that needs to be set is the master value so the minion knows where to locate its master.

By default, the salt-minion configuration will be in `/etc/salt/minion`. A notable exception is FreeBSD, where the configuration will be in `/usr/local/etc/salt/minion`.

22.9.1 Minion Primary Configuration

master

Default: `salt`

The hostname or ipv4 of the master.

Default: `salt`

```
master: salt
```

The option can can also be set to a list of masters, enabling *multi-master* mode.

```
master:
  - address1
  - address2
```

Changed in version 2014.7.0: The master can be dynamically configured. The `master` value can be set to an module function which will be executed and will assume that the returning value is the ip or hostname of the desired master. If a function is being specified, then the `master_type` option must be set to `func`, to tell the minion that the value is a function to be run and not a fully-qualified domain name.

```
master: module.function
master_type: func
```

In addition, instead of using multi-master mode, the minion can be configured to use the list of master addresses as a failover list, trying the first address, then the second, etc. until the minion successfully connects. To enable this behavior, set `master_type` to `failover`:

```
master:
  - address1
  - address2
master_type: failover
```

master_type

New in version 2014.7.0.

Default: `str`

The type of the `master` variable. Can be either `func` or `failover`.

If the master needs to be dynamically assigned by executing a function instead of reading in the static master value, set this to `func`. This can be used to manage the minion's master setting from an execution module. By simply changing the algorithm in the module to return a new master ip/fqdn, restart the minion and it will connect to the new master.

```
master_type: func
```

If this option is set to `failover`, `master` must be a list of master addresses. The minion will then try each master in the order specified in the list until it successfully connects.

```
master_type: failover
```

master_shuffle

New in version 2014.7.0.

Default: `False`

If `master` is a list of addresses, shuffle them before trying to connect to distribute the minions over all available masters. This uses Python's `random.shuffle` method.

```
master_shuffle: True
```

retry_dns

Default: `30`

Set the number of seconds to wait before attempting to resolve the master hostname if name resolution fails. Defaults to 30 seconds. Set to zero if the minion should shutdown and not retry.

```
retry_dns: 30
```

master_port

Default: `4506`

The port of the master ret server, this needs to coincide with the ret_port option on the Salt master.

```
master_port: 4506
```

user

Default: `root`

The user to run the Salt processes

```
user: root
```

pidfile

Default: `/var/run/salt-minion.pid`

The location of the daemon's process ID file

```
pidfile: /var/run/salt-minion.pid
```

root_dir

Default: `/`

This directory is prepended to the following options: `pki_dir`, `cachedir`, `log_file`, `sock_dir`, and `pidfile`.

```
root_dir: /
```

pki_dir

Default: `/etc/salt/pki`

The directory used to store the minion's public and private keys.

```
pki_dir: /etc/salt/pki
```

id

Default: the system's hostname

See also:

Salt Walkthrough

The **Setting up a Salt Minion** section contains detailed information on how the hostname is determined.

Explicitly declare the id for this minion to use. Since Salt uses detached ids it is possible to run multiple minions on the same machine but with different ids.

```
id: foo.bar.com
```

append_domain

Default: `None`

Append a domain to a hostname in the event that it does not exist. This is useful for systems where `socket.getfqdn()` does not actually result in a FQDN (for instance, Solaris).

```
append_domain: foo.org
```

cachedir

Default: `/var/cache/salt`

The location for minion cache data.

```
cachedir: /var/cache/salt
```

verify_env

Default: `True`

Verify and set permissions on configuration directories at startup.

```
verify_env: True
```

Note: When marked as True the verify_env option requires WRITE access to the configuration directory (/etc/salt/). In certain situations such as mounting /etc/salt/ as read-only for templating this will create a stack trace when state.highstate is called.

---

cache_jobs

Default: `False`

The minion can locally cache the return data from jobs sent to it, this can be a good way to keep track of the minion side of the jobs the minion has executed. By default this feature is disabled, to enable set cache_jobs to `True`.

```
cache_jobs: False
```

sock_dir

Default: `/var/run/salt/minion`

The directory where Unix sockets will be kept.

```
sock_dir: /var/run/salt/minion
```

backup_mode

Default: `[]`

Backup files replaced by file.managed and file.recurse under cachedir.

```
backup_mode: minion
```

acceptance_wait_time

Default: `10`

The number of seconds to wait until attempting to re-authenticate with the master.

```
acceptance_wait_time: 10
```

random_reauth_delay

When the master key changes, the minion will try to re-auth itself to receive the new master key. In larger environments this can cause a syn-flood on the master because all minions try to re-auth immediately. To prevent this and have a minion wait for a random amount of time, use this optional parameter. The wait-time will be a random number of seconds between 0 and the defined value.

```
random_reauth_delay: 60
```

acceptance_wait_time_max

Default: `None`

The maximum number of seconds to wait until attempting to re-authenticate with the master. If set, the wait will increase by acceptance_wait_time seconds each iteration.

```
acceptance_wait_time_max: None
```

recon_default

Default: `1000`

The interval in milliseconds that the socket should wait before trying to reconnect to the master (1000ms = 1 second).

```
recon_default: 1000
```

recon_max

Default: `10000`

The maximum time a socket should wait. Each interval the time to wait is calculated by doubling the previous time. If recon_max is reached, it starts again at the recon_default.

Short example:

- reconnect 1: the socket will wait `recon_default' milliseconds
- reconnect 2: `recon_default' * 2
- reconnect 3: (`recon_default' * 2) * 2
- reconnect 4: value from previous interval * 2
- reconnect 5: value from previous interval * 2
- reconnect x: if value >= recon_max, it starts again with recon_default

```
recon_max: 10000
```

recon_randomize

Default: `True`

Generate a random wait time on minion start. The wait time will be a random value between recon_default and recon_default and recon_max. Having all minions reconnect with the same recon_default and recon_max value kind of defeats the purpose of being able to change these settings. If all minions have the same values and the setup is quite large (several thousand minions), they will still flood the master. The desired behavior is to have time-frame within all minions try to reconnect.

```
recon_randomize: True
```

dns_check

Default: `True`

When healing, a dns_check is run. This is to make sure that the originally resolved dns has not changed. If this is something that does not happen in your environment, set this value to `False`.

```
dns_check: True
```

cache_sreqs

Default: `True`

The connection to the master ret_port is kept open. When set to False, the minion creates a new connection for every return to the master. environment, set this value to `False`.

```
cache_sreqs: True
```

ipc_mode

Default: `ipc`

Windows platforms lack POSIX IPC and must rely on slower TCP based inter- process communications. Set ipc_mode to `tcp` on such systems.

```
ipc_mode: ipc
```

tcp_pub_port

Default: `4510`

Publish port used when `ipc_mode` is set to `tcp`.

```
tcp_pub_port: 4510
```

tcp_pull_port

Default: `4511`

Pull port used when `ipc_mode` is set to `tcp`.

```
tcp_pull_port: 4511
```

22.9.2 Minion Module Management

disable_modules

Default: `[]` (all modules are enabled by default)

The event may occur in which the administrator desires that a minion should not be able to execute a certain module. The sys module is built into the minion and cannot be disabled.

This setting can also tune the minion, as all modules are loaded into ram disabling modules will lover the minion's ram footprint.

```
disable_modules:
  - test
  - solr
```

disable_returners

Default: `[]` (all returners are enabled by default)

If certain returners should be disabled, this is the place

```
disable_returners:
  - mongo_return
```

module_dirs

Default: []

A list of extra directories to search for Salt modules

```
module_dirs:
  - /var/lib/salt/modules
```

returner_dirs

Default: []

A list of extra directories to search for Salt returners

```
returners_dirs:
  - /var/lib/salt/returners
```

states_dirs

Default: []

A list of extra directories to search for Salt states

```
states_dirs:
  - /var/lib/salt/states
```

grains_dirs

Default: []

A list of extra directories to search for Salt grains

```
grains_dirs:
  - /var/lib/salt/grains
```

render_dirs

Default: []

A list of extra directories to search for Salt renderers

```
render_dirs:
  - /var/lib/salt/renderers
```

cython_enable

Default: `False`

Set this value to true to enable auto-loading and compiling of `.pyx` modules, This setting requires that `gcc` and `cython` are installed on the minion

```
cython_enable: False
```

providers

Default: (empty)

A module provider can be statically overwritten or extended for the minion via the `providers` option. This can be done *on an individual basis in an SLS file*, or globally here in the minion config, like below.

```
providers:
  service: systemd
```

22.9.3 State Management Settings

renderer

Default: `yaml_jinja`

The default renderer used for local state executions

```
renderer: yaml_jinja
```

state_verbose

Default: `False`

state_verbose allows for the data returned from the minion to be more verbose. Normally only states that fail or states that have changes are returned, but setting state_verbose to `True` will return all states that were checked

```
state_verbose: True
```

state_output

Default: `full`

The state_output setting changes if the output is the full multi line output for each changed state if set to `full', but if set to `terse' the output will be shortened to a single line.

```
state_output: full
```

autoload_dynamic_modules

Default: `True`

autoload_dynamic_modules Turns on automatic loading of modules found in the environments on the master. This is turned on by default, to turn of auto-loading modules when states run set this value to `False`

```
autoload_dynamic_modules: True
```

Default: `True`

clean_dynamic_modules keeps the dynamic modules on the minion in sync with the dynamic modules on the master, this means that if a dynamic module is not on the master it will be deleted from the minion. By default this is enabled and can be disabled by changing this value to `False`

```
clean_dynamic_modules: True
```

environment

Default: None

Normally the minion is not isolated to any single environment on the master when running states, but the environment can be isolated on the minion side by statically setting it. Remember that the recommended way to manage environments is to isolate via the top file.

```
environment: None
```

22.9.4 File Directory Settings

file_client

Default: remote

The client defaults to looking on the master server for files, but can be directed to look on the minion by setting this parameter to local.

```
file_client: remote
```

use_master_when_local

Default: False

When using a local file_client, this parameter is used to allow the client to connect to a master for remote execution.

```
use_master_when_local: False
```

file_roots

Default:

```
base:
  - /srv/salt
```

When using a local file_client, this parameter is used to setup the fileserver's environments. This parameter operates identically to the master config parameter of the same name.

```
file_roots:
  base:
    - /srv/salt
  dev:
    - /srv/salt/dev/services
    - /srv/salt/dev/states
  prod:
    - /srv/salt/prod/services
    - /srv/salt/prod/states
```

hash_type

Default: md5

The hash_type is the hash to use when discovering the hash of a file on the local fileserver. The default is md5, but sha1, sha224, sha256, sha384, and sha512 are also supported.

```
hash_type: md5
```

pillar_roots

Default:

```
base:
  - /srv/pillar
```

When using a local `file_client`, this parameter is used to setup the pillar environments.

```
pillar_roots:
  base:
    - /srv/pillar
  dev:
    - /srv/pillar/dev
  prod:
    - /srv/pillar/prod
```

22.9.5 Security Settings

open_mode

Default: `False`

Open mode can be used to clean out the PKI key received from the Salt master, turn on open mode, restart the minion, then turn off open mode and restart the minion to clean the keys.

```
open_mode: False
```

verify_master_pubkey_sign

Default: `False`

Enables verification of the master-public-signature returned by the master in auth-replies. Please see the tutorial on how to configure this properly Multimaster-PKI with Failover Tutorial

New in version 2014.7.0.

```
verify_master_pubkey_sign: True
```

If this is set to `True`, `master_sign_pubkey` must be also set to `True` in the master configuration file.

master_sign_key_name

Default: `master_sign`

The filename without the *.pub* suffix of the public key that should be used for verifying the signature from the master. The file must be located in the minion's pki directory.

New in version 2014.7.0.

```
master_sign_key_name: <filename_without_suffix>
```

always_verify_signature

Default: False

If verify_master_pubkey_sign is enabled, the signature is only verified, if the public-key of the master changes. If the signature should always be verified, this can be set to True.

New in version 2014.7.0.

```
always_verify_signature: True
```

22.9.6 Thread Settings

Default: True

Disable multiprocessing support by default when a minion receives a publication a new process is spawned and the command is executed therein.

```
multiprocessing: True
```

22.9.7 Minion Logging Settings

log_file

Default: /var/log/salt/minion

The minion log can be sent to a regular file, local path name, or network location. See also log_file.

Examples:

```
log_file: /var/log/salt/minion
```

```
log_file: file:///dev/log
```

```
log_file: udp://loghost:10514
```

log_level

Default: warning

The level of messages to send to the console. See also log_level.

```
log_level: warning
```

log_level_logfile

Default: warning

The level of messages to send to the log file. See also log_level_logfile.

```
log_level_logfile: warning
```

log_datefmt

Default: %H:%M:%S

The date and time format used in console log messages. See also log_datefmt.

```
log_datefmt: '%H:%M:%S'
```

log_datefmt_logfile

Default: %Y-%m-%d %H:%M:%S

The date and time format used in log file messages. See also log_datefmt_logfile.

```
log_datefmt_logfile: '%Y-%m-%d %H:%M:%S'
```

log_fmt_console

Default: [%(levelname)-8s] %(message)s

The format of the console logging messages. See also log_fmt_console.

```
log_fmt_console: '[%(levelname)-8s] %(message)s'
```

log_fmt_logfile

Default: %(asctime)s,%(msecs)03.0f [%(name)-17s][%(levelname)-8s] %(message)s

The format of the log file logging messages. See also log_fmt_logfile.

```
log_fmt_logfile: '%(asctime)s,%(msecs)03.0f [%(name)-17s][%(levelname)-8s] %(message)s'
```

log_granular_levels

Default: {}

This can be used to control logging levels more specifically. See also log_granular_levels.

failhard

Default: False

Set the global failhard flag, this informs all states to stop running states at the moment a single state fails

```
failhard: False
```

22.9.8 Include Configuration

default_include

Default: `minion.d/*.conf`

The minion can include configuration from other files. Per default the minion will automatically include all config files from *minion.d/*.conf* where minion.d is relative to the directory of the minion configuration file.

include

Default: `not defined`

The minion can include configuration from other files. To enable this, pass a list of paths to this option. The paths can be either relative or absolute; if relative, they are considered to be relative to the directory the main minion configuration file lives in. Paths can make use of shell-style globbing. If no files are matched by a path passed to this option then the minion will log a warning message.

```
# Include files from a minion.d directory in the same
# directory as the minion config file
include: minion.d/*.conf

# Include a single extra file into the configuration
include: /etc/roles/webserver

# Include several files and the minion.d directory
include:
  - extra_config
  - minion.d/*
  - /etc/roles/webserver
```

22.9.9 Frozen Build Update Settings

These options control how `salt.modules.saltutil.update()` works with esky frozen apps. For more information look at https://github.com/cloudmatrix/esky/.

update_url

Default: `False` (Update feature is disabled)

The url to use when looking for application updates. Esky depends on directory listings to search for new versions. A webserver running on your Master is a good starting point for most setups.

```
update_url: 'http://salt.example.com/minion-updates'
```

update_restart_services

Default: `[]` (service restarting on update is disabled)

A list of services to restart when the minion software is updated. This would typically just be a list containing the minion's service name, but you may have other services that need to go with it.

```
update_restart_services: ['salt-minion']
```

22.10 Running the Salt Master/Minion as an Unprivileged User

While the default setup runs the master and minion as the root user, some may consider it an extra measure of security to run the master as a non-root user. Keep in mind that doing so does not change the master's capability to access minions as the user they are running as. Due to this many feel that running the master as a non-root user does not grant any real security advantage which is why the master has remained as root by default.

Note: Some of Salt's operations cannot execute correctly when the master is not running as root, specifically the pam external auth system, as this system needs root access to check authentication.

As of Salt 0.9.10 it is possible to run Salt as a non-root user. This can be done by setting the `user` parameter in the master configuration file. and restarting the `salt-master` service.

The minion has it's own `user` parameter as well, but running the minion as an unprivileged user will keep it from making changes to things like users, installed packages, etc. unless access controls (sudo, etc.) are setup on the minion to permit the non-root user to make the needed changes.

In order to allow Salt to successfully run as a non-root user, ownership, and permissions need to be set such that the desired user can read from and write to the following directories (and their subdirectories, where applicable):

- /etc/salt
- /var/cache/salt
- /var/log/salt
- /var/run/salt

Ownership can be easily changed with `chown`, like so:

```
# chown -R user /etc/salt /var/cache/salt /var/log/salt /var/run/salt
```

Warning: Running either the master or minion with the `root_dir` parameter specified will affect these paths, as will setting options like `pki_dir`, `cachedir`, `log_file`, and other options that normally live in the above directories.

22.11 Logging

The salt project tries to get the logging to work for you and help us solve any issues you might find along the way.

If you want to get some more information on the nitty-gritty of salt's logging system, please head over to the *logging development document*, if all you're after is salt's logging configurations, please continue reading.

22.11.1 Available Configuration Settings

`log_file`

The log records can be sent to a regular file, local path name, or network location. Remote logging works best when configured to use rsyslogd(8) (e.g.: `file:///dev/log`), with rsyslogd(8) configured for network logging. The format for remote addresses is: `<file|udp|tcp>://<host|socketpath>:<port-if-required>/<log-facility>`.

Default: Dependent of the binary being executed, for example, for `salt-master`, `/var/log/salt/master`.

Examples:

```
log_file: /var/log/salt/master
```

```
log_file: /var/log/salt/minion
```

```
log_file: file:///dev/log
```

```
log_file: udp://loghost:10514
```

log_level

Default: `warning`

The level of log record messages to send to the console. One of `all`, `garbage`, `trace`, `debug`, `info`, `warning`, `error`, `critical`, `quiet`.

```
log_level: warning
```

log_level_logfile

Default: `warning`

The level of messages to send to the log file. One of `all`, `garbage`, `trace`, `debug`, `info`, `warning`, `error`, `critical`, `quiet`.

```
log_level_logfile: warning
```

log_datefmt

Default: `%H:%M:%S`

The date and time format used in console log messages. Allowed date/time formatting can be seen on `time.strftime`.

```
log_datefmt: '%H:%M:%S'
```

log_datefmt_logfile

Default: `%Y-%m-%d %H:%M:%S`

The date and time format used in log file messages. Allowed date/time formatting can be seen on `time.strftime`.

```
log_datefmt_logfile: '%Y-%m-%d %H:%M:%S'
```

log_fmt_console

Default: `[%(levelname)-8s] %(message)s`

The format of the console logging messages. Allowed formatting options can be seen on the *LogRecord attributes*.

```
log_fmt_console: '[%(levelname)-8s] %(message)s'
```

log_fmt_logfile

Default: `%(asctime)s,%(msecs)03.0f [%(name)-17s][%(levelname)-8s] %(message)s`

The format of the log file logging messages. Allowed formatting options can be seen on the *LogRecord attributes*.

```
log_fmt_logfile: '%(asctime)s,%(msecs)03.0f [%(name)-17s][%(levelname)-8s] %(message)s'
```

log_granular_levels

Default: {}

This can be used to control logging levels more specifically. The example sets the main salt library at the `warning' level, but sets `salt.modules` to log at the `debug` level:

```
log_granular_levels:
  'salt': 'warning'
  'salt.modules': 'debug'
```

External Logging Handlers

Besides the internal logging handlers used by salt, there are some external which can be used, see the *external logging handlers* document.

22.12 External Logging Handlers

logstash_mod	Logstash Logging Handler
sentry_mod	Sentry Logging Handler

22.12.1 Logstash Logging Handler

New in version 0.17.0.

This module provides some Logstash logging handlers.

UDP Logging Handler

For versions of Logstash before 1.2.0:

In the salt configuration file:

```
logstash_udp_handler:
  host: 127.0.0.1
  port: 9999
  version: 0
```

In the Logstash configuration file:

```
input {
  udp {
    type => "udp-type"
    format => "json_event"
```

```
  }
}
```

For version 1.2.0 of Logstash and newer:

In the salt configuration file:

```
logstash_udp_handler:
  host: 127.0.0.1
  port: 9999
  version: 1
```

In the Logstash configuration file:

```
input {
  udp {
    port => 9999
    codec => json
  }
}
```

Please read the UDP input configuration page for additional information.

ZeroMQ Logging Handler

For versions of Logstash before 1.2.0:

In the salt configuration file:

```
logstash_zmq_handler:
  address: tcp://127.0.0.1:2021
  version: 0
```

In the Logstash configuration file:

```
input {
  zeromq {
    type => "zeromq-type"
    mode => "server"
    topology => "pubsub"
    address => "tcp://0.0.0.0:2021"
    charset => "UTF-8"
    format => "json_event"
  }
}
```

For version 1.2.0 of Logstash and newer:

In the salt configuration file:

```
logstash_zmq_handler:
  address: tcp://127.0.0.1:2021
  version: 1
```

In the Logstash configuration file:

```
input {
  zeromq {
    topology => "pubsub"
    address => "tcp://0.0.0.0:2021"
```

```
    codec => json
  }
}
```

Please read the ZeroMQ input configuration page for additional information.

Important Logstash Setting

One of the most important settings that you should not forget on your Logstash configuration file regarding these logging handlers is `format`. Both the *UDP* and *ZeroMQ* inputs need to have `format` as `json_event` which is what we send over the wire.

Log Level

Both the `logstash_udp_handler` and the `logstash_zmq_handler` configuration sections accept an additional setting `log_level`. If not set, the logging level used will be the one defined for `log_level` in the global configuration file section.

HWM

The high water mark for the ZMQ socket setting. Only applicable for the `logstash_zmq_handler`.

Inspiration

This work was inspired in pylogstash, python-logstash, canary and the PyZMQ logging handler.

22.12.2 Sentry Logging Handler

New in version 0.17.0.

This module provides a Sentry logging handler.

Note

The Raven library needs to be installed on the system for this logging handler to be available.

Configuring the python Sentry client, Raven, should be done under the `sentry_handler` configuration key. At the bare minimum, you need to define the DSN. As an example:

```
sentry_handler:
  dsn: https://pub-key:secret-key@app.getsentry.com/app-id
```

More complex configurations can be achieved, for example:

```
sentry_handler:
  servers:
    - https://sentry.example.com
    - http://192.168.1.1
  project: app-id
  public_key: deadbeefdeadbeefdeadbeefdeadbeef
  secret_key: beefdeadbeefdeadbeefdeadbeefdead
```

All the client configuration keys are supported, please see the Raven client documentation.

The default logging level for the sentry handler is `ERROR`. If you wish to define a different one, define `log_level` under the `sentry_handler` configuration key:

```
sentry_handler:
  dsn: https://pub-key:secret-key@app.getsentry.com/app-id
  log_level: warning
```

The available log levels are those also available for the salt `cli` tools and configuration; `salt --help` should give you the required information.

Threaded Transports

Raven's documents rightly suggest using its threaded transport for critical applications. However, don't forget that if you start having troubles with Salt after enabling the threaded transport, please try switching to a non-threaded transport to see if that fixes your problem.

22.13 Salt File Server

Salt comes with a simple file server suitable for distributing files to the Salt minions. The file server is a stateless ZeroMQ server that is built into the Salt master.

The main intent of the Salt file server is to present files for use in the Salt state system. With this said, the Salt file server can be used for any general file transfer from the master to the minions.

22.13.1 File Server Backends

In Salt 0.12.0, the modular fileserver was introduced. This feature added the ability for the Salt Master to integrate different file server backends. File server backends allow the Salt file server to act as a transparent bridge to external resources. A good example of this is the `git` backend, which allows Salt to serve files sourced from one or more git repositories, but there are several others as well. Click *here* for a full list of Salt's fileserver backends.

Enabling a Fileserver Backend

Fileserver backends can be enabled with the `fileserver_backend` option.

```
fileserver_backend:
  - git
```

See the *documentation* for each backend to find the correct value to add to `fileserver_backend` in order to enable them.

Using Multiple Backends

If `fileserver_backend` is not defined in the Master config file, Salt will use the `roots` backend, but the `fileserver_backend` option supports multiple backends. When more than one backend is in use, the files from the enabled backends are merged into a single virtual filesystem. When a file is requested, the backends will be searched in order for that file, and the first backend to match will be the one which returns the file.

```
fileserver_backend:
  - roots
  - git
```

With this configuration, the environments and files defined in the `file_roots` parameter will be searched first, and if the file is not found then the git repositories defined in `gitfs_remotes` will be searched.

Environments

Just as the order of the values in `fileserver_backend` matters, so too does the order in which different sources are defined within a fileserver environment. For example, given the below `file_roots` configuration, if both `/srv/salt/dev/foo.txt` and `/srv/salt/prod/foo.txt` exist on the Master, then `salt://foo.txt` would point to `/srv/salt/dev/foo.txt` in the `dev` environment, but it would point to `/srv/salt/prod/foo.txt` in the `base` environment.

```
file_roots:
  base:
    - /srv/salt/prod
  qa:
    - /srv/salt/qa
    - /srv/salt/prod
  dev:
    - /srv/salt/dev
    - /srv/salt/qa
    - /srv/salt/prod
```

Similarly, when using the `git` backend, if both repositories defined below have a `hotfix23` branch/tag, and both of them also contain the file `bar.txt` in the root of the repository at that branch/tag, then `salt://bar.txt` in the `hotfix23` environment would be served from the `first` repository.

```
gitfs_remotes:
  - https://mydomain.tld/repos/first.git
  - https://mydomain.tld/repos/second.git
```

Note: Environments map differently based on the fileserver backend. For instance, the mappings are explicitly defined in `roots` backend, while in the VCS backends (`git`, `hg`, `svn`) the environments are created from branches/tags/bookmarks/etc. For the `minion` backend, the files are all in a single environment, which is specified by the `minionfs_env` option.

See the documentation for each backend for a more detailed explanation of how environments are mapped.

22.13.2 Dynamic Module Distribution

New in version 0.9.5.

Salt Python modules can be distributed automatically via the Salt file server. Under the root of any environment defined via the `file_roots` option on the master server directories corresponding to the type of module can be used.

The directories are prepended with an underscore:

1. `_modules`
2. `_grains`
3. `_renderers`

4. `_returners`
5. `_states`

The contents of these directories need to be synced over to the minions after Python modules have been created in them. There are a number of ways to sync the modules.

Sync Via States

The minion configuration contains an option `autoload_dynamic_modules` which defaults to True. This option makes the state system refresh all dynamic modules when states are run. To disable this behavior set `autoload_dynamic_modules` to False in the minion config.

When dynamic modules are autoloaded via states, modules only pertinent to the environments matched in the master's top file are downloaded.

This is important to remember, because modules can be manually loaded from any specific environment that environment specific modules will be loaded when a state run is executed.

Sync Via the saltutil Module

The saltutil module has a number of functions that can be used to sync all or specific dynamic modules. The saltutil module function `saltutil.sync_all` will sync all module types over to a minion. For more information see: `salt.modules.saltutil`

22.13.3 File Server Configuration

The Salt file server is a high performance file server written in ZeroMQ. It manages large files quickly and with little overhead, and has been optimized to handle small files in an extremely efficient manner.

The Salt file server is an environment aware file server. This means that files can be allocated within many root directories and accessed by specifying both the file path and the environment to search. The individual environments can span across multiple directory roots to create overlays and to allow for files to be organized in many flexible ways.

Environments

The Salt file server defaults to the mandatory `base` environment. This environment **MUST** be defined and is used to download files when no environment is specified.

Environments allow for files and sls data to be logically separated, but environments are not isolated from each other. This allows for logical isolation of environments by the engineer using Salt, but also allows for information to be used in multiple environments.

Directory Overlay

The `environment` setting is a list of directories to publish files from. These directories are searched in order to find the specified file and the first file found is returned.

This means that directory data is prioritized based on the order in which they are listed. In the case of this `file_roots` configuration:

```
file_roots:
  base:
    - /srv/salt/base
    - /srv/salt/failover
```

If a file's URI is `salt://httpd/httpd.conf`, it will first search for the file at `/srv/salt/base/httpd/httpd.conf`. If the file is found there it will be returned. If the file is not found there, then `/srv/salt/failover/httpd/httpd.conf` will be used for the source.

This allows for directories to be overlaid and prioritized based on the order they are defined in the configuration.

It is also possible to have `file_roots` which supports multiple environments:

```
file_roots:
  base:
    - /srv/salt/base
  dev:
    - /srv/salt/dev
    - /srv/salt/base
  prod:
    - /srv/salt/prod
    - /srv/salt/base
```

This example ensures that each environment will check the associated environment directory for files first. If a file is not found in the appropriate directory, the system will default to using the base directory.

Local File Server

New in version 0.9.8.

The file server can be rerouted to run from the minion. This is primarily to enable running Salt states without a Salt master. To use the local file server interface, copy the file server data to the minion and set the file_roots option on the minion to point to the directories copied from the master. Once the minion `file_roots` option has been set, change the `file_client` option to local to make sure that the local file server interface is used.

22.13.4 The cp Module

The cp module is the home of minion side file server operations. The cp module is used by the Salt state system, salt-cp, and can be used to distribute files presented by the Salt file server.

Environments

Since the file server is made to work with the Salt state system, it supports environments. The environments are defined in the master config file and when referencing an environment the file specified will be based on the root directory of the environment.

get_file

The cp.get_file function can be used on the minion to download a file from the master, the syntax looks like this:

```
# salt '*' cp.get_file salt://vimrc /etc/vimrc
```

This will instruct all Salt minions to download the vimrc file and copy it to /etc/vimrc

Template rendering can be enabled on both the source and destination file names like so:

```
# salt '*' cp.get_file "salt://{{grains.os}}/vimrc" /etc/vimrc template=jinja
```

This example would instruct all Salt minions to download the vimrc from a directory with the same name as their OS grain and copy it to /etc/vimrc

For larger files, the cp.get_file module also supports gzip compression. Because gzip is CPU-intensive, this should only be used in scenarios where the compression ratio is very high (e.g. pretty-printed JSON or YAML files).

To use compression, use the `gzip` named argument. Valid values are integers from 1 to 9, where 1 is the lightest compression and 9 the heaviest. In other words, 1 uses the least CPU on the master (and minion), while 9 uses the most.

```
# salt '*' cp.get_file salt://vimrc /etc/vimrc gzip=5
```

Finally, note that by default cp.get_file does *not* create new destination directories if they do not exist. To change this, use the `makedirs` argument:

```
# salt '*' cp.get_file salt://vimrc /etc/vim/vimrc makedirs=True
```

In this example, /etc/vim/ would be created if it didn't already exist.

get_dir

The cp.get_dir function can be used on the minion to download an entire directory from the master. The syntax is very similar to get_file:

```
# salt '*' cp.get_dir salt://etc/apache2 /etc
```

cp.get_dir supports template rendering and gzip compression arguments just like get_file:

```
# salt '*' cp.get_dir salt://etc/{{pillar.webserver}} /etc gzip=5 template=jinja
```

22.13.5 File Server Client API

A client API is available which allows for modules and applications to be written which make use of the Salt file server.

The file server uses the same authentication and encryption used by the rest of the Salt system for network communication.

FileClient Class

The FileClient class is used to set up the communication from the minion to the master. When creating a FileClient object the minion configuration needs to be passed in. When using the FileClient from within a minion module the built in `__opts__` data can be passed:

```
import salt.minion

def get_file(path, dest, env='base'):
    '''
    Used to get a single file from the Salt master

    CLI Example:
    salt '*' cp.get_file salt://vimrc /etc/vimrc
    '''
```

```
    # Create the FileClient object
    client = salt.minion.FileClient(__opts__)
    # Call get_file
    return client.get_file(path, dest, False, env)
```

Using the FileClient class outside of a minion module where the `__opts__` data is not available, it needs to be generated:

```
import salt.minion
import salt.config

def get_file(path, dest, env='base'):
    '''
    Used to get a single file from the Salt master
    '''
    # Get the configuration data
    opts = salt.config.minion_config('/etc/salt/minion')
    # Create the FileClient object
    client = salt.minion.FileClient(opts)
    # Call get_file
    return client.get_file(path, dest, False, env)
```

22.14 Full list of builtin fileserver modules

gitfs	Git Fileserver Backend
hgfs	Mercurial Fileserver Backend
minionfs	Fileserver backend which serves files pushed to the Master
roots	The default file server backend
s3fs	Amazon S3 Fileserver Backend
svnfs	Subversion Fileserver Backend

22.14.1 salt.fileserver.gitfs

Git Fileserver Backend

With this backend, branches and tags in a remote git repository are exposed to salt as different environments.

To enable, add `git` to the `fileserver_backend` option in the Master config file.

```
fileserver_backend:
  - git
```

As of Salt 2014.7.0, the Git fileserver backend supports GitPython, pygit2, and dulwich to provide the Python interface to git. If more than one of these are present, the order of preference for which one will be chosen is the same as the order in which they were listed: pygit2, GitPython, dulwich (keep in mind, this order is subject to change).

An optional master config parameter (`gitfs_provider`) can be used to specify which provider should be used.

More detailed information on how to use gitfs can be found in the *Gitfs Walkthrough*.

Note: Minimum requirements

To use GitPython for gitfs requires a minimum GitPython version of 0.3.0, as well as the git CLI utility. Instructions for installing GitPython can be found *here*.

To use pygit2 for gitfs requires a minimum pygit2 version of 0.20.3. pygit2 0.20.3 requires libgit2 0.20.0. pygit2 and libgit2 are developed alongside one another, so it is recommended to keep them both at the same major release to avoid unexpected behavior. For example, pygit2 0.21.x requires libgit2 0.21.x, pygit2 0.22.x will require libgit2 0.22.x, etc.

To find stale refs, pygit2 additionally requires the git CLI utility to be installed.

`salt.fileserver.gitfs.`**`clear_cache`**`()`
Completely clear gitfs cache

`salt.fileserver.gitfs.`**`clear_lock`**`(`*remote=None*`)`
Clear update.lk

`remote` can either be a dictionary containing repo configuration information, or a pattern. If the latter, then remotes for which the URL matches the pattern will be locked.

`salt.fileserver.gitfs.`**`dir_list`**`(`*load*`)`
Return a list of all directories on the master

`salt.fileserver.gitfs.`**`envs`**`(`*ignore_cache=False*, *skip_clean=False*`)`
Return a list of refs that can be used as environments

`salt.fileserver.gitfs.`**`file_hash`**`(`*load*, *fnd*`)`
Return a file hash, the hash type is set in the master config file

`salt.fileserver.gitfs.`**`file_list`**`(`*load*`)`
Return a list of all files on the file server in a specified environment

`salt.fileserver.gitfs.`**`file_list_emptydirs`**`(`*load*`)`
Return a list of all empty directories on the master

`salt.fileserver.gitfs.`**`find_file`**`(`*path*, *tgt_env='base'*, ***kwargs*`)`
Find the first file to match the path and ref, read the file out of git and send the path to the newly cached file

`salt.fileserver.gitfs.`**`init`**`()`
Return the git repo object for this session

`salt.fileserver.gitfs.`**`lock`**`(`*remote=None*`)`
Place an update.lk

`remote` can either be a dictionary containing repo configuration information, or a pattern. If the latter, then remotes for which the URL matches the pattern will be locked.

`salt.fileserver.gitfs.`**`serve_file`**`(`*load*, *fnd*`)`
Return a chunk from a file based on the data received

`salt.fileserver.gitfs.`**`symlink_list`**`(`*load*`)`
Return a dict of all symlinks based on a given path in the repo

`salt.fileserver.gitfs.`**`update`**`()`
Execute a git fetch on all of the repos

22.14.2 salt.fileserver.hgfs

Mercurial Fileserver Backend

To enable, add `hg` to the `fileserver_backend` option in the Master config file.

```
fileserver_backend:
  - hg
```

After enabling this backend, branches, bookmarks, and tags in a remote mercurial repository are exposed to salt as different environments. This feature is managed by the `fileserver_backend` option in the salt master config file.

This fileserver has an additional option `hgfs_branch_method` that will set the desired branch method. Possible values are: `branches`, `bookmarks`, or `mixed`. If using `branches` or `mixed`, the `default` branch will be mapped to `base`.

Changed in version 2014.1.0: The `hgfs_base` master config parameter was added, allowing for a branch other than `default` to be used for the `base` environment, and allowing for a `base` environment to be specified when using an `hgfs_branch_method` of `bookmarks`.

depends

- mercurial
- python bindings for mercurial (`python-hglib`)

`salt.fileserver.hgfs.`**`clear_cache`**`()`
Completely clear hgfs cache

`salt.fileserver.hgfs.`**`clear_lock`**`(`*remote=None*`)`
Clear update.lk

`remote` can either be a dictionary containing repo configuration information, or a pattern. If the latter, then remotes for which the URL matches the pattern will be locked.

`salt.fileserver.hgfs.`**`dir_list`**`(`*load*`)`
Return a list of all directories on the master

`salt.fileserver.hgfs.`**`envs`**`(`*ignore_cache=False*`)`
Return a list of refs that can be used as environments

`salt.fileserver.hgfs.`**`file_hash`**`(`*load*, *fnd*`)`
Return a file hash, the hash type is set in the master config file

`salt.fileserver.hgfs.`**`file_list`**`(`*load*`)`
Return a list of all files on the file server in a specified environment

`salt.fileserver.hgfs.`**`file_list_emptydirs`**`(`*load*`)`
Return a list of all empty directories on the master

`salt.fileserver.hgfs.`**`find_file`**`(`*path*, *tgt_env='base'*, ***kwargs*`)`
Find the first file to match the path and ref, read the file out of hg and send the path to the newly cached file

`salt.fileserver.hgfs.`**`init`**`()`
Return a list of hglib objects for the various hgfs remotes

`salt.fileserver.hgfs.`**`lock`**`(`*remote=None*`)`
Place an update.lk

`remote` can either be a dictionary containing repo configuration information, or a pattern. If the latter, then remotes for which the URL matches the pattern will be locked.

`salt.fileserver.hgfs.`**`serve_file`**`(`*load*, *fnd*`)`
Return a chunk from a file based on the data received

`salt.fileserver.hgfs.`**`update`**`()`
Execute an hg pull on all of the repos

22.14.3 salt.fileserver.minionfs

Fileserver backend which serves files pushed to the Master

The cp.push function allows Minions to push files up to the Master. Using this backend, these pushed files are exposed to other Minions via the Salt fileserver.

To enable minionfs, file_recv needs to be set to True in the master config file (otherwise cp.push will not be allowed to push files to the Master), and minion must be added to the fileserver_backends list.

```
fileserver_backend:
  - minion
```

Other minionfs settings include: minionfs_whitelist, minionfs_blacklist, minionfs_mountpoint, and minionfs_env.

See also:

MinionFS Backend Walkthrough

salt.fileserver.minionfs.**dir_list**(*load*)
Return a list of all directories on the master

CLI Example:

```
$ salt 'source-minion' cp.push /absolute/path/file  # Push the file to the master
$ salt 'destination-minion' cp.list_master_dirs
destination-minion:
    - source-minion/absolute
    - source-minion/absolute/path
```

salt.fileserver.minionfs.**envs**()
Returns the one environment specified for minionfs in the master configuration.

salt.fileserver.minionfs.**file_hash**(*load*, *fnd*)
Return a file hash, the hash type is set in the master config file

salt.fileserver.minionfs.**file_list**(*load*)
Return a list of all files on the file server in a specified environment

salt.fileserver.minionfs.**find_file**(*path*, *tgt_env='base'*, ***kwargs*)
Search the environment for the relative path

salt.fileserver.minionfs.**serve_file**(*load*, *fnd*)
Return a chunk from a file based on the data received

CLI Example:

```
# Push the file to the master
$ salt 'source-minion' cp.push /path/to/the/file
$ salt 'destination-minion' cp.get_file salt://source-minion/path/to/the/file /destination/file
```

salt.fileserver.minionfs.**update**()
When we are asked to update (regular interval) lets reap the cache

22.14.4 salt.fileserver.roots

The default file server backend

This fileserver backend serves files from the Master's local filesystem. If fileserver_backend is not defined in the Master config file, then this backend is enabled by default. If it *is* defined then roots must be in the fileserver_backend list to enable this backend.

```
fileserver_backend:
  - roots
```

Fileserver environments are defined using the `file_roots` configuration option.

`salt.fileserver.roots.`**`dir_list`**(*load*)
Return a list of all directories on the master

`salt.fileserver.roots.`**`envs`**()
Return the file server environments

`salt.fileserver.roots.`**`file_hash`**(*load*, *fnd*)
Return a file hash, the hash type is set in the master config file

`salt.fileserver.roots.`**`file_list`**(*load*)
Return a list of all files on the file server in a specified environment

`salt.fileserver.roots.`**`file_list_emptydirs`**(*load*)
Return a list of all empty directories on the master

`salt.fileserver.roots.`**`find_file`**(*path*, *saltenv='base'*, *env=None*, ***kwargs*)
Search the environment for the relative path

`salt.fileserver.roots.`**`serve_file`**(*load*, *fnd*)
Return a chunk from a file based on the data received

`salt.fileserver.roots.`**`symlink_list`**(*load*)
Return a dict of all symlinks based on a given path on the Master

`salt.fileserver.roots.`**`update`**()
When we are asked to update (regular interval) lets reap the cache

22.14.5 salt.fileserver.s3fs

Amazon S3 Fileserver Backend

This backend exposes directories in S3 buckets as Salt environments. To enable this backend, add `s3fs` to the `fileserver_backend` option in the Master config file.

```
fileserver_backend:
  - s3fs
```

S3 credentials must also be set in the master config file:

```
s3.keyid: GKTADJGHEIQSXMKKRBJ08H
s3.key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs
```

Alternatively, if on EC2 these credentials can be automatically loaded from instance metadata.

This fileserver supports two modes of operation for the buckets:

1. **A single bucket per environment**

```
    s3.buckets:
      production:
        - bucket1
        - bucket2
      staging:
        - bucket3
        - bucket4
```

2. **Multiple environments per bucket**

```
    s3.buckets:
      - bucket1
      - bucket2
      - bucket3
      - bucket4
```

Note that bucket names must be all lowercase both in the AWS console and in Salt, otherwise you may encounter `SignatureDoesNotMatch` errors.

A multiple-environment bucket must adhere to the following root directory structure:

```
s3://<bucket name>/<environment>/<files>
```

Note: This fileserver back-end requires the use of the MD5 hashing algorithm. MD5 may not be compliant with all security policies.

`salt.fileserver.s3fs.`**`dir_list`**(*load*)
Return a list of all directories on the master

`salt.fileserver.s3fs.`**`envs`**()
Return a list of directories within the bucket that can be used as environments.

`salt.fileserver.s3fs.`**`file_hash`**(*load*, *fnd*)
Return an MD5 file hash

`salt.fileserver.s3fs.`**`file_list`**(*load*)
Return a list of all files on the file server in a specified environment

`salt.fileserver.s3fs.`**`file_list_emptydirs`**(*load*)
Return a list of all empty directories on the master

`salt.fileserver.s3fs.`**`find_file`**(*path*, *saltenv='base'*, *env=None*, ***kwargs*)
Look through the buckets cache file for a match. If the field is found, it is retrieved from S3 only if its cached version is missing, or if the MD5 does not match.

`salt.fileserver.s3fs.`**`serve_file`**(*load*, *fnd*)
Return a chunk from a file based on the data received

`salt.fileserver.s3fs.`**`update`**()
Update the cache file for the bucket.

22.14.6 salt.fileserver.svnfs

Subversion Fileserver Backend

After enabling this backend, branches, and tags in a remote subversion repository are exposed to salt as different environments. To enable this backend, add `svn` to the `fileserver_backend` option in the Master config file.

```
fileserver_backend:
  - svn
```

This backend assumes a standard svn layout with directories for `branches`, `tags`, and `trunk`, at the repository root.

depends

- subversion
- pysvn

Changed in version 2014.7.0: The paths to the trunk, branches, and tags have been made configurable, via the config options `svnfs_trunk`, `svnfs_branches`, and `svnfs_tags`. `svnfs_mountpoint` was also added. Finally, support for per-remote configuration parameters was added. See the `documentation` for more information.

`salt.fileserver.svnfs.`**`clear_cache`**`()`
Completely clear svnfs cache

`salt.fileserver.svnfs.`**`clear_lock`**`(`*remote=None*`)`
Clear update.lk

`remote` can either be a dictionary containing repo configuration information, or a pattern. If the latter, then remotes for which the URL matches the pattern will be locked.

`salt.fileserver.svnfs.`**`dir_list`**`(`*load*`)`
Return a list of all directories on the master

`salt.fileserver.svnfs.`**`envs`**`(`*ignore_cache=False*`)`
Return a list of refs that can be used as environments

`salt.fileserver.svnfs.`**`file_hash`**`(`*load*, *fnd*`)`
Return a file hash, the hash type is set in the master config file

`salt.fileserver.svnfs.`**`file_list`**`(`*load*`)`
Return a list of all files on the file server in a specified environment

`salt.fileserver.svnfs.`**`file_list_emptydirs`**`(`*load*`)`
Return a list of all empty directories on the master

`salt.fileserver.svnfs.`**`find_file`**`(`*path*, *tgt_env='base'*, ***kwargs*`)`
Find the first file to match the path and ref. This operates similarly to the roots file sever but with assumptions of the directory structure based on svn standard practices.

`salt.fileserver.svnfs.`**`init`**`()`
Return the list of svn remotes and their configuration information

`salt.fileserver.svnfs.`**`lock`**`(`*remote=None*`)`
Place an update.lk

`remote` can either be a dictionary containing repo configuration information, or a pattern. If the latter, then remotes for which the URL matches the pattern will be locked.

`salt.fileserver.svnfs.`**`serve_file`**`(`*load*, *fnd*`)`
Return a chunk from a file based on the data received

`salt.fileserver.svnfs.`**`update`**`()`
Execute an svn update on all of the repos

22.15 Salt code and internals

Reference documentation on Salt's internal code.

22.15.1 Contents

salt.serializers

salt.utils.aggregation

This library makes it possible to introspect dataset and aggregate nodes when it is instructed.

Note: The following examples with be expressed in YAML for convenience's sake:

- !aggr-scalar will refer to Scalar python function
- !aggr-map will refer to Map python object
- !aggr-seq will refer for Sequence python object

How to instructs merging This yaml document has duplicate keys:

```
foo: !aggr-scalar first
foo: !aggr-scalar second
bar: !aggr-map {first: foo}
bar: !aggr-map {second: bar}
baz: !aggr-scalar 42
```

but tagged values instruct Salt that overlapping values they can be merged together:

```
foo: !aggr-seq [first, second]
bar: !aggr-map {first: foo, second: bar}
baz: !aggr-seq [42]
```

Default merge strategy is keep untouched For example, this yaml document still has duplicate keys, but does not instruct aggregation:

```
foo: first
foo: second
bar: {first: foo}
bar: {second: bar}
baz: 42
```

So the late found values prevail:

```
foo: second
bar: {second: bar}
baz: 42
```

Limitations Aggregation is permitted between tagged objects that share the same type. If not, the default merge strategy prevails.

For example, these examples:

```
foo: {first: value}
foo: !aggr-map {second: value}

bar: !aggr-map {first: value}
bar: 42

baz: !aggr-seq [42]
baz: [fail]

qux: 42
qux: !aggr-scalar fail
```

are interpreted like this:

```
foo: !aggr-map{second: value}

bar: 42

baz: [fail]

qux: !aggr-seq [fail]
```

Introspection TODO: write this part

salt.utils.aggregation.**aggregate**(*obj_a*, *obj_b*, *level=False*, *map_class=<class 'salt.utils.aggregation.Map'>*, *sequence_class=<class 'salt.utils.aggregation.Sequence'>*)

Merge obj_b into obj_a.

```
>>> aggregate('first', 'second', True) == ['first', 'second']
True
```

class salt.utils.aggregation.**Aggregate**

Aggregation base.

class salt.utils.aggregation.**Map**(**args*, ***kwds*)

Map aggregation.

salt.utils.aggregation.**Scalar**(*obj*)

Shortcut for Sequence creation

```
>>> Scalar('foo') == Sequence(['foo'])
True
```

class salt.utils.aggregation.**Sequence**

Sequence aggregation.

Exceptions

Salt-specific exceptions should be thrown as often as possible so the various interfaces to Salt (CLI, API, etc) can handle those errors appropriately and display error messages appropriately.

salt.exceptions	This module is a central location for all salt exceptions

salt.exceptions

This module is a central location for all salt exceptions

exception salt.exceptions.**AuthenticationError**(*message=''*)

If sha256 signature fails during decryption

exception salt.exceptions.**AuthorizationError**(*message=''*)

Thrown when runner or wheel execution fails due to permissions

exception salt.exceptions.**CommandExecutionError**(*message=''*)

Used when a module runs a command which returns an error and wants to show the user the output gracefully instead of dying

exception salt.exceptions.**CommandNotFoundError**(*message=''*)

Used in modules or grains when a required binary is not available

exception salt.exceptions.**EauthAuthenticationError**(*message=''*)
Thrown when eauth authentication fails

exception salt.exceptions.**FileserverConfigError**(*message=''*)
Used when invalid fileserver settings are detected

exception salt.exceptions.**LoaderError**(*message=''*)
Problems loading the right renderer

exception salt.exceptions.**MasterExit**
Rise when the master exits

exception salt.exceptions.**MinionError**(*message=''*)
Minion problems reading uris such as salt:// or http://

exception salt.exceptions.**PkgParseError**(*message=''*)
Used when of the pkg modules cannot correctly parse the output from the CLI tool (pacman, yum, apt, aptitude, etc)

exception salt.exceptions.**PublishError**(*message=''*)
Problems encountered when trying to publish a command

exception salt.exceptions.**SaltClientError**(*message=''*)
Problem reading the master root key

exception salt.exceptions.**SaltClientTimeout**(*msg*, *jid=None*, **args*, ***kwargs*)
Thrown when a job sent through one of the Client interfaces times out

Takes the `jid` as a parameter

exception salt.exceptions.**SaltCloudConfigError**(*message=''*)
Raised when a configuration setting is not found and should exist.

exception salt.exceptions.**SaltCloudException**(*message=''*)
Generic Salt Cloud Exception

exception salt.exceptions.**SaltCloudExecutionFailure**(*message=''*)
Raised when too much failures have occurred while querying/waiting for data.

exception salt.exceptions.**SaltCloudExecutionTimeout**(*message=''*)
Raised when too much time has passed while querying/waiting for data.

exception salt.exceptions.**SaltCloudNotFound**(*message=''*)
Raised when some cloud provider function cannot find what's being searched.

exception salt.exceptions.**SaltCloudPasswordError**(*message=''*)
Raise when virtual terminal password input failed

exception salt.exceptions.**SaltCloudSystemExit**(*message*, *exit_code=1*)
This exception is raised when the execution should be stopped.

exception salt.exceptions.**SaltDaemonNotRunning**(*message=''*)
Throw when a running master/minion/syndic is not running but is needed to perform the requested operation (e.g., eauth).

exception salt.exceptions.**SaltException**(*message=''*)
Base exception class; all Salt-specific exceptions should subclass this

pack()
Pack this exception into a serializable dictionary that is safe for transport via msgpack

exception salt.exceptions.**SaltInvocationError**(*message=''*)
Used when the wrong number of arguments are sent to modules or invalid arguments are specified on the command line

exception salt.exceptions.**SaltMasterError**(*message=''*)
Problem reading the master root key

exception salt.exceptions.**SaltRenderError**(*message*, *line_num=None*, *buf=''*, *marker='<======================'*, *trace=None*)
Used when a renderer needs to raise an explicit error. If a line number and buffer string are passed, get_context will be invoked to get the location of the error.

exception salt.exceptions.**SaltReqTimeoutError**(*message=''*)
Thrown when a salt master request call fails to return within the timeout

exception salt.exceptions.**SaltRunnerError**(*message=''*)
Problem in runner

exception salt.exceptions.**SaltSyndicMasterError**(*message=''*)
Problem while proxying a request in the syndication master

exception salt.exceptions.**SaltSystemExit**(*code=0*, *msg=None*)
This exception is raised when an unsolvable problem is found. There's nothing else to do, salt should just exit.

exception salt.exceptions.**SaltWheelError**(*message=''*)
Problem in wheel

exception salt.exceptions.**TimedProcTimeoutError**(*message=''*)
Thrown when a timed subprocess does not terminate within the timeout, or if the specified timeout is not an int or a float

exception salt.exceptions.**TokenAuthenticationError**(*message=''*)
Thrown when token authentication fails

salt.exceptions

This module is a central location for all salt exceptions

exception salt.exceptions.**AuthenticationError**(*message=''*)
If sha256 signature fails during decryption

exception salt.exceptions.**AuthorizationError**(*message=''*)
Thrown when runner or wheel execution fails due to permissions

exception salt.exceptions.**CommandExecutionError**(*message=''*)
Used when a module runs a command which returns an error and wants to show the user the output gracefully instead of dying

exception salt.exceptions.**CommandNotFoundError**(*message=''*)
Used in modules or grains when a required binary is not available

exception salt.exceptions.**EauthAuthenticationError**(*message=''*)
Thrown when eauth authentication fails

exception salt.exceptions.**FileserverConfigError**(*message=''*)
Used when invalid fileserver settings are detected

exception salt.exceptions.**LoaderError**(*message=''*)
Problems loading the right renderer

exception salt.exceptions.**MasterExit**
Rise when the master exits

exception salt.exceptions.**MinionError**(*message=''*)
Minion problems reading uris such as salt:// or http://

exception salt.exceptions.**PkgParseError**(*message=''*)
Used when of the pkg modules cannot correctly parse the output from the CLI tool (pacman, yum, apt, aptitude, etc)

exception salt.exceptions.**PublishError**(*message=''*)
Problems encountered when trying to publish a command

exception salt.exceptions.**SaltClientError**(*message=''*)
Problem reading the master root key

exception salt.exceptions.**SaltClientTimeout**(*msg*, *jid=None*, **args*, ***kwargs*)
Thrown when a job sent through one of the Client interfaces times out

Takes the `jid` as a parameter

exception salt.exceptions.**SaltCloudConfigError**(*message=''*)
Raised when a configuration setting is not found and should exist.

exception salt.exceptions.**SaltCloudException**(*message=''*)
Generic Salt Cloud Exception

exception salt.exceptions.**SaltCloudExecutionFailure**(*message=''*)
Raised when too much failures have occurred while querying/waiting for data.

exception salt.exceptions.**SaltCloudExecutionTimeout**(*message=''*)
Raised when too much time has passed while querying/waiting for data.

exception salt.exceptions.**SaltCloudNotFound**(*message=''*)
Raised when some cloud provider function cannot find what's being searched.

exception salt.exceptions.**SaltCloudPasswordError**(*message=''*)
Raise when virtual terminal password input failed

exception salt.exceptions.**SaltCloudSystemExit**(*message*, *exit_code=1*)
This exception is raised when the execution should be stopped.

exception salt.exceptions.**SaltDaemonNotRunning**(*message=''*)
Throw when a running master/minion/syndic is not running but is needed to perform the requested operation (e.g., eauth).

exception salt.exceptions.**SaltException**(*message=''*)
Base exception class; all Salt-specific exceptions should subclass this

pack()
Pack this exception into a serializable dictionary that is safe for transport via msgpack

exception salt.exceptions.**SaltInvocationError**(*message=''*)
Used when the wrong number of arguments are sent to modules or invalid arguments are specified on the command line

exception salt.exceptions.**SaltMasterError**(*message=''*)
Problem reading the master root key

exception salt.exceptions.**SaltRenderError**(*message*, *line_num=None*, *buf=''*, *marker='<======================'*, *trace=None*)
Used when a renderer needs to raise an explicit error. If a line number and buffer string are passed, get_context will be invoked to get the location of the error.

exception salt.exceptions.**SaltReqTimeoutError**(*message=''*)
Thrown when a salt master request call fails to return within the timeout

exception salt.exceptions.**SaltRunnerError**(*message=''*)
Problem in runner

exception salt.exceptions.**SaltSyndicMasterError**(*message=''*)
Problem while proxying a request in the syndication master

exception salt.exceptions.**SaltSystemExit**(*code=0*, *msg=None*)
This exception is raised when an unsolvable problem is found. There's nothing else to do, salt should just exit.

exception salt.exceptions.**SaltWheelError**(*message=''*)
Problem in wheel

exception salt.exceptions.**TimedProcTimeoutError**(*message=''*)
Thrown when a timed subprocess does not terminate within the timeout, or if the specified timeout is not an int or a float

exception salt.exceptions.**TokenAuthenticationError**(*message=''*)
Thrown when token authentication fails

salt.serializers

salt.utils.serializers

This module implements all the serializers needed by salt. Each serializer offers the same functions and attributes:

deserialize function for deserializing string or stream

serialize function for serializing a Python object

available flag that tells if the serializer is available (all dependencies are met etc.)

salt.utils.serializers.json

Implements JSON serializer.

It's just a wrapper around json (or simplejson if available).

salt.utils.serializers.json.**deserialize**(*stream_or_string*, ***options*)
Deserialize any string of stream like object into a Python data structure.

Parameters

- **stream_or_string** -- stream or string to deserialize.
- **options** -- options given to lower json/simplejson module.

salt.utils.serializers.json.**serialize**(*obj*, ***options*)
Serialize Python data to JSON.

Parameters

- **obj** -- the data structure to serialize
- **options** -- options given to lower json/simplejson module.

salt.utils.serializers.yaml

Implements YAML serializer.

Underneath, it is based on pyyaml and use the safe dumper and loader. It also use C bindings if they are available.

salt.utils.serializers.yaml.**deserialize**(*stream_or_string*, ***options*)
Deserialize any string of stream like object into a Python data structure.

Parameters

- **stream_or_string** -- stream or string to deserialize.
- **options** -- options given to lower yaml module.

salt.utils.serializers.yaml.**serialize**(*obj*, ***options*)
Serialize Python data to YAML.

Parameters

- **obj** -- the data structure to serialize
- **options** -- options given to lower yaml module.

salt.utils.serializers.msgpack

Implements MsgPack serializer.

salt.utils.serializers.msgpack.**deserialize**(*stream_or_string*, ***options*)
Deserialize any string of stream like object into a Python data structure.

Parameters

- **stream_or_string** -- stream or string to deserialize.
- **options** -- options given to lower msgpack module.

salt.utils.serializers.msgpack.**serialize**(*obj*, ***options*)
Serialize Python data to MsgPack.

Parameters

- **obj** -- the data structure to serialize
- **options** -- options given to lower msgpack module.

22.16 Full list of builtin execution modules

Virtual modules

22.16.1 salt.modules.pkg

pkg is a virtual module that is fulfilled by one of the following modules:

- salt.modules.aptpkg
- salt.modules.brew
- salt.modules.ebuild
- salt.modules.freebsdpkg
- salt.modules.openbsdpkg
- salt.modules.pacman
- salt.modules.pkgin
- salt.modules.pkgng
- salt.modules.pkgutil

- salt.modules.solarispkg
- salt.modules.win_pkg
- salt.modules.yumpkg
- salt.modules.zypper

aliases	Manage the information in the aliases file
alternatives	Support for Alternatives system
apache	Support for Apache
aptpkg	Support for APT (Advanced Packaging Tool)
archive	A module to wrap (non-Windows) archive calls
artifactory	Module for fetching artifacts from Artifactory
at	Wrapper module for at(1)
augeas_cfg	Manages configuration files via augeas
aws_sqs	Support for the Amazon Simple Queue Service.
blockdev	Module for managing block devices
bluez	Support for Bluetooth (using BlueZ in Linux).
boto_asg	Connection module for Amazon Autoscale Groups
boto_cfn	Connection module for Amazon Cloud Formation
boto_cloudwatch	Connection module for Amazon CloudWatch
boto_dynamodb	Connection module for Amazon DynamoDB
boto_elasticache	Connection module for Amazon Elasticache
boto_elb	Connection module for Amazon ELB
boto_iam	Connection module for Amazon IAM
boto_route53	Connection module for Amazon Route53
boto_secgroup	Connection module for Amazon Security Groups
boto_sns	Connection module for Amazon SNS
boto_sqs	Connection module for Amazon SQS
boto_vpc	Connection module for Amazon VPC
brew	Homebrew for Mac OS X
bridge	Module for gathering and managing bridging information
bsd_shadow	Manage the password database on BSD systems
btrfs	Module for managing BTRFS file systems.
cassandra	Cassandra NoSQL Database Module
chef	Execute chef in server or solo mode
chocolatey	A dead simple module wrapping calls to the Chocolatey package manager
cloud	Salt-specific interface for calling Salt Cloud directly
cmdmod	A module for shelling out
composer	Use composer to install PHP dependencies for a directory
config	Return config information
container_resource	Common resources for LXC and systemd-nspawn containers
cp	Minion side functions for salt-cp
cpan	Manage Perl modules using CPAN
cron	Work with cron
cyg	Manage cygwin packages.
daemontools	daemontools service module. This module will create daemontools type
darwin_sysctl	Module for viewing and modifying sysctl parameters
data	Manage a local persistent data structure that can hold any arbitrary data
ddns	Support for RFC 2136 dynamic DNS updates.
deb_apache	Support for Apache

Table 22.6 -- continued from previous page

debconfmod	Support for Debconf
debian_ip	The networking module for Debian based distros
debian_service	Service support for Debian systems (uses update-rc.d and /sbin/service)
defaults	
devmap	Device-Mapper module
dig	Compendium of generic DNS utilities
disk	Module for gathering disk information
djangomod	Manage Django sites
dnsmasq	Module for managing dnsmasq
dnsutil	Compendium of generic DNS utilities
dockerio	Management of Docker Containers
dpkg	Support for DEB packages
drac	Manage Dell DRAC
drbd	DRBD administration module
ebuild	Support for Portage
eix	Support for Eix
elasticsearch	Connection module for Elasticsearch
environ	Support for getting and setting the environment variables of the current salt process.
eselect	Support for eselect, Gentoo's configuration and management tool.
etcd_mod	Execution module to work with etcd
event	Use the *Salt Event System* to fire events from the master to the minion and vice-versa.
extfs	Module for managing ext2/3/4 file systems
file	Manage information about regular files, directories,
firewalld	Support for firewalld
freebsd_sysctl	Module for viewing and modifying sysctl parameters
freebsdjail	The jail module for FreeBSD
freebsdkmod	Module to manage FreeBSD kernel modules
freebsdpkg	Remote package support using `pkg_add(1)`
freebsdports	Install software from the FreeBSD `ports(7)` system
freebsdservice	The service module for FreeBSD
fsutils	Run-time utilities
gem	Manage ruby gems.
genesis	Module for managing container and VM images
gentoo_service	Top level package command wrapper, used to translate the os detected by grains
gentoolkitmod	Support for Gentoolkit
git	Support for the Git SCM
glance	Module for handling openstack glance calls.
glusterfs	Manage a glusterfs pool
gnomedesktop	GNOME implementations
gpg	Manage a GPG keychains, add keys, create keys, retrieve keys from keyservers.
grains	Return/control aspects of the grains data
groupadd	Manage groups on Linux, OpenBSD and NetBSD
grub_legacy	Support for GRUB Legacy
guestfs	Interact with virtual machine images via libguestfs
hadoop	Support for hadoop
haproxyconn	Support for haproxy
hashutil	A collection of hashing and encoding functions
hg	Support for the Mercurial SCM
hipchat	
hosts	Manage the information in the hosts file

Table 22.6 -- continued from previous page

htpasswd	Support for htpasswd command
http	Module for making various web calls.
ilo	Manage HP ILO
img	Virtual machine image management tools
incron	Work with incron
influx	InfluxDB - A distributed time series database
ini_manage	Edit ini files
introspect	Functions to perform introspection on a minion, and return data in a format
ipset	Support for ipset
iptables	Support for iptables
jboss7	Module for managing JBoss AS 7 through the CLI interface.
jboss7_cli	Module for low-level interaction with JbossAS7 through CLI.
junos	Module for interfacing to Junos devices
key	Functions to view the minion's public key information
keyboard	Module for managing keyboards on supported POSIX-like systems using systemd, or such as Redha
keystone	Module for handling openstack keystone calls.
kmod	Module to manage Linux kernel modules
launchctl	Module for the management of MacOS systems that use launchd/launchctl
layman	Support for Layman
ldapmod	Salt interface to LDAP commands
linux_acl	Support for Linux File Access Control Lists
linux_lvm	Support for Linux LVM2
linux_sysctl	Module for viewing and modifying sysctl parameters
localemod	Module for managing locales on POSIX-like systems.
locate	Module for using the locate utilities
logadm	Module for managing Solaris logadm based log rotations.
logrotate	Module for managing logrotate.
lvs	Support for LVS (Linux Virtual Server)
lxc	Control Linux Containers via Salt
mac_group	Manage groups on Mac OS 10.7+
mac_user	Manage users on Mac OS 10.7+
macports	Support for MacPorts under Mac OSX.
makeconf	Support for modifying make.conf under Gentoo
match	The match module allows for match routines to be run and determine target specs
mdadm	Salt module to manage RAID arrays with mdadm
memcached	Module for Management of Memcached Keys
mine	The function cache system allows for data to be stored on the master so it can be easily read by oth
mod_random	New in version 2014.7.0.
modjk	Control Modjk via the Apache Tomcat ``Status'' worker
mongodb	Module to provide MongoDB functionality to Salt
monit	Monit service module.
moosefs	Module for gathering and managing information about MooseFS
mount	Salt module to manage unix mounts and the fstab file
munin	Run munin plugins/checks from salt and format the output as data.
mysql	Module to provide MySQL compatibility to salt.
nagios	Run nagios plugins/checks from salt and get the return as data.
netbsd_sysctl	Module for viewing and modifying sysctl parameters
netbsdservice	The service module for NetBSD
netscaler	
network	Module for gathering and managing network information

Table 22.6 -- continued from previous page

neutron	Module for handling OpenStack Neutron calls
nfs3	Module for managing NFS version 3.
nftables	Support for nftables
nginx	Support for nginx
nova	Module for handling OpenStack Nova calls
npm	Manage and query NPM packages.
omapi	This module interacts with an ISC DHCP Server via OMAPI.
openbsd_sysctl	Module for viewing and modifying OpenBSD sysctl parameters
openbsdpkg	Package support for OpenBSD
openbsdrcctl	The rcctl service module for OpenBSD
openbsdservice	The service module for OpenBSD
openstack_config	Modify, retrieve, or delete values from OpenStack configuration files.
oracle	Oracle DataBase connection module
osxdesktop	Mac OS X implementations of various commands in the ``desktop'' interface
pacman	A module to wrap pacman calls, since Arch is the best
pagerduty	Module for Firing Events via PagerDuty
pam	Support for pam
parted	Module for managing partitions on POSIX-like systems.
pecl	Manage PHP pecl extensions.
pillar	Extract the pillar data for this minion
pip	Install Python packages with pip to either the system or a virtualenv
pkg_resource	Resources needed by pkg providers
pkgin	Package support for pkgin based systems, inspired from freebsdpkg module
pkgng	Support for pkgng, the new package manager for FreeBSD
pkgutil	Pkgutil support for Solaris
portage_config	Configure portage(5)
postfix	Support for Postfix
postgres	Module to provide Postgres compatibility to salt.
poudriere	Support for poudriere
powerpath	powerpath support.
ps	A salt interface to psutil, a system and process library.
publish	Publish a command from a minion to a target
puppet	Execute puppet routines
pw_group	Manage groups on FreeBSD
pw_user	Manage users with the useradd command
pyenv	Manage python installations with pyenv.
qemu_img	Qemu-img Command Wrapper
qemu_nbd	Qemu Command Wrapper
quota	Module for managing quotas on POSIX-like systems.
rabbitmq	Module to provide RabbitMQ compatibility to Salt.
raet_publish	Publish a command from a minion to a target
random_org	
rbenv	Manage ruby installations with rbenv.
rdp	Manage RDP Service on Windows servers
redismod	Module to provide redis functionality to Salt
reg	Manage the registry on Windows
rest_package	Service support for the REST example
rest_sample	Module for interfacing to the REST example
rest_service	Service support for the REST example
ret	Module to integrate with the returner system and retrieve data sent to a salt returner

Table 22.6 -- continued from previous page

rh_ip	The networking module for RHEL/Fedora based distros
rh_service	Service support for RHEL-based systems, including support for both upstart and sysvinit
riak	Riak Salt Module
rpm	Support for rpm
rsync	Wrapper for rsync
runit	runit service module
rvm	Manage ruby installations and gemsets with RVM, the Ruby Version Manager.
s3	Connection module for Amazon S3
saltcloudmod	Control a salt cloud system
saltutil	The Saltutil module is used to manage the state of the salt minion itself.
schedule	Module for managing the Salt schedule on a minion
scsi	SCSI administration module
sdb	Module for Manipulating Data via the Salt DB API
seed	Virtual machine image management tools
selinux	Execute calls on selinux
sensors	Read lm-sensors
serverdensity_device	Wrapper around Server Density API
service	The default service module, if not otherwise specified salt will fall back
shadow	Manage the shadow file
slack_notify	Module for sending messages to Slack
smartos_imgadm	Module for running imgadm command on SmartOS
smartos_vmadm	Module for managing VMs on SmartOS
smf	Service support for Solaris 10 and 11, should work with other systems that use SMF also.
smtp	Module for Sending Messages via SMTP
softwareupdate	Support for the softwareupdate command on MacOS.
solaris_group	Manage groups on Solaris
solaris_shadow	Manage the password database on Solaris systems
solaris_user	Manage users with the useradd command
solarisips	IPS pkg support for Solaris
solarispkg	Package support for Solaris
solr	Apache Solr Salt Module
splunk_search	Module for interop with the Splunk API
sqlite3	Support for SQLite3
ssh	Manage client ssh components
state	Control the state system on the minion
status	Module for returning various status data about a minion.
sudo	Allow for the calling of execution modules via sudo
supervisord	Provide the service module for system supervisord or supervisord in a
svn	Subversion SCM
swift	Module for handling OpenStack Swift calls
sysbench	The `sysbench' module is used to analyze the performance of the minions, right from the master! It
syslog_ng	Module for getting information about syslog-ng
sysmod	The sys module provides information about the available functions on the minion
sysrc	sysrc module for FreeBSD
system	Support for reboot, shutdown, etc
systemd	Provide the service module for systemd
test	Module for running arbitrary tests
test_virtual	Module for running arbitrary tests with a __virtual__ function
timezone	Module for managing timezone on POSIX-like systems.
tls	A salt module for SSL/TLS.

Table 22.6 -- continued from previous page

tomcat	Support for Tomcat
twilio_notify	Module for notifications via Twilio
upstart	Module for the management of upstart systems.
useradd	Manage users with the useradd command
uwsgi	uWSGI stats server http://uwsgi-docs.readthedocs.org/en/latest/StatsServer.html
varnish	Support for Varnish
vbox_guest	VirtualBox Guest Additions installer
virt	Work with virtual machines managed by libvirt
virtualenv_mod	Create virtualenv environments
win_autoruns	Module for listing programs that automatically run on startup
win_disk	Module for gathering disk information on Windows
win_dns_client	Module for configuring DNS Client on Windows systems
win_file	Manage information about files on the minion, set/read user, group
win_firewall	Module for configuring Windows Firewall
win_groupadd	Manage groups on Windows
win_ip	The networking module for Windows based systems
win_network	Module for gathering and managing network information
win_ntp	Management of NTP servers on Windows
win_path	Manage the Windows System PATH
win_pkg	A module to manage software on Windows
win_repo	Module to manage Windows software repo on a Standalone Minion
win_servermanager	Manage Windows features via the ServerManager powershell module
win_service	Windows Service module.
win_shadow	Manage the shadow file
win_status	Module for returning various status data about a minion.
win_system	Support for reboot, shutdown, etc
win_timezone	Module for managing timezone on Windows systems.
win_update	Module for running windows updates.
win_useradd	Manage Windows users with the net user command
xapi	This module (mostly) uses the XenAPI to manage Xen virtual machines.
xfs	Module for managing XFS file systems.
xmpp	Module for Sending Messages via XMPP (a.k.a.
yumpkg	Support for YUM
zcbuildout	Management of zc.buildout
zfs	Salt interface to ZFS commands
zk_concurrency	Concurrency controls in zookeeper
znc	znc - An advanced IRC bouncer
zpool	Module for running ZFS zpool command
zypper	Package support for openSUSE via the zypper package manager

22.16.2 salt.modules.aliases

Manage the information in the aliases file

salt.modules.aliases.**get_target**(*alias*)

Return the target associated with an alias

CLI Example:

```
salt '*' aliases.get_target alias
```

salt.modules.aliases.**has_target**(*alias*, *target*)

Return true if the alias/target is set

CLI Example:

```
salt '*' aliases.has_target alias target
```

salt.modules.aliases.**list_aliases**()
Return the aliases found in the aliases file in this format:

```
{'alias': 'target'}
```

CLI Example:

```
salt '*' aliases.list_aliases
```

salt.modules.aliases.**rm_alias**(*alias*)
Remove an entry from the aliases file

CLI Example:

```
salt '*' aliases.rm_alias alias
```

salt.modules.aliases.**set_target**(*alias*, *target*)
Set the entry in the aliases file for the given alias, this will overwrite any previous entry for the given alias or create a new one if it does not exist.

CLI Example:

```
salt '*' aliases.set_target alias target
```

22.16.3 salt.modules.alternatives

Support for Alternatives system

codeauthor Radek Rada <radek.rada@gmail.com>

salt.modules.alternatives.**auto**(*name*)
Trigger alternatives to set the path for <name> as specified by priority.

CLI Example:

```
salt '*' alternatives.auto name
```

salt.modules.alternatives.**check_installed**(*name*, *path*)
Check if the current highest-priority match for a given alternatives link is set to the desired path

CLI Example:

```
salt '*' alternatives.check_installed name path
```

salt.modules.alternatives.**display**(*name*)
Display alternatives settings for defined command name

CLI Example:

```
salt '*' alternatives.display editor
```

salt.modules.alternatives.**install**(*name*, *link*, *path*, *priority*)
Install symbolic links determining default commands

CLI Example:

```
salt '*' alternatives.install editor /usr/bin/editor /usr/bin/emacs23 50
```

salt.modules.alternatives.**remove**(*name*, *path*)
Remove symbolic links determining the default commands.

CLI Example:

```
salt '*' alternatives.remove name path
```

salt.modules.alternatives.**set_**(*name*, *path*)
Manually set the alternative <path> for <name>.

CLI Example:

```
salt '*' alternatives.set name path
```

salt.modules.alternatives.**show_current**(*name*)
Display the current highest-priority alternative for a given alternatives link

CLI Example:

```
salt '*' alternatives.show_current editor
```

22.16.4 salt.modules.apache

Support for Apache

Note: The functions in here are generic functions designed to work with all implementations of Apache. Debian-specific functions have been moved into deb_apache.py, but will still load under the `apache` namespace when a Debian-based system is detected.

---

salt.modules.apache.**config**(*name*, *config*, *edit=True*)
Create VirtualHost configuration files

name File for the virtual host

config VirtualHost configurations

Note: This function is not meant to be used from the command line. Config is meant to be an ordered dict of all of the apache configs.

CLI Example:

```
salt '*' apache.config /etc/httpd/conf.d/ports.conf config="[{'Listen': '22'}]"
```

salt.modules.apache.**directives**()
Return list of directives together with expected arguments and places where the directive is valid (`apachectl -L`)

CLI Example:

```
salt '*' apache.directives
```

salt.modules.apache.**fullversion**()
Return server version (`apachectl -V`)

CLI Example:

```
salt '*' apache.fullversion
```

salt.modules.apache.**modules**()
Return list of static and shared modules (`apachectl -M`)

CLI Example:

```
salt '*' apache.modules
```

salt.modules.apache.**server_status**(*profile='default'*)
Get Information from the Apache server-status handler

Note: The server-status handler is disabled by default. In order for this function to work it needs to be enabled. See http://httpd.apache.org/docs/2.2/mod/mod_status.html

---

The following configuration needs to exists in pillar/grains. Each entry nested in `apache.server-status` is a profile of a vhost/server. This would give support for multiple apache servers/vhosts.

```
apache.server-status:
  default:
    url: http://localhost/server-status
    user: someuser
    pass: password
    realm: 'authentication realm for digest passwords'
    timeout: 5
```

CLI Examples:

```
salt '*' apache.server_status
salt '*' apache.server_status other-profile
```

salt.modules.apache.**servermods**()
Return list of modules compiled into the server (`apachectl -l`)

CLI Example:

```
salt '*' apache.servermods
```

salt.modules.apache.**signal**(*signal=None*)
Signals httpd to start, restart, or stop.

CLI Example:

```
salt '*' apache.signal restart
```

salt.modules.apache.**useradd**(*pwfile*, *user*, *password*, *opts=''*)
Add HTTP user using the `htpasswd` command. If the `htpasswd` file does not exist, it will be created. Valid options that can be passed are:

```
n  Don't update file; display results on stdout.
m  Force MD5 encryption of the password (default).
d  Force CRYPT encryption of the password.
p  Do not encrypt the password (plaintext).
s  Force SHA encryption of the password.
```

CLI Examples:

```
salt '*' apache.useradd /etc/httpd/htpasswd larry badpassword
salt '*' apache.useradd /etc/httpd/htpasswd larry badpass opts=ns
```

salt.modules.apache.**userdel**(*pwfile*, *user*)
Delete HTTP user from the specified `htpasswd` file.

CLI Example:

```
salt '*' apache.userdel /etc/httpd/htpasswd larry
```

salt.modules.apache.**version**()
Return server version (`apachectl -v`)

CLI Example:

```
salt '*' apache.version
```

salt.modules.apache.**vhosts**()
Show the settings as parsed from the config file (currently only shows the virtualhost settings) (`apachectl -S`). Because each additional virtual host adds to the execution time, this command may require a long timeout be specified by using `-t 10`.

CLI Example:

```
salt -t 10 '*' apache.vhosts
```

22.16.5 salt.modules.aptpkg

Support for APT (Advanced Packaging Tool)

Note: For virtual package support, either the `python-apt` or `dctrl-tools` package must be installed.

For repository management, the `python-apt` package must be installed.

---

salt.modules.aptpkg.**autoremove**(*list_only=False*)
New in version 2015.5.0.

Remove packages not required by another package using `apt-get autoremove`.

list_only [False] Only retrieve the list of packages to be auto-removed, do not actually perform the auto-removal.

CLI Example:

```
salt '*' pkg.autoremove
salt '*' pkg.autoremove list_only=True
```

salt.modules.aptpkg.**del_repo**(*repo*, ***kwargs*)
Delete a repo from the sources.list / sources.list.d

If the .list file is in the sources.list.d directory and the file that the repo exists in does not contain any other repo configuration, the file itself will be deleted.

The repo passed in must be a fully formed repository definition string.

CLI Examples:

```
salt '*' pkg.del_repo "myrepo definition"
```

salt.modules.aptpkg.**expand_repo_def**(*repokwargs*)
Take a repository definition and expand it to the full pkg repository dict that can be used for comparison. This is a helper function to make the Debian/Ubuntu apt sources sane for comparison in the pkgrepo states.

There is no use to calling this function via the CLI.

salt.modules.aptpkg.**file_dict**(**packages*)

List the files that belong to a package, grouped by package. Not specifying any packages will return a list of _every_ file on the system's package database (not generally recommended).

CLI Examples:

```
salt '*' pkg.file_list httpd
salt '*' pkg.file_list httpd postfix
salt '*' pkg.file_list
```

salt.modules.aptpkg.**file_list**(**packages*)

List the files that belong to a package. Not specifying any packages will return a list of _every_ file on the system's package database (not generally recommended).

CLI Examples:

```
salt '*' pkg.file_list httpd
salt '*' pkg.file_list httpd postfix
salt '*' pkg.file_list
```

salt.modules.aptpkg.**get_repo**(*repo*, ***kwargs*)

Display a repo from the sources.list / sources.list.d

The repo passed in needs to be a complete repo entry.

CLI Examples:

```
salt '*' pkg.get_repo "myrepo definition"
```

salt.modules.aptpkg.**get_selections**(*pattern=None*, *state=None*)

View package state from the dpkg database.

Returns a dict of dicts containing the state, and package names:

```
{'<host>':
    {'<state>': ['pkg1',
                 ...
                ]
    },
    ...
}
```

CLI Example:

```
salt '*' pkg.get_selections
salt '*' pkg.get_selections 'python-*'
salt '*' pkg.get_selections state=hold
salt '*' pkg.get_selections 'openssh*' state=hold
```

salt.modules.aptpkg.**hold**(*name=None*, *pkgs=None*, *sources=None*, ***kwargs*)

New in version 2014.7.0.

Set package in `hold' state, meaning it will not be upgraded.

name The name of the package, e.g., `tmux'

CLI Example:

```
salt '*' pkg.hold <package name>
```

pkgs A list of packages to hold. Must be passed as a python list.

CLI Example:

```
salt '*' pkg.hold pkgs='["foo", "bar"]'
```

salt.modules.aptpkg.**install**(*name=None, refresh=False, fromrepo=None, skip_verify=False, debconf=None, pkgs=None, sources=None, **kwargs*)

Install the passed package, add refresh=True to update the dpkg database.

name The name of the package to be installed. Note that this parameter is ignored if either ``pkgs'' or ``sources'' is passed. Additionally, please note that this option can only be used to install packages from a software repository. To install a package file manually, use the ``sources'' option.

32-bit packages can be installed on 64-bit systems by appending the architecture designation (`:i386`, etc.) to the end of the package name.

CLI Example:

```
salt '*' pkg.install <package name>
```

refresh Whether or not to refresh the package database before installing.

fromrepo Specify a package repository to install from (e.g., `apt-get -t unstable install somepackage`)

skip_verify Skip the GPG verification check (e.g., `--allow-unauthenticated`, or `--force-bad-verify` for install from package file).

debconf Provide the path to a debconf answers file, processed before installation.

version Install a specific version of the package, e.g. 1.2.3~0ubuntu0. Ignored if ``pkgs'' or ``sources'' is passed.

Multiple Package Installation Options:

pkgs A list of packages to install from a software repository. Must be passed as a python list.

CLI Example:

```
salt '*' pkg.install pkgs='["foo", "bar"]'
salt '*' pkg.install pkgs='["foo", {"bar": "1.2.3-0ubuntu0"}]'
```

sources A list of DEB packages to install. Must be passed as a list of dicts, with the keys being package names, and the values being the source URI or local path to the package. Dependencies are automatically resolved and marked as auto-installed.

32-bit packages can be installed on 64-bit systems by appending the architecture designation (`:i386`, etc.) to the end of the package name.

Changed in version 2014.7.0.

CLI Example:

```
salt '*' pkg.install sources='[{"foo": "salt://foo.deb"},{"bar": "salt://bar.deb"}]'
```

force_yes Passes `--force-yes` to the apt-get command. Don't use this unless you know what you're doing.

New in version 0.17.4.

install_recommends Whether to install the packages marked as recommended. Default is True.

New in version 2015.5.0.

only_upgrade Only upgrade the packages, if they are already installed. Default is False.

New in version 2015.5.0.

Returns a dict containing the new package names and versions:

```
{'<package>': {'old': '<old-version>',
               'new': '<new-version>'}}
```

salt.modules.aptpkg.**latest_version**(**names*, ***kwargs*)

Return the latest version of the named package available for upgrade or installation. If more than one package name is specified, a dict of name/version pairs is returned.

If the latest version of a given package is already installed, an empty string will be returned for that package.

A specific repo can be requested using the `fromrepo` keyword argument.

CLI Example:

```
salt '*' pkg.latest_version <package name>
salt '*' pkg.latest_version <package name> fromrepo=unstable
salt '*' pkg.latest_version <package1> <package2> <package3> ...
```

salt.modules.aptpkg.**list_pkgs**(*versions_as_list=False*, *removed=False*, *purge_desired=False*, ***kwargs*)

List the packages currently installed in a dict:

```
{'<package_name>': '<version>'}
```

removed If `True`, then only packages which have been removed (but not purged) will be returned.

purge_desired If `True`, then only packages which have been marked to be purged, but can't be purged due to their status as dependencies for other installed packages, will be returned. Note that these packages will appear in installed

Changed in version 2014.1.1: Packages in this state now correctly show up in the output of this function.

Note: External dependencies

Virtual package resolution requires the `dctrl-tools` package to be installed. Virtual packages will show a version of `1`.

CLI Example:

```
salt '*' pkg.list_pkgs
salt '*' pkg.list_pkgs versions_as_list=True
```

salt.modules.aptpkg.**list_repos**()

Lists all repos in the sources.list (and sources.lists.d) files

CLI Example:

```
salt '*' pkg.list_repos
salt '*' pkg.list_repos disabled=True
```

salt.modules.aptpkg.**list_upgrades**(*refresh=True*)

List all available package upgrades.

CLI Example:

```
salt '*' pkg.list_upgrades
```

salt.modules.aptpkg.**mod_repo**(*repo*, *saltenv='base'*, ***kwargs*)

Modify one or more values for a repo. If the repo does not exist, it will be created, so long as the definition is

well formed. For Ubuntu the ``ppa:<project>/repo'' format is acceptable. ``ppa:'' format can only be used to create a new repository.

The following options are available to modify a repo definition:

```
comps (a comma separated list of components for the repo, e.g. "main")
file (a file name to be used)
keyserver (keyserver to get gpg key from)
keyid (key id to load with the keyserver argument)
key_url (URL to a gpg key to add to the apt gpg keyring)
consolidate (if true, will attempt to de-dup and consolidate sources)

* Note: Due to the way keys are stored for apt, there is a known issue
        where the key wont be updated unless another change is made
        at the same time.  Keys should be properly added on initial
        configuration.
```

CLI Examples:

```
salt '*' pkg.mod_repo 'myrepo definition' uri=http://new/uri
salt '*' pkg.mod_repo 'myrepo definition' comps=main,universe
```

`salt.modules.aptpkg.`**`owner`**`(`*`*paths`*`)`

New in version 2014.7.0.

Return the name of the package that owns the file. Multiple file paths can be passed. Like `pkg.version`, if a single path is passed, a string will be returned, and if multiple paths are passed, a dictionary of file/package name pairs will be returned.

If the file is not owned by a package, or is not present on the minion, then an empty string will be returned for that path.

CLI Example:

salt `*' pkg.owner /usr/bin/apachectl salt `*' pkg.owner /usr/bin/apachectl /usr/bin/basename

`salt.modules.aptpkg.`**`purge`**`(`*`name=None, pkgs=None, **kwargs`*`)`

Remove packages via `apt-get purge` along with all configuration files.

name The name of the package to be deleted.

Multiple Package Options:

pkgs A list of packages to delete. Must be passed as a python list. The `name` parameter will be ignored if this option is passed.

New in version 0.16.0.

Returns a dict containing the changes.

CLI Example:

```
salt '*' pkg.purge <package name>
salt '*' pkg.purge <package1>,<package2>,<package3>
salt '*' pkg.purge pkgs='["foo", "bar"]'
```

`salt.modules.aptpkg.`**`refresh_db`**`()`

Updates the APT database to latest packages based upon repositories

Returns a dict, with the keys being package databases and the values being the result of the update attempt. Values can be one of the following:

- `True`: Database updated successfully

- `False`: Problem updating database
- `None`: Database already up-to-date

CLI Example:

```
salt '*' pkg.refresh_db
```

`salt.modules.aptpkg.`**`remove`**(*name=None*, *pkgs=None*, ***kwargs*)

Remove packages using `apt-get remove`.

name The name of the package to be deleted.

Multiple Package Options:

pkgs A list of packages to delete. Must be passed as a python list. The `name` parameter will be ignored if this option is passed.

New in version 0.16.0.

Returns a dict containing the changes.

CLI Example:

```
salt '*' pkg.remove <package name>
salt '*' pkg.remove <package1>,<package2>,<package3>
salt '*' pkg.remove pkgs='["foo", "bar"]'
```

`salt.modules.aptpkg.`**`set_selections`**(*path=None*, *selection=None*, *clear=False*, *saltenv='base'*)

Change package state in the dpkg database.

The state can be any one of, documented in `dpkg(1)`:

- install
- hold
- deinstall
- purge

This command is commonly used to mark specific packages to be held from being upgraded, that is, to be kept at a certain version. When a state is changed to anything but being held, then it is typically followed by `apt-get -u dselect-upgrade`.

Note: Be careful with the `clear` argument, since it will start with setting all packages to deinstall state.

Returns a dict of dicts containing the package names, and the new and old versions:

```
{'<host>':
    {'<package>': {'new': '<new-state>',
                   'old': '<old-state>'}
    },
    ...
}
```

CLI Example:

```
salt '*' pkg.set_selections selection='{"install": ["netcat"]}'
salt '*' pkg.set_selections selection='{"hold": ["openssh-server", "openssh-client"]}'
salt '*' pkg.set_selections salt://path/to/file
salt '*' pkg.set_selections salt://path/to/file clear=True
```

salt.modules.aptpkg.**unhold**(*name=None*, *pkgs=None*, *sources=None*, ***kwargs*)
New in version 2014.7.0.

Set package current in `hold' state to install state, meaning it will be upgraded.

name The name of the package, e.g., `tmux'

CLI Example:

```
salt '*' pkg.unhold <package name>
```

pkgs A list of packages to hold. Must be passed as a python list.

CLI Example:

```
salt '*' pkg.unhold pkgs='["foo", "bar"]'
```

salt.modules.aptpkg.**upgrade**(*refresh=True*, *dist_upgrade=True*)
Upgrades all packages via `apt-get dist-upgrade`

Returns a dict containing the changes.

{`<package>': {`old': `<old-version>', `new': `<new-version>'}}

dist_upgrade Whether to perform the upgrade using dist-upgrade vs upgrade. Default is to use dist-upgrade.

New in version 2014.7.0.

CLI Example:

```
salt '*' pkg.upgrade
```

salt.modules.aptpkg.**upgrade_available**(*name*)
Check whether or not an upgrade is available for a given package

CLI Example:

```
salt '*' pkg.upgrade_available <package name>
```

salt.modules.aptpkg.**version**(**names*, ***kwargs*)
Returns a string representing the package version or an empty string if not installed. If more than one package name is specified, a dict of name/version pairs is returned.

CLI Example:

```
salt '*' pkg.version <package name>
salt '*' pkg.version <package1> <package2> <package3> ...
```

salt.modules.aptpkg.**version_cmp**(*pkg1*, *pkg2*)
Do a cmp-style comparison on two packages. Return -1 if pkg1 < pkg2, 0 if pkg1 == pkg2, and 1 if pkg1 > pkg2. Return None if there was a problem making the comparison.

CLI Example:

```
salt '*' pkg.version_cmp '0.2.4-0ubuntu1' '0.2.4.1-0ubuntu1'
```

22.16.6 salt.modules.archive

A module to wrap (non-Windows) archive calls

New in version 2014.1.0.

salt.modules.archive.**cmd_unzip**(*zip_file*, *dest*, *excludes=None*, *template=None*, *options=None*, *runas=None*)

New in version 2015.5.0: In versions 2014.7.x and earlier, this function was known as `archive.unzip`.

Uses the `unzip` command to unpack zip files. This command is part of the Info-ZIP suite of tools, and is typically packaged as simply `unzip`.

zip_file Path of zip file to be unpacked

dest The destination directory into which the file should be unpacked

excludes [None] Comma-separated list of files not to unpack. Can also be passed in a Python list.

template [None] Can be set to `jinja' or another supported template engine to render the command arguments before execution:

```
salt '*' archive.cmd_unzip template=jinja /tmp/zipfile.zip /tmp/{{grains.id}}/ excludes=file
```

options [None] Additional command-line options to pass to the `unzip` binary.

runas [None] Unpack the zip file as the specified user. Defaults to the user under which the minion is running.

New in version 2015.5.0.

options [None] Additional command-line options to pass to the `unzip` binary.

CLI Example:

```
salt '*' archive.cmd_unzip /tmp/zipfile.zip /home/strongbad/ excludes=file_1,file_2
```

salt.modules.archive.**cmd_zip**(*zip_file*, *sources*, *template=None*, *cwd=None*, *runas=None*)

New in version 2015.5.0: In versions 2014.7.x and earlier, this function was known as `archive.zip`.

Uses the `zip` command to create zip files. This command is part of the Info-ZIP suite of tools, and is typically packaged as simply `zip`.

zip_file Path of zip file to be created

sources Comma-separated list of sources to include in the zip file. Sources can also be passed in a Python list.

template [None] Can be set to `jinja' or another supported template engine to render the command arguments before execution:

```
salt '*' archive.cmd_zip template=jinja /tmp/zipfile.zip /tmp/sourcefile1,/tmp/{{grains.id}}
```

cwd [None] Use this argument along with relative paths in `sources` to create zip files which do not contain the leading directories. If not specified, the zip file will be created as if the cwd was `/`, and creating a zip file of `/foo/bar/baz.txt` will contain the parent directories `foo` and `bar`. To create a zip file containing just `baz.txt`, the following command would be used:

```
salt '*' archive.cmd_zip /tmp/baz.zip baz.txt cwd=/foo/bar
```

New in version 2014.7.1.

runas [None] Create the zip file as the specified user. Defaults to the user under which the minion is running.

New in version 2015.5.0.

CLI Example:

```
salt '*' archive.cmd_zip /tmp/zipfile.zip /tmp/sourcefile1,/tmp/sourcefile2
```

salt.modules.archive.**gunzip**(*gzipfile*, *template=None*, *runas=None*)

Uses the gunzip command to unpack gzip files

template [None] Can be set to `jinja' or another supported template engine to render the command arguments before execution:

```
salt '*' archive.gunzip template=jinja /tmp/{{grains.id}}.txt.gz
```

CLI Example:

```
# Create /tmp/sourcefile.txt
salt '*' archive.gunzip /tmp/sourcefile.txt.gz
```

salt.modules.archive.**gzip**(*sourcefile*, *template=None*, *runas=None*)

Uses the gzip command to create gzip files

template [None] Can be set to `jinja' or another supported template engine to render the command arguments before execution:

```
salt '*' archive.gzip template=jinja /tmp/{{grains.id}}.txt
```

CLI Example:

```
# Create /tmp/sourcefile.txt.gz
salt '*' archive.gzip /tmp/sourcefile.txt
```

salt.modules.archive.**rar**(*rarfile*, *sources*, *template=None*, *cwd=None*, *runas=None*)

Uses rar for Linux to create rar files

rarfile Path of rar file to be created

sources Comma-separated list of sources to include in the rar file. Sources can also be passed in a Python list.

cwd [None] Run the rar command from the specified directory. Use this argument along with relative file paths to create rar files which do not contain the leading directories. If not specified, this will default to the home directory of the user under which the salt minion process is running.

New in version 2014.7.1.

template [None] Can be set to `jinja' or another supported template engine to render the command arguments before execution:

```
salt '*' archive.rar template=jinja /tmp/rarfile.rar '/tmp/sourcefile1,/tmp/{{grains.id}}.tx
```

CLI Example:

```
salt '*' archive.rar /tmp/rarfile.rar /tmp/sourcefile1,/tmp/sourcefile2
```

salt.modules.archive.**tar**(*options*, *tarfile*, *sources=None*, *dest=None*, *cwd=None*, *template=None*, *runas=None*)

Note: This function has changed for version 0.17.0. In prior versions, the `cwd` and `template` arguments must be specified, with the source directories/files coming as a space-separated list at the end of the command. Beginning with 0.17.0, `sources` must be a comma-separated list, and the `cwd` and `template` arguments are optional.

Uses the tar command to pack, unpack, etc. tar files

options Options to pass to the tar command

tarfile The filename of the tar archive to pack/unpack

sources Comma delimited list of files to **pack** into the tarfile. Can also be passed as a Python list.

dest The destination directory into which to **unpack** the tarfile

cwd [None] The directory in which the tar command should be executed. If not specified, will default to the home directory of the user under which the salt minion process is running.

template [None] Can be set to `jinja' or another supported template engine to render the command arguments before execution:

```
salt '*' archive.tar cjvf /tmp/salt.tar.bz2 {{grains.saltpath}} template=jinja
```

CLI Examples:

```
# Create a tarfile
salt '*' archive.tar cjvf /tmp/tarfile.tar.bz2 /tmp/file_1,/tmp/file_2
# Unpack a tarfile
salt '*' archive.tar xf foo.tar dest=/target/directory
```

salt.modules.archive.**unrar**(*rarfile*, *dest*, *excludes=None*, *template=None*, *runas=None*)

Uses rar for Linux to unpack rar files

rarfile Name of rar file to be unpacked

dest The destination directory into which to **unpack** the rar file

template [None] Can be set to `jinja' or another supported template engine to render the command arguments before execution:

```
salt '*' archive.unrar template=jinja /tmp/rarfile.rar /tmp/{{grains.id}}/ excludes=file_1,f
```

CLI Example:

```
salt '*' archive.unrar /tmp/rarfile.rar /home/strongbad/ excludes=file_1,file_2
```

salt.modules.archive.**unzip**(*zip_file*, *dest*, *excludes=None*, *template=None*, *runas=None*)

Uses the `zipfile` Python module to unpack zip files

Changed in version 2015.5.0: This function was rewritten to use Python's native zip file support. The old functionality has been preserved in the new function `archive.cmd_unzip`. For versions 2014.7.x and earlier, see the `archive.cmd_zip` documentation.

zip_file Path of zip file to be unpacked

dest The destination directory into which the file should be unpacked

excludes [None] Comma-separated list of files not to unpack. Can also be passed in a Python list.

template [None] Can be set to `jinja' or another supported template engine to render the command arguments before execution:

```
salt '*' archive.unzip template=jinja /tmp/zipfile.zip /tmp/{{grains.id}}/ excludes=file_1,f
```

runas [None] Unpack the zip file as the specified user. Defaults to the user under which the minion is running.

CLI Example:

```
salt '*' archive.unzip /tmp/zipfile.zip /home/strongbad/ excludes=file_1,file_2
```

salt.modules.archive.**zip_**(*zip_file*, *sources*, *template=None*, *cwd=None*, *runas=None*)

Uses the `zipfile` Python module to create zip files

Changed in version 2015.5.0: This function was rewritten to use Python's native zip file support. The old functionality has been preserved in the new function `archive.cmd_zip`. For versions 2014.7.x and earlier, see the `archive.cmd_zip` documentation.

zip_file Path of zip file to be created

sources Comma-separated list of sources to include in the zip file. Sources can also be passed in a Python list.

template [None] Can be set to `jinja' or another supported template engine to render the command arguments before execution:

```
salt '*' archive.zip template=jinja /tmp/zipfile.zip /tmp/sourcefile1,/tmp/{{grains.id}}.txt
```

cwd [None] Use this argument along with relative paths in `sources` to create zip files which do not contain the leading directories. If not specified, the zip file will be created as if the cwd was `/`, and creating a zip file of `/foo/bar/baz.txt` will contain the parent directories `foo` and `bar`. To create a zip file containing just `baz.txt`, the following command would be used:

```
salt '*' archive.zip /tmp/baz.zip baz.txt cwd=/foo/bar
```

runas [None] Create the zip file as the specified user. Defaults to the user under which the minion is running.

CLI Example:

```
salt '*' archive.zip /tmp/zipfile.zip /tmp/sourcefile1,/tmp/sourcefile2
```

22.16.7 salt.modules.artifactory

Module for fetching artifacts from Artifactory

exception `salt.modules.artifactory.`**`ArtifactoryError`**(*value*)

`salt.modules.artifactory.`**`get_latest_snapshot`**(*artifactory_url*, *repository*, *group_id*, *artifact_id*, *packaging*, *target_dir='/tmp'*, *target_file=None*)

Gets latest snapshot of the given artifact

artifactory_url URL of artifactory instance

repository Snapshot repository in artifactory to retrieve artifact from, for example: libs-snapshots

group_id Group Id of the artifact

artifact_id Artifact Id of the artifact

packaging Packaging type (jar,war,ear,etc)

target_dir Target directory to download artifact to (default: /tmp)

target_file Target file to download artifact to (by default it is target_dir/artifact_id-snapshot_version.packaging)

`salt.modules.artifactory.`**`get_release`**(*artifactory_url*, *repository*, *group_id*, *artifact_id*, *packaging*, *version*, *target_dir='/tmp'*, *target_file=None*)

Gets the specified release of the artifact

artifactory_url URL of artifactory instance

repository Release repository in artifactory to retrieve artifact from, for example: libs-releases

group_id Group Id of the artifact

artifact_id Artifact Id of the artifact

packaging Packaging type (jar,war,ear,etc)

version Version of the artifact

target_dir Target directory to download artifact to (default: /tmp)

target_file Target file to download artifact to (by default it is target_dir/artifact_id-version.packaging)

salt.modules.artifactory.**get_snapshot**(*artifactory_url*, *repository*, *group_id*, *artifact_id*, *packaging*, *version*, *snapshot_version=None*, *target_dir='/tmp'*, *target_file=None*)

Gets snapshot of the desired version of the artifact

artifactory_url URL of artifactory instance

repository Snapshot repository in artifactory to retrieve artifact from, for example: libs-snapshots

group_id Group Id of the artifact

artifact_id Artifact Id of the artifact

packaging Packaging type (jar,war,ear,etc)

version Version of the artifact

target_dir Target directory to download artifact to (default: /tmp)

target_file Target file to download artifact to (by default it is target_dir/artifact_id-snapshot_version.packaging)

22.16.8 salt.modules.at

Wrapper module for at(1)

Also, a `tag' feature has been added to more easily tag jobs.

salt.modules.at.**at**(**args*, ***kwargs*)

Add a job to the queue.

The `timespec' follows the format documented in the at(1) manpage.

CLI Example:

```
salt '*' at.at <timespec> <cmd> [tag=<tag>] [runas=<user>]
salt '*' at.at 12:05am '/sbin/reboot' tag=reboot
salt '*' at.at '3:05am +3 days' 'bin/myscript' tag=nightly runas=jim
```

salt.modules.at.**atc**(*jobid*)

Print the at(1) script that will run for the passed job id. This is mostly for debugging so the output will just be text.

CLI Example:

```
salt '*' at.atc <jobid>
```

salt.modules.at.**atq**(*tag=None*)

List all queued and running jobs or only those with an optional `tag'.

CLI Example:

```
salt '*' at.atq
salt '*' at.atq [tag]
salt '*' at.atq [job number]
```

salt.modules.at.**atrm**(**args*)

Remove jobs from the queue.

CLI Example:

```
    salt '*' at.atrm <jobid> <jobid> .. <jobid>
    salt '*' at.atrm all
    salt '*' at.atrm all [tag]
```

salt.modules.at.**jobcheck**(***kwargs*)

Check the job from queue. The kwargs dict include `hour minute day month year tag runas' Other parameters will be ignored.

CLI Example:

```
    salt '*' at.jobcheck runas=jam day=13
    salt '*' at.jobcheck day=13 month=12 year=13 tag=rose
```

22.16.9 salt.modules.augeas_cfg

Manages configuration files via augeas

This module requires the `augeas` Python module.

Warning: Minimal installations of Debian and Ubuntu have been seen to have packaging bugs with python-augeas, causing the augeas module to fail to import. If the minion has the augeas module installed, but the functions in this execution module fail to run due to being unavailable, first restart the salt-minion service. If the problem persists past that, the following command can be run from the master to determine what is causing the import to fail:

```
salt minion-id cmd.run 'python -c "from augeas import Augeas"'
```

For affected Debian/Ubuntu hosts, installing `libpython2.7` has been known to resolve the issue.

salt.modules.augeas_cfg.**execute**(*context=None*, *lens=None*, *commands=()*)

Execute Augeas commands

New in version 2014.7.0.

CLI Example:

```
    salt '*' augeas.execute /files/etc/redis/redis.conf commands='["set bind 0.0.0.0", "set maxmemor
```

salt.modules.augeas_cfg.**get**(*path*, *value=''*)

Get a value for a specific augeas path

CLI Example:

```
    salt '*' augeas.get /files/etc/hosts/1/ ipaddr
```

salt.modules.augeas_cfg.**ls**(*path*)

List the direct children of a node

CLI Example:

```
    salt '*' augeas.ls /files/etc/passwd
```

salt.modules.augeas_cfg.**match**(*path*, *value=''*)

Get matches for path expression

CLI Example:

```
salt '*' augeas.match /files/etc/services/service-name ssh
```

salt.modules.augeas_cfg.**remove**(*path*)

Get matches for path expression

CLI Example:

```
salt '*' augeas.remove /files/etc/sysctl.conf/net.ipv4.conf.all.log_martians
```

salt.modules.augeas_cfg.**setvalue**(**args*)

Set a value for a specific augeas path

CLI Example:

```
salt '*' augeas.setvalue /files/etc/hosts/1/canonical localhost
```

This will set the first entry in /etc/hosts to localhost

CLI Example:

```
salt '*' augeas.setvalue /files/etc/hosts/01/ipaddr 192.168.1.1 \
                         /files/etc/hosts/01/canonical test
```

Adds a new host to /etc/hosts the ip address 192.168.1.1 and hostname test

CLI Example:

```
salt '*' augeas.setvalue prefix=/files/etc/sudoers/ \
         "spec[user = '%wheel']/user" "%wheel" \
         "spec[user = '%wheel']/host_group/host" 'ALL' \
         "spec[user = '%wheel']/host_group/command[1]" 'ALL' \
         "spec[user = '%wheel']/host_group/command[1]/tag" 'PASSWD' \
         "spec[user = '%wheel']/host_group/command[2]" '/usr/bin/apt-get' \
         "spec[user = '%wheel']/host_group/command[2]/tag" NOPASSWD
```

Ensures that the following line is present in /etc/sudoers:

```
%wheel ALL = PASSWD : ALL , NOPASSWD : /usr/bin/apt-get , /usr/bin/aptitude
```

salt.modules.augeas_cfg.**tree**(*path*)

Returns recursively the complete tree of a node

CLI Example:

```
salt '*' augeas.tree /files/etc/
```

22.16.10 salt.modules.aws_sqs

Support for the Amazon Simple Queue Service.

salt.modules.aws_sqs.**create_queue**(*name*, *region*, *opts=None*, *user=None*)

Creates a queue with the correct name.

name Name of the SQS queue to create

region Region to create the SQS queue in

opts [None] Any additional options to add to the command line

user [None] Run hg as a user other than what the minion runs as

salt.modules.aws_sqs.**delete_message**(*queue*, *region*, *receipthandle*, *opts=None*, *user=None*)
Delete one or more messages from a queue in a region

queue The name of the queue to delete messages from

region Region where SQS queues exists

receipthandle The ReceiptHandle of the message to delete. The ReceiptHandle is obtained in the return from receive_message

opts [None] Any additional options to add to the command line

user [None] Run as a user other than what the minion runs as

CLI Example:

```
salt '*' aws_sqs.delete_message <sqs queue> <region> receipthandle='<sqs ReceiptHandle>'
```

New in version 2014.7.0.

salt.modules.aws_sqs.**delete_queue**(*name*, *region*, *opts=None*, *user=None*)
Deletes a queue in the region.

name Name of the SQS queue to deletes

region Name of the region to delete the queue from

opts [None] Any additional options to add to the command line

user [None] Run hg as a user other than what the minion runs as

salt.modules.aws_sqs.**list_queues**(*region*, *opts=None*, *user=None*)
List the queues in the selected region.

region Region to list SQS queues for

opts [None] Any additional options to add to the command line

user [None] Run hg as a user other than what the minion runs as

salt.modules.aws_sqs.**queue_exists**(*name*, *region*, *opts=None*, *user=None*)
Returns True or False on whether the queue exists in the region

name Name of the SQS queue to search for

region Name of the region to search for the queue in

opts [None] Any additional options to add to the command line

user [None] Run hg as a user other than what the minion runs as

salt.modules.aws_sqs.**receive_message**(*queue*, *region*, *num=1*, *opts=None*, *user=None*)
Receive one or more messages from a queue in a region

queue The name of the queue to receive messages from

region Region where SQS queues exists

num [1] The max number of messages to receive

opts [None] Any additional options to add to the command line

user [None] Run as a user other than what the minion runs as

CLI Example:

```
salt '*' aws_sqs.receive_message <sqs queue> <region>
salt '*' aws_sqs.receive_message <sqs queue> <region> num=10
```

New in version 2014.7.0.

22.16.11 salt.modules.blockdev

Module for managing block devices

New in version 2014.7.0.

salt.modules.blockdev.**dump**(*device*, *args=None*)
Return all contents of dumpe2fs for a specified device

CLI Example: .. code-block:: bash

salt `*' extfs.dump /dev/sda1

salt.modules.blockdev.**resize2fs**(*device*)
Resizes the filesystem.

CLI Example: .. code-block:: bash

salt `*' blockdev.resize2fs /dev/sda1

salt.modules.blockdev.**tune**(*device*, ***kwargs*)
Set attributes for the specified device

CLI Example:

```
salt '*' blockdev.tune /dev/sda1 read-ahead=1024 read-write=True
```

Valid options are: `read-ahead`, `filesystem-read-ahead`, `read-only`, `read-write`.

See the `blockdev(8)` manpage for a more complete description of these options.

salt.modules.blockdev.**wipe**(*device*)
Remove the filesystem information

CLI Example:

```
salt '*' blockdev.wipe /dev/sda1
```

22.16.12 salt.modules.bluez

Support for Bluetooth (using BlueZ in Linux).

The following packages are required packages for this module:

bluez >= 5.7 bluez-libs >= 5.7 bluez-utils >= 5.7 pybluez >= 0.18

salt.modules.bluez.**address_**()
Get the many addresses of the Bluetooth adapter

CLI Example:

```
salt '*' bluetooth.address
```

salt.modules.bluez.**block**(*bdaddr*)
Block a specific bluetooth device by BD Address

CLI Example:

```
salt '*' bluetooth.block DE:AD:BE:EF:CA:FE
```

salt.modules.bluez.**discoverable**(*dev*)
Enable this bluetooth device to be discoverable.

CLI Example:

```
salt '*' bluetooth.discoverable hci0
```

salt.modules.bluez.**noscan**(*dev*)
Turn off scanning modes on this device.

CLI Example:

```
salt '*' bluetooth.noscan hci0
```

salt.modules.bluez.**pair**(*address*, *key*)
Pair the bluetooth adapter with a device

CLI Example:

```
salt '*' bluetooth.pair DE:AD:BE:EF:CA:FE 1234
```

Where DE:AD:BE:EF:CA:FE is the address of the device to pair with, and 1234 is the passphrase.

TODO: This function is currently broken, as the bluez-simple-agent program no longer ships with BlueZ >= 5.0. It needs to be refactored.

salt.modules.bluez.**power**(*dev*, *mode*)
Power a bluetooth device on or off

CLI Examples:

```
salt '*' bluetooth.power hci0 on
salt '*' bluetooth.power hci0 off
```

salt.modules.bluez.**scan**()
Scan for bluetooth devices in the area

CLI Example:

```
salt '*' bluetooth.scan
```

salt.modules.bluez.**start**()
Start the bluetooth service.

CLI Example:

```
salt '*' bluetooth.start
```

salt.modules.bluez.**stop**()
Stop the bluetooth service.

CLI Example:

```
salt '*' bluetooth.stop
```

salt.modules.bluez.**unblock**(*bdaddr*)
Unblock a specific bluetooth device by BD Address

CLI Example:

```
salt '*' bluetooth.unblock DE:AD:BE:EF:CA:FE
```

salt.modules.bluez.**unpair**(*address*)
Unpair the bluetooth adapter from a device

CLI Example:

```
salt '*' bluetooth.unpair DE:AD:BE:EF:CA:FE
```

Where DE:AD:BE:EF:CA:FE is the address of the device to unpair.

TODO: This function is currently broken, as the bluez-simple-agent program no longer ships with BlueZ >= 5.0. It needs to be refactored.

salt.modules.bluez.**version**()
Return Bluez version from bluetoothd -v

CLI Example:

```
salt '*' bluetoothd.version
```

22.16.13 salt.modules.boto_asg

Connection module for Amazon Autoscale Groups

New in version 2014.7.0.

configuration This module accepts explicit autoscale credentials but can also utilize IAM roles assigned to the instance trough Instance Profiles. Dynamic credentials are then automatically obtained from AWS API and no further configuration is necessary. More Information available at:

```
http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/iam-roles-for-amazon-ec2.html
```

If IAM roles are not used you need to specify them either in a pillar or in the minion's config file:

```
asg.keyid: GKTADJGHEIQSXMKKRBJ08H
asg.key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs
```

A region may also be specified in the configuration:

```
asg.region: us-east-1
```

If a region is not specified, the default is us-east-1.

It's also possible to specify key, keyid and region via a profile, either as a passed in dict, or as a string to pull from pillars or minion config:

myprofile: keyid: GKTADJGHEIQSXMKKRBJ08H key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs region: us-east-1

depends boto

salt.modules.boto_asg.**create**(*name*, *launch_config_name*, *availability_zones*, *min_size*, *max_size*, *desired_capacity=None*, *load_balancers=None*, *default_cooldown=None*, *health_check_type=None*, *health_check_period=None*, *placement_group=None*, *vpc_zone_identifier=None*, *tags=None*, *termination_policies=None*, *suspended_processes=None*, *scaling_policies=None*, *region=None*, *key=None*, *keyid=None*, *profile=None*)
Create an autoscale group.

CLI example:

```
salt myminion boto_asg.create myasg mylc '["us-east-1a", "us-east-1e"]' 1 10 load_balancers='["m
```

salt.modules.boto_asg.**create_launch_configuration**(*name*, *image_id*, *key_name=None*, *security_groups=None*, *user_data=None*, *instance_type='m1.small'*, *kernel_id=None*, *ramdisk_id=None*, *block_device_mappings=None*, *instance_monitoring=False*, *spot_price=None*, *instance_profile_name=None*, *ebs_optimized=False*, *associate_public_ip_address=None*, *volume_type=None*, *delete_on_termination=True*, *iops=None*, *use_block_device_types=False*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Create a launch configuration.

CLI example:

```
salt myminion boto_asg.create_launch_configuration mylc image_id=ami-0b9c9f62 key_name='mykey' s
```

salt.modules.boto_asg.**delete**(*name*, *force=False*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Delete an autoscale group.

CLI example:

```
salt myminion boto_asg.delete myasg region=us-east-1
```

salt.modules.boto_asg.**delete_launch_configuration**(*name*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Delete a launch configuration.

CLI example:

```
salt myminion boto_asg.delete_launch_configuration mylc
```

salt.modules.boto_asg.**exists**(*name*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Check to see if an autoscale group exists.

CLI example:

```
salt myminion boto_asg.exists myasg region=us-east-1
```

salt.modules.boto_asg.**get_cloud_init_mime**(*cloud_init*)

Get a mime multipart encoded string from a cloud-init dict. Currently supports scripts and cloud-config.

CLI Example:

```
salt myminion boto.get_cloud_init_mime <cloud init>
```

salt.modules.boto_asg.**get_config**(*name*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Get the configuration for an autoscale group.

CLI example:

```
salt myminion boto_asg.get_config myasg region=us-east-1
```

salt.modules.boto_asg.**get_instances**(*name*, *lifecycle_state='InService'*, *health_status='Healthy'*, *attribute='private_ip_address'*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

return attribute of all instances in the named autoscale group.

CLI example:

```
salt-call boto_asg.get_instances my_autoscale_group_name
```

salt.modules.boto_asg.**get_scaling_policy_arn**(*as_group*, *scaling_policy_name*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Return the arn for a scaling policy in a specific autoscale group or None if not found. Mainly used as a helper method for boto_cloudwatch_alarm, for linking alarms to scaling policies.

CLI Example:

```
salt '*' boto_asg.get_scaling_policy_arn mygroup mypolicy
```

salt.modules.boto_asg.**launch_configuration_exists**(*name*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Check for a launch configuration's existence.

CLI example:

```
salt myminion boto_asg.launch_configuration_exists mylc
```

salt.modules.boto_asg.**update**(*name*, *launch_config_name*, *availability_zones*, *min_size*, *max_size*, *desired_capacity=None*, *load_balancers=None*, *default_cooldown=None*, *health_check_type=None*, *health_check_period=None*, *placement_group=None*, *vpc_zone_identifier=None*, *tags=None*, *termination_policies=None*, *suspended_processes=None*, *scaling_policies=None*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Update an autoscale group.

CLI example:

```
salt myminion boto_asg.update myasg mylc '["us-east-1a", "us-east-1e"]' 1 10 load_balancers='["m
```

22.16.14 salt.modules.boto_cfn

Connection module for Amazon Cloud Formation

configuration This module accepts explicit AWS credentials but can also utilize IAM roles assigned to the instance trough Instance Profiles. Dynamic credentials are then automatically obtained from AWS API and no further configuration is necessary. More Information available at:

```
http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/iam-roles-for-amazon-ec2.html
```

If IAM roles are not used you need to specify them either in a pillar or in the minion's config file:

```
cfn.keyid: GKTADJGHEIQSXMKKRBJ08H
cfn.key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs
```

A region may also be specified in the configuration:

```
        cfn.region: us-east-1
```

If a region is not specified, the default is us-east-1.

It's also possible to specify key, keyid and region via a profile, either as a passed in dict, or as a string to pull from pillars or minion config:

myprofile: keyid: GKTADJGHEIQSXMKKRBJ08H key: askdjghsdfjkghWupUjasdflkd-fklgjsdfjajkghs region: us-east-1

depends boto

`salt.modules.boto_cfn.`**`create`**(*name*, *template_url=None*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Create a CFN stack.

CLI example to create a stack:

```
    salt myminion boto_cfn.create mystack template_url='https://s3.amazonaws.com/bucket/template.cft
```

`salt.modules.boto_cfn.`**`delete`**(*name*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Delete a CFN stack.

CLI example to delete a stack:

```
    salt myminion boto_cfn.delete mystack region=us-east-1
```

`salt.modules.boto_cfn.`**`exists`**(*name*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Check to see if a stack exists.

CLI example:

```
    salt myminion boto_cfn.exists mystack region=us-east-1
```

`salt.modules.boto_cfn.`**`get_template`**(*name*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Check to see if attributes are set on a CFN stack.

CLI example:

```
    salt myminion boto_cfn.get_template mystack
```

22.16.15 salt.modules.boto_cloudwatch

Connection module for Amazon CloudWatch

New in version 2014.7.0.

configuration This module accepts explicit credentials but can also utilize IAM roles assigned to the instance trough Instance Profiles. Dynamic credentials are then automatically obtained from AWS API and no further configuration is necessary. More Information available at:

```
        http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/iam-roles-for-amazon-ec2.html
```

If IAM roles are not used you need to specify them either in a pillar or in the minion's config file:

```
        cloudwatch.keyid: GKTADJGHEIQSXMKKRBJ08H
        cloudwatch.key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs
```

A region may also be specified in the configuration:

```
cloudwatch.region: us-east-1
```

If a region is not specified, the default is us-east-1.

It's also possible to specify key, keyid and region via a profile, either as a passed in dict, or as a string to pull from pillars or minion config:

myprofile: keyid: GKTADJGHEIQSXMKKRBJ08H key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs region: us-east-1

depends boto

`salt.modules.boto_cloudwatch.`**`convert_to_arn`**(*arns*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Convert a list of strings into actual arns. Converts convenience names such as `scaling_policy:...'

CLI Example:

```
salt '*' convert_to_arn 'scaling_policy:'
```

`salt.modules.boto_cloudwatch.`**`create_or_update_alarm`**(*connection=None*, *name=None*, *metric=None*, *namespace=None*, *statistic=None*, *comparison=None*, *threshold=None*, *period=None*, *evaluation_periods=None*, *unit=None*, *description=''*, *dimensions=None*, *alarm_actions=None*, *insufficient_data_actions=None*, *ok_actions=None*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Create or update a cloudwatch alarm.

Params are the same as: http://boto.readthedocs.org/en/latest/ref/cloudwatch.html#boto.ec2.cloudwatch.alarm.MetricAlarm.

Dimensions must be a dict. If the value of Dimensions is a string, it will be json decoded to produce a dict. alarm_actions, insufficient_data_actions, and ok_actions must be lists of string. If the passed-in value is a string, it will be split on "," to produce a list. The strings themselves for alarm_actions, insufficient_data_actions, and ok_actions must be Amazon resource names (ARN's); however, this method also supports an arn lookup notation, as follows:

arn:aws:.... ARN as per http://docs.aws.amazon.com/general/latest/gr/aws-arns-and-namespaces.html scaling_policy:<as_name>:<scaling_policy_name> The named autoscale group scaling policy, for the named group (e.g. scaling_policy:my-asg:ScaleDown)

This is convenient for setting up autoscaling as follows. First specify a boto_asg.present state for an ASG with scaling_policies, and then set up boto_cloudwatch_alarm.present states which have alarm_actions that reference the scaling_policy.

CLI example:

salt myminion boto_cloudwatch.create_alarm name=myalarm ... region=us-east-1

`salt.modules.boto_cloudwatch.`**`delete_alarm`**(*name*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Delete a cloudwatch alarm

CLI example to delete a queue:

```
salt myminion boto_cloudwatch.delete_alarm myalarm region=us-east-1
```

salt.modules.boto_cloudwatch.**get_alarm**(*name*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Get alarm details. Also can be used to check to see if an alarm exists.

CLI example:

```
salt myminion boto_cloudwatch.get_alarm myalarm region=us-east-1
```

salt.modules.boto_cloudwatch.**get_all_alarms**(*region=None*, *prefix=None*, *key=None*, *keyid=None*, *profile=None*)

Get all alarm details. Produces results that can be used to create an sls file.

If prefix parameter is given, alarm names in the output will be prepended with the prefix; alarms that have the prefix will be skipped. This can be used to convert existing alarms to be managed by salt, as follows:

1. **Make a ``backup'' of all existing alarms** $ salt-call boto_cloudwatch.get_all_alarms --out=txt | sed ``s/local: //'' > legacy_alarms.sls
2. **Get all alarms with new prefixed names** $ salt-call boto_cloudwatch.get_all_alarms ``prefix=**MANAGED BY SALT** '' --out=txt | sed ``s/local: //'' > managed_alarms.sls
3. **Insert the managed alarms into cloudwatch** $ salt-call state.template managed_alarms.sls
4. Manually verify that the new alarms look right
5. Delete the original alarms $ sed s/present/absent/ legacy_alarms.sls > remove_legacy_alarms.sls $ salt-call state.template remove_legacy_alarms.sls
6. Get all alarms again, verify no changes $ salt-call boto_cloudwatch.get_all_alarms --out=txt | sed ``s/local: //'' > final_alarms.sls $ diff final_alarms.sls managed_alarms.sls

CLI example:

```
salt myminion boto_cloudwatch.get_all_alarms region=us-east-1 --out=txt
```

22.16.16 salt.modules.boto_dynamodb

Connection module for Amazon DynamoDB

New in version 2015.5.0.

configuration This module accepts explicit DynamoDB credentials but can also utilize IAM roles assigned to the instance trough Instance Profiles. Dynamic credentials are then automatically obtained from AWS API and no further configuration is necessary. More Information available at:

```
http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/iam-roles-for-amazon-ec2.html
```

If IAM roles are not used you need to specify them either in a pillar or in the minion's config file:

```
keyid: GKTADJGHEIQSXMKKRBJ08H
key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs
```

A region may also be specified in the configuration:

```
region: us-east-1
```

If a region is not specified, the default is us-east-1.

It's also possible to specify key, keyid and region via a profile, either as a passed in dict, or as a string to pull from pillars or minion config:

myprofile: keyid: GKTADJGHEIQSXMKKRBJ08H key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs region: us-east-1

depends boto

salt.modules.boto_dynamodb.**create_table**(*table_name*, *region=None*, *key=None*, *keyid=None*, *profile=None*, *read_capacity_units=None*, *write_capacity_units=None*, *hash_key=None*, *hash_key_data_type=None*, *range_key=None*, *range_key_data_type=None*, *local_indexes=None*, *global_indexes=None*)

Creates a DynamoDB table.

CLI example:

```
salt myminion boto_dynamodb.create_table table_name /
region=us-east-1 /
hash_key=id /
hash_key_data_type=N /
range_key=created_at /
range_key_data_type=N /
read_capacity_units=1 /
write_capacity_units=1
```

salt.modules.boto_dynamodb.**delete**(*table_name*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Delete a DynamoDB table.

CLI example:

```
salt myminion boto_dynamodb.delete table_name region=us-east-1
```

salt.modules.boto_dynamodb.**exists**(*table_name*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Check to see if a table exists.

CLI example:

```
salt myminion boto_dynamodb.exists table_name region=us-east-1
```

22.16.17 salt.modules.boto_elasticache

Connection module for Amazon Elasticache

New in version 2014.7.0.

configuration This module accepts explicit elasticache credentials but can also utilize IAM roles assigned to the instance trough Instance Profiles. Dynamic credentials are then automatically obtained from AWS API and no further configuration is necessary. More Information available at:

```
http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/iam-roles-for-amazon-ec2.html
```

If IAM roles are not used you need to specify them either in a pillar or in the minion's config file:

```
elasticache.keyid: GKTADJGHEIQSXMKKRBJ08H
elasticache.key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs
```

A region may also be specified in the configuration:

```
elasticache.region: us-east-1
```

If a region is not specified, the default is us-east-1.

It's also possible to specify key, keyid and region via a profile, either as a passed in dict, or as a string to pull from pillars or minion config:

myprofile: keyid: GKTADJGHEIQSXMKKRBJ08H key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs region: us-east-1

depends boto

salt.modules.boto_elasticache.**authorize_cache_security_group_ingress**(*name*, *ec2_security_group_name*, *ec2_security_group_owner_id*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Authorize network ingress from an ec2 security group to a cache security group.

CLI example:

```
salt myminion boto_elasticache.authorize_cache_security_group_ingress myelasticachesg myec2sg 87
```

salt.modules.boto_elasticache.**create**(*name*, *num_cache_nodes*, *engine*, *cache_node_type*, *replication_group_id=None*, *engine_version=None*, *cache_parameter_group_name=None*, *cache_subnet_group_name=None*, *cache_security_group_names=None*, *security_group_ids=None*, *snapshot_arns=None*, *preferred_availability_zone=None*, *preferred_maintenance_window=None*, *port=None*, *notification_topic_arn=None*, *auto_minor_version_upgrade=True*, *wait=False*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Create a cache cluster.

CLI example:

```
salt myminion boto_elasticache.create myelasticache 1 redis cache.t1.micro cache_security_group_
```

salt.modules.boto_elasticache.**create_cache_security_group**(*name*, *description*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Create a cache security group.

CLI example:

```
salt myminion boto_elasticache.create_cache_security_group myelasticachesg 'My Cache Security Gr
```

salt.modules.boto_elasticache.**delete**(*name*, *wait=False*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Delete a cache cluster.

CLI example:

```
salt myminion boto_elasticache.delete myelasticache
```

salt.modules.boto_elasticache.**delete_cache_security_group**(*name*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Delete a cache security group.

CLI example:

```
salt myminion boto_elasticache.delete_cache_security_group myelasticachesg 'My Cache Security Gr
```

salt.modules.boto_elasticache.**exists**(*name*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Check to see if a cache cluster exists.

CLI example:

```
salt myminion boto_elasticache.exists myelasticache
```

salt.modules.boto_elasticache.**get_cache_subnet_group**(*name*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Get information about a cache subnet group.

CLI example:

```
salt myminion boto_elasticache.get_cache_subnet_group mycache_subnet_group
```

salt.modules.boto_elasticache.**get_config**(*name*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Get the configuration for a cache cluster.

CLI example:

```
salt myminion boto_elasticache.get_config myelasticache
```

salt.modules.boto_elasticache.**revoke_cache_security_group_ingress**(*name*, *ec2_security_group_name*, *ec2_security_group_owner_id*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Revoke network ingress from an ec2 security group to a cache security group.

CLI example:

```
salt myminion boto_elasticache.revoke_cache_security_group_ingress myelasticachesg myec2sg 87987
```

22.16.18 salt.modules.boto_elb

Connection module for Amazon ELB

New in version 2014.7.0.

configuration This module accepts explicit elb credentials but can also utilize IAM roles assigned to the instance trough Instance Profiles. Dynamic credentials are then automatically obtained from AWS API and no further configuration is necessary. More Information available at:

```
        http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/iam-roles-for-amazon-ec2.html
```

If IAM roles are not used you need to specify them either in a pillar or in the minion's config file:

```
        elb.keyid: GKTADJGHEIQSXMKKRBJ08H
        elb.key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs
```

A region may also be specified in the configuration:

```
        elb.region: us-east-1
```

If a region is not specified, the default is us-east-1.

It's also possible to specify key, keyid and region via a profile, either as a passed in dict, or as a string to pull from pillars or minion config:

myprofile: keyid: GKTADJGHEIQSXMKKRBJ08H key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs region: us-east-1

depends boto

salt.modules.boto_elb.**attach_subnets**(*name*, *subnets*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Attach ELB to subnets.

CLI example:

```
    salt myminion boto_elb.attach_subnets myelb '["mysubnet"]'
```

salt.modules.boto_elb.**create**(*name*, *availability_zones*, *listeners=None*, *subnets=None*, *security_groups=None*, *scheme='internet-facing'*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Create an ELB

CLI example to create an ELB:

```
    salt myminion boto_elb.create myelb '["us-east-1a", "us-east-1e"]' listeners='[["HTTPS", "HTTP",
```

salt.modules.boto_elb.**create_listeners**(*name*, *listeners=None*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Create listeners on an ELB.

CLI example:

```
    salt myminion boto_elb.create_listeners myelb listeners='[["HTTPS", "HTTP", 443, 80, "arn:aws:ia
```

salt.modules.boto_elb.**delete**(*name*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Delete an ELB.

CLI example to delete an ELB:

```
    salt myminion boto_elb.delete myelb region=us-east-1
```

salt.modules.boto_elb.**delete_listeners**(*name*, *ports*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Delete listeners on an ELB.

CLI example:

```
    salt myminion boto_elb.delete_listeners myelb '[80,443]'
```

salt.modules.boto_elb.**deregister_instances**(*name*, *instances*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Deregister instances with an ELB. Instances is either a string instance id or a list of string instance id's.

Returns:

- `True`: instance(s) deregistered successfully
- `False`: instance(s) failed to be deregistered
- `None`: instance(s) not valid or not registered, no action taken

CLI example:

```
salt myminion boto_elb.deregister_instances myelb instance_id
salt myminion boto_elb.deregister_instances myelb "[instance_id, instance_id]"
```

salt.modules.boto_elb.**detach_subnets**(*name*, *subnets*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Detach ELB from subnets.

CLI example:

```
salt myminion boto_elb.detach_subnets myelb '["mysubnet"]'
```

salt.modules.boto_elb.**disable_availability_zones**(*name*, *availability_zones*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Disable availability zones for ELB.

CLI example:

```
salt myminion boto_elb.disable_availability_zones myelb '["us-east-1a"]'
```

salt.modules.boto_elb.**enable_availability_zones**(*name*, *availability_zones*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Enable availability zones for ELB.

CLI example:

```
salt myminion boto_elb.enable_availability_zones myelb '["us-east-1a"]'
```

salt.modules.boto_elb.**exists**(*name*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Check to see if an ELB exists.

CLI example:

```
salt myminion boto_elb.exists myelb region=us-east-1
```

salt.modules.boto_elb.**get_attributes**(*name*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Check to see if attributes are set on an ELB.

CLI example:

```
salt myminion boto_elb.get_attributes myelb
```

salt.modules.boto_elb.**get_elb_config**(*name*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Check to see if an ELB exists.

CLI example:

```
salt myminion boto_elb.exists myelb region=us-east-1
```

salt.modules.boto_elb.**get_health_check**(*name*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Get the health check configured for this ELB.

CLI example:

```
salt myminion boto_elb.get_health_check myelb
```

salt.modules.boto_elb.**get_instance_health**(*name*, *region=None*, *key=None*, *keyid=None*, *profile=None*, *instances=None*)

Get a list of instances and their health state

CLI example:

```
salt myminion boto_elb.get_instance_health myelb
salt myminion boto_elb.get_instance_health myelb region=us-east-1 instances="[instance_id,instan
```

salt.modules.boto_elb.**register_instances**(*name*, *instances*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Register instances with an ELB. Instances is either a string instance id or a list of string instance id's.

Returns:

- `True`: instance(s) registered successfully
- `False`: instance(s) failed to be registered

CLI example:

```
salt myminion boto_elb.register_instances myelb instance_id
salt myminion boto_elb.register_instances myelb "[instance_id,instance_id]"
```

salt.modules.boto_elb.**set_attributes**(*name*, *attributes*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Set attributes on an ELB.

CLI example to set attributes on an ELB:

```
salt myminion boto_elb.set_attributes myelb '{"access_log": {"enabled": "true", "s3_bucket_name"
```

salt.modules.boto_elb.**set_health_check**(*name*, *health_check*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Set attributes on an ELB.

CLI example to set attributes on an ELB:

```
salt myminion boto_elb.set_health_check myelb '{"target": "HTTP:80/"}'
```

22.16.19 salt.modules.boto_iam

Connection module for Amazon IAM

New in version 2014.7.0.

configuration This module accepts explicit iam credentials but can also utilize IAM roles assigned to the instance trough Instance Profiles. Dynamic credentials are then automatically obtained from AWS API and no further configuration is necessary. More Information available at:

```
        http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/iam-roles-for-amazon-ec2.html
```

If IAM roles are not used you need to specify them either in a pillar or in the minion's config file:

```
        iam.keyid: GKTADJGHEIQSXMKKRBJ08H
        iam.key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs
        iam.region: us-east-1
```

It's also possible to specify key, keyid and region via a profile, either as a passed in dict, or as a string to pull from pillars or minion config:

myprofile: keyid: GKTADJGHEIQSXMKKRBJ08H key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs region: us-east-1

depends boto

salt.modules.boto_iam.**associate_profile_to_role**(*profile_name*, *role_name*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Associate an instance profile with an IAM role.

CLI example:

```
    salt myminion boto_iam.associate_profile_to_role myirole myiprofile
```

salt.modules.boto_iam.**create_instance_profile**(*name*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Create an instance profile.

CLI example:

```
    salt myminion boto_iam.create_instance_profile myiprofile
```

salt.modules.boto_iam.**create_role**(*name*, *policy_document=None*, *path=None*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Create an instance role.

CLI example:

```
    salt myminion boto_iam.create_role myrole
```

salt.modules.boto_iam.**create_role_policy**(*role_name*, *policy_name*, *policy*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Create or modify a role policy.

CLI example:

```
    salt myminion boto_iam.create_role_policy myirole mypolicy '{"MyPolicy": "Statement": [{"Action"
```

salt.modules.boto_iam.**delete_instance_profile**(*name*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Delete an instance profile.

CLI example:

```
    salt myminion boto_iam.delete_instance_profile myiprofile
```

salt.modules.boto_iam.**delete_role**(*name*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Delete an IAM role.

CLI example:

```
salt myminion boto_iam.delete_role myirole
```

salt.modules.boto_iam.**delete_role_policy**(*role_name*, *policy_name*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Delete a role policy.

CLI example:

```
salt myminion boto_iam.delete_role_policy myirole mypolicy
```

salt.modules.boto_iam.**disassociate_profile_from_role**(*profile_name*, *role_name*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Disassociate an instance profile from an IAM role.

CLI example:

```
salt myminion boto_iam.disassociate_profile_from_role myirole myiprofile
```

salt.modules.boto_iam.**get_account_id**(*region=None*, *key=None*, *keyid=None*, *profile=None*)

Get a the AWS account id associated with the used credentials.

CLI example:

```
salt myminion boto_iam.get_account_id
```

salt.modules.boto_iam.**get_role_policy**(*role_name*, *policy_name*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Get a role policy.

CLI example:

```
salt myminion boto_iam.get_role_policy myirole mypolicy
```

salt.modules.boto_iam.**instance_profile_exists**(*name*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Check to see if an instance profile exists.

CLI example:

```
salt myminion boto_iam.instance_profile_exists myiprofile
```

salt.modules.boto_iam.**list_role_policies**(*role_name*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Get a list of policy names from a role.

CLI example:

```
salt myminion boto_iam.list_role_policies myirole
```

salt.modules.boto_iam.**profile_associated**(*role_name*, *profile_name*, *region*, *key*, *keyid*, *profile*)

Check to see if an instance profile is associated with an IAM role.

CLI example:

```
salt myminion boto_iam.profile_associated myirole myiprofile
```

salt.modules.boto_iam.**role_exists**(*name*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Check to see if an IAM role exists.

CLI example:

```
salt myminion boto_iam.role_exists myirole
```

22.16.20 salt.modules.boto_route53

Connection module for Amazon Route53

New in version 2014.7.0.

configuration This module accepts explicit route53 credentials but can also utilize IAM roles assigned to the instance trough Instance Profiles. Dynamic credentials are then automatically obtained from AWS API and no further configuration is necessary. More Information available at:

```
http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/iam-roles-for-amazon-ec2.html
```

If IAM roles are not used you need to specify them either in a pillar or in the minion's config file:

```
route53.keyid: GKTADJGHEIQSXMKKRBJ08H
route53.key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs
```

A region may also be specified in the configuration:

```
route53.region: us-east-1
```

If a region is not specified, the default is us-east-1.

It's also possible to specify key, keyid and region via a profile, either as a passed in dict, or as a string to pull from pillars or minion config:

myprofile: keyid: GKTADJGHEIQSXMKKRBJ08H key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs region: us-east-1

depends boto

salt.modules.boto_route53.**add_record**(*name*, *value*, *zone*, *record_type*, *identifier=None*, *ttl=None*, *region=None*, *key=None*, *keyid=None*, *profile=None*, *sync_wait=False*)

Add a record to a zone.

CLI example:

```
salt myminion boto_route53.add_record test.example.org 1.1.1.1 example.org A
```

salt.modules.boto_route53.**delete_record**(*name*, *zone*, *record_type*, *identifier=None*, *all_records=False*, *region=None*, *key=None*, *keyid=None*, *profile=None*, *sync_wait=False*)

Modify a record in a zone.

CLI example:

```
salt myminion boto_route53.delete_record test.example.org example.org A
```

salt.modules.boto_route53.**get_record**(*name*, *zone*, *record_type*, *fetch_all=False*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Get a record from a zone.

CLI example:

```
salt myminion boto_route53.get_record test.example.org example.org A
```

salt.modules.boto_route53.**update_record**(*name*, *value*, *zone*, *record_type*, *identifier=None*, *ttl=None*, *region=None*, *key=None*, *keyid=None*, *profile=None*, *sync_wait=False*)

Modify a record in a zone.

CLI example:

```
salt myminion boto_route53.modify_record test.example.org 1.1.1.1 example.org A
```

22.16.21 salt.modules.boto_secgroup

Connection module for Amazon Security Groups

New in version 2014.7.0.

configuration This module accepts explicit ec2 credentials but can also utilize IAM roles assigned to the instance trough Instance Profiles. Dynamic credentials are then automatically obtained from AWS API and no further configuration is necessary. More Information available at:

```
http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/iam-roles-for-amazon-ec2.html
```

If IAM roles are not used you need to specify them either in a pillar or in the minion's config file:

```
secgroup.keyid: GKTADJGHEIQSXMKKRBJ08H
secgroup.key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs
```

A region may also be specified in the configuration:

```
secgroup.region: us-east-1
```

If a region is not specified, the default is us-east-1.

It's also possible to specify key, keyid and region via a profile, either as a passed in dict, or as a string to pull from pillars or minion config:

myprofile: keyid: GKTADJGHEIQSXMKKRBJ08H key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs region: us-east-1

depends boto

salt.modules.boto_secgroup.**authorize**(*name=None*, *source_group_name=None*, *source_group_owner_id=None*, *ip_protocol=None*, *from_port=None*, *to_port=None*, *cidr_ip=None*, *group_id=None*, *source_group_group_id=None*, *region=None*, *key=None*, *keyid=None*, *profile=None*, *vpc_id=None*)

Add a new rule to an existing security group.

CLI example:

```
salt myminion boto_secgroup.authorize mysecgroup ip_protocol=tcp from_port=80 to_port=80 cidr_ip
```

salt.modules.boto_secgroup.**convert_to_group_ids**(*groups*, *vpc_id*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Given a list of security groups and a vpc_id, convert_to_group_ids will convert all list items in the given list to security group ids.

CLI example:

```
salt myminion boto_secgroup.convert_to_group_ids mysecgroup vpc-89yhh7h
```

salt.modules.boto_secgroup.**create**(*name*, *description*, *vpc_id=None*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Create an autoscale group.

CLI example:

```
salt myminion boto_secgroup.create mysecgroup 'My Security Group'
```

salt.modules.boto_secgroup.**delete**(*name=None*, *group_id=None*, *region=None*, *key=None*, *keyid=None*, *profile=None*, *vpc_id=None*)

Delete an autoscale group.

CLI example:

```
salt myminion boto_secgroup.delete mysecgroup
```

salt.modules.boto_secgroup.**exists**(*name=None*, *region=None*, *key=None*, *keyid=None*, *profile=None*, *vpc_id=None*, *group_id=None*)

Check to see if an security group exists.

CLI example:

```
salt myminion boto_secgroup.exists mysecgroup
```

salt.modules.boto_secgroup.**get_config**(*name=None*, *group_id=None*, *region=None*, *key=None*, *keyid=None*, *profile=None*, *vpc_id=None*)

Get the configuration for a security group.

CLI example:

```
salt myminion boto_secgroup.get_config mysecgroup
```

salt.modules.boto_secgroup.**get_group_id**(*name*, *vpc_id=None*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Get a Group ID given a Group Name or Group Name and VPC ID

CLI example:

```
salt myminion boto_secgroup.get_group_id mysecgroup
```

salt.modules.boto_secgroup.**revoke**(*name=None*, *source_group_name=None*, *source_group_owner_id=None*, *ip_protocol=None*, *from_port=None*, *to_port=None*, *cidr_ip=None*, *group_id=None*, *source_group_group_id=None*, *region=None*, *key=None*, *keyid=None*, *profile=None*, *vpc_id=None*)

Remove a rule from an existing security group.

CLI example:

```
salt myminion boto_secgroup.revoke mysecgroup ip_protocol=tcp from_port=80 to_port=80 cidr_ip='1
```

22.16.22 salt.modules.boto_sns

Connection module for Amazon SNS

configuration This module accepts explicit sns credentials but can also utilize IAM roles assigned to the instance trough Instance Profiles. Dynamic credentials are then automatically obtained from AWS API and no further configuration is necessary. More Information available at:

```
        http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/iam-roles-for-amazon-ec2.html
```

If IAM roles are not used you need to specify them either in a pillar or in the minion's config file:

```
sns.keyid: GKTADJGHEIQSXMKKRBJ08H
sns.key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs
```

A region may also be specified in the configuration:

```
sns.region: us-east-1
```

If a region is not specified, the default is us-east-1.

It's also possible to specify key, keyid and region via a profile, either as a passed in dict, or as a string to pull from pillars or minion config:

myprofile: keyid: GKTADJGHEIQSXMKKRBJ08H key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs region: us-east-1

depends boto

salt.modules.boto_sns.**create**(*name*, *region=None*, *key=None*, *keyid=None*, *profile=None*)
Create an SNS topic.

CLI example to create a topic:

```
salt myminion boto_sns.create mytopic region=us-east-1
```

salt.modules.boto_sns.**delete**(*name*, *region=None*, *key=None*, *keyid=None*, *profile=None*)
Delete an SNS topic.

CLI example to delete a topic:

```
salt myminion boto_sns.delete mytopic region=us-east-1
```

salt.modules.boto_sns.**exists**(*name*, *region=None*, *key=None*, *keyid=None*, *profile=None*)
Check to see if an SNS topic exists.

CLI example:

```
salt myminion boto_sns.exists mytopic region=us-east-1
```

salt.modules.boto_sns.**get_all_topics**(*region=None*, *key=None*, *keyid=None*, *profile=None*)
Returns a list of the all topics..

CLI example:

```
salt myminion boto_sns.get_all_topics
```

salt.modules.boto_sns.**get_arn**(*name*, *region=None*, *key=None*, *keyid=None*, *profile=None*)
Returns the full ARN for a given topic name.

CLI example:

```
salt myminion boto_sns.get_arn mytopic
```

22.16.23 salt.modules.boto_sqs

Connection module for Amazon SQS

New in version 2014.7.0.

configuration This module accepts explicit sqs credentials but can also utilize IAM roles assigned to the instance trough Instance Profiles. Dynamic credentials are then automatically obtained from AWS API and no further configuration is necessary. More Information available at:

```
http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/iam-roles-for-amazon-ec2.html
```

If IAM roles are not used you need to specify them either in a pillar or in the minion's config file:

```
sqs.keyid: GKTADJGHEIQSXMKKRBJ08H
sqs.key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs
```

A region may also be specified in the configuration:

```
sqs.region: us-east-1
```

If a region is not specified, the default is us-east-1.

It's also possible to specify key, keyid and region via a profile, either as a passed in dict, or as a string to pull from pillars or minion config:

myprofile: keyid: GKTADJGHEIQSXMKKRBJ08H key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs region: us-east-1

depends boto

salt.modules.boto_sqs.**create**(*name*, *region=None*, *key=None*, *keyid=None*, *profile=None*)
Create an SQS queue.

CLI example to create a queue:

```
salt myminion boto_sqs.create myqueue region=us-east-1
```

salt.modules.boto_sqs.**delete**(*name*, *region=None*, *key=None*, *keyid=None*, *profile=None*)
Delete an SQS queue.

CLI example to delete a queue:

```
salt myminion boto_sqs.delete myqueue region=us-east-1
```

salt.modules.boto_sqs.**exists**(*name*, *region=None*, *key=None*, *keyid=None*, *profile=None*)
Check to see if a queue exists.

CLI example:

```
salt myminion boto_sqs.exists myqueue region=us-east-1
```

salt.modules.boto_sqs.**get_attributes**(*name*, *region=None*, *key=None*, *keyid=None*, *profile=None*)
Check to see if attributes are set on an SQS queue.

CLI example:

```
salt myminion boto_sqs.get_attributes myqueue
```

salt.modules.boto_sqs.**set_attributes**(*name*, *attributes*, *region=None*, *key=None*, *keyid=None*, *profile=None*)
Set attributes on an SQS queue.

CLI example to set attributes on a queue:

```
salt myminion boto_sqs.set_attributes myqueue '{ReceiveMessageWaitTimeSeconds: 20}' region=us-ea
```

22.16.24 salt.modules.boto_vpc

Connection module for Amazon VPC

New in version 2014.7.0.

configuration This module accepts explicit VPC credentials but can also utilize IAM roles assigned to the instance trough Instance Profiles. Dynamic credentials are then automatically obtained from AWS API and no further configuration is necessary. More Information available at:

```
http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/iam-roles-for-amazon-ec2.html
```

If IAM roles are not used you need to specify them either in a pillar or in the minion's config file:

```
vpc.keyid: GKTADJGHEIQSXMKKRBJ08H
vpc.key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs
```

A region may also be specified in the configuration:

```
vpc.region: us-east-1
```

If a region is not specified, the default is us-east-1.

It's also possible to specify key, keyid and region via a profile, either as a passed in dict, or as a string to pull from pillars or minion config:

myprofile: keyid: GKTADJGHEIQSXMKKRBJ08H key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs region: us-east-1

depends boto

salt.modules.boto_vpc.**associate_dhcp_options_to_vpc**(*dhcp_options_id*, *vpc_id*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Given valid DHCP options id and a valid VPC id, associate the DHCP options record with the VPC.

Returns True if the DHCP options record were associated and returns False if the DHCP options record was not associated.

CLI example:

```
.. code-block:: bash
```

salt myminion boto_vpc.associate_dhcp_options_to_vpc `dhcp-a0bl34pp' `vpc-6b1fe402'

salt.modules.boto_vpc.**associate_network_acl_to_subnet**(*network_acl_id*, *subnet_id*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Given a network acl ID and a subnet ID, associates a network acl to a subnet.

Returns the association ID if successful, otherwise returns False.

CLI Example:

```
.. code-block:: bash
```

salt myminion boto_vpc.associate_network_acl_to_subnet `acl-5fb85d36' `subnet-6a1fe403'

salt.modules.boto_vpc.**associate_new_dhcp_options_to_vpc**(*vpc_id*, *domain_name=None*, *domain_name_servers=None*, *ntp_servers=None*, *netbios_name_servers=None*, *netbios_node_type=None*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Given valid DHCP options and a valid VPC id, create and associate the DHCP options record with the VPC.

Returns True if the DHCP options record were created and associated and returns False if the DHCP options record was not created and associated.

CLI example:

```
.. code-block:: bash
```

salt myminion boto_vpc.associate_new_dhcp_options_to_vpc `vpc-6b1fe402' domain_name='example.com' domain_name_servers='[1.2.3.4]' ntp_servers='[5.6.7.8]' netbios_name_servers='[10.0.0.1]' netbios_node_type=1

salt.modules.boto_vpc.**associate_new_network_acl_to_subnet**(*vpc_id*, *subnet_id*, *network_acl_name=None*, *tags=None*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Given a vpc ID and a subnet ID, associates a new network act to a subnet.

Returns a dictionary containing the network acl id and the new association id if successful. If unsuccessful, returns False.

CLI Example:

```
.. code-block:: bash
```

salt myminion boto_vpc.associate_new_network_acl_to_subnet `vpc-6b1fe402' `subnet-6a1fe403'

salt.modules.boto_vpc.**associate_route_table**(*route_table_id*, *subnet_id*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Given a route table ID and a subnet ID, associates the route table with the subnet.

CLI Example:

```
.. code-block:: bash
```

salt myminion boto_vpc.associate_route_table `rtb-1f382e7d' `subnet-6a1fe403'

salt.modules.boto_vpc.**create**(*cidr_block*, *instance_tenancy=None*, *vpc_name=None*, *tags=None*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Given a valid CIDR block, create a VPC.

An optional instance_tenancy argument can be provided. If provided, the valid values are `default' or `dedicated' An optional vpc_name argument can be provided.

Returns True if the VPC was created and returns False if the VPC was not created.

CLI example:

```
.. code-block:: bash
```

salt myminion boto_vpc.create `10.0.0.0/24'

salt.modules.boto_vpc.**create_customer_gateway**(*vpn_connection_type*, *ip_address*, *bgp_asn*, *customer_gateway_name=None*, *tags=None*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Given a valid VPN connection type, a static IP address and a customer gateway's Border Gateway Protocol (BGP) Autonomous System Number, create a customer gateway.

Returns True if the customer gateway was created and returns False if the customer gateway was not created.

CLI example:

```
.. code-block:: bash
```

salt myminion boto_vpc.create_customer_gateway `ipsec.1', `12.1.2.3', 65534

salt.modules.boto_vpc.**create_dhcp_options**(*domain_name=None*, *domain_name_servers=None*, *ntp_servers=None*, *netbios_name_servers=None*, *netbios_node_type=None*, *dhcp_options_name=None*, *tags=None*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Given valid DHCP options, create a DHCP options record.

Returns True if the DHCP options record was created and returns False if the DHCP options record was not deleted.

CLI example:

```
.. code-block:: bash
```

salt myminion boto_vpc.create_dhcp_options domain_name='example.com' domain_name_servers='[1.2.3.4]' ntp_servers='[5.6.7.8]' netbios_name_servers='[10.0.0.1]' netbios_node_type=1

salt.modules.boto_vpc.**create_network_acl**(*vpc_id*, *network_acl_name=None*, *tags=None*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Given a vpc_id, creates a network acl.

Returns the network acl id if successful, otherwise returns False.

CLI Example:

```
.. code-block:: bash
```

salt myminion boto_vpc.create_network_acl `vpc-6b1fe402'

salt.modules.boto_vpc.**create_network_acl_entry**(*network_acl_id*, *rule_number*, *protocol*, *rule_action*, *cidr_block*, *egress=None*, *icmp_code=None*, *icmp_type=None*, *port_range_from=None*, *port_range_to=None*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Creates a network acl entry.

CLI Example:

```
.. code-block:: bash
```

salt myminion boto_vpc.create_network_acl_entry `acl-5fb85d36' `32767' `-1' `deny' `0.0.0.0/0'

salt.modules.boto_vpc.**create_route**(*route_table_id*, *destination_cidr_block*, *gateway_id=None*, *instance_id=None*, *interface_id=None*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Creates a route.

CLI Example:

```
.. code-block:: bash
```

salt myminion boto_vpc.create_route `rtb-1f382e7d' `10.0.0.0/16'

salt.modules.boto_vpc.**create_route_table**(*vpc_id*, *route_table_name=None*, *tags=None*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Creates a route table.

CLI Example:

```
.. code-block:: bash
```

salt myminion boto_vpc.create_route_table `vpc-6b1fe402'

salt.modules.boto_vpc.**create_subnet**(*vpc_id*, *cidr_block*, *availability_zone=None*, *subnet_name=None*, *tags=None*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Given a valid VPC ID and a CIDR block, create a subnet for the VPC.

An optional availability zone argument can be provided.

Returns True if the VPC subnet was created and returns False if the VPC subnet was not created.

CLI example:

```
.. code-block:: bash
```

salt myminion boto_vpc.create_subnet `vpc-6b1fe402' `10.0.0.0/25'

salt.modules.boto_vpc.**customer_gateway_exists**(*customer_gateway_id*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Given a customer gateway ID, check if the customer gateway ID exists.

Returns True if the customer gateway ID exists; Returns False otherwise.

CLI Example:

```
.. code-block:: bash
```

salt myminion boto_vpc.customer_gateway_exists `cgw-b6a247df'

salt.modules.boto_vpc.**delete**(*vpc_id*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Given a VPC ID, delete the VPC.

Returns True if the VPC was deleted and returns False if the VPC was not deleted.

CLI example:

```
.. code-block:: bash
```

salt myminion boto_vpc.delete `vpc-6b1fe402'

salt.modules.boto_vpc.**delete_customer_gateway**(*customer_gateway_id*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Given a customer gateway ID, delete the customer gateway.

Returns True if the customer gateway was deleted and returns False if the customer gateway was not deleted.

CLI example:

```
.. code-block:: bash
```

salt myminion boto_vpc.delete_customer_gateway `cgw-b6a247df'

salt.modules.boto_vpc.**delete_network_acl**(*network_acl_id*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Deletes a network acl based on the network_acl_id provided.

Returns True if the network acl was deleted successfully, otherwise returns False.

CLI Example:

```
.. code-block:: bash
```

salt myminion boto_vpc.delete_network_acl `acl-5fb85d36'

salt.modules.boto_vpc.**delete_network_acl_entry**(*network_acl_id*, *rule_number*, *egress=None*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Deletes a network acl entry.

CLI Example:

```
.. code-block:: bash
```

salt myminion boto_vpc.delete_network_acl_entry `acl-5fb85d36' `32767'

salt.modules.boto_vpc.**delete_route**(*route_table_id*, *destination_cidr_block*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Deletes a route.

CLI Example:

```
.. code-block:: bash
```

salt myminion boto_vpc.delete_route `rtb-1f382e7d' `10.0.0.0/16'

salt.modules.boto_vpc.**delete_route_table**(*route_table_id*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Deletes a route table.

CLI Example:

```
.. code-example:: bash
```

salt myminion boto_vpc.delete_route_table `rtb-1f382e7d'

salt.modules.boto_vpc.**delete_subnet**(*subnet_id*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Given a subnet ID, delete the subnet.

Returns True if the subnet was deleted and returns False if the subnet was not deleted.

CLI example:

```
.. code-block:: bash
```

salt myminion boto_vpc.delete_subnet `subnet-6a1fe403'

salt.modules.boto_vpc.**describe**(*vpc_id=None*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Given a VPC ID describe it's properties.

Returns a dictionary of interesting properties. CLI example:

```
.. code-block:: bash
```

salt myminion boto_vpc.describe vpc_id=vpc-123456

salt.modules.boto_vpc.**dhcp_options_exists**(*dhcp_options_id=None*, *name=None*, *tags=None*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Check if a dhcp option exists.

Returns True if the dhcp option exists; Returns False otherwise.

CLI Example:

```
.. code-block:: bash
```

salt myminion boto_vpc.dhcp_options_exists dhcp_options_id='dhcp-a0bl34pp'

salt.modules.boto_vpc.**disassociate_network_acl**(*subnet_id*, *vpc_id=None*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Given a subnet ID, disassociates a network acl.

CLI Example:

```
.. code-block:: bash
```

salt myminion boto_vpc.disassociate_network_acl `subnet-6a1fe403'

salt.modules.boto_vpc.**disassociate_route_table**(*association_id*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Dissassociates a route table.

association_id The Route Table Association ID to disassociate

CLI Example:

```
.. code-block:: bash
```

salt myminion boto_vpc.disassociate_route_table `rtbassoc-d8ccddba'

salt.modules.boto_vpc.**exists**(*vpc_id=None*, *name=None*, *cidr=None*, *tags=None*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Given a VPC ID, check to see if the given VPC ID exists.

Returns True if the given VPC ID exists and returns False if the given VPC ID does not exist.

CLI example:

```
.. code-block:: bash
```

salt myminion boto_vpc.exists myvpc

salt.modules.boto_vpc.**get_id**(*name=None*, *cidr=None*, *tags=None*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Given a VPC properties, return VPC ID if exist.

CLI example:

```
.. code-block:: bash
```

salt myminion boto_vpc.get_id myvpc

salt.modules.boto_vpc.**get_subnet_association**(*subnets*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Given a subnet (aka: a vpc zone identifier) or list of subnets, returns vpc association.

Returns a VPC ID if the given subnets are associated with the same VPC ID. Returns False on an error or if the given subnets are associated with different VPC IDs.

CLI Examples:

```
.. code-block:: bash
```

salt myminion boto_vpc.get_subnet_association subnet-61b47516

```
salt myminion boto_vpc.get_subnet_association ['subnet-61b47516','subnet-2cb9785b']
```

salt.modules.boto_vpc.**network_acl_exists**(*network_acl_id=None*, *name=None*, *tags=None*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Checks if a network acl exists.

Returns True if the network acl exists or returns False if it doesn't exist.

CLI Example:

```
.. code-block:: bash
```

salt myminion boto_vpc.network_acl_exists network_acl_id='acl-5fb85d36'

salt.modules.boto_vpc.**replace_network_acl_entry**(*network_acl_id*, *rule_number*, *protocol*, *rule_action*, *cidr_block*, *egress=None*, *icmp_code=None*, *icmp_type=None*, *port_range_from=None*, *port_range_to=None*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Replaces a network acl entry.

CLI Example:

```
.. code-block:: bash
```

salt myminion boto_vpc.replace_network_acl_entry `acl-5fb85d36' `32767' `-1' `deny' `0.0.0.0/0'

salt.modules.boto_vpc.**replace_route**(*route_table_id*, *destination_cidr_block*, *gateway_id=None*, *instance_id=None*, *interface_id=None*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Replaces a route.

CLI Example:

```
.. code-block:: bash
```

salt myminion boto_vpc.replace_route `rtb-1f382e7d' `10.0.0.0/16'

salt.modules.boto_vpc.**replace_route_table_association**(*association_id*, *route_table_id*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Replaces a route table association.

CLI Example:

```
.. code-block:: bash
```

salt myminion boto_vpc.replace_route_table_association `rtbassoc-d8ccddba' `rtb-1f382e7d'

salt.modules.boto_vpc.**route_table_exists**(*route_table_id=None*, *name=None*, *tags=None*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Checks if a route table exists.

CLI Example:

```
.. code-block:: bash
```

salt myminion boto_vpc.route_table_exists route_table_id='rtb-1f382e7d'

salt.modules.boto_vpc.**subnet_exists**(*subnet_id=None*, *name=None*, *tags=None*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Check if a subnet exists.

Returns True if the subnet exists, otherwise returns False.

CLI Example:

```
.. code-block:: bash
```

salt myminion boto_vpc.subnet_exists subnet_id='subnet-6a1fe403'

22.16.25 salt.modules.brew

Homebrew for Mac OS X

salt.modules.brew.**install**(*name=None*, *pkgs=None*, *taps=None*, *options=None*, ***kwargs*)

Install the passed package(s) with `brew install`

name The name of the formula to be installed. Note that this parameter is ignored if ``pkgs'' is passed.

CLI Example:

```
salt '*' pkg.install <package name>
```

taps Unofficial Github repos to use when updating and installing formulas.

CLI Example:

```
salt '*' pkg.install <package name> tap='<tap>'
salt '*' pkg.install zlib taps='homebrew/dupes'
salt '*' pkg.install php54 taps='["josegonzalez/php", "homebrew/dupes"]'
```

options Options to pass to brew. Only applies to initial install. Due to how brew works, modifying chosen options requires a full uninstall followed by a fresh install. Note that if ``pkgs'' is used, all options will be passed to all packages. Unrecognized options for a package will be silently ignored by brew.

CLI Example:

```
salt '*' pkg.install <package name> tap='<tap>'
salt '*' pkg.install php54 taps='["josegonzalez/php", "homebrew/dupes"]' options='["--with-f
```

Multiple Package Installation Options:

pkgs A list of formulas to install. Must be passed as a python list.

CLI Example:

```
salt '*' pkg.install pkgs='["foo","bar"]'
```

Returns a dict containing the new package names and versions:

```
{'<package>': {'old': '<old-version>',
               'new': '<new-version>'}}
```

CLI Example:

```
salt '*' pkg.install 'package package package'
```

`salt.modules.brew.`**`latest_version`**(**names*, ***kwargs*)

Return the latest version of the named package available for upgrade or installation

Note that this currently not fully implemented but needs to return something to avoid a traceback when calling pkg.latest.

CLI Example:

```
salt '*' pkg.latest_version <package name>
salt '*' pkg.latest_version <package1> <package2> <package3>
```

`salt.modules.brew.`**`list_pkgs`**(*versions_as_list=False*, ***kwargs*)

List the packages currently installed in a dict:

```
{'<package_name>': '<version>'}
```

CLI Example:

```
salt '*' pkg.list_pkgs
```

`salt.modules.brew.`**`list_upgrades`**(*refresh=True*)

Check whether or not an upgrade is available for all packages

CLI Example:

```
salt '*' pkg.list_upgrades
```

salt.modules.brew.**refresh_db**()
Update the homebrew package repository.

CLI Example:

```
salt '*' pkg.refresh_db
```

salt.modules.brew.**remove**(*name=None*, *pkgs=None*, ***kwargs*)
Removes packages with `brew uninstall`.

name The name of the package to be deleted.

Multiple Package Options:

pkgs A list of packages to delete. Must be passed as a python list. The `name` parameter will be ignored if this option is passed.

New in version 0.16.0.

Returns a dict containing the changes.

CLI Example:

```
salt '*' pkg.remove <package name>
salt '*' pkg.remove <package1>,<package2>,<package3>
salt '*' pkg.remove pkgs='["foo", "bar"]'
```

salt.modules.brew.**upgrade**(*refresh=True*)
Upgrade outdated, unpinned brews.

refresh Fetch the newest version of Homebrew and all formulae from GitHub before installing.

Return a dict containing the new package names and versions:

```
{'<package>': {'old': '<old-version>',
               'new': '<new-version>'}}
```

CLI Example:

```
salt '*' pkg.upgrade
```

salt.modules.brew.**upgrade_available**(*pkg*)
Check whether or not an upgrade is available for a given package

CLI Example:

```
salt '*' pkg.upgrade_available <package name>
```

salt.modules.brew.**version**(**names*, ***kwargs*)
Returns a string representing the package version or an empty string if not installed. If more than one package name is specified, a dict of name/version pairs is returned.

CLI Example:

```
salt '*' pkg.version <package name>
salt '*' pkg.version <package1> <package2> <package3>
```

22.16.26 salt.modules.bridge

Module for gathering and managing bridging information

salt.modules.bridge.**add**(*br=None*)
Creates a bridge

CLI Example:

```
salt '*' bridge.add br0
```

salt.modules.bridge.**addif**(*br=None*, *iface=None*)
Adds an interface to a bridge

CLI Example:

```
salt '*' bridge.addif br0 eth0
```

salt.modules.bridge.**delete**(*br=None*)
Deletes a bridge

CLI Example:

```
salt '*' bridge.delete br0
```

salt.modules.bridge.**delif**(*br=None*, *iface=None*)
Removes an interface from a bridge

CLI Example:

```
salt '*' bridge.delif br0 eth0
```

salt.modules.bridge.**find_interfaces**(**args*)
Returns the bridge to which the interfaces are bond to

CLI Example:

```
salt '*' bridge.find_interfaces eth0 [eth1...]
```

salt.modules.bridge.**interfaces**(*br=None*)
Returns interfaces attached to a bridge

CLI Example:

```
salt '*' bridge.interfaces br0
```

salt.modules.bridge.**list_**()
Returns the machine's bridges list

CLI Example:

```
salt '*' bridge.list
```

salt.modules.bridge.**show**(*br=None*)
Returns bridges interfaces along with enslaved physical interfaces. If no interface is given, all bridges are shown, else only the specified bridge values are returned.

CLI Example:

```
salt '*' bridge.show
salt '*' bridge.show br0
```

salt.modules.bridge.**stp**(*br=None*, *state='disable'*, *iface=None*)
Sets Spanning Tree Protocol state for a bridge

CLI Example:

```
salt '*' bridge.stp br0 enable
salt '*' bridge.stp br0 disable
```

For BSD-like operating systems, it is required to add the interface on which to enable the STP.

CLI Example:

```
salt '*' bridge.stp bridge0 enable fxp0
salt '*' bridge.stp bridge0 disable fxp0
```

22.16.27 salt.modules.bsd_shadow

Manage the password database on BSD systems

salt.modules.bsd_shadow.**default_hash**()
Returns the default hash used for unset passwords

CLI Example:

```
salt '*' shadow.default_hash
```

salt.modules.bsd_shadow.**info**(*name*)
Return information for the specified user

CLI Example:

```
salt '*' shadow.info someuser
```

salt.modules.bsd_shadow.**set_change**(*name*, *change*)
Sets the time at which the password expires (in seconds since the EPOCH). See man usermod on NetBSD and OpenBSD or man pw on FreeBSD. ``0'' means the password never expires.

CLI Example:

```
salt '*' shadow.set_change username 1419980400
```

salt.modules.bsd_shadow.**set_expire**(*name*, *expire*)
Sets the time at which the account expires (in seconds since the EPOCH). See man usermod on NetBSD and OpenBSD or man pw on FreeBSD. ``0'' means the account never expires.

CLI Example:

```
salt '*' shadow.set_expire username 1419980400
```

salt.modules.bsd_shadow.**set_password**(*name*, *password*)
Set the password for a named user. The password must be a properly defined hash. The password hash can be generated with this command:

```
python -c "import crypt; print crypt.crypt('password', ciphersalt)"
```

NOTE: When constructing the `ciphersalt` string, you must escape any dollar signs, to avoid them being interpolated by the shell.

`'password'` is, of course, the password for which you want to generate a hash.

`ciphersalt` is a combination of a cipher identifier, an optional number of rounds, and the cryptographic salt. The arrangement and format of these fields depends on the cipher and which flavor of BSD you are using.

For more information on this, see the manpage for `crpyt(3)`. On NetBSD, additional information is available in `passwd.conf(5)`.

It is important to make sure that a supported cipher is used.

CLI Example:

```
salt '*' shadow.set_password someuser '$1$UYCIxa628.9qXjpQCjM4a..'
```

22.16.28 salt.modules.btrfs

Module for managing BTRFS file systems.

`salt.modules.btrfs.add(mountpoint, *devices, **kwargs)`
Add a devices to a BTRFS filesystem.

General options:

- **nodiscard**: Do not perform whole device TRIM
- **force**: Force overwrite existing filesystem on the disk

CLI Example:

```
salt '*' btrfs.add /mountpoint /dev/sda1 /dev/sda2
```

`salt.modules.btrfs.convert(device, permanent=False, keeplf=False)`
Convert ext2/3/4 to BTRFS. Device should be mounted.

Filesystem can be converted temporarily so the further processing and rollback is possible, or permanently, where previous extended filesystem image gets deleted. Please note, permanent conversion takes a while as BTRFS filesystem needs to be properly rebalanced afterwards.

General options:

- **permanent**: Specify if the migration should be permanent (false by default)
- **keeplf**: Keep **`lost+found`** of the partition (removed by default, but still in the image, if not permanent migration)

CLI Example:

```
salt '*' btrfs.convert /dev/sda1
salt '*' btrfs.convert /dev/sda1 permanent=True
```

`salt.modules.btrfs.defragment(path)`
Defragment mounted BTRFS filesystem. In order to defragment a filesystem, device should be properly mounted and writable.

If passed a device name, then defragmented whole filesystem, mounted on in. If passed a moun tpoint of the filesystem, then only this mount point is defragmented.

CLI Example:

```
salt '*' btrfs.defragment /dev/sda1
salt '*' btrfs.defragment /path/on/filesystem
```

`salt.modules.btrfs.delete(mountpoint, *devices, **kwargs)`
Remove devices from a BTRFS filesystem.

CLI Example:

```
salt '*' btrfs.delete /mountpoint /dev/sda1 /dev/sda2
```

salt.modules.btrfs.**devices**()

Get known BTRFS formatted devices on the system.

CLI Example:

```
salt '*' btrfs.devices
```

salt.modules.btrfs.**features**()

List currently available BTRFS features.

CLI Example:

```
salt '*' btrfs.mkfs_features
```

salt.modules.btrfs.**info**(*device*)

Get BTRFS filesystem information.

CLI Example:

```
salt '*' btrfs.info /dev/sda1
```

salt.modules.btrfs.**mkfs**(**devices*, ***kwargs*)

Create a file system on the specified device. By default wipes out with force.

General options:

- **allocsize**: Specify the BTRFS offset from the start of the device.
- **bytecount**: Specify the size of the resultant filesystem.
- **nodesize**: Node size.
- **leafsize**: Specify the nodesize, the tree block size in which btrfs stores data.
- **noforce**: Prevent force overwrite when an existing filesystem is detected on the device.
- **sectorsize**: Specify the sectorsize, the minimum data block allocation unit.
- **nodiscard**: Do not perform whole device TRIM operation by default.
- **uuid**: Pass UUID or pass True to generate one.

Options:

- **dto**: **(raid0|raid1|raid5|raid6|raid10|single|dup)** Specify how the data must be spanned across the devices specified.
- **mto**: **(raid0|raid1|raid5|raid6|raid10|single|dup)** Specify how metadata must be spanned across the devices specified.
- **fts**: Features (call `salt <host> btrfs.features` for full list of available features)

See the `mkfs.btrfs(8)` manpage for a more complete description of corresponding options description.

CLI Example:

```
salt '*' btrfs.mkfs /dev/sda1
salt '*' btrfs.mkfs /dev/sda1 noforce=True
```

salt.modules.btrfs.**properties**(*obj*, *type=None*, *set=None*)

List properties for given btrfs object. The object can be path of BTRFS device, mount point, or any directories/files inside the BTRFS filesystem.

General options:

•**type**: Possible types are s[ubvol], f[ilesystem], i[node] and d[evice].

•**force**: Force overwrite existing filesystem on the disk

•**set**: <key=value,key1=value1...> Options for a filesystem properties.

CLI Example:

```
salt '*' btrfs.properties /mountpoint
salt '*' btrfs.properties /dev/sda1 type=subvol set='ro=false,label="My Storage"'
```

salt.modules.btrfs.**resize**(*mountpoint*, *size*)

Resize filesystem.

General options:

•**mountpoint**: Specify the BTRFS mountpoint to resize.

•**size**: ([+/-]<newsize>[kKmMgGtTpPeE]|max) Specify the new size of the target.

CLI Example:

```
salt '*' btrfs.resize /mountpoint size=+1g
salt '*' btrfs.resize /dev/sda1 size=max
```

salt.modules.btrfs.**usage**(*path*)

Show in which disk the chunks are allocated.

CLI Example:

```
salt '*' btrfs.usage /your/mountpoint
```

salt.modules.btrfs.**version**()

Return BTRFS version.

CLI Example:

```
salt '*' btrfs.version
```

22.16.29 salt.modules.cassandra

Cassandra NoSQL Database Module

depends

- pycassa Cassandra Python adapter

configuration The location of the `nodetool' command, host, and thrift port needs to be specified via pillar:

```
cassandra.nodetool: /usr/local/bin/nodetool
cassandra.host: localhost
cassandra.thrift_port: 9160
```

salt.modules.cassandra.**column_families**(*keyspace=None*)

Return existing column families for all keyspaces or just the provided one.

CLI Example:

```
salt '*' cassandra.column_families
salt '*' cassandra.column_families <keyspace>
```

salt.modules.cassandra.**column_family_definition**(*keyspace=None*, *column_family=None*)

Return a dictionary of column family definitions for the given keyspace/column_family

CLI Example:

```
salt '*' cassandra.column_family_definition <keyspace> <column_family>
```

salt.modules.cassandra.**compactionstats**()

Return compactionstats info

CLI Example:

```
salt '*' cassandra.compactionstats
```

salt.modules.cassandra.**info**()

Return cassandra node info

CLI Example:

```
salt '*' cassandra.info
```

salt.modules.cassandra.**keyspaces**()

Return existing keyspaces

CLI Example:

```
salt '*' cassandra.keyspaces
```

salt.modules.cassandra.**netstats**()

Return netstats info

CLI Example:

```
salt '*' cassandra.netstats
```

salt.modules.cassandra.**ring**()

Return cassandra ring info

CLI Example:

```
salt '*' cassandra.ring
```

salt.modules.cassandra.**tpstats**()

Return tpstats info

CLI Example:

```
salt '*' cassandra.tpstats
```

salt.modules.cassandra.**version**()

Return the cassandra version

CLI Example:

```
salt '*' cassandra.version
```

22.16.30 salt.modules.chef

Execute chef in server or solo mode

salt.modules.chef.**client**(*whyrun=False*, *localmode=False*, *logfile=None*, ***kwargs*)
Execute a chef client run and return a dict with the stderr, stdout, return code, and pid.

CLI Example:

```
salt '*' chef.client server=https://localhost
```

server The chef server URL

client_key Set the client key file location

config The configuration file to use

config-file-jail Directory under which config files are allowed to be loaded (no client.rb or knife.rb outside this path will be loaded).

environment Set the Chef Environment on the node

group Group to set privilege to

json-attributes Load attributes from a JSON file or URL

localmode Point chef-client at local repository if True

log_level Set the log level (debug, info, warn, error, fatal)

logfile Set the log file location

node-name The node name for this client

override-runlist Replace current run list with specified items for a single run

pid Set the PID file location, defaults to /tmp/chef-client.pid

run-lock-timeout Set maximum duration to wait for another client run to finish, default is indefinitely.

runlist Permanently replace current run list with specified items

user User to set privilege to

validation_key Set the validation key file location, used for registering new clients

whyrun Enable whyrun mode when set to True

salt.modules.chef.**solo**(*whyrun=False*, *logfile=None*, ***kwargs*)
Execute a chef solo run and return a dict with the stderr, stdout, return code, and pid.

CLI Example:

```
salt '*' chef.solo override-runlist=test
```

config The configuration file to use

environment Set the Chef Environment on the node

group Group to set privilege to

json-attributes Load attributes from a JSON file or URL

log_level Set the log level (debug, info, warn, error, fatal)

logfile Set the log file location

node-name The node name for this client

override-runlist Replace current run list with specified items for a single run

recipe-url Pull down a remote gzipped tarball of recipes and untar it to the cookbook cache

run-lock-timeout Set maximum duration to wait for another client run to finish, default is indefinitely.

user User to set privilege to

whyrun Enable whyrun mode when set to True

22.16.31 salt.modules.chocolatey

A dead simple module wrapping calls to the Chocolatey package manager (http://chocolatey.org)

New in version 2014.1.0.

`salt.modules.chocolatey.`**`bootstrap`**(*force=False*)

Download and install the latest version of the Chocolatey package manager via the official bootstrap.

Chocolatey requires Windows PowerShell and the .NET v4.0 runtime. Depending on the host's version of Windows, chocolatey.bootstrap will attempt to ensure these prerequisites are met by downloading and executing the appropriate installers from Microsoft.

Note that if PowerShell is installed, you may have to restart the host machine for Chocolatey to work.

force Run the bootstrap process even if Chocolatey is found in the path.

CLI Example:

```
salt '*' chocolatey.bootstrap
salt '*' chocolatey.bootstrap force=True
```

`salt.modules.chocolatey.`**`chocolatey_version`**()

New in version 2014.7.0.

Returns the version of Chocolatey installed on the minion.

CLI Example:

```
salt '*' chocolatey.chocolatey_version
```

`salt.modules.chocolatey.`**`install`**(*name*, *version=None*, *source=None*, *force=False*)

Instructs Chocolatey to install a package.

name The name of the package to be installed. Only accepts a single argument.

version Install a specific version of the package. Defaults to latest version.

source Chocolatey repository (directory, share or remote URL feed) the package comes from. Defaults to the official Chocolatey feed.

force Reinstall the current version of an existing package.

CLI Example:

```
salt '*' chocolatey.install <package name>
salt '*' chocolatey.install <package name> version=<package version>
```

`salt.modules.chocolatey.`**`install_cygwin`**(*name*)

Instructs Chocolatey to install a package via Cygwin.

name The name of the package to be installed. Only accepts a single argument.

CLI Example:

```
salt '*' chocolatey.install_cygwin <package name>
```

salt.modules.chocolatey.**install_gem**(*name*, *version=None*)
Instructs Chocolatey to install a package via Ruby's Gems.

name The name of the package to be installed. Only accepts a single argument.

version Install a specific version of the package. Defaults to latest version available.

CLI Example:

```
salt '*' chocolatey.install_gem <package name>
salt '*' chocolatey.install_gem <package name> version=<package version>
```

salt.modules.chocolatey.**install_missing**(*name*, *version=None*, *source=None*)
Instructs Chocolatey to install a package if it doesn't already exist.

Changed in version 2014.7.0: If the minion has Chocolatey >= 0.9.8.24 installed, this function calls `chocolatey.install` instead, as `installmissing` is deprecated as of that version and will be removed in Chocolatey 1.0.

name The name of the package to be installed. Only accepts a single argument.

version Install a specific version of the package. Defaults to latest version available.

source Chocolatey repository (directory, share or remote URL feed) the package comes from. Defaults to the official Chocolatey feed.

CLI Example:

```
salt '*' chocolatey.install_missing <package name>
salt '*' chocolatey.install_missing <package name> version=<package version>
```

salt.modules.chocolatey.**install_python**(*name*, *version=None*)
Instructs Chocolatey to install a package via Python's easy_install.

name The name of the package to be installed. Only accepts a single argument.

version Install a specific version of the package. Defaults to latest version available.

CLI Example:

```
salt '*' chocolatey.install_python <package name>
salt '*' chocolatey.install_python <package name> version=<package version>
```

salt.modules.chocolatey.**install_webpi**(*name*)
Instructs Chocolatey to install a package via the Microsoft Web PI service.

name The name of the package to be installed. Only accepts a single argument.

CLI Example:

```
salt '*' chocolatey.install_webpi <package name>
```

salt.modules.chocolatey.**install_windowsfeatures**(*name*)
Instructs Chocolatey to install a Windows Feature via the Deployment Image Servicing and Management tool.

name The name of the feature to be installed. Only accepts a single argument.

CLI Example:

```
salt '*' chocolatey.install_windowsfeatures <package name>
```

salt.modules.chocolatey.**list_**(*narrow*, *all_versions=False*, *pre_versions=False*, *source=None*)
Instructs Chocolatey to pull a vague package list from the repository.

narrow Term used to narrow down results. Searches against name/description/tag.

all_versions Display all available package versions in results. Defaults to False.

pre_versions Display pre-release packages in results. Defaults to False.

source Chocolatey repository (directory, share or remote URL feed) the package comes from. Defaults to the official Chocolatey feed.

CLI Example:

```
salt '*' chocolatey.list <narrow>
salt '*' chocolatey.list <narrow> all_versions=True
```

salt.modules.chocolatey.**list_webpi**()

Instructs Chocolatey to pull a full package list from the Microsoft Web PI repository.

CLI Example:

```
salt '*' chocolatey.list_webpi
```

salt.modules.chocolatey.**list_windowsfeatures**()

Instructs Chocolatey to pull a full package list from the Windows Features list, via the Deployment Image Servicing and Management tool.

CLI Example:

```
salt '*' chocolatey.list_windowsfeatures
```

salt.modules.chocolatey.**uninstall**(*name*, *version=None*)

Instructs Chocolatey to uninstall a package.

name The name of the package to be uninstalled. Only accepts a single argument.

version Uninstalls a specific version of the package. Defaults to latest version installed.

CLI Example:

```
salt '*' chocolatey.uninstall <package name>
salt '*' chocolatey.uninstall <package name> version=<package version>
```

salt.modules.chocolatey.**update**(*name*, *source=None*, *pre_versions=False*)

Instructs Chocolatey to update packages on the system.

name The name of the package to update, or ``all'' to update everything installed on the system.

source Chocolatey repository (directory, share or remote URL feed) the package comes from. Defaults to the official Chocolatey feed.

pre_versions Include pre-release packages in comparison. Defaults to False.

CLI Example:

```
salt "*" chocolatey.update all
salt "*" chocolatey.update <package name> pre_versions=True
```

salt.modules.chocolatey.**version**(*name*, *check_remote=False*, *source=None*, *pre_versions=False*)

Instructs Chocolatey to check an installed package version, and optionally compare it to one available from a remote feed.

name The name of the package to check.

check_remote Get the version number of the latest package from the remote feed. Defaults to False.

source Chocolatey repository (directory, share or remote URL feed) the package comes from. Defaults to the official Chocolatey feed.

pre_versions Include pre-release packages in comparison. Defaults to False.

CLI Example:

```
salt "*" chocolatey.version <package name>
salt "*" chocolatey.version <package name> check_remote=True
```

22.16.32 salt.modules.cloud

Salt-specific interface for calling Salt Cloud directly

`salt.modules.cloud.`**`action`**(*fun=None*, *cloudmap=None*, *names=None*, *provider=None*, *instance=None*, ***kwargs*)

Execute a single action on the given provider/instance

CLI Example:

```
salt '*' cloud.action start instance=myinstance
salt '*' cloud.action stop instance=myinstance
salt '*' cloud.action show_image provider=my-ec2-config image=ami-1624987f
```

`salt.modules.cloud.`**`create`**(*provider*, *names*, ***kwargs*)

Create an instance using Salt Cloud

CLI Example:

```
salt minionname cloud.create my-ec2-config myinstance image=ami-1624987f size='t1.micro' ssh_use
```

`salt.modules.cloud.`**`destroy`**(*names*)

Destroy the named VM(s)

CLI Example:

```
salt '*' cloud.destroy myinstance
```

`salt.modules.cloud.`**`full_query`**(*query_type='list_nodes_full'*)

List all available cloud provider data

CLI Example:

```
salt '*' cloud.full_query
```

`salt.modules.cloud.`**`get_instance`**(*name*, *provider=None*)

Return details on an instance.

Similar to the cloud action show_instance but returns only the instance details.

CLI Example:

```
salt '*' cloud.get_instance myinstance
```

SLS Example:

```
{{ salt['cloud.get_instance']('myinstance')['mac_address'] }}
```

`salt.modules.cloud.`**`has_instance`**(*name*, *provider=None*)

Return true if the instance is found on a provider

CLI Example:

```
salt '*' cloud.has_instance myinstance
```

salt.modules.cloud.**list_images**(*provider='all'*)
List cloud provider images for the given providers

CLI Example:

```
salt '*' cloud.list_images my-gce-config
```

salt.modules.cloud.**list_locations**(*provider='all'*)
List cloud provider locations for the given providers

CLI Example:

```
salt '*' cloud.list_locations my-gce-config
```

salt.modules.cloud.**list_sizes**(*provider='all'*)
List cloud provider sizes for the given providers

CLI Example:

```
salt '*' cloud.list_sizes my-gce-config
```

salt.modules.cloud.**network_create**(*provider*, *names*, ***kwargs*)
Create private network

CLI Example:

```
salt minionname cloud.network_create my-nova names=['salt'] cidr='192.168.100.0/24'
```

salt.modules.cloud.**network_list**(*provider*)
List private networks

CLI Example:

```
salt minionname cloud.network_list my-nova
```

salt.modules.cloud.**profile_**(*profile*, *names*, *vm_overrides=None*, ***kwargs*)
Spin up an instance using Salt Cloud

CLI Example:

```
salt '*' cloud.profile my-gce-config myinstance
```

salt.modules.cloud.**query**(*query_type='list_nodes'*)
List cloud provider data for all providers

CLI Examples:

```
salt '*' cloud.query
salt '*' cloud.query list_nodes_full
salt '*' cloud.query list_nodes_select
```

salt.modules.cloud.**select_query**(*query_type='list_nodes_select'*)
List selected nodes

CLI Example:

```
salt '*' cloud.select_query
```

salt.modules.cloud.**virtual_interface_create**(*provider*, *names*, ***kwargs*)
Attach private interfaces to a server

CLI Example:

```
salt minionname cloud.virtual_interface_create my-nova names=['salt-master'] net_name='salt'
```

salt.modules.cloud.**virtual_interface_list**(*provider*, *names*, ***kwargs*)
List virtual interfaces on a server

CLI Example:

```
salt minionname cloud.virtual_interface_list my-nova names=['salt-master']
```

salt.modules.cloud.**volume_attach**(*provider*, *names*, ***kwargs*)
Attach volume to a server

CLI Example:

```
salt minionname cloud.volume_attach my-nova myblock server_name=myserver device='/dev/xvdf'
```

salt.modules.cloud.**volume_create**(*provider*, *names*, ***kwargs*)
Create volume

CLI Example:

```
salt minionname cloud.volume_create my-nova myblock size=100 voltype=SSD
```

salt.modules.cloud.**volume_delete**(*provider*, *names*, ***kwargs*)
Delete volume

CLI Example:

```
salt minionname cloud.volume_delete my-nova myblock
```

salt.modules.cloud.**volume_detach**(*provider*, *names*, ***kwargs*)
Detach volume from a server

CLI Example:

```
salt minionname cloud.volume_detach my-nova myblock server_name=myserver
```

salt.modules.cloud.**volume_list**(*provider*)
List block storage volumes

CLI Example:

```
salt minionname cloud.volume_list my-nova
```

22.16.33 salt.modules.cmdmod

A module for shelling out

Keep in mind that this module is insecure, in that it can give whomever has access to the master root execution access to all salt minions.

salt.modules.cmdmod.**exec_code**(*lang*, *code*, *cwd=None*)
Pass in two strings, the first naming the executable language, aka - python2, python3, ruby, perl, lua, etc. the second string containing the code you wish to execute. The stdout will be returned.

CLI Example:

```
salt '*' cmd.exec_code ruby 'puts "cheese"'
```

salt.modules.cmdmod.**exec_code_all**(*lang*, *code*, *cwd=None*)

Pass in two strings, the first naming the executable language, aka - python2, python3, ruby, perl, lua, etc. the second string containing the code you wish to execute. All cmd artifacts (stdout, stderr, retcode, pid) will be returned.

CLI Example:

```
salt '*' cmd.exec_code_all ruby 'puts "cheese"'
```

salt.modules.cmdmod.**has_exec**(*cmd*)

Returns true if the executable is available on the minion, false otherwise

CLI Example:

```
salt '*' cmd.has_exec cat
```

salt.modules.cmdmod.**retcode**(*cmd*, *cwd=None*, *stdin=None*, *runas=None*, *shell='/bin/bash'*, *python_shell=None*, *env=None*, *clean_env=False*, *template=None*, *umask=None*, *output_loglevel='debug'*, *timeout=None*, *reset_system_locale=True*, *ignore_retcode=False*, *saltenv='base'*, *use_vt=False*, ***kwargs*)

Execute a shell command and return the command's return code.

Note that `env` represents the environment variables for the command, and should be formatted as a dict, or a YAML string which resolves to a dict.

Return type int

Return type None

Returns Return Code as an int or None if there was an exception.

CLI Example:

```
salt '*' cmd.retcode "file /bin/bash"
```

The template arg can be set to `jinja' or another supported template engine to render the command arguments before execution. For example:

```
salt '*' cmd.retcode template=jinja "file {{grains.pythonpath[0]}}/python"
```

A string of standard input can be specified for the command to be run using the `stdin` parameter. This can be useful in cases where sensitive information must be read from standard input.:

```
salt '*' cmd.retcode "grep f" stdin='one\ntwo\nthree\nfour\nfive\n'
```

salt.modules.cmdmod.**run**(*cmd*, *cwd=None*, *stdin=None*, *runas=None*, *shell='/bin/bash'*, *python_shell=None*, *env=None*, *clean_env=False*, *template=None*, *rstrip=True*, *umask=None*, *output_loglevel='debug'*, *timeout=None*, *reset_system_locale=True*, *ignore_retcode=False*, *saltenv='base'*, *use_vt=False*, ***kwargs*)

Execute the passed command and return the output as a string

Note that `env` represents the environment variables for the command, and should be formatted as a dict, or a YAML string which resolves to a dict.

Warning: Warning This function does not process commands through a shell unless the python_shell flag is set to True. This means that any shell-specific functionality such as `echo' or the use of pipes, redirection or &&, should either be migrated to cmd.shell or have the python_shell=True flag set here.
The use of python_shell=True means that the shell will accept _any_ input including potentially malicious commands such as `good_command;rm -rf /'. Be absolutely certain that you have sanitized your input prior to using python_shell=True

CLI Example:

```
salt '*' cmd.run "ls -l | awk '/foo/{print \$2}'"
```

The template arg can be set to `jinja' or another supported template engine to render the command arguments before execution. For example:

```
salt '*' cmd.run template=jinja "ls -l /tmp/{{grains.id}} | awk '/foo/{print \$2}'"
```

Specify an alternate shell with the shell parameter:

```
salt '*' cmd.run "Get-ChildItem C:\ " shell='powershell'
```

A string of standard input can be specified for the command to be run using the `stdin` parameter. This can be useful in cases where sensitive information must be read from standard input.:

```
salt '*' cmd.run "grep f" stdin='one\ntwo\nthree\nfour\nfive\n'
```

If an equal sign (`=`) appears in an argument to a Salt command it is interpreted as a keyword argument in the format `key=val`. That processing can be bypassed in order to pass an equal sign through to the remote shell command by manually specifying the kwarg:

```
salt '*' cmd.run cmd='sed -e s/=/:/g'
```

`salt.modules.cmdmod.`**`run_all`**(*cmd*, *cwd=None*, *stdin=None*, *runas=None*, *shell='/bin/bash'*, *python_shell=None*, *env=None*, *clean_env=False*, *template=None*, *rstrip=True*, *umask=None*, *output_loglevel='debug'*, *timeout=None*, *reset_system_locale=True*, *ignore_retcode=False*, *saltenv='base'*, *use_vt=False*, ***kwargs*)

Execute the passed command and return a dict of return data

Note that `env` represents the environment variables for the command, and should be formatted as a dict, or a YAML string which resolves to a dict.

CLI Example:

```
salt '*' cmd.run_all "ls -l | awk '/foo/{print \$2}'"
```

The template arg can be set to `jinja' or another supported template engine to render the command arguments before execution. For example:

```
salt '*' cmd.run_all template=jinja "ls -l /tmp/{{grains.id}} | awk '/foo/{print \$2}'"
```

A string of standard input can be specified for the command to be run using the `stdin` parameter. This can be useful in cases where sensitive information must be read from standard input.:

```
salt '*' cmd.run_all "grep f" stdin='one\ntwo\nthree\nfour\nfive\n'
```

salt.modules.cmdmod.**run_chroot**(*root*, *cmd*, *cwd=None*, *stdin=None*, *runas=None*, *shell='/bin/bash'*, *python_shell=True*, *env=None*, *clean_env=False*, *template=None*, *rstrip=True*, *umask=None*, *output_loglevel='quiet'*, *quiet=False*, *timeout=None*, *reset_system_locale=True*, *ignore_retcode=False*, *saltenv='base'*, *use_vt=False*, ***kwargs*)

New in version 2014.7.0.

This function runs `cmd.run_all` wrapped within a chroot, with dev and proc mounted in the chroot

stdin [None] Standard input to be used for the command

New in version 2014.7.1.

output_loglevel [debug] Level at which to log the output from the command. Set to `quiet` to suppress logging.

New in version 2014.7.1.

CLI Example:

```
salt '*' cmd.run_chroot /var/lib/lxc/container_name/rootfs 'sh /tmp/bootstrap.sh'
```

salt.modules.cmdmod.**run_stderr**(*cmd*, *cwd=None*, *stdin=None*, *runas=None*, *shell='/bin/bash'*, *python_shell=None*, *env=None*, *clean_env=False*, *template=None*, *rstrip=True*, *umask=None*, *output_loglevel='debug'*, *timeout=None*, *reset_system_locale=True*, *ignore_retcode=False*, *saltenv='base'*, *use_vt=False*, ***kwargs*)

Execute a command and only return the standard error

Note that `env` represents the environment variables for the command, and should be formatted as a dict, or a YAML string which resolves to a dict.

CLI Example:

```
salt '*' cmd.run_stderr "ls -l | awk '/foo/{print \$2}'"
```

The template arg can be set to `jinja' or another supported template engine to render the command arguments before execution. For example:

```
salt '*' cmd.run_stderr template=jinja "ls -l /tmp/{{grains.id}} | awk '/foo/{print \$2}'"
```

A string of standard input can be specified for the command to be run using the `stdin` parameter. This can be useful in cases where sensitive information must be read from standard input.:

```
salt '*' cmd.run_stderr "grep f" stdin='one\ntwo\nthree\nfour\nfive\n'
```

salt.modules.cmdmod.**run_stdout**(*cmd*, *cwd=None*, *stdin=None*, *runas=None*, *shell='/bin/bash'*, *python_shell=None*, *env=None*, *clean_env=False*, *template=None*, *rstrip=True*, *umask=None*, *output_loglevel='debug'*, *timeout=None*, *reset_system_locale=True*, *ignore_retcode=False*, *saltenv='base'*, *use_vt=False*, ***kwargs*)

Execute a command, and only return the standard out

Note that `env` represents the environment variables for the command, and should be formatted as a dict, or a YAML string which resolves to a dict.

CLI Example:

```
salt '*' cmd.run_stdout "ls -l | awk '/foo/{print \$2}'"
```

The template arg can be set to `jinja' or another supported template engine to render the command arguments before execution. For example:

```
salt '*' cmd.run_stdout template=jinja "ls -l /tmp/{{grains.id}} | awk '/foo/{print \$2}'"
```

A string of standard input can be specified for the command to be run using the `stdin` parameter. This can be useful in cases where sensitive information must be read from standard input.:

```
salt '*' cmd.run_stdout "grep f" stdin='one\ntwo\nthree\nfour\nfive\n'
```

salt.modules.cmdmod.**script**(*source*, *args=None*, *cwd=None*, *stdin=None*, *runas=None*, *shell='/bin/bash'*, *python_shell=None*, *env=None*, *template=None*, *umask=None*, *output_loglevel='debug'*, *quiet=False*, *timeout=None*, *reset_system_locale=True*, *__env__=None*, *saltenv='base'*, *use_vt=False*, ***kwargs*)

Download a script from a remote location and execute the script locally. The script can be located on the salt master file server or on an HTTP/FTP server.

The script will be executed directly, so it can be written in any available programming language.

The script can also be formatted as a template, the default is jinja. Arguments for the script can be specified as well.

CLI Example:

```
salt '*' cmd.script salt://scripts/runme.sh
salt '*' cmd.script salt://scripts/runme.sh 'arg1 arg2 "arg 3"'
salt '*' cmd.script salt://scripts/windows_task.ps1 args=' -Input c:\tmp\infile.txt' shell='powe
```

A string of standard input can be specified for the command to be run using the `stdin` parameter. This can be useful in cases where sensitive information must be read from standard input.:

```
salt '*' cmd.script salt://scripts/runme.sh stdin='one\ntwo\nthree\nfour\nfive\n'
```

salt.modules.cmdmod.**script_retcode**(*source*, *args=None*, *cwd=None*, *stdin=None*, *runas=None*, *shell='/bin/bash'*, *python_shell=None*, *env=None*, *template='jinja'*, *umask=None*, *timeout=None*, *reset_system_locale=True*, *__env__=None*, *saltenv='base'*, *output_loglevel='debug'*, *use_vt=False*, ***kwargs*)

Download a script from a remote location and execute the script locally. The script can be located on the salt master file server or on an HTTP/FTP server.

The script will be executed directly, so it can be written in any available programming language.

The script can also be formatted as a template, the default is jinja.

Only evaluate the script return code and do not block for terminal output

CLI Example:

```
salt '*' cmd.script_retcode salt://scripts/runme.sh
salt '*' cmd.script_retcode salt://scripts/runme.sh 'arg1 arg2 "arg 3"'
salt '*' cmd.script_retcode salt://scripts/windows_task.ps1 args=' -Input c:\tmp\infile.txt' she
```

A string of standard input can be specified for the command to be run using the `stdin` parameter. This can be useful in cases where sensitive information must be read from standard input.:

```
salt '*' cmd.script_retcode salt://scripts/runme.sh stdin='one\ntwo\nthree\nfour\nfive\n'
```

salt.modules.cmdmod.**shell**(*cmd*, *cwd=None*, *stdin=None*, *runas=None*, *shell='/bin/bash'*, *env=None*, *clean_env=False*, *template=None*, *rstrip=True*, *umask=None*, *output_loglevel='debug'*, *quiet=False*, *timeout=None*, *reset_system_locale=True*, *ignore_retcode=False*, *saltenv='base'*, *use_vt=False*, ***kwargs*)

Execute the passed command and return the output as a string.

New in version 2015.5.0.

Warning: Warning This passes the cmd argument directly to the shell without any further processing! Be absolutely sure that you have properly santized the command passed to this function and do not use untrusted inputs.
Note that `env` represents the environment variables for the command, and should be formatted as a dict, or a YAML string which resolves to a dict.

CLI Example:

```
salt '*' cmd.shell "ls -l | awk '/foo/{print \$2}'"
```

The template arg can be set to `jinja' or another supported template engine to render the command arguments before execution. For example:

```
salt '*' cmd.shell template=jinja "ls -l /tmp/{{grains.id}} | awk '/foo/{print \$2}'"
```

Specify an alternate shell with the shell parameter:

```
salt '*' cmd.shell "Get-ChildItem C:\ " shell='powershell'
```

A string of standard input can be specified for the command to be run using the `stdin` parameter. This can be useful in cases where sensitive information must be read from standard input.:

```
salt '*' cmd.shell "grep f" stdin='one\ntwo\nthree\nfour\nfive\n'
```

If an equal sign (`=`) appears in an argument to a Salt command it is interpreted as a keyword argument in the format `key=val`. That processing can be bypassed in order to pass an equal sign through to the remote shell command by manually specifying the kwarg:

```
salt '*' cmd.shell cmd='sed -e s/=/:/g'
```

`salt.modules.cmdmod.`**`shells`**`()`
Lists the valid shells on this system via the /etc/shells file

New in version 2015.5.0.

CLI Example:

```
salt '*' cmd.shells
```

`salt.modules.cmdmod.`**`tty`**`(`*device*, *echo=None*`)`
Echo a string to a specific tty

CLI Example:

```
salt '*' cmd.tty tty0 'This is a test'
salt '*' cmd.tty pts3 'This is a test'
```

`salt.modules.cmdmod.`**`which`**`(`*cmd*`)`
Returns the path of an executable available on the minion, None otherwise

CLI Example:

```
salt '*' cmd.which cat
```

`salt.modules.cmdmod.`**`which_bin`**`(`*cmds*`)`
Returns the first command found in a list of commands

CLI Example:

```
salt '*' cmd.which_bin '[pip2, pip, pip-python]'
```

22.16.34 salt.modules.composer

Use composer to install PHP dependencies for a directory

salt.modules.composer.**did_composer_install**(*dir*)

Test to see if the vendor directory exists in this directory

dir Directory location of the composer.json file

CLI Example:

```
salt '*' composer.did_composer_install /var/www/application
```

salt.modules.composer.**install**(*dir*, *composer=None*, *php=None*, *runas=None*, *prefer_source=None*, *prefer_dist=None*, *no_scripts=None*, *no_plugins=None*, *optimize=None*, *no_dev=None*, *quiet=False*, *composer_home='/root'*)

Install composer dependencies for a directory.

If composer has not been installed globally making it available in the system PATH & making it executable, the `composer` and `php` parameters will need to be set to the location of the executables.

dir Directory location of the composer.json file.

composer Location of the composer.phar file. If not set composer will just execute ``composer'' as if it is installed globally. (i.e. /path/to/composer.phar)

php Location of the php executable to use with composer. (i.e. /usr/bin/php)

runas Which system user to run composer as.

prefer_source --prefer-source option of composer.

prefer_dist --prefer-dist option of composer.

no_scripts --no-scripts option of composer.

no_plugins --no-plugins option of composer.

optimize --optimize-autoloader option of composer. Recommended for production.

no_dev --no-dev option for composer. Recommended for production.

quiet --quiet option for composer. Whether or not to return output from composer.

composer_home $COMPOSER_HOME environment variable

CLI Example:

```
salt '*' composer.install /var/www/application

salt '*' composer.install /var/www/application              no_dev=True optimize=True
```

salt.modules.composer.**selfupdate**(*composer=None*, *php=None*, *runas=None*, *quiet=False*, *composer_home='/root'*)

Update composer itself.

If composer has not been installed globally making it available in the system PATH & making it executable, the `composer` and `php` parameters will need to be set to the location of the executables.

composer Location of the composer.phar file. If not set composer will just execute ``composer'' as if it is installed globally. (i.e. /path/to/composer.phar)

php Location of the php executable to use with composer. (i.e. /usr/bin/php)

runas Which system user to run composer as.

quiet --quiet option for composer. Whether or not to return output from composer.

composer_home $COMPOSER_HOME environment variable

CLI Example:

```
salt '*' composer.selfupdate
```

`salt.modules.composer.`**`update`**(*dir*, *composer=None*, *php=None*, *runas=None*, *prefer_source=None*, *prefer_dist=None*, *no_scripts=None*, *no_plugins=None*, *optimize=None*, *no_dev=None*, *quiet=False*, *composer_home='/root'*)

Update composer dependencies for a directory.

If *composer install* has not yet been run, this runs *composer install* instead.

If composer has not been installed globally making it available in the system PATH & making it executable, the `composer` and `php` parameters will need to be set to the location of the executables.

dir Directory location of the composer.json file.

composer Location of the composer.phar file. If not set composer will just execute ``composer'' as if it is installed globally. (i.e. /path/to/composer.phar)

php Location of the php executable to use with composer. (i.e. /usr/bin/php)

runas Which system user to run composer as.

prefer_source --prefer-source option of composer.

prefer_dist --prefer-dist option of composer.

no_scripts --no-scripts option of composer.

no_plugins --no-plugins option of composer.

optimize --optimize-autoloader option of composer. Recommended for production.

no_dev --no-dev option for composer. Recommended for production.

quiet --quiet option for composer. Whether or not to return output from composer.

composer_home $COMPOSER_HOME environment variable

CLI Example:

```
salt '*' composer.update /var/www/application

salt '*' composer.update /var/www/application            no_dev=True optimize=True
```

22.16.35 salt.modules.config

Return config information

`salt.modules.config.`**`backup_mode`**(*backup=''*)

Return the backup mode

CLI Example:

```
salt '*' config.backup_mode
```

salt.modules.config.**dot_vals**(*value*)

Pass in a configuration value that should be preceded by the module name and a dot, this will return a list of all read key/value pairs

CLI Example:

```
salt '*' config.dot_vals host
```

salt.modules.config.**gather_bootstrap_script**(*bootstrap=None*)

Download the salt-bootstrap script, and return its location

bootstrap URL of alternate bootstrap script

CLI Example:

```
salt '*' config.gather_bootstrap_script
```

salt.modules.config.**get**(*key*, *default=''*, *delimiter=':'*, *merge=None*)

Attempt to retrieve the named value from the minion config file, pillar, grains or the master config. If the named value is not available, return the value specified by `default`. If not specified, the default is an empty string.

Values can also be retrieved from nested dictionaries. Assume the below data structure:

```
{'pkg': {'apache': 'httpd'}}
```

To retrieve the value associated with the `apache` key, in the sub-dictionary corresponding to the `pkg` key, the following command can be used:

```
salt myminion config.get pkg:apache
```

The `:` (colon) is used to represent a nested dictionary level.

Changed in version 2015.5.0: The `delimiter` argument was added, to allow delimiters other than `:` to be used.

This function traverses these data stores in this order:

- Minion config file
- Minion's grains
- Minion's pillar data
- Master config file

delimiter New in version 2015.5.0.

Override the delimiter used to separate nested levels of a data structure.

merge New in version 2015.5.0.

If passed, this parameter will change the behavior of the function so that, instead of traversing each data store above in order and returning the first match, the data stores are first merged together and then searched. The pillar data is merged into the master config data, then the grains are merged, followed by the Minion config data. The resulting data structure is then searched for a match. This allows for configurations to be more flexible.

Note: The merging described above does not mean that grain data will end up in the Minion's pillar

data, or pillar data will end up in the master config data, etc. The data is just combined for the purposes of searching an amalgam of the different data stores.

The supported merge strategies are as follows:

- **recurse** - If a key exists in both dictionaries, and the new value is not a dictionary, it is replaced. Otherwise, the sub-dictionaries are merged together into a single dictionary, recursively on down, following the same criteria. For example:

```
>>> dict1 = {'foo': {'bar': 1, 'qux': True},
             'hosts': ['a', 'b', 'c'],
             'only_x': None}
>>> dict2 = {'foo': {'baz': 2, 'qux': False},
             'hosts': ['d', 'e', 'f'],
             'only_y': None}
>>> merged
{'foo': {'bar': 1, 'baz': 2, 'qux': False},
 'hosts': ['d', 'e', 'f'],
 'only_dict1': None,
 'only_dict2': None}
```

- **overwrite** - If a key exists in the top level of both dictionaries, the new value completely overwrites the old. For example:

```
>>> dict1 = {'foo': {'bar': 1, 'qux': True},
             'hosts': ['a', 'b', 'c'],
             'only_x': None}
>>> dict2 = {'foo': {'baz': 2, 'qux': False},
             'hosts': ['d', 'e', 'f'],
             'only_y': None}
>>> merged
{'foo': {'baz': 2, 'qux': False},
 'hosts': ['d', 'e', 'f'],
 'only_dict1': None,
 'only_dict2': None}
```

CLI Example:

```
salt '*' config.get pkg:apache
salt '*' config.get lxc.container_profile:centos merge=recurse
```

`salt.modules.config.`**`manage_mode`**(*mode*)

Return a mode value, normalized to a string

CLI Example:

```
salt '*' config.manage_mode
```

`salt.modules.config.`**`merge`**(*value*, *default=''*, *omit_opts=False*, *omit_master=False*, *omit_pillar=False*)

Retrieves an option based on key, merging all matches.

Same as `option()` except that it merges all matches, rather than taking the first match.

CLI Example:

```
salt '*' config.merge schedule
```

salt.modules.config.**option**(*value*, *default=''*, *omit_opts=False*, *omit_master=False*, *omit_pillar=False*)

Pass in a generic option and receive the value that will be assigned

CLI Example:

```
salt '*' config.option redis.host
```

salt.modules.config.**valid_fileproto**(*uri*)

Returns a boolean value based on whether or not the URI passed has a valid remote file protocol designation

CLI Example:

```
salt '*' config.valid_fileproto salt://path/to/file
```

22.16.36 salt.modules.container_resource

Common resources for LXC and systemd-nspawn containers

These functions are not designed to be called directly, but instead from the `lxc` and the (future) `nspawn` execution modules.

salt.modules.container_resource.**run**(*name*, *cmd*, *output=None*, *no_start=False*, *stdin=None*, *python_shell=True*, *output_loglevel='debug'*, *ignore_retcode=False*, *use_vt=False*)

Common logic for running shell commands in containers

Requires the full command to be passed to `cmd.run/cmd.run_all`

22.16.37 salt.modules.cp

Minion side functions for salt-cp

salt.modules.cp.**cache_dir**(*path*, *saltenv='base'*, *include_empty=False*, *include_pat=None*, *exclude_pat=None*, *env=None*)

Download and cache everything under a directory from the master

include_pat [None] Glob or regex to narrow down the files cached from the given path. If matching with a regex, the regex must be prefixed with `E@`, otherwise the expression will be interpreted as a glob.

New in version 2014.7.0.

exclude_pat [None] Glob or regex to exclude certain files from being cached from the given path. If matching with a regex, the regex must be prefixed with `E@`, otherwise the expression will be interpreted as a glob.

Note: If used with `include_pat`, files matching this pattern will be excluded from the subset of files defined by `include_pat`.

New in version 2014.7.0.

CLI Examples:

```
salt '*' cp.cache_dir salt://path/to/dir
salt '*' cp.cache_dir salt://path/to/dir include_pat='E@*.py$'
```

salt.modules.cp.**cache_file**(*path*, *saltenv='base'*, *env=None*)

Used to cache a single file on the salt-minion Returns the location of the new cached file on the minion

CLI Example:

```
salt '*' cp.cache_file salt://path/to/file
```

salt.modules.cp.**cache_files**(*paths*, *saltenv='base'*, *env=None*)

Used to gather many files from the master, the gathered files will be saved in the minion cachedir reflective to the paths retrieved from the master.

CLI Example:

```
salt '*' cp.cache_files salt://pathto/file1,salt://pathto/file1
```

salt.modules.cp.**cache_local_file**(*path*)

Cache a local file on the minion in the localfiles cache

CLI Example:

```
salt '*' cp.cache_local_file /etc/hosts
```

salt.modules.cp.**cache_master**(*saltenv='base'*, *env=None*)

Retrieve all of the files on the master and cache them locally

CLI Example:

```
salt '*' cp.cache_master
```

salt.modules.cp.**get_dir**(*path*, *dest*, *saltenv='base'*, *template=None*, *gzip=None*, *env=None*)

Used to recursively copy a directory from the salt master

CLI Example:

```
salt '*' cp.get_dir salt://path/to/dir/ /minion/dest
```

get_dir supports the same template and gzip arguments as get_file.

salt.modules.cp.**get_file**(*path*, *dest*, *saltenv='base'*, *makedirs=False*, *template=None*, *gzip=None*, *env=None*)

Used to get a single file from the salt master

CLI Example:

```
salt '*' cp.get_file salt://path/to/file /minion/dest
```

Template rendering can be enabled on both the source and destination file names like so:

```
salt '*' cp.get_file "salt://{{grains.os}}/vimrc" /etc/vimrc template=jinja
```

This example would instruct all Salt minions to download the vimrc from a directory with the same name as their os grain and copy it to /etc/vimrc

For larger files, the cp.get_file module also supports gzip compression. Because gzip is CPU-intensive, this should only be used in scenarios where the compression ratio is very high (e.g. pretty-printed JSON or YAML files).

Use the *gzip* named argument to enable it. Valid values are 1..9, where 1 is the lightest compression and 9 the heaviest. 1 uses the least CPU on the master (and minion), 9 uses the most.

salt.modules.cp.**get_file_str**(*path*, *saltenv='base'*, *env=None*)

Return the contents of a file from a URL

CLI Example:

```
salt '*' cp.get_file_str salt://my/file
```

salt.modules.cp.**get_template**(*path*, *dest*, *template='jinja'*, *saltenv='base'*, *env=None*, *makedirs=False*, ***kwargs*)

Render a file as a template before setting it down. Warning, order is not the same as in fileclient.cp for non breaking old API.

CLI Example:

```
salt '*' cp.get_template salt://path/to/template /minion/dest
```

salt.modules.cp.**get_url**(*path*, *dest*, *saltenv='base'*, *env=None*)

Used to get a single file from a URL.

The default behaviuor is to write the fetched file to the given destination path. To simply return the text contents instead, set destination to None.

CLI Example:

```
salt '*' cp.get_url salt://my/file /tmp/mine
salt '*' cp.get_url http://www.slashdot.org /tmp/index.html
```

salt.modules.cp.**hash_file**(*path*, *saltenv='base'*, *env=None*)

Return the hash of a file, to get the hash of a file on the salt master file server prepend the path with salt://<file on server> otherwise, prepend the file with / for a local file.

CLI Example:

```
salt '*' cp.hash_file salt://path/to/file
```

salt.modules.cp.**is_cached**(*path*, *saltenv='base'*, *env=None*)

Return a boolean if the given path on the master has been cached on the minion

CLI Example:

```
salt '*' cp.is_cached salt://path/to/file
```

salt.modules.cp.**list_master**(*saltenv='base'*, *prefix=''*, *env=None*)

List all of the files stored on the master

CLI Example:

```
salt '*' cp.list_master
```

salt.modules.cp.**list_master_dirs**(*saltenv='base'*, *prefix=''*, *env=None*)

List all of the directories stored on the master

CLI Example:

```
salt '*' cp.list_master_dirs
```

salt.modules.cp.**list_master_symlinks**(*saltenv='base'*, *prefix=''*, *env=None*)

List all of the symlinks stored on the master

CLI Example:

```
salt '*' cp.list_master_symlinks
```

salt.modules.cp.**list_minion**(*saltenv='base'*, *env=None*)

List all of the files cached on the minion

CLI Example:

```
salt '*' cp.list_minion
```

salt.modules.cp.**list_states**(*saltenv='base'*, *env=None*)
List all of the available state modules in an environment

CLI Example:

```
salt '*' cp.list_states
```

salt.modules.cp.**push**(*path*, *keep_symlinks=False*)
Push a file from the minion up to the master, the file will be saved to the salt master in the master's minion files cachedir (defaults to `/var/cache/salt/master/minions/minion-id/files`)

Since this feature allows a minion to push a file up to the master server it is disabled by default for security purposes. To enable, set `file_recv` to `True` in the master configuration file, and restart the master.

keep_symlinks Keep the path value without resolving its canonical form

CLI Example:

```
salt '*' cp.push /etc/fstab
salt '*' cp.push /etc/system-release keep_symlinks=True
```

salt.modules.cp.**push_dir**(*path*, *glob=None*)
Push a directory from the minion up to the master, the files will be saved to the salt master in the master's minion files cachedir (defaults to `/var/cache/salt/master/minions/minion-id/files`). It also has a glob for matching specific files using globbing.

New in version 2014.7.0.

Since this feature allows a minion to push files up to the master server it is disabled by default for security purposes. To enable, set `file_recv` to `True` in the master configuration file, and restart the master.

CLI Example:

```
salt '*' cp.push /usr/lib/mysql
salt '*' cp.push_dir /etc/modprobe.d/ glob='*.conf'
```

salt.modules.cp.**recv**(*files*, *dest*)
Used with salt-cp, pass the files dict, and the destination.

This function receives small fast copy files from the master via salt-cp. It does not work via the CLI.

22.16.38 salt.modules.cpan

Manage Perl modules using CPAN

New in version 2015.5.0.

salt.modules.cpan.**install**(*module*)
Install a Perl module from CPAN

CLI Example:

```
salt '*' cpan.install Template::Alloy
```

salt.modules.cpan.**list_**()
List installed Perl modules, and the version installed

CLI Example:

```
salt '*' cpan.list
```

salt.modules.cpan.**remove**(*module*, *details=False*)
Attempt to remove a Perl module that was installed from CPAN. Because the `cpan` command doesn't actually support ``uninstall''-like functionality, this function will attempt to do what it can, with what it has from CPAN.

Until this function is declared stable, USE AT YOUR OWN RISK!

CLI Example:

```
salt '*' cpan.remove Old::Package
```

salt.modules.cpan.**show**(*module*)
Show information about a specific Perl module

CLI Example:

```
salt '*' cpan.show Template::Alloy
```

salt.modules.cpan.**show_config**()
Return a dict of CPAN configuration values

CLI Example:

```
salt '*' cpan.show_config
```

22.16.39 salt.modules.cron

Work with cron

salt.modules.cron.**list_tab**(*user*)
Return the contents of the specified user's crontab

CLI Example:

```
salt '*' cron.list_tab root
```

salt.modules.cron.**ls**(*user*)
Return the contents of the specified user's crontab

CLI Example:

```
salt '*' cron.list_tab root
```

salt.modules.cron.**raw_cron**(*user*)
Return the contents of the user's crontab

CLI Example:

```
salt '*' cron.raw_cron root
```

salt.modules.cron.**rm**(*user*, *cmd*, *minute=None*, *hour=None*, *daymonth=None*, *month=None*, *dayweek=None*, *identifier=None*)
Remove a cron job for a specified user. If any of the day/time params are specified, the job will only be removed if the specified params match.

CLI Example:

```
    salt '*' cron.rm_job root /usr/local/weekly
    salt '*' cron.rm_job root /usr/bin/foo dayweek=1
```

salt.modules.cron.**rm_env**(*user*, *name*)

Remove cron environment variable for a specified user.

CLI Example:

```
    salt '*' cron.rm_env root MAILTO
```

salt.modules.cron.**rm_job**(*user*, *cmd*, *minute=None*, *hour=None*, *daymonth=None*, *month=None*, *dayweek=None*, *identifier=None*)

Remove a cron job for a specified user. If any of the day/time params are specified, the job will only be removed if the specified params match.

CLI Example:

```
    salt '*' cron.rm_job root /usr/local/weekly
    salt '*' cron.rm_job root /usr/bin/foo dayweek=1
```

salt.modules.cron.**set_env**(*user*, *name*, *value=None*)

Set up an environment variable in the crontab.

CLI Example:

```
    salt '*' cron.set_env root MAILTO user@example.com
```

salt.modules.cron.**set_job**(*user*, *minute*, *hour*, *daymonth*, *month*, *dayweek*, *cmd*, *comment=None*, *identifier=None*)

Sets a cron job up for a specified user.

CLI Example:

```
    salt '*' cron.set_job root '*' '*' '*' '*' 1 /usr/local/weekly
```

salt.modules.cron.**set_special**(*user*, *special*, *cmd*)

Set up a special command in the crontab.

CLI Example:

```
    salt '*' cron.set_special root @hourly 'echo foobar'
```

salt.modules.cron.**write_cron_file**(*user*, *path*)

Writes the contents of a file to a user's crontab

CLI Example:

```
    salt '*' cron.write_cron_file root /tmp/new_cron
```

salt.modules.cron.**write_cron_file_verbose**(*user*, *path*)

Writes the contents of a file to a user's crontab and return error message on error

CLI Example:

```
    salt '*' cron.write_cron_file_verbose root /tmp/new_cron
```

22.16.40 salt.modules.cyg

Manage cygwin packages.

Module file to accompany the cyg state.

salt.modules.cyg.**check_valid_package**(*package*, *cyg_arch='x86_64'*, *mirrors=None*)
Check if the package is valid on the given mirrors.

salt.modules.cyg.**install**(*packages=None*, *cyg_arch='x86_64'*, *mirrors=None*)
Install one or several packages.

packages [None] The packages to install

cyg_arch [x86_64] Specify the architecture to install the package under Current options are x86 and x86_64

CLI Example:

```
salt '*' cyg.install dos2unix
salt '*' cyg.install dos2unix mirrors=[{'http://mirror': 'http://url/to/public/key}]
```

salt.modules.cyg.**list_**(*package=''*, *cyg_arch='x86_64'*)
List locally installed packages.

package [``] package name to check. else all

cyg_arch : Cygwin architecture to use Options are x86 and x86_64

CLI Example:

```
salt '*' cyg.list
```

salt.modules.cyg.**uninstall**(*packages*, *cyg_arch='x86_64'*, *mirrors=None*)
Uninstall one or several packages.

packages The packages to uninstall.

cyg_arch [x86_64] Specify the architecture to remove the package from Current options are x86 and x86_64

CLI Example:

```
salt '*' cyg.uninstall dos2unix
salt '*' cyg.uninstall dos2unix mirrors=[{'http://mirror': 'http://url/to/public/key}]
```

salt.modules.cyg.**update**(*cyg_arch='x86_64'*, *mirrors=None*)
Update all packages.

cyg_arch [x86_64] Specify the cygwin architecture update Current options are x86 and x86_64

CLI Example:

```
salt '*' cyg.update
salt '*' cyg.update dos2unix mirrors=[{'http://mirror': 'http://url/to/public/key}]
```

22.16.41 salt.modules.daemontools

daemontools service module. This module will create daemontools type service watcher.

This module is compatible with the `service` states, so it can be used to maintain services using the `provider` argument:

```
myservice:
  service.running:
    - provider: daemontools
```

salt.modules.daemontools.**available**(*name*)
Returns `True` if the specified service is available, otherwise returns `False`.

CLI Example:

```
salt '*' daemontools.available foo
```

salt.modules.daemontools.**full_restart**(*name*)
Calls daemontools.restart() function

CLI Example:

```
salt '*' daemontools.full_restart <service name>
```

salt.modules.daemontools.**get_all**()
Return a list of all available services

CLI Example:

```
salt '*' daemontools.get_all
```

salt.modules.daemontools.**missing**(*name*)
The inverse of daemontools.available. Returns `True` if the specified service is not available, otherwise returns `False`.

CLI Example:

```
salt '*' daemontools.missing foo
```

salt.modules.daemontools.**reload_**(*name*)
Wrapper for term()

CLI Example:

```
salt '*' daemontools.reload <service name>
```

salt.modules.daemontools.**restart**(*name*)
Restart service via daemontools. This will stop/start service

CLI Example:

```
salt '*' daemontools.restart <service name>
```

salt.modules.daemontools.**start**(*name*)
Starts service via daemontools

CLI Example:

```
salt '*' daemontools.start <service name>
```

salt.modules.daemontools.**status**(*name*, *sig=None*)
Return the status for a service via daemontools, return pid if running

CLI Example:

```
salt '*' daemontools.status <service name>
```

salt.modules.daemontools.**stop**(*name*)
Stops service via daemontools

CLI Example:

```
salt '*' daemontools.stop <service name>
```

salt.modules.daemontools.**term**(*name*)
Send a TERM to service via daemontools

CLI Example:

```
    salt '*' daemontools.term <service name>
```

22.16.42 salt.modules.darwin_sysctl

Module for viewing and modifying sysctl parameters

salt.modules.darwin_sysctl.**assign**(*name*, *value*)
Assign a single sysctl parameter for this minion

name The name of the sysctl value to edit.

value The sysctl value to apply.

CLI Example:

```
    salt '*' sysctl.assign net.inet.icmp.icmplim 50
```

salt.modules.darwin_sysctl.**get**(*name*)
Return a single sysctl parameter for this minion

name The name of the sysctl value to display.

CLI Example:

```
    salt '*' sysctl.get hw.physmem
```

salt.modules.darwin_sysctl.**persist**(*name*, *value*, *config='/etc/sysctl.conf'*, *apply_change=False*)
Assign and persist a simple sysctl parameter for this minion

name The name of the sysctl value to edit.

value The sysctl value to apply.

config The location of the sysctl configuration file.

apply_change Default is False; Default behavior only creates or edits the sysctl.conf file. If apply is set to True, the changes are applied to the system.

CLI Example:

```
    salt '*' sysctl.persist net.inet.icmp.icmplim 50
    salt '*' sysctl.persist coretemp_load NO config=/etc/sysctl.conf
```

salt.modules.darwin_sysctl.**show**(*config_file=False*)
Return a list of sysctl parameters for this minion

CLI Example:

```
    salt '*' sysctl.show
```

22.16.43 salt.modules.data

Manage a local persistent data structure that can hold any arbitrary data specific to the minion

salt.modules.data.**cas**(*key*, *value*, *old_value*)
Check and set a value in the minion datastore

CLI Example:

```
salt '*' data.cas <key> <value> <old_value>
```

salt.modules.data.**clear**()
Clear out all of the data in the minion datastore, this function is destructive!

CLI Example:

```
salt '*' data.clear
```

salt.modules.data.**dump**(*new_data*)
Replace the entire datastore with a passed data structure

CLI Example:

```
salt '*' data.dump '{'eggs': 'spam'}'
```

salt.modules.data.**getval**(*key*)
Get a value from the minion datastore

CLI Example:

```
salt '*' data.getval <key>
```

salt.modules.data.**getvals**(**keys*)
Get values from the minion datastore

CLI Example:

```
salt '*' data.getvals <key> [<key> ...]
```

salt.modules.data.**load**()
Return all of the data in the minion datastore

CLI Example:

```
salt '*' data.load
```

salt.modules.data.**update**(*key*, *value*)
Update a key with a value in the minion datastore

CLI Example:

```
salt '*' data.update <key> <value>
```

22.16.44 salt.modules.ddns

Support for RFC 2136 dynamic DNS updates.

depends

- dnspython Python module

configuration If you want to use TSIG authentication for the server, there are a couple of optional configuration parameters made available to support this (the keyname is only needed if the keyring contains more than one key):

```
keyring: keyring file (default=None)
keyname: key name in file (default=None)
```

The keyring file needs to be in json format and the key name needs to end with an extra period in the file, similar to this:

```
        {"keyname.": "keycontent"}
```

salt.modules.ddns.**add_host**(*zone*, *name*, *ttl*, *ip*, *nameserver=`127.0.0.1'*, *replace=True*, ***kwargs*)
Add, replace, or update the A and PTR (reverse) records for a host.

CLI Example:

```
salt ns1 ddns.add_host example.com host1 60 10.1.1.1
```

salt.modules.ddns.**delete**(*zone*, *name*, *rdtype=None*, *data=None*, *nameserver=`127.0.0.1'*, ***kwargs*)
Delete a DNS record.

CLI Example:

```
salt ns1 ddns.delete example.com host1 A
```

salt.modules.ddns.**delete_host**(*zone*, *name*, *nameserver=`127.0.0.1'*, ***kwargs*)
Delete the forward and reverse records for a host.

Returns true if any records are deleted.

CLI Example:

```
salt ns1 ddns.delete_host example.com host1
```

salt.modules.ddns.**update**(*zone*, *name*, *ttl*, *rdtype*, *data*, *nameserver=`127.0.0.1'*, *replace=False*, ***kwargs*)
Add, replace, or update a DNS record. nameserver must be an IP address and the minion running this module must have update privileges on that server. If replace is true, first deletes all records for this name and type.

CLI Example:

```
salt ns1 ddns.update example.com host1 60 A 10.0.0.1
```

22.16.45 salt.modules.deb_apache

Support for Apache

Please note: The functions in here are Debian-specific. Placing them in this separate file will allow them to load only on Debian-based systems, while still loading under the `apache` namespace.

salt.modules.deb_apache.**a2dismod**(*mod*)
Runs a2dismod for the given mod.

This will only be functional on Debian-based operating systems (Ubuntu, Mint, etc).

CLI Examples:

```
salt '*' apache.a2dismod vhost_alias
```

salt.modules.deb_apache.**a2dissite**(*site*)
Runs a2dissite for the given site.

This will only be functional on Debian-based operating systems (Ubuntu, Mint, etc).

CLI Examples:

```
salt '*' apache.a2dissite example.com
```

salt.modules.deb_apache.**a2enmod**(*mod*)
Runs a2enmod for the given mod.

This will only be functional on Debian-based operating systems (Ubuntu, Mint, etc).

CLI Examples:

```
salt '*' apache.a2enmod vhost_alias
```

salt.modules.deb_apache.**a2ensite**(*site*)
Runs a2ensite for the given site.

This will only be functional on Debian-based operating systems (Ubuntu, Mint, etc).

CLI Examples:

```
salt '*' apache.a2ensite example.com
```

salt.modules.deb_apache.**check_mod_enabled**(*mod*)
Checks to see if the specific mod symlink is in /etc/apache2/mods-enabled.

This will only be functional on Debian-based operating systems (Ubuntu, Mint, etc).

CLI Examples:

```
salt '*' apache.check_mod_enabled status.conf
salt '*' apache.check_mod_enabled status.load
```

salt.modules.deb_apache.**check_site_enabled**(*site*)
Checks to see if the specific Site symlink is in /etc/apache2/sites-enabled.

This will only be functional on Debian-based operating systems (Ubuntu, Mint, etc).

CLI Examples:

```
salt '*' apache.check_site_enabled example.com
```

22.16.46 salt.modules.debconfmod

Support for Debconf

salt.modules.debconfmod.**get_selections**(*fetchempty=True*)
Answers to debconf questions for all packages in the following format:

```
{'package': [['question', 'type', 'value'], ...]}
```

CLI Example:

```
salt '*' debconf.get_selections
```

salt.modules.debconfmod.**set_**(*package*, *question*, *type*, *value*, **extra*)
Set answers to debconf questions for a package.

CLI Example:

```
salt '*' debconf.set <package> <question> <type> <value> [<value> ...]
```

salt.modules.debconfmod.**set_file**(*path*, *saltenv='base'*, ***kwargs*)
Set answers to debconf questions from a file.

CLI Example:

```
salt '*' debconf.set_file salt://pathto/pkg.selections
```

salt.modules.debconfmod.**set_template**(*path*, *template*, *context*, *defaults*, *saltenv='base'*, ***kwargs*)

Set answers to debconf questions from a template.

path location of the file containing the package selections

template template format

context variables to add to the template environment

default default values for the template environment

CLI Example:

```
salt '*' debconf.set_template salt://pathto/pkg.selections.jinja jinja None None
```

salt.modules.debconfmod.**show**(*name*)

Answers to debconf questions for a package in the following format:

```
[['question', 'type', 'value'], ...]
```

If debconf doesn't know about a package, we return None.

CLI Example:

```
salt '*' debconf.show <package name>
```

22.16.47 salt.modules.debian_ip

The networking module for Debian based distros

References:

- http://www.debian.org/doc/manuals/debian-reference/ch05.en.html

salt.modules.debian_ip.**apply_network_settings**(***settings*)

Apply global network configuration.

CLI Example:

```
salt '*' ip.apply_network_settings
```

salt.modules.debian_ip.**build_bond**(*iface*, ***settings*)

Create a bond script in /etc/modprobe.d with the passed settings and load the bonding kernel module.

CLI Example:

```
salt '*' ip.build_bond bond0 mode=balance-alb
```

salt.modules.debian_ip.**build_interface**(*iface*, *iface_type*, *enabled*, ***settings*)

Build an interface script for a network interface.

CLI Example:

```
salt '*' ip.build_interface eth0 eth <settings>
```

salt.modules.debian_ip.**build_network_settings**(***settings*)

Build the global network script.

CLI Example:

```
salt '*' ip.build_network_settings <settings>
```

salt.modules.debian_ip.**build_routes**(*iface*, ***settings*)
Add route scripts for a network interface using up commands.

CLI Example:

```
salt '*' ip.build_routes eth0 <settings>
```

salt.modules.debian_ip.**down**(*iface*, *iface_type*)
Shutdown a network interface

CLI Example:

```
salt '*' ip.down eth0
```

salt.modules.debian_ip.**get_bond**(*iface*)
Return the content of a bond script

CLI Example:

```
salt '*' ip.get_bond bond0
```

salt.modules.debian_ip.**get_interface**(*iface*)
Return the contents of an interface script

CLI Example:

```
salt '*' ip.get_interface eth0
```

salt.modules.debian_ip.**get_network_settings**()
Return the contents of the global network script.

CLI Example:

```
salt '*' ip.get_network_settings
```

salt.modules.debian_ip.**get_routes**(*iface*)
Return the routes for the interface

CLI Example:

```
salt '*' ip.get_interface eth0
```

salt.modules.debian_ip.**up**(*iface*, *iface_type*)
Start up a network interface

CLI Example:

```
salt '*' ip.up eth0
```

22.16.48 salt.modules.debian_service

Service support for Debian systems (uses update-rc.d and /sbin/service)

salt.modules.debian_service.**available**(*name*)
Returns `True` if the specified service is available, otherwise returns `False`.

CLI Example:

```
salt '*' service.available sshd
```

salt.modules.debian_service.**disable**(*name*, ***kwargs*)
Disable the named service to start at boot

CLI Example:

```
salt '*' service.disable <service name>
```

salt.modules.debian_service.**disabled**(*name*)
Return True if the named service is enabled, false otherwise

CLI Example:

```
salt '*' service.disabled <service name>
```

salt.modules.debian_service.**enable**(*name*, ***kwargs*)
Enable the named service to start at boot

CLI Example:

```
salt '*' service.enable <service name>
```

salt.modules.debian_service.**enabled**(*name*, ***kwargs*)
Return True if the named service is enabled, false otherwise

CLI Example:

```
salt '*' service.enabled <service name>
```

salt.modules.debian_service.**force_reload**(*name*)
Force-reload the named service

CLI Example:

```
salt '*' service.force_reload <service name>
```

salt.modules.debian_service.**get_all**()
Return all available boot services

CLI Example:

```
salt '*' service.get_all
```

salt.modules.debian_service.**get_disabled**()
Return a set of services that are installed but disabled

CLI Example:

```
salt '*' service.get_disabled
```

salt.modules.debian_service.**get_enabled**()
Return a list of service that are enabled on boot

CLI Example:

```
salt '*' service.get_enabled
```

salt.modules.debian_service.**missing**(*name*)
The inverse of service.available. Returns `True` if the specified service is not available, otherwise returns `False`.

CLI Example:

```
salt '*' service.missing sshd
```

salt.modules.debian_service.**reload_**(*name*)
Reload the named service

CLI Example:

```
salt '*' service.reload <service name>
```

salt.modules.debian_service.**restart**(*name*)
Restart the named service

CLI Example:

```
salt '*' service.restart <service name>
```

salt.modules.debian_service.**start**(*name*)
Start the specified service

CLI Example:

```
salt '*' service.start <service name>
```

salt.modules.debian_service.**status**(*name*, *sig=None*)
Return the status for a service, pass a signature to use to find the service via ps

CLI Example:

```
salt '*' service.status <service name>
```

salt.modules.debian_service.**stop**(*name*)
Stop the specified service

CLI Example:

```
salt '*' service.stop <service name>
```

22.16.49 salt.modules.defaults

salt.modules.defaults.**get**(*key*, *default=''*)
defaults.get is used much like pillar.get except that it will read a default value for a pillar from defaults.json or defaults.yaml files that are stored in the root of a salt formula.

When called from the CLI it works exactly like pillar.get.

CLI Example:

```
salt '*' defaults.get core:users:root
```

When called from an SLS file, it works by first reading a defaults.json and second a defaults.yaml file. If the key exists in these files and does not exist in a pillar named after the formula, the value from the defaults file is used.

Example core/defaults.json file for the `core' formula:

```
{
    "users": {
        "root": 0
    }
}
```

With this, from a state file you can use salt['defaults.get'](`users:root') to read the `0' value from defaults.json if a core:users:root pillar key is not defined.

22.16.50 salt.modules.devmap

Device-Mapper module

salt.modules.devmap.**multipath_flush**(*device*)
Device-Mapper Multipath flush

CLI Example:

```
salt '*' devmap.multipath_flush mpath1
```

salt.modules.devmap.**multipath_list**()
Device-Mapper Multipath list

CLI Example:

```
salt '*' devmap.multipath_list
```

22.16.51 salt.modules.dig

Compendium of generic DNS utilities

salt.modules.dig.**A**(*host*, *nameserver=None*)
Return the A record for `host`.

Always returns a list.

CLI Example:

```
salt ns1 dig.A www.google.com
```

salt.modules.dig.**AAAA**(*host*, *nameserver=None*)
Return the AAAA record for `host`.

Always returns a list.

CLI Example:

```
salt ns1 dig.AAAA www.google.com
```

salt.modules.dig.**MX**(*domain*, *resolve=False*, *nameserver=None*)
Return a list of lists for the MX of `domain`.

If the `resolve` argument is True, resolve IPs for the servers.

It's limited to one IP, because although in practice it's very rarely a round robin, it is an acceptable configuration and pulling just one IP lets the data be similar to the non-resolved version. If you think an MX has multiple IPs, don't use the resolver here, resolve them in a separate step.

CLI Example:

```
salt ns1 dig.MX google.com
```

salt.modules.dig.**NS**(*domain*, *resolve=True*, *nameserver=None*)
Return a list of IPs of the nameservers for `domain`

If `resolve` is False, don't resolve names.

CLI Example:

```
salt ns1 dig.NS google.com
```

salt.modules.dig.**SPF**(*domain*, *record='SPF'*, *nameserver=None*)
Return the allowed IPv4 ranges in the SPF record for `domain`.

If record is `SPF` and the SPF record is empty, the TXT record will be searched automatically. If you know the domain uses TXT and not SPF, specifying that will save a lookup.

CLI Example:

```
salt ns1 dig.SPF google.com
```

salt.modules.dig.**TXT**(*host*, *nameserver=None*)
Return the TXT record for `host`.

Always returns a list.

CLI Example:

```
salt ns1 dig.TXT google.com
```

salt.modules.dig.**check_ip**(*addr*)
Check if address is a valid IP. returns True if valid, otherwise False.

CLI Example:

```
salt ns1 dig.check_ip 127.0.0.1
salt ns1 dig.check_ip 1111:2222:3333:4444:5555:6666:7777:8888
```

22.16.52 salt.modules.disk

Module for gathering disk information

salt.modules.disk.**blkid**(*device=None*)
Return block device attributes: UUID, LABEL, etc. This function only works on systems where blkid is available.

CLI Example:

```
salt '*' disk.blkid
salt '*' disk.blkid /dev/sda
```

salt.modules.disk.**inodeusage**(*args=None*)
Return inode usage information for volumes mounted on this minion

CLI Example:

```
salt '*' disk.inodeusage
```

salt.modules.disk.**percent**(*args=None*)
Return partition information for volumes mounted on this minion

CLI Example:

```
salt '*' disk.percent /var
```

salt.modules.disk.**usage**(*args=None*)
Return usage information for volumes mounted on this minion

CLI Example:

```
salt '*' disk.usage
```

22.16.53 salt.modules.djangomod

Manage Django sites

salt.modules.djangomod.**collectstatic**(*settings_module*, *bin_env=None*, *no_post_process=False*, *ignore=None*, *dry_run=False*, *clear=False*, *link=False*, *no_default_ignore=False*, *pythonpath=None*, *env=None*)

Collect static files from each of your applications into a single location that can easily be served in production.

CLI Example:

```
salt '*' django.collectstatic <settings_module>
```

salt.modules.djangomod.**command**(*settings_module*, *command*, *bin_env=None*, *pythonpath=None*, *env=None*, **args*, ***kwargs*)

Run arbitrary django management command

CLI Example:

```
salt '*' django.command <settings_module> <command>
```

salt.modules.djangomod.**createsuperuser**(*settings_module*, *username*, *email*, *bin_env=None*, *database=None*, *pythonpath=None*, *env=None*)

Create a super user for the database. This function defaults to use the `--noinput` flag which prevents the creation of a password for the superuser.

CLI Example:

```
salt '*' django.createsuperuser <settings_module> user user@example.com
```

salt.modules.djangomod.**loaddata**(*settings_module*, *fixtures*, *bin_env=None*, *database=None*, *pythonpath=None*, *env=None*)

Load fixture data

Fixtures: comma separated list of fixtures to load

CLI Example:

```
salt '*' django.loaddata <settings_module> <comma delimited list of fixtures>
```

salt.modules.djangomod.**syncdb**(*settings_module*, *bin_env=None*, *migrate=False*, *database=None*, *pythonpath=None*, *env=None*, *noinput=True*)

Run syncdb

Execute the Django-Admin syncdb command, if South is available on the minion the `migrate` option can be passed as `True` calling the migrations to run after the syncdb completes

CLI Example:

```
salt '*' django.syncdb <settings_module>
```

22.16.54 salt.modules.dnsmasq

Module for managing dnsmasq

salt.modules.dnsmasq.**fullversion**()
Shows installed version of dnsmasq and compile options.

CLI Example:

```
salt '*' dnsmasq.version
```

salt.modules.dnsmasq.**get_config**(*config_file='/etc/dnsmasq.conf'*)
Dumps all options from the config file.

CLI Examples:

```
salt '*' dnsmasq.get_config
salt '*' dnsmasq.get_config file=/etc/dnsmasq.conf
```

salt.modules.dnsmasq.**set_config**(*config_file='/etc/dnsmasq.conf'*, *follow=True*, ***kwargs*)
Sets a value or a set of values in the specified file. By default, if conf-dir is configured in this file, salt will attempt to set the option in any file inside the conf-dir where it has already been enabled. If it does not find it inside any files, it will append it to the main config file. Setting follow to False will turn off this behavior.

If a config option currently appears multiple times (such as dhcp-host, which is specified at least once per host), the new option will be added to the end of the main config file (and not to any includes). If you need an option added to a specific include file, specify it as the config_file.

CLI Examples:

```
salt '*' dnsmasq.set_config domain=mydomain.com
salt '*' dnsmasq.set_config follow=False domain=mydomain.com
salt '*' dnsmasq.set_config file=/etc/dnsmasq.conf domain=mydomain.com
```

salt.modules.dnsmasq.**version**()
Shows installed version of dnsmasq.

CLI Example:

```
salt '*' dnsmasq.version
```

22.16.55 salt.modules.dnsutil

Compendium of generic DNS utilities

salt.modules.dnsutil.**A**(*host*, *nameserver=None*)
Return the A record(s) for *host*.

Always returns a list.

CLI Example:

```
salt ns1 dnsutil.A www.google.com
```

salt.modules.dnsutil.**AAAA**(*host*, *nameserver=None*)
Return the AAAA record(s) for *host*.

Always returns a list.

New in version 2014.7.5.

CLI Example:

```
salt ns1 dnsutil.AAAA www.google.com
```

salt.modules.dnsutil.**MX**(*domain*, *resolve=False*, *nameserver=None*)
Return a list of lists for the MX of `domain`.

If the `resolve' argument is True, resolve IPs for the servers.

It's limited to one IP, because although in practice it's very rarely a round robin, it is an acceptable configuration and pulling just one IP lets the data be similar to the non-resolved version. If you think an MX has multiple IPs, don't use the resolver here, resolve them in a separate step.

CLI Example:

```
salt ns1 dig.MX google.com
```

salt.modules.dnsutil.**NS**(*domain*, *resolve=True*, *nameserver=None*)
Return a list of IPs of the nameservers for `domain`

If `resolve' is False, don't resolve names.

CLI Example:

```
salt ns1 dig.NS google.com
```

salt.modules.dnsutil.**SPF**(*domain*, *record='SPF'*, *nameserver=None*)
Return the allowed IPv4 ranges in the SPF record for `domain`.

If record is `SPF` and the SPF record is empty, the TXT record will be searched automatically. If you know the domain uses TXT and not SPF, specifying that will save a lookup.

CLI Example:

```
salt ns1 dig.SPF google.com
```

salt.modules.dnsutil.**check_ip**(*ip_addr*)
Check that string ip_addr is a valid IP

CLI Example:

```
salt ns1 dig.check_ip 127.0.0.1
```

salt.modules.dnsutil.**hosts_append**(*hostsfile='/etc/hosts'*, *ip_addr=None*, *entries=None*)
Append a single line to the /etc/hosts file.

CLI Example:

```
salt '*' dnsutil.hosts_append /etc/hosts 127.0.0.1 ad1.yuk.co,ad2.yuk.co
```

salt.modules.dnsutil.**hosts_remove**(*hostsfile='/etc/hosts'*, *entries=None*)
Remove a host from the /etc/hosts file. If doing so will leave a line containing only an IP address, then the line will be deleted. This function will leave comments and blank lines intact.

CLI Examples:

```
salt '*' dnsutil.hosts_remove /etc/hosts ad1.yuk.co
salt '*' dnsutil.hosts_remove /etc/hosts ad2.yuk.co,ad1.yuk.co
```

salt.modules.dnsutil.**parse_hosts**(*hostsfile='/etc/hosts'*, *hosts=None*)
Parse /etc/hosts file.

CLI Example:

```
salt '*' dnsutil.parse_hosts
```

salt.modules.dnsutil.**parse_zone**(*zonefile=None*, *zone=None*)
Parses a zone file. Can be passed raw zone data on the API level.

CLI Example:

```
salt ns1 dnsutil.parse_zone /var/lib/named/example.com.zone
```

salt.modules.dnsutil.**serial**(*zone=''*, *update=False*)
Return, store and update a dns serial for your zone files.

zone: a keywork for a specific zone

update: store an updated version of the serial in a grain

If `update` is False, the function will retrieve an existing serial or return the current date if no serial is stored. Nothing will be stored

If `update` is True, the function will set the serial to the current date if none exist or if the existing serial is for a previous date. If a serial for greater than the current date is already stored, the function will increment it.

This module stores the serial in a grain, you can explicitly set the stored value as a grain named `dnsserial_<zone_name>`.

CLI Example:

```
salt ns1 dnsutil.serial example.com
```

22.16.56 salt.modules.dockerio

Management of Docker Containers

New in version 2014.1.0.

Note: The DockerIO integration is still in beta; the API is subject to change

General Notes

As we use states, we don't want to be continuously popping dockers, so we will map each container id (or image) with a grain whenever it is relevant.

As a corollary, we will resolve a container id either directly by the id or try to find a container id matching something stocked in grain.

Installation Prerequisites

- You will need the `docker-py` python package in your python installation path that is running salt. Its version should support Docker Remote API v1.12.

  Currently, `docker-py 0.5.0` is known to support Docker Remote API v1.12

```
pip install docker-py==0.5.0
```

Prerequisite Pillar Configuration for Authentication

- To push or pull you will need to be authenticated as the `docker-py` bindings require it
- For this to happen, you will need to configure a mapping in the pillar representing your per URL authentication bits:

```
docker-registries:
    registry_url:
        email: foo@foo.com
        password: s3cr3t
        username: foo
```

- You need at least an entry to the default docker index:

```
docker-registries:
    https://index.docker.io/v1/:
        email: foo@foo.com
        password: s3cr3t
        username: foo
```

- You can define multiple registry blocks for them to be aggregated. The only thing to keep in mind is that their ID must finish with `-docker-registries`:

```
ac-docker-registries:
    https://index.bar.io/v1/:
        email: foo@foo.com
        password: s3cr3t
        username: foo

ab-docker-registries:
    https://index.foo.io/v1/:
        email: foo@foo.com
        password: s3cr3t
        username: foo
```

This could be also written as:

```
docker-registries:
    https://index.bar.io/v1/:
        email: foo@foo.com
        password: s3cr3t
        username: foo
    https://index.foo.io/v1/:
        email: foo@foo.com
        password: s3cr3t
        username: foo
```

Methods

- **Registry Dialog**
  - `login`
  - `push`
  - `pull`
- **Docker Management**

  - version
  - info

- **Image Management**
  - search
  - inspect_image
  - get_images
  - remove_image
  - import_image
  - build
  - tag
  - save
  - load

- **Container Management**
  - start
  - stop
  - restart
  - kill
  - wait
  - get_containers
  - inspect_container
  - remove_container
  - is_running
  - top
  - port
  - logs
  - diff
  - commit
  - create_container
  - export
  - get_container_root

Runtime Execution within a specific, already existing/running container

Idea is to use lxc-attach to execute inside the container context. We do not want to use `docker run` but want to execute something inside a running container.

These are the available methods:

- retcode

- run
- run_all
- run_stderr
- run_stdout
- script
- script_retcode

salt.modules.dockerio.**build**(*path=None*, *tag=None*, *quiet=False*, *fileobj=None*, *nocache=False*, *rm=True*, *timeout=None*)

Build a docker image from a dockerfile or an URL

path url/branch/docker_dir or path on the filesystem to the dockerfile

tag tag of the image

quiet quiet mode, Default is `False`

nocache do not use docker image cache, Default is `False`

rm remove intermediate commits, Default is `True`

timeout timeout value before aborting (in seconds)

CLI Example:

```
salt '*' docker.build vieux/apache
salt '*' docker.build github.com/creack/docker-firefox
```

salt.modules.dockerio.**commit**(*container*, *repository=None*, *tag=None*, *message=None*, *author=None*, *conf=None*)

Commit a container (promotes it to an image)

container container id

repository repository/image to commit to

tag tag of the image (Optional)

message commit message (Optional)

author author name (Optional)

conf conf (Optional)

CLI Example:

```
salt '*' docker.commit <container id>
```

salt.modules.dockerio.**create_container**(*image*, *command=None*, *hostname=None*, *user=None*, *detach=True*, *stdin_open=False*, *tty=False*, *mem_limit=0*, *ports=None*, *environment=None*, *dns=None*, *volumes=None*, *volumes_from=None*, *name=None*, *cpu_shares=None*, *cpuset=None*)

Create a new container

image image to create the container from

command command to execute while starting

hostname hostname of the container

user user to run docker as

detach daemon mode, Default is `True`

environment environment variable mapping (`{'foo':'BAR'}`)

ports port redirections (`{'222': {}}`)

volumes list of volume mappings:

```
        (['/mountpoint/in/container:/guest/foo', '/same/path/mounted/point'])
```

tty attach ttys, Default is `False`

stdin_open let stdin open, Default is `False`

name name given to container

cpu_shares CPU shares (relative weight)

cpuset CPUs in which to allow execution (`0-3' or `0,1')

CLI Example:

```
    salt '*' docker.create_container o/ubuntu volumes="['/s','/m:/f']"
```

`salt.modules.dockerio.`**`diff`**(*container*)

Get container diffs

container container id

CLI Example:

```
    salt '*' docker.diff <container id>
```

`salt.modules.dockerio.`**`exists`**(*container*)

Check if a given container exists

container container id

Returns `True` if container exists otherwise returns `False`

CLI Example:

```
    salt '*' docker.exists <container id>
```

`salt.modules.dockerio.`**`export`**(*container*, *path*)

Export a container to a file

container container id

path path to which file is to be exported

CLI Example:

```
    salt '*' docker.export <container id>
```

`salt.modules.dockerio.`**`get_container_root`**(*container*)

Get the container rootfs path

container container id or grain

CLI Example:

```
    salt '*' docker.get_container_root <container id>
```

`salt.modules.dockerio.`**`get_containers`**(*all=True*, *trunc=False*, *since=None*, *before=None*, *limit=-1*, *host=False*, *inspect=False*)

Get a list of mappings representing all containers

all return all containers, Default is `True`

trunc set it to True to have the short ID, Default is `False`

host include the Docker host's ipv4 and ipv6 address in return, Default is `False`

inspect Get more granular information about each container by running a docker inspect

CLI Example:

```
salt '*' docker.get_containers
salt '*' docker.get_containers host=True
salt '*' docker.get_containers host=True inspect=True
```

`salt.modules.dockerio.`**`get_images`**(*name=None*, *quiet=False*, *all=True*)

List docker images

name repository name

quiet only show image id, Default is `False`

all show all images, Default is `True`

CLI Example:

```
salt '*' docker.get_images <name> [quiet=True|False] [all=True|False]
```

`salt.modules.dockerio.`**`import_image`**(*src*, *repo*, *tag=None*)

Import content from a local tarball or a URL to a docker image

src content to import (URL or absolute path to a tarball)

repo repository to import to

tag set tag of the image (Optional)

CLI Example:

```
salt '*' docker.import_image <src> <repo> [tag]
```

`salt.modules.dockerio.`**`info`**()

Get the version information about docker. This is similar to `docker info` command

CLI Example:

```
salt '*' docker.info
```

`salt.modules.dockerio.`**`inspect_container`**(*container*)

Get container information. This is similar to `docker inspect` command but only for containers

container container id

CLI Example:

```
salt '*' docker.inspect_container <container id>
```

`salt.modules.dockerio.`**`inspect_image`**(*image*)

Inspect the status of an image and return relative data. This is similar to `docker inspect` command but only for images

image name of the image

CLI Example:

```
salt '*' docker.inspect_image <image>
```

salt.modules.dockerio.**is_running**(*container*)
Check if the specified container is running

container container id

Returns `True` if container is running otherwise returns `False`

CLI Example:

```
salt '*' docker.is_running <container id>
```

salt.modules.dockerio.**kill**(*container*)
Kill a running container

container container id

CLI Example:

```
salt '*' docker.kill <container id>
```

salt.modules.dockerio.**load**(*imagepath*)
Load the specified file at imagepath into docker that was generated from a docker save command e.g. *docker load < imagepath*

imagepath imagepath to docker tar file

CLI Example:

```
salt '*' docker.load /path/to/image
```

salt.modules.dockerio.**login**(*url=None*, *username=None*, *password=None*, *email=None*)
Wrapper to the `docker.py` login method (does not do much yet)

url registry url to authenticate to

username username to authenticate

password password to authenticate

email email to authenticate

CLI Example:

```
salt '*' docker.login <url> <username> <password> <email>
```

salt.modules.dockerio.**logs**(*container*)
Return logs for a specified container

container container id

CLI Example:

```
salt '*' docker.logs <container id>
```

salt.modules.dockerio.**port**(*container*, *private_port*)
Private port mapping allocation information. This method is broken on docker-py side. Just use the result of inspect to mangle port allocation

container container id

private_port private port on the container to query for

CLI Example:

```
salt '*' docker.port <container id> <private port>
```

salt.modules.dockerio.**pull**(*repo*, *tag=None*, *insecure_registry=False*)

Pulls an image from any registry. See documentation at top of this page to configure authenticated access

repo name of repository

tag specific tag to pull (Optional)

insecure_registry set as `True` to use insecure (non HTTPS) registry. Default is `False` (only available if using docker-py >= 0.5.0)

CLI Example:

```
salt '*' docker.pull <repository> [tag]
```

salt.modules.dockerio.**push**(*repo*, *tag=None*, *quiet=False*, *insecure_registry=False*)

Pushes an image to any registry. See documentation at top of this page to configure authenticated access

repo name of repository

tag specific tag to push (Optional)

quiet set as `True` to quiet output, Default is `False`

insecure_registry set as `True` to use insecure (non HTTPS) registry. Default is `False` (only available if using docker-py >= 0.5.0)

CLI Example:

```
salt '*' docker.push <repository> [tag] [quiet=True|False]
```

salt.modules.dockerio.**remove_container**(*container*, *force=False*, *v=False*)

Remove a container from a docker installation

container container id

force remove a running container, Default is `False`

v remove the volumes associated to the container, Default is `False`

CLI Example:

```
salt '*' docker.remove_container <container id> [force=True|False] [v=True|False]
```

salt.modules.dockerio.**remove_image**(*image*)

Remove an image from a system.

image name of image

CLI Example:

```
salt '*' docker.remove_image <image>
```

salt.modules.dockerio.**restart**(*container*, *timeout=10*)

Restart a running container

container container id

timeout timeout for container to exit gracefully before killing it, Default is `10` seconds

CLI Example:

```
salt '*' docker.restart <container id> [timeout=20]
```

salt.modules.dockerio.**retcode**(*container*, *cmd*)

Wrapper for `cmdmod.retcode` inside a container context

container container id (or grain)

cmd command to execute

Note: The return is a bit different as we use the docker struct. Output of the command is in `out' and result is `False` if command failed to execute.

Warning: Be advised that this function allows for raw shell access to the named container! If allowing users to execute this directly it may allow more rights than intended!

CLI Example:

```
salt '*' docker.retcode <container id> 'ls -l /etc'
```

salt.modules.dockerio.**run**(*container*, *cmd*)

Wrapper for `cmdmod.run` inside a container context

container container id (or grain)

cmd command to execute

Note: The return is a bit different as we use the docker struct. Output of the command is in `out' and result is always `True`.

Warning: Be advised that this function allows for raw shell access to the named container! If allowing users to execute this directly it may allow more rights than intended!

CLI Example:

```
salt '*' docker.run <container id> 'ls -l /etc'
```

salt.modules.dockerio.**run_all**(*container*, *cmd*)

Wrapper for `cmdmod.run_all` inside a container context

container container id (or grain)

cmd command to execute

Note: The return is a bit different as we use the docker struct. Output of the command is in `out' and result is `False` if command failed to execute.

Warning: Be advised that this function allows for raw shell access to the named container! If allowing users to execute this directly it may allow more rights than intended!

CLI Example:

```
salt '*' docker.run_all <container id> 'ls -l /etc'
```

salt.modules.dockerio.**run_stderr**(*container*, *cmd*)

Wrapper for `cmdmod.run_stderr` inside a container context

container container id (or grain)

cmd command to execute

Note: The return is a bit different as we use the docker struct. Output of the command is in `out' and result is always `True`.

Warning: Be advised that this function allows for raw shell access to the named container! If allowing users to execute this directly it may allow more rights than intended!

CLI Example:

```
salt '*' docker.run_stderr <container id> 'ls -l /etc'
```

`salt.modules.dockerio.`**`run_stdout`**(*container*, *cmd*)

Wrapper for `cmdmod.run_stdout` inside a container context

container container id (or grain)

cmd command to execute

Note: The return is a bit different as we use the docker struct. Output of the command is in `out' and result is always `True`.

Warning: Be advised that this function allows for raw shell access to the named container! If allowing users to execute this directly it may allow more rights than intended!

CLI Example:

```
salt '*' docker.run_stdout <container id> 'ls -l /etc'
```

`salt.modules.dockerio.`**`save`**(*image*, *filename*)

New in version 2015.5.0.

Save the specified image to filename from docker e.g. *docker save image > filename*

image name of image

filename The filename of the saved docker image

CLI Example:

```
salt '*' docker.save arch_image /path/to/save/image
```

`salt.modules.dockerio.`**`script`**(*container*, *source*, *args=None*, *cwd=None*, *stdin=None*, *runas=None*, *shell='/bin/bash'*, *env=None*, *template='jinja'*, *umask=None*, *timeout=None*, *reset_system_locale=True*, *no_clean=False*, *saltenv='base'*)

Wrapper for `cmdmod.script` inside a container context

container container id (or grain)

additional parameters See `cmd.script`

Warning: Be advised that this function allows for raw shell access to the named container! If allowing users to execute this directly it may allow more rights than intended!

Download a script from a remote location and execute the script in the container. The script can be located on the salt master file server or on an HTTP/FTP server.

The script will be executed directly, so it can be written in any available programming language.

The script can also be formatted as a template, the default is jinja. Arguments for the script can be specified as well.

CLI Example:

```
salt '*' docker.script <container id> salt://docker_script.py
salt '*' docker.script <container id> salt://scripts/runme.sh 'arg1 arg2 "arg 3"'
salt '*' docker.script <container id> salt://scripts/windows_task.ps1 args=' -Input c:\tmp\infil
```

A string of standard input can be specified for the command to be run using the stdin parameter. This can be useful in cases where sensitive information must be read from standard input:

CLI Example:

```
salt '*' docker.script <container id> salt://scripts/runme.sh stdin='one\ntwo\nthree\nfour\nfive
```

salt.modules.dockerio.**script_retcode**(*container*, *source*, *cwd=None*, *stdin=None*, *runas=None*, *shell='/bin/bash'*, *env=None*, *template='jinja'*, *umask=None*, *timeout=None*, *reset_system_locale=True*, *no_clean=False*, *saltenv='base'*)

Wrapper for `cmdmod.script_retcode` inside a container context

container container id (or grain)

additional parameters See `cmd.script_retcode`

> **Warning:** Be advised that this function allows for raw shell access to the named container! If allowing users to execute this directly it may allow more rights than intended!

CLI Example:

```
salt '*' docker.script_retcode <container id> salt://docker_script.py
```

salt.modules.dockerio.**search**(*term*)

Search for an image on the registry

term search keyword

CLI Example:

```
salt '*' docker.search <term>
```

salt.modules.dockerio.**start**(*container*, *binds=None*, *port_bindings=None*, *lxc_conf=None*, *publish_all_ports=None*, *links=None*, *privileged=False*, *dns=None*, *volumes_from=None*, *network_mode=None*, *restart_policy=None*, *cap_add=None*, *cap_drop=None*)

Start the specified container

container container id

CLI Example:

```
salt '*' docker.start <container id>
```

salt.modules.dockerio.**stop**(*container*, *timeout=10*)

Stop a running container

container container id

timeout timeout for container to exit gracefully before killing it, Default is `10` seconds

CLI Example:

```
salt '*' docker.stop <container id> [timeout=20]
```

salt.modules.dockerio.**tag**(*image*, *repository*, *tag=None*, *force=False*)
Tag an image into a repository

image name of image

repository name of repository

tag tag to apply (Optional)

force force apply tag, Default is `False`

CLI Example:

```
salt '*' docker.tag <image> <repository> [tag] [force=True|False]
```

salt.modules.dockerio.**top**(*container*)
Run the docker top command on a specific container

container container id

CLI Example:

```
salt '*' docker.top <container id>
```

salt.modules.dockerio.**version**()
Get docker version

CLI Example:

```
salt '*' docker.version
```

salt.modules.dockerio.**wait**(*container*)
Wait for a container to exit gracefully

container container id

CLI Example:

```
salt '*' docker.wait <container id>
```

22.16.57 salt.modules.dpkg

Support for DEB packages

salt.modules.dpkg.**file_dict**(**packages*)
List the files that belong to a package, grouped by package. Not specifying any packages will return a list of _every_ file on the system's package database (not generally recommended).

CLI Examples:

```
salt '*' lowpkg.file_list httpd
salt '*' lowpkg.file_list httpd postfix
salt '*' lowpkg.file_list
```

salt.modules.dpkg.**file_list**(**packages*)
List the files that belong to a package. Not specifying any packages will return a list of _every_ file on the system's package database (not generally recommended).

CLI Examples:

```
    salt '*' lowpkg.file_list httpd
    salt '*' lowpkg.file_list httpd postfix
    salt '*' lowpkg.file_list
```

salt.modules.dpkg.**list_pkgs**(**packages*)
List the packages currently installed in a dict:

```
    {'<package_name>': '<version>'}
```

External dependencies:

```
    Virtual package resolution requires aptitude. Because this function
    uses dpkg, virtual packages will be reported as not installed.
```

CLI Example:

```
    salt '*' lowpkg.list_pkgs
    salt '*' lowpkg.list_pkgs httpd
```

salt.modules.dpkg.**unpurge**(**packages*)
Change package selection for each package specified to `install'

CLI Example:

```
    salt '*' lowpkg.unpurge curl
```

22.16.58 salt.modules.drac

Manage Dell DRAC

salt.modules.drac.**change_password**(*username*, *password*, *uid=None*)
Change users password

CLI Example:

```
    salt dell drac.change_password [USERNAME] [PASSWORD] [UID - optional]
    salt dell drac.change_password diana secret
```

salt.modules.drac.**create_user**(*username*, *password*, *permissions*, *users=None*)
Create user accounts

CLI Example:

```
    salt dell drac.create_user [USERNAME] [PASSWORD] [PRIVELEGES]
    salt dell drac.create_user diana secret login,test_alerts,clear_logs
```

DRAC Privileges

- login : Login to iDRAC
- drac : Configure iDRAC
- user_management : Configure Users
- clear_logs : Clear Logs
- server_control_commands : Execute Server Control Commands
- console_redirection : Access Console Redirection
- virtual_media : Access Virtual Media

- test_alerts : Test Alerts
- debug_commands : Execute Debug Commands

salt.modules.drac.**delete_user**(*username*, *uid=None*)
Delete a user

CLI Example:

```
salt dell drac.delete_user [USERNAME] [UID - optional]
salt dell drac.delete_user diana 4
```

salt.modules.drac.**email_alerts**(*action*)
Enable/Disable email alerts

CLI Example:

```
salt dell drac.email_alerts True
salt dell drac.email_alerts False
```

salt.modules.drac.**list_users**()
List all DRAC users

CLI Example:

```
salt dell drac.list_users
```

salt.modules.drac.**nameservers**(**ns*)
Configure the nameservers on the DRAC

CLI Example:

```
salt dell drac.nameservers [NAMESERVERS]
salt dell drac.nameservers ns1.example.com ns2.example.com
```

salt.modules.drac.**network_info**()
Return Network Configuration

CLI Example:

```
salt dell drac.getniccfg
```

salt.modules.drac.**server_hardreset**()
Performs a reset (reboot) operation on the managed server.

CLI Example:

```
salt dell drac.server_hardreset
```

salt.modules.drac.**server_poweroff**()
Powers down the managed server.

CLI Example:

```
salt dell drac.server_poweroff
```

salt.modules.drac.**server_poweron**()
Powers up the managed server.

CLI Example:

```
salt dell drac.server_poweron
```

salt.modules.drac.**server_pxe**()
Configure server to PXE perform a one off PXE boot

CLI Example:

```
salt dell drac.server_pxe
```

salt.modules.drac.**server_reboot**()
Issues a power-cycle operation on the managed server. This action is similar to pressing the power button on the system's front panel to power down and then power up the system.

CLI Example:

```
salt dell drac.server_reboot
```

salt.modules.drac.**set_network**(*ip*, *netmask*, *gateway*)
Configure Network

CLI Example:

```
salt dell drac.set_network [DRAC IP] [NETMASK] [GATEWAY]
salt dell drac.set_network 192.168.0.2 255.255.255.0 192.168.0.1
```

salt.modules.drac.**set_permissions**(*username*, *permissions*, *uid=None*)
Configure users permissions

CLI Example:

```
salt dell drac.set_permissions [USERNAME] [PRIVELEGES] [USER INDEX - optional]
salt dell drac.set_permissions diana login,test_alerts,clear_logs 4
```

DRAC Privileges

- login : Login to iDRAC
- drac : Configure iDRAC
- user_management : Configure Users
- clear_logs : Clear Logs
- server_control_commands : Execute Server Control Commands
- console_redirection : Access Console Redirection
- virtual_media : Access Virtual Media
- test_alerts : Test Alerts
- debug_commands : Execute Debug Commands

salt.modules.drac.**set_snmp**(*community*)
Configure SNMP community string

CLI Example:

```
salt dell drac.set_snmp [COMMUNITY]
salt dell drac.set_snmp public
```

salt.modules.drac.**syslog**(*server*, *enable=True*)
Configure syslog remote logging, by default syslog will automatically be enabled if a server is specified. However, if you want to disable syslog you will need to specify a server followed by False

CLI Example:

```
salt dell drac.syslog [SYSLOG IP] [ENABLE/DISABLE]
salt dell drac.syslog 0.0.0.0 False
```

salt.modules.drac.**system_info**()
Return System information

CLI Example:

```
salt dell drac.getsysinfo
```

22.16.59 salt.modules.drbd

DRBD administration module

salt.modules.drbd.**overview**()
Show status of the DRBD devices

CLI Example:

```
salt '*' drbd.overview
```

22.16.60 salt.modules.ebuild

Support for Portage

optdepends

- portage Python adapter

For now all package names *MUST* include the package category, i.e. `'vim'` will not work, `'app-editors/vim'` will.

salt.modules.ebuild.**check_db**(**names*, ***kwargs*)
New in version 0.17.0.

Returns a dict containing the following information for each specified package:

1. A key `found`, which will be a boolean value denoting if a match was found in the package database.
2. If `found` is `False`, then a second key called `suggestions` will be present, which will contain a list of possible matches. This list will be empty if the package name was specified in `category/pkgname` format, since the suggestions are only intended to disambiguate ambiguous package names (ones submitted without a category).

CLI Examples:

```
salt '*' pkg.check_db <package1> <package2> <package3>
```

salt.modules.ebuild.**check_extra_requirements**(*pkgname*, *pkgver*)
Check if the installed package already has the given requirements.

CLI Example:

```
salt '*' pkg.check_extra_requirements 'sys-devel/gcc' '~>4.1.2:4.1::gentoo[nls,fortran]'
```

salt.modules.ebuild.**depclean**(*name=None*, *slot=None*, *fromrepo=None*, *pkgs=None*)

Portage has a function to remove unused dependencies. If a package is provided, it will only removed the package if no other package depends on it.

name The name of the package to be cleaned.

slot Restrict the remove to a specific slot. Ignored if `name` is None.

fromrepo Restrict the remove to a specific slot. Ignored if `name` is None.

pkgs Clean multiple packages. `slot` and `fromrepo` arguments are ignored if this argument is present. Must be passed as a python list.

Return a list containing the removed packages:

CLI Example:

```
salt '*' pkg.depclean <package name>
```

salt.modules.ebuild.**ex_mod_init**(*low*)

If the config option `ebuild.enforce_nice_config` is set to True, this module will enforce a nice tree structure for /etc/portage/package.* configuration files.

New in version 0.17.0: Initial automatic enforcement added when pkg is used on a Gentoo system.

Changed in version 2014.1.0-Hydrogen: Configure option added to make this behaviour optional, defaulting to off.

See also:

`ebuild.ex_mod_init` is called automatically when a state invokes a pkg state on a Gentoo system. `salt.states.pkg.mod_init()`

`ebuild.ex_mod_init` uses `portage_config.enforce_nice_config` to do the lifting. `salt.modules.portage_config.enforce_nice_config()`

CLI Example:

```
salt '*' pkg.ex_mod_init
```

salt.modules.ebuild.**install**(*name=None*, *refresh=False*, *pkgs=None*, *sources=None*, *slot=None*, *fromrepo=None*, *uses=None*, *binhost=None*, ***kwargs*)

Install the passed package(s), add refresh=True to sync the portage tree before package is installed.

name The name of the package to be installed. Note that this parameter is ignored if either ``pkgs'' or ``sources'' is passed. Additionally, please note that this option can only be used to emerge a package from the portage tree. To install a tbz2 package manually, use the ``sources'' option described below.

CLI Example:

```
salt '*' pkg.install <package name>
```

refresh Whether or not to sync the portage tree before installing.

version Install a specific version of the package, e.g. 1.0.9-r1. Ignored if ``pkgs'' or ``sources'' is passed.

slot Similar to version, but specifies a valid slot to be installed. It will install the latest available version in the specified slot. Ignored if ``pkgs'' or ``sources'' or ``version'' is passed.

CLI Example:

```
salt '*' pkg.install sys-devel/gcc slot='4.4'
```

fromrepo Similar to slot, but specifies the repository from the package will be installed. It will install the latest available version in the specified repository. Ignored if ``pkgs'' or ``sources'' or ``version'' is passed.

CLI Example:

```
salt '*' pkg.install salt fromrepo='gentoo'
```

uses Similar to slot, but specifies a list of use flag. Ignored if ``pkgs'' or ``sources'' or ``version'' is passed.

CLI Example:

```
salt '*' pkg.install sys-devel/gcc uses='["nptl","-nossp"]'
```

Multiple Package Installation Options:

pkgs A list of packages to install from the portage tree. Must be passed as a python list.

CLI Example:

```
salt '*' pkg.install pkgs='["foo","bar","~category/package:slot::repository[use]"]'
```

sources A list of tbz2 packages to install. Must be passed as a list of dicts, with the keys being package names, and the values being the source URI or local path to the package.

CLI Example:

```
salt '*' pkg.install sources='[{"foo": "salt://foo.tbz2"},{"bar": "salt://bar.tbz2"}]'
```

binhost has two options try and force. try - tells emerge to try and install the package from a configured binhost. force - forces emerge to install the package from a binhost otherwise it fails out.

Returns a dict containing the new package names and versions:

```
{'<package>': {'old': '<old-version>',
               'new': '<new-version>'}}
```

salt.modules.ebuild.**latest_version**(**names*, ***kwargs*)

Return the latest version of the named package available for upgrade or installation. If more than one package name is specified, a dict of name/version pairs is returned.

If the latest version of a given package is already installed, an empty string will be returned for that package.

CLI Example:

```
salt '*' pkg.latest_version <package name>
salt '*' pkg.latest_version <package1> <package2> <package3> ...
```

salt.modules.ebuild.**list_pkgs**(*versions_as_list=False*, ***kwargs*)

List the packages currently installed in a dict:

```
{'<package_name>': '<version>'}
```

CLI Example:

```
salt '*' pkg.list_pkgs
```

salt.modules.ebuild.**list_upgrades**(*refresh=True*)

List all available package upgrades.

CLI Example:

```
salt '*' pkg.list_upgrades
```

salt.modules.ebuild.**porttree_matches**(*name*)

Returns a list containing the matches for a given package name from the portage tree. Note that the specific version of the package will not be provided for packages that have several versions in the portage tree, but rather the name of the package (i.e. ``dev-python/paramiko'').

salt.modules.ebuild.**purge**(*name=None, slot=None, fromrepo=None, pkgs=None, **kwargs*)

Portage does not have a purge, this function calls remove followed by depclean to emulate a purge process

name The name of the package to be deleted.

slot Restrict the remove to a specific slot. Ignored if name is None.

fromrepo Restrict the remove to a specific slot. Ignored if `name` is None.

Multiple Package Options:

pkgs Uninstall multiple packages. `slot` and `fromrepo` arguments are ignored if this argument is present. Must be passed as a python list.

New in version 0.16.0.

Returns a dict containing the changes.

CLI Example:

```
salt '*' pkg.purge <package name>
salt '*' pkg.purge <package name> slot=4.4
salt '*' pkg.purge <package1>,<package2>,<package3>
salt '*' pkg.purge pkgs='["foo", "bar"]'
```

salt.modules.ebuild.**refresh_db**()

Updates the portage tree (emerge --sync). Uses eix-sync if available.

CLI Example:

```
salt '*' pkg.refresh_db
```

salt.modules.ebuild.**remove**(*name=None, slot=None, fromrepo=None, pkgs=None, **kwargs*)

Remove packages via emerge --unmerge.

name The name of the package to be deleted.

slot Restrict the remove to a specific slot. Ignored if `name` is None.

fromrepo Restrict the remove to a specific slot. Ignored if `name` is None.

Multiple Package Options:

pkgs Uninstall multiple packages. `slot` and `fromrepo` arguments are ignored if this argument is present. Must be passed as a python list.

New in version 0.16.0.

Returns a dict containing the changes.

CLI Example:

```
salt '*' pkg.remove <package name>
salt '*' pkg.remove <package name> slot=4.4 fromrepo=gentoo
salt '*' pkg.remove <package1>,<package2>,<package3>
salt '*' pkg.remove pkgs='["foo", "bar"]'
```

salt.modules.ebuild.**update**(*pkg*, *slot=None*, *fromrepo=None*, *refresh=False*, *binhost=None*)
Updates the passed package (emerge --update package)

slot Restrict the update to a particular slot. It will update to the latest version within the slot.

fromrepo Restrict the update to a particular repository. It will update to the latest version within the repository.

binhost has two options try and force. try - tells emerge to try and install the package from a configured binhost. force - forces emerge to install the package from a binhost otherwise it fails out.

Return a dict containing the new package names and versions:

```
{'<package>': {'old': '<old-version>',
               'new': '<new-version>'}}
```

CLI Example:

```
salt '*' pkg.update <package name>
```

salt.modules.ebuild.**upgrade**(*refresh=True*, *binhost=None*)
Run a full system upgrade (emerge --update world)

binhost has two options try and force. try - tells emerge to try and install the package from a configured binhost. force - forces emerge to install the package from a binhost otherwise it fails out.

Return a dict containing the new package names and versions:

```
{'<package>': {'old': '<old-version>',
               'new': '<new-version>'}}
```

CLI Example:

```
salt '*' pkg.upgrade
```

salt.modules.ebuild.**upgrade_available**(*name*)
Check whether or not an upgrade is available for a given package

CLI Example:

```
salt '*' pkg.upgrade_available <package name>
```

salt.modules.ebuild.**version**(**names*, ***kwargs*)
Returns a string representing the package version or an empty string if not installed. If more than one package name is specified, a dict of name/version pairs is returned.

CLI Example:

```
salt '*' pkg.version <package name>
salt '*' pkg.version <package1> <package2> <package3> ...
```

salt.modules.ebuild.**version_clean**(*version*)
Clean the version string removing extra data.

CLI Example:

```
salt '*' pkg.version_clean <version_string>
```

salt.modules.ebuild.**version_cmp**(*pkg1*, *pkg2*)
Do a cmp-style comparison on two packages. Return -1 if pkg1 < pkg2, 0 if pkg1 == pkg2, and 1 if pkg1 > pkg2. Return None if there was a problem making the comparison.

CLI Example:

```
salt '*' pkg.version_cmp '0.2.4-0' '0.2.4.1-0'
```

22.16.61 salt.modules.eix

Support for Eix

salt.modules.eix.**sync**()
Sync portage/overlay trees and update the eix database

CLI Example:

```
salt '*' eix.sync
```

salt.modules.eix.**update**()
Update the eix database

CLI Example:

```
salt '*' eix.update
```

22.16.62 salt.modules.elasticsearch

Connection module for Elasticsearch

depends elasticsearch

salt.modules.elasticsearch.**delete**(*index*, *doc_type*, *id*, *hosts=None*, *profile='elasticsearch'*)
Delete the document specified by the id in the index.

CLI example:

```
salt myminion elasticsearch.delete grafana-dash dashboard mydash profile='grafana'
```

salt.modules.elasticsearch.**exists**(*index*, *id*, *doc_type='_all'*, *hosts=None*, *profile='elasticsearch'*)
Check for the existence of an elasticsearch document specified by id in the index.

CLI example:

```
salt myminion elasticsearch.exists grafana-dash mydash profile='grafana'
```

salt.modules.elasticsearch.**get**(*index*, *id*, *doc_type='_all'*, *hosts=None*, *profile='elasticsearch'*)
Get the contents of the specifed id from the index.

CLI example:

```
salt myminion elasticsearch.get grafana-dash mydash profile='grafana'
```

salt.modules.elasticsearch.**index**(*index*, *doc_type*, *body*, *id=None*, *hosts=None*, *profile='elasticsearch'*)
Create or update an index with the specified body for the specified id.

CLI example:

```
salt myminion elasticsearch.index grafana-dash dashboard '{"user":"guest","group":"guest","body"
```

22.16.63 salt.modules.environ

Support for getting and setting the environment variables of the current salt process.

salt.modules.environ.**get**(*key*, *default=''*)
Get a single salt process environment variable.

key String used as the key for environment lookup.

default If the key is not found in the environment, return this value. Default: ``'

CLI Example:

```
salt '*' environ.get foo
salt '*' environ.get baz default=False
```

salt.modules.environ.**has_value**(*key*, *value=None*)
Determine whether the key exists in the current salt process environment dictionary. Optionally compare the current value of the environment against the supplied value string.

key Must be a string. Used as key for environment lookup.

value: Optional. If key exists in the environment, compare the current value with this value. Return True if they are equal.

CLI Example:

```
salt '*' environ.has_value foo
```

salt.modules.environ.**item**(*keys*, *default=''*)
Get one or more salt process environment variables. Returns a dict.

keys Either a string or a list of strings that will be used as the keys for environment lookup.

default If the key is not found in the environment, return this value. Default: ``'

CLI Example:

```
salt '*' environ.item foo
salt '*' environ.item '[foo, baz]' default=None
```

salt.modules.environ.**items**()
Return a dict of the entire environment set for the salt process

CLI Example:

```
salt '*' environ.items
```

salt.modules.environ.**setenv**(*environ*, *false_unsets=False*, *clear_all=False*, *update_minion=False*)
Set multiple salt process environment variables from a dict. Returns a dict.

environ Must be a dict. The top-level keys of the dict are the names of the environment variables to set. Each key's value must be a string or False. Refer to the `false_unsets' parameter for behavior when a value set to False.

false_unsets If a key's value is False and false_unsets is True, then the key will be removed from the salt processes environment dict entirely. If a key's value is False and false_unsets is not True, then the key's value will be set to an empty string. Default: False

clear_all USE WITH CAUTION! This option can unset environment variables needed for salt to function properly. If clear_all is True, then any environment variables not defined in the environ dict will be deleted. Default: False

update_minion If True, apply these environ changes to the main salt-minion process. If False, the environ changes will only affect the current salt subprocess. Default: False

CLI Example:

```
salt '*' environ.setenv '{"foo": "bar", "baz": "quux"}'
salt '*' environ.setenv '{"a": "b", "c": False}' false_unsets=True
```

salt.modules.environ.**setval**(*key*, *val*, *false_unsets=False*)

Set a single salt process environment variable. Returns True on success.

key The environment key to set. Must be a string.

val The value to set. Must be a string or False. Refer to the `false_unsets' parameter for behavior when set to False.

false_unsets If val is False and false_unsets is True, then the key will be removed from the salt processes environment dict entirely. If val is False and false_unsets is not True, then the key's value will be set to an empty string. Default: False.

CLI Example:

```
salt '*' environ.setval foo bar
salt '*' environ.setval baz val=False false_unsets=True
```

22.16.64 salt.modules.eselect

Support for eselect, Gentoo's configuration and management tool.

salt.modules.eselect.**exec_action**(*module*, *action*, *module_parameter=None*, *action_parameter=None*, *state_only=False*)

Execute an arbitrary action on a module.

module name of the module to be executed

action name of the module's action to be run

module_parameter additional params passed to the defined module

action_parameter additional params passed to the defined action

state_only don't return any output but only the success/failure of the operation

CLI Example (updating the `php` implementation used for `apache2`):

```
salt '*' eselect.exec_action php update action_parameter='apache2'
```

salt.modules.eselect.**get_current_target**(*module*, *module_parameter=None*, *action_parameter=None*)

Get the currently selected target for the given module.

module name of the module to be queried for its current target

module_parameter additional params passed to the defined module

action_parameter additional params passed to the `show' action

CLI Example (current target of system-wide `java-vm`):

```
salt '*' eselect.get_current_target java-vm action_parameter='system'
```

CLI Example (current target of `kernel` symlink):

```
salt '*' eselect.get_current_target kernel
```

salt.modules.eselect.**get_modules**()
List available `eselect` modules.

CLI Example:

```
salt '*' eselect.get_modules
```

salt.modules.eselect.**get_target_list**(*module*)
List available targets for the given module.

module name of the module to be queried for its targets

CLI Example:

```
salt '*' eselect.get_target_list kernel
```

salt.modules.eselect.**set_target**(*module*, *target*, *module_parameter=None*, *action_parameter=None*)
Set the target for the given module. Target can be specified by index or name.

module name of the module for which a target should be set

target name of the target to be set for this module

module_parameter additional params passed to the defined module

action_parameter additional params passed to the defined action

CLI Example (setting target of system-wide `java-vm`):

```
salt '*' eselect.set_target java-vm icedtea-bin-7 action_parameter='system'
```

CLI Example (setting target of `kernel` symlink):

```
salt '*' eselect.set_target kernel linux-3.17.5-gentoo
```

22.16.65 salt.modules.etcd_mod

Execution module to work with etcd

depends

- python-etcd

In order to use an etcd server, a profile should be created in the master configuration file:

```
my_etd_config:
  etcd.host: 127.0.0.1
  etcd.port: 4001
```

It is technically possible to configure etcd without using a profile, but this is not considered to be a best practice, especially when multiple etcd servers or clusters are available.

```
etcd.host: 127.0.0.1
etcd.port: 4001
```

salt.modules.etcd_mod.**get_**(*key*, *recurse=False*, *profile=None*)
New in version 2014.7.0.

Get a value from etcd, by direct path

CLI Examples:

```
salt myminion etcd.get /path/to/key
salt myminion etcd.get /path/to/key profile=my_etcd_config
salt myminion etcd.get /path/to/key recurse=True profile=my_etcd_config
```

salt.modules.etcd_mod.**ls_**(*path='/'*, *profile=None*)
New in version 2014.7.0.

Return all keys and dirs inside a specific path

CLI Example:

```
salt myminion etcd.ls /path/to/dir/
salt myminion etcd.ls /path/to/dir/ profile=my_etcd_config
```

salt.modules.etcd_mod.**rm_**(*key*, *recurse=False*, *profile=None*)
New in version 2014.7.0.

Delete a key from etcd

CLI Example:

```
salt myminion etcd.rm /path/to/key
salt myminion etcd.rm /path/to/key profile=my_etcd_config
salt myminion etcd.rm /path/to/dir recurse=True profile=my_etcd_config
```

salt.modules.etcd_mod.**set_**(*key*, *value*, *profile=None*)
New in version 2014.7.0.

Set a value in etcd, by direct path

CLI Example:

```
salt myminion etcd.set /path/to/key value
salt myminion etcd.set /path/to/key value profile=my_etcd_config
```

salt.modules.etcd_mod.**tree**(*path='/'*, *profile=None*)
New in version 2014.7.0.

Recurse through etcd and return all values

CLI Example:

```
salt myminion etcd.tree
salt myminion etcd.tree profile=my_etcd_config
salt myminion etcd.tree /path/to/keys profile=my_etcd_config
```

22.16.66 salt.modules.event

Use the *Salt Event System* to fire events from the master to the minion and vice-versa.

salt.modules.event.**fire**(*data*, *tag*)
Fire an event on the local minion event bus. Data must be formed as a dict.

CLI Example:

```
salt '*' event.fire '{"data":"my event data"}' 'tag'
```

salt.modules.event.**fire_master**(*data*, *tag*, *preload=None*)
Fire an event off up to the master server

CLI Example:

```
salt '*' event.fire_master '{"data":"my event data"}' 'tag'
```

salt.modules.event.**send**(*tag*, *data=None*, *preload=None*, *with_env=False*, *with_grains=False*, *with_pillar=False*, ***kwargs*)

Send an event to the Salt Master

New in version 2014.7.0.

Parameters

- **tag** -- A tag to give the event. Use slashes to create a namespace for related events. E.g., `myco/build/buildserver1/start`, `myco/build/buildserver1/success`, `myco/build/buildserver1/failure`.
- **data** -- A dictionary of data to send in the event. This is free-form. Send any data points that are needed for whoever is consuming the event. Arguments on the CLI are interpreted as YAML so complex data structures are possible.
- **with_env** (Specify `True` to include all environment variables, or specify a list of strings of variable names to include.) -- Include environment variables from the current shell environment in the event data as `environ`.. This is a short-hand for working with systems that seed the environment with relevant data such as Jenkins.
- **with_grains** (Specify `True` to include all grains, or specify a list of strings of grain names to include.) -- Include grains from the current minion in the event data as `grains`.
- **with_pillar** (Specify `True` to include all Pillar values, or specify a list of strings of Pillar keys to include. It is a best-practice to only specify a relevant subset of Pillar data.) -- Include Pillar values from the current minion in the event data as `pillar`. Remember Pillar data is often sensitive data so be careful. This is useful for passing ephemeral Pillar values through an event. Such as passing the `pillar={}` kwarg in `state.sls` from the Master, through an event on the Minion, then back to the Master.
- **kwargs** -- Any additional keyword arguments passed to this function will be interpreted as key-value pairs and included in the event data. This provides a convenient alternative to YAML for simple values.

CLI Example:

```
salt-call event.send myco/mytag foo=Foo bar=Bar
salt-call event.send 'myco/mytag' '{foo: Foo, bar: Bar}'
```

A convenient way to allow Jenkins to execute `salt-call` is via sudo. The following rule in sudoers will allow the `jenkins` user to run only the following command.

`/etc/sudoers` (allow preserving the environment):

```
jenkins ALL=(ALL) NOPASSWD:SETENV: /usr/bin/salt-call event.send*
```

Call Jenkins via sudo (preserve the environment):

```
sudo -E salt-call event.send myco/jenkins/build/success with_env=[BUILD_ID, BUILD_URL, GIT_BRANC
```

22.16.67 salt.modules.extfs

Module for managing ext2/3/4 file systems

salt.modules.extfs.**attributes**(*device*, *args=None*)
Return attributes from dumpe2fs for a specified device

CLI Example:

```
salt '*' extfs.attributes /dev/sda1
```

salt.modules.extfs.**blocks**(*device*, *args=None*)
Return block and inode info from dumpe2fs for a specified device

CLI Example:

```
salt '*' extfs.blocks /dev/sda1
```

salt.modules.extfs.**dump**(*device*, *args=None*)
Return all contents of dumpe2fs for a specified device

CLI Example:

```
salt '*' extfs.dump /dev/sda1
```

salt.modules.extfs.**mkfs**(*device*, *fs_type*, ***kwargs*)
Create a file system on the specified device

CLI Example:

```
salt '*' extfs.mkfs /dev/sda1 fs_type=ext4 opts='acl,noexec'
```

Valid options are:

- **block_size**: 1024, 2048 or 4096
- **check**: check for bad blocks
- **direct**: use direct IO
- **ext_opts**: extended file system options (comma-separated)
- **fragment_size**: size of fragments
- **force**: setting force to True will cause mke2fs to specify the -F option twice (it is already set once); this is truly dangerous
- **blocks_per_group**: number of blocks in a block group
- **number_of_groups**: ext4 option for a virtual block group
- **bytes_per_inode**: set the bytes/inode ratio
- **inode_size**: size of the inode
- **journal**: set to True to create a journal (default on ext3/4)
- **journal_opts**: options for the fs journal (comma separated)
- **blocks_file**: read bad blocks from file
- **label**: label to apply to the file system
- **reserved**: percentage of blocks reserved for super-user
- **last_dir**: last mounted directory
- **test**: set to True to not actually create the file system (mke2fs -n)
- **number_of_inodes**: override default number of inodes

- **creator_os**: override ``creator operating system'' field
- **opts**: mount options (comma separated)
- **revision**: set the filesystem revision (default 1)
- **super**: write superblock and group descriptors only
- **fs_type**: set the filesystem type (REQUIRED)
- **usage_type**: how the filesystem is going to be used
- **uuid**: set the UUID for the file system

See the `mke2fs(8)` manpage for a more complete description of these options.

`salt.modules.extfs.`**`tune`**(*device*, ***kwargs*)

Set attributes for the specified device (using tune2fs)

CLI Example:

```
salt '*' extfs.tune /dev/sda1 force=True label=wildstallyns opts='acl,noexec'
```

Valid options are:

- **max**: max mount count
- **count**: mount count
- **error**: error behavior
- **extended_opts**: extended options (comma separated)
- **force**: force, even if there are errors (set to True)
- **group**: group name or gid that can use the reserved blocks
- **interval**: interval between checks
- **journal**: set to True to create a journal (default on ext3/4)
- **journal_opts**: options for the fs journal (comma separated)
- **label**: label to apply to the file system
- **reserved**: percentage of blocks reserved for super-user
- **last_dir**: last mounted directory
- **opts**: mount options (comma separated)
- **feature**: set or clear a feature (comma separated)
- **mmp_check**: mmp check interval
- **reserved**: reserved blocks count
- **quota_opts**: quota options (comma separated)
- **time**: time last checked
- **user**: user or uid who can use the reserved blocks
- **uuid**: set the UUID for the file system

See the `mke2fs(8)` manpage for a more complete description of these options.

22.16.68 salt.modules.file

Manage information about regular files, directories, and special files on the minion, set/read user, group, mode, and data

salt.modules.file.**access**(*path*, *mode*)

New in version 2014.1.0.

Test whether the Salt process has the specified access to the file. One of the following modes must be specified:

CLI Example:

```
salt '*' file.access /path/to/file f
salt '*' file.access /path/to/file x
```

salt.modules.file.**append**(*path*, **args*, ***kwargs*)

New in version 0.9.5.

Append text to the end of a file

path path to file

***args** strings to append to file

CLI Example:

```
salt '*' file.append /etc/motd \
        "With all thine offerings thou shalt offer salt." \
        "Salt is what makes things taste bad when it isn't in them."
```

Attention

If you need to pass a string to append and that string contains an equal sign, you **must** include the argument name, args. For example:

```
salt '*' file.append /etc/motd args='cheese=spam'

salt '*' file.append /etc/motd args="['cheese=spam','spam=cheese']"
```

---

salt.modules.file.**basename**(*path*)

Returns the final component of a pathname

New in version 2015.5.0.

This can be useful at the CLI but is frequently useful when scripting.

```
{%- set filename = salt['file.basename'](source_file) %}
```

CLI Example:

```
salt '*' file.basename 'test/test.config'
```

salt.modules.file.**blockreplace**(*path*, *marker_start='#-- start managed zone --`*, *marker_end='#-- end managed zone --`*, *content='`*, *append_if_not_found=False*, *prepend_if_not_found=False*, *backup='.bak'*, *dry_run=False*, *show_changes=True*)

New in version 2014.1.0.

Replace content of a text block in a file, delimited by line markers

A block of content delimited by comments can help you manage several lines entries without worrying about old entries removal.

Note: This function will store two copies of the file in-memory (the original version and the edited version) in order to detect changes and only edit the targeted file if necessary.

path Filesystem path to the file to be edited

marker_start The line content identifying a line as the start of the content block. Note that the whole line containing this marker will be considered, so whitespace or extra content before or after the marker is included in final output

marker_end The line content identifying a line as the end of the content block. Note that the whole line containing this marker will be considered, so whitespace or extra content before or after the marker is included in final output

content The content to be used between the two lines identified by marker_start and marker_stop.

append_if_not_found [False] If markers are not found and set to `True` then, the markers and content will be appended to the file.

prepend_if_not_found [False] If markers are not found and set to `True` then, the markers and content will be prepended to the file.

backup The file extension to use for a backup of the file if any edit is made. Set to `False` to skip making a backup.

dry_run Don't make any edits to the file.

show_changes Output a unified diff of the old file and the new file. If `False`, return a boolean if any changes were made.

CLI Example:

```
salt '*' file.blockreplace /etc/hosts '#-- start managed zone foobar : DO NOT EDIT --' \
'#-- end managed zone foobar --' $'10.0.1.1 foo.foobar\n10.0.1.2 bar.foobar' True
```

`salt.modules.file.`**`check_file_meta`**(*name*, *sfn*, *source*, *source_sum*, *user*, *group*, *mode*, *saltenv*, *template=None*, *contents=None*)

Check for the changes in the file metadata.

CLI Example:

```
salt '*' file.check_file_meta /etc/httpd/conf.d/httpd.conf salt://http/httpd.conf '{hash_type: '
```

Note: Supported hash types include sha512, sha384, sha256, sha224, sha1, and md5.

`salt.modules.file.`**`check_hash`**(*path*, *file_hash*)

Check if a file matches the given hash string

Returns true if the hash matched, otherwise false. Raises ValueError if the hash was not formatted correctly.

path A file path

hash A string in the form <hash_type>:<hash_value>. For example: `md5:e138491e9d5b97023cea823fe17bac22`

CLI Example:

```
salt '*' file.check_hash /etc/fstab md5:<md5sum>
```

salt.modules.file.**check_managed**(*name*, *source*, *source_hash*, *user*, *group*, *mode*, *template*, *context*, *defaults*, *saltenv*, *contents=None*, ***kwargs*)

Check to see what changes need to be made for a file

CLI Example:

```
salt '*' file.check_managed /etc/httpd/conf.d/httpd.conf salt://http/httpd.conf '{hash_type: 'md
```

salt.modules.file.**check_managed_changes**(*name*, *source*, *source_hash*, *user*, *group*, *mode*, *template*, *context*, *defaults*, *saltenv*, *contents=None*, ***kwargs*)

Return a dictionary of what changes need to be made for a file

CLI Example:

```
salt '*' file.check_managed_changes /etc/httpd/conf.d/httpd.conf salt://http/httpd.conf '{hash_t
```

salt.modules.file.**check_perms**(*name*, *ret*, *user*, *group*, *mode*, *follow_symlinks=False*)

Check the permissions on files and chown if needed

CLI Example:

```
salt '*' file.check_perms /etc/sudoers '{}' root root 400
```

Changed in version 2014.1.3: `follow_symlinks` option added

salt.modules.file.**chgrp**(*path*, *group*)

Change the group of a file

path path to the file or directory

group group owner

CLI Example:

```
salt '*' file.chgrp /etc/passwd root
```

salt.modules.file.**chown**(*path*, *user*, *group*)

Chown a file, pass the file the desired user and group

path path to the file or directory

user user owner

group group owner

CLI Example:

```
salt '*' file.chown /etc/passwd root root
```

salt.modules.file.**comment**(*path*, *regex*, *char='#'*, *backup='.bak'*)

Deprecated since version 0.17.0: Use `replace()` instead.

Comment out specified lines in a file

path The full path to the file to be edited

regex A regular expression used to find the lines that are to be commented; this pattern will be wrapped in parenthesis and will move any preceding/trailing `^` or `$` characters outside the parenthesis (e.g., the pattern `^foo$` will be rewritten as `^(foo)$`)

char [#] The character to be inserted at the beginning of a line in order to comment it out

backup [`.bak`] The file will be backed up before edit with this file extension

> **Warning**: This backup will be overwritten each time `sed` / `comment` / `uncomment` is called. Meaning the backup will only be useful after the first invocation.

CLI Example:

```
salt '*' file.comment /etc/modules pcspkr
```

`salt.modules.file.`**`contains`**(*path*, *text*)

Deprecated since version 0.17.0: Use `search()` instead.

Return `True` if the file at `path` contains `text`

CLI Example:

```
salt '*' file.contains /etc/crontab 'mymaintenance.sh'
```

`salt.modules.file.`**`contains_glob`**(*path*, *glob_expr*)

Deprecated since version 0.17.0: Use `search()` instead.

Return `True` if the given glob matches a string in the named file

CLI Example:

```
salt '*' file.contains_glob /etc/foobar '*cheese*'
```

`salt.modules.file.`**`contains_regex`**(*path*, *regex*, *lchar=''*)

Deprecated since version 0.17.0: Use `search()` instead.

Return True if the given regular expression matches on any line in the text of a given file.

If the lchar argument (leading char) is specified, it will strip *lchar* from the left side of each line before trying to match

CLI Example:

```
salt '*' file.contains_regex /etc/crontab
```

`salt.modules.file.`**`contains_regex_multiline`**(*path*, *regex*)

Deprecated since version 0.17.0: Use `search()` instead.

Return True if the given regular expression matches anything in the text of a given file

Traverses multiple lines at a time, via the salt BufferedReader (reads in chunks)

CLI Example:

```
salt '*' file.contains_regex_multiline /etc/crontab '^maint'
```

`salt.modules.file.`**`copy`**(*src*, *dst*, *recurse=False*, *remove_existing=False*)

Copy a file or directory from source to dst

In order to copy a directory, the recurse flag is required, and will by default overwrite files in the destination with the same path, and retain all other existing files. (similar to cp -r on unix)

remove_existing will remove all files in the target directory, and then copy files from the source.

CLI Example:

```
salt '*' file.copy /path/to/src /path/to/dst
salt '*' file.copy /path/to/src_dir /path/to/dst_dir recurse=True
salt '*' file.copy /path/to/src_dir /path/to/dst_dir recurse=True remove_existing=True
```

salt.modules.file.**delete_backup**(*path*, *backup_id*)
New in version 0.17.0.

Delete a previous version of a file that was backed up using Salt's *file state backup* system.

path The path on the minion to check for backups

backup_id The numeric id for the backup you wish to delete, as found using `file.list_backups`

CLI Example:

```
salt '*' file.restore_backup /foo/bar/baz.txt 0
```

salt.modules.file.**directory_exists**(*path*)
Tests to see if path is a valid directory. Returns True/False.

CLI Example:

```
salt '*' file.directory_exists /etc
```

salt.modules.file.**dirname**(*path*)
Returns the directory component of a pathname

New in version 2015.5.0.

This can be useful at the CLI but is frequently useful when scripting.

```
{%- from salt['file.dirname'](tpldir) + '/vars.jinja' import parent_vars %}
```

CLI Example:

```
salt '*' file.dirname 'test/path/filename.config'
```

salt.modules.file.**diskusage**(*path*)
Recursively calculate disk usage of path and return it in bytes

CLI Example:

```
salt '*' file.diskusage /path/to/check
```

salt.modules.file.**extract_hash**(*hash_fn*, *hash_type='sha256'*, *file_name=''*)
This routine is called from the `file.managed` state to pull a hash from a remote file. Regular expressions are used line by line on the `source_hash` file, to find a potential candidate of the indicated hash type. This avoids many problems of arbitrary file lay out rules. It specifically permits pulling hash codes from debian `*.dsc` files.

For example:

```
openerp_7.0-latest-1.tar.gz:
  file.managed:
    - name: /tmp/openerp_7.0-20121227-075624-1_all.deb
    - source: http://nightly.openerp.com/7.0/nightly/deb/openerp_7.0-20121227-075624-1.tar.gz
    - source_hash: http://nightly.openerp.com/7.0/nightly/deb/openerp_7.0-20121227-075624-1.dsc
```

CLI Example:

```
salt '*' file.extract_hash /etc/foo sha512 /path/to/hash/file
```

salt.modules.file.**file_exists**(*path*)
Tests to see if path is a valid file. Returns True/False.

CLI Example:

```
salt '*' file.file_exists /etc/passwd
```

salt.modules.file.**find**(*path*, **args*, ***kwargs*)

Approximate the Unix `find(1)` command and return a list of paths that meet the specified criteria.

The options include match criteria:

```
name    = path-glob                 # case sensitive
iname   = path-glob                 # case insensitive
regex   = path-regex                # case sensitive
iregex  = path-regex                # case insensitive
type    = file-types                # match any listed type
user    = users                     # match any listed user
group   = groups                    # match any listed group
size    = [+-]number[size-unit]     # default unit = byte
mtime   = interval                  # modified since date
grep    = regex                     # search file contents
```

and/or actions:

```
delete [= file-types]               # default type = 'f'
exec    = command [arg ...]         # where {} is replaced by pathname
print  [= print-opts]
```

and/or depth criteria:

```
maxdepth = maximum depth to transverse in path
mindepth = minimum depth to transverse before checking files or directories
```

The default action is `print=path`

`path-glob`:

```
*               = match zero or more chars
?               = match any char
[abc]           = match a, b, or c
[!abc] or [^abc] = match anything except a, b, and c
[x-y]           = match chars x through y
[!x-y] or [^x-y] = match anything except chars x through y
{a,b,c}         = match a or b or c
```

`path-regex`: a Python Regex (regular expression) pattern to match pathnames

`file-types`: a string of one or more of the following:

```
a: all file types
b: block device
c: character device
d: directory
p: FIFO (named pipe)
f: plain file
l: symlink
s: socket
```

`users`: a space and/or comma separated list of user names and/or uids

`groups`: a space and/or comma separated list of group names and/or gids

`size-unit`:

```
    b: bytes
    k: kilobytes
    m: megabytes
    g: gigabytes
    t: terabytes
```

interval:

```
    [<num>w] [<num>d] [<num>h] [<num>m] [<num>s]

    where:
        w: week
        d: day
        h: hour
        m: minute
        s: second
```

print-opts: a comma and/or space separated list of one or more of the following:

```
    group: group name
    md5:   MD5 digest of file contents
    mode:  file permissions (as integer)
    mtime: last modification time (as time_t)
    name:  file basename
    path:  file absolute path
    size:  file size in bytes
    type:  file type
    user:  user name
```

CLI Examples:

```
    salt '*' file.find / type=f name=\*.bak size=+10m
    salt '*' file.find /var mtime=+30d size=+10m print=path,size,mtime
    salt '*' file.find /var/log name=\*.[0-9] mtime=+30d size=+10m delete
```

salt.modules.file.**get_devmm**(*name*)

Get major/minor info from a device

CLI Example:

```
    salt '*' file.get_devmm /dev/chr
```

salt.modules.file.**get_diff**(*minionfile*, *masterfile*, *env=None*, *saltenv='base'*)

Return unified diff of file compared to file on master

CLI Example:

```
    salt '*' file.get_diff /home/fred/.vimrc salt://users/fred/.vimrc
```

salt.modules.file.**get_gid**(*path*, *follow_symlinks=True*)

Return the id of the group that owns a given file

path file or directory of which to get the gid

follow_symlinks indicated if symlinks should be followed

CLI Example:

```
    salt '*' file.get_gid /etc/passwd
```

Changed in version 0.16.4: `follow_symlinks` option added

salt.modules.file.**get_group**(*path*, *follow_symlinks=True*)
Return the group that owns a given file

path file or directory of which to get the group

follow_symlinks indicated if symlinks should be followed

CLI Example:

```
salt '*' file.get_group /etc/passwd
```

Changed in version 0.16.4: `follow_symlinks` option added

salt.modules.file.**get_hash**(*path*, *form='sha256'*, *chunk_size=65536*)
Get the hash sum of a file

This is better than `get_sum` for the following reasons:

- It does not read the entire file into memory.
- **It does not return a string on error. The returned value of** `get_sum` cannot really be trusted since it is vulnerable to collisions: `get_sum(..., 'xyz') == 'Hash xyz not supported'`

path path to the file or directory

form desired sum format

chunk_size amount to sum at once

CLI Example:

```
salt '*' file.get_hash /etc/shadow
```

salt.modules.file.**get_managed**(*name*, *template*, *source*, *source_hash*, *user*, *group*, *mode*, *saltenv*, *context*, *defaults*, ***kwargs*)
Return the managed file data for file.managed

name location where the file lives on the server

template template format

source managed source file

source_hash hash of the source file

user user owner

group group owner

mode file mode

context variables to add to the environment

default default values of for context_dict

CLI Example:

```
salt '*' file.get_managed /etc/httpd/conf.d/httpd.conf jinja salt://http/httpd.conf '{hash_type:
```

salt.modules.file.**get_mode**(*path*, *follow_symlinks=True*)
Return the mode of a file

path file or directory of which to get the mode

follow_symlinks indicated if symlinks should be followed

CLI Example:

```
salt '*' file.get_mode /etc/passwd
```

Changed in version 2014.1.0: `follow_symlinks` option added

`salt.modules.file.`**`get_selinux_context`**(*path*)
Get an SELinux context from a given path

CLI Example:

```
salt '*' file.get_selinux_context /etc/hosts
```

`salt.modules.file.`**`get_sum`**(*path*, *form='sha256'*)
Return the checksum for the given file. The following checksum algorithms are supported:

- md5
- sha1
- sha224
- sha256 **(default)**
- sha384
- sha512

path path to the file or directory

form desired sum format

CLI Example:

```
salt '*' file.get_sum /etc/passwd sha512
```

`salt.modules.file.`**`get_uid`**(*path*, *follow_symlinks=True*)
Return the id of the user that owns a given file

path file or directory of which to get the uid

follow_symlinks indicated if symlinks should be followed

CLI Example:

```
salt '*' file.get_uid /etc/passwd
```

Changed in version 0.16.4: `follow_symlinks` option added

`salt.modules.file.`**`get_user`**(*path*, *follow_symlinks=True*)
Return the user that owns a given file

path file or directory of which to get the user

follow_symlinks indicated if symlinks should be followed

CLI Example:

```
salt '*' file.get_user /etc/passwd
```

Changed in version 0.16.4: `follow_symlinks` option added

`salt.modules.file.`**`gid_to_group`**(*gid*)
Convert the group id to the group name on this system

gid gid to convert to a group name

CLI Example:

```
salt '*' file.gid_to_group 0
```

salt.modules.file.**grep**(*path*, *pattern*, **args*)

Grep for a string in the specified file

Note: This function's return value is slated for refinement in future versions of Salt

path A file path

pattern A string. For example: `test a[0-5]`

args grep options. For example: `" -v" " -i -B2"`

CLI Example:

```
salt '*' file.grep /etc/passwd nobody
salt '*' file.grep /etc/sysconfig/network-scripts/ifcfg-eth0 ipaddr " -i"
salt '*' file.grep /etc/sysconfig/network-scripts/ifcfg-eth0 ipaddr " -i -B2"
salt '*' file.grep "/etc/sysconfig/network-scripts/*" ipaddr " -i -l"
```

salt.modules.file.**group_to_gid**(*group*)

Convert the group to the gid on this system

group group to convert to its gid

CLI Example:

```
salt '*' file.group_to_gid root
```

salt.modules.file.**is_blkdev**(*name*)

Check if a file exists and is a block device.

CLI Example:

```
salt '*' file.is_blkdev /dev/blk
```

salt.modules.file.**is_chrdev**(*name*)

Check if a file exists and is a character device.

CLI Example:

```
salt '*' file.is_chrdev /dev/chr
```

salt.modules.file.**is_fifo**(*name*)

Check if a file exists and is a FIFO.

CLI Example:

```
salt '*' file.is_fifo /dev/fifo
```

salt.modules.file.**is_link**(*path*)

Check if the path is a symlink

CLI Example:

```
salt '*' file.is_link /path/to/link
```

salt.modules.file.**join**(**args*)

Return a normalized file system path for the underlying OS

New in version 2014.7.0.

This can be useful at the CLI but is frequently useful when scripting combining path variables:

```
{% set www_root = '/var' %}
{% set app_dir = 'myapp' %}

myapp_config:
  file:
    - managed
    - name: {{ salt['file.join'](www_root, app_dir, 'config.yaml') }}
```

CLI Example:

```
salt '*' file.join '/' 'usr' 'local' 'bin'
```

salt.modules.file.**lchown**(*path*, *user*, *group*)

Chown a file, pass the file the desired user and group without following symlinks.

path path to the file or directory

user user owner

group group owner

CLI Example:

```
salt '*' file.chown /etc/passwd root root
```

salt.modules.file.**link**(*src*, *path*)

New in version 2014.1.0.

Create a hard link to a file

CLI Example:

```
salt '*' file.link /path/to/file /path/to/link
```

salt.modules.file.**list_backups**(*path*, *limit=None*)

New in version 0.17.0.

Lists the previous versions of a file backed up using Salt's *file state backup* system.

path The path on the minion to check for backups

limit Limit the number of results to the most recent N backups

CLI Example:

```
salt '*' file.list_backups /foo/bar/baz.txt
```

salt.modules.file.**list_backups_dir**(*path*, *limit=None*)

Lists the previous versions of a directory backed up using Salt's *file state backup* system.

path The directory on the minion to check for backups

limit Limit the number of results to the most recent N backups

CLI Example:

```
salt '*' file.list_backups_dir /foo/bar/baz/
```

salt.modules.file.**lstat**(*path*)

New in version 2014.1.0.

Returns the lstat attributes for the given file or dir. Does not support symbolic links.

CLI Example:

```
salt '*' file.lstat /path/to/file
```

salt.modules.file.**makedirs_**(*path*, *user=None*, *group=None*, *mode=None*)

Ensure that the directory containing this path is available.

Note: The path must end with a trailing slash otherwise the directory/directories will be created up to the parent directory. For example if path is `/opt/code`, then it would be treated as `/opt/` but if the path ends with a trailing slash like `/opt/code/`, then it would be treated as `/opt/code/`.

CLI Example:

```
salt '*' file.makedirs /opt/code/
```

salt.modules.file.**makedirs_perms**(*name*, *user=None*, *group=None*, *mode=`0755'*)

Taken and modified from os.makedirs to set user, group and mode for each directory created.

CLI Example:

```
salt '*' file.makedirs_perms /opt/code
```

salt.modules.file.**manage_file**(*name*, *sfn*, *ret*, *source*, *source_sum*, *user*, *group*, *mode*, *saltenv*, *backup*, *makedirs=False*, *template=None*, *show_diff=True*, *contents=None*, *dir_mode=None*, *follow_symlinks=True*)

Checks the destination against what was retrieved with get_managed and makes the appropriate modifications (if necessary).

name location to place the file

sfn location of cached file on the minion

This is the path to the file stored on the minion. This file is placed on the minion using cp.cache_file. If the hash sum of that file matches the source_sum, we do not transfer the file to the minion again.

This file is then grabbed and if it has template set, it renders the file to be placed into the correct place on the system using salt.files.utils.copyfile()

source file reference on the master

source_hash sum hash for source

user user owner

group group owner

backup backup_mode

makedirs make directories if they do not exist

template format of templating

show_diff Include diff in state return

contents: contents to be placed in the file

dir_mode mode for directories created with makedirs

CLI Example:

```
salt '*' file.manage_file /etc/httpd/conf.d/httpd.conf '' '{}' salt://http/httpd.conf '{hash_typ
```

Changed in version 2014.7.0: `follow_symlinks` option added

`salt.modules.file.`**`mkdir`**(*dir_path*, *user=None*, *group=None*, *mode=None*)
Ensure that a directory is available.

CLI Example:

```
salt '*' file.mkdir /opt/jetty/context
```

`salt.modules.file.`**`mknod`**(*name*, *ntype*, *major=0*, *minor=0*, *user=None*, *group=None*, *mode=`0600'*)
New in version 0.17.0.

Create a block device, character device, or fifo pipe. Identical to the gnu mknod.

CLI Examples:

```
salt '*' file.mknod /dev/chr c 180 31
salt '*' file.mknod /dev/blk b 8 999
salt '*' file.nknod /dev/fifo p
```

`salt.modules.file.`**`mknod_blkdev`**(*name*, *major*, *minor*, *user=None*, *group=None*, *mode=`0660'*)
New in version 0.17.0.

Create a block device.

CLI Example:

```
salt '*' file.mknod_blkdev /dev/blk 8 999
```

`salt.modules.file.`**`mknod_chrdev`**(*name*, *major*, *minor*, *user=None*, *group=None*, *mode=`0660'*)
New in version 0.17.0.

Create a character device.

CLI Example:

```
salt '*' file.mknod_chrdev /dev/chr 180 31
```

`salt.modules.file.`**`mknod_fifo`**(*name*, *user=None*, *group=None*, *mode=`0660'*)
New in version 0.17.0.

Create a FIFO pipe.

CLI Example:

```
salt '*' file.mknod_fifo /dev/fifo
```

`salt.modules.file.`**`move`**(*src*, *dst*)
Move a file or directory

CLI Example:

```
salt '*' file.move /path/to/src /path/to/dst
```

`salt.modules.file.`**`normpath`**(*path*)
Returns Normalize path, eliminating double slashes, etc.

New in version 2015.5.0.

This can be useful at the CLI but is frequently useful when scripting.

```
{%- from salt['file.normpath'](tpldir + '/../vars.jinja') import parent_vars %}
```

CLI Example:

```
salt '*' file.normpath 'a/b/c/..'
```

salt.modules.file.**open_files**(*by_pid=False*)

Return a list of all physical open files on the system.

CLI Examples:

```
salt '*' file.open_files
salt '*' file.open_files by_pid=True
```

salt.modules.file.**pardir**()

Return the relative parent directory path symbol for underlying OS

New in version 2014.7.0.

This can be useful when constructing Salt Formulas.

```
{% set pardir = salt['file.pardir']() %}
{% set final_path = salt['file.join']('subdir', pardir, 'confdir') %}
```

CLI Example:

```
salt '*' file.pardir
```

salt.modules.file.**patch**(*originalfile*, *patchfile*, *options=''*, *dry_run=False*)

New in version 0.10.4.

Apply a patch to a file

Equivalent to:

```
patch <options> <originalfile> <patchfile>
```

originalfile The full path to the file or directory to be patched

patchfile A patch file to apply to `originalfile`

options Options to pass to patch.

CLI Example:

```
salt '*' file.patch /opt/file.txt /tmp/file.txt.patch
```

salt.modules.file.**path_exists_glob**(*path*)

Tests to see if path after expansion is a valid path (file or directory). Expansion allows usage of ? * and character ranges []. Tilde expansion is not supported. Returns True/False.

New in version Hellium.

CLI Example:

```
salt '*' file.path_exists_glob /etc/pam*/pass*
```

salt.modules.file.**prepend**(*path*, **args*, ***kwargs*)

New in version 2014.7.0.

Prepend text to the beginning of a file

path path to file

***args** strings to prepend to the file

CLI Example:

```
salt '*' file.prepend /etc/motd \
        "With all thine offerings thou shalt offer salt." \
        "Salt is what makes things taste bad when it isn't in them."
```

Attention

If you need to pass a string to append and that string contains an equal sign, you **must** include the argument name, args. For example:

```
salt '*' file.prepend /etc/motd args='cheese=spam'

salt '*' file.prepend /etc/motd args="['cheese=spam','spam=cheese']"
```

`salt.modules.file.`**`psed`**(*path*, *before*, *after*, *limit=''*, *backup='.bak'*, *flags='gMS'*, *escape_all=False*, *multi=False*)

Deprecated since version 0.17.0: Use `replace()` instead.

Make a simple edit to a file (pure Python version)

Equivalent to:

```
sed <backup> <options> "/<limit>/ s/<before>/<after>/<flags> <file>"
```

path The full path to the file to be edited

before A pattern to find in order to replace with `after`

after Text that will replace `before`

limit [`''`] An initial pattern to search for before searching for `before`

backup [`.bak`] The file will be backed up before edit with this file extension; **WARNING**: each time `sed`/`comment`/`uncomment` is called will overwrite this backup

flags [`gMS`]

Flags to modify the search. Valid values are:

- `g`: Replace all occurrences of the pattern, not just the first.
- `I`: Ignore case.
- `L`: Make `\w`, `\W`, `\b`, `\B`, `\s` and `\S` dependent on the locale.
- `M`: Treat multiple lines as a single line.
- `S`: Make `.` match all characters, including newlines.
- `U`: Make `\w`, `\W`, `\b`, `\B`, `\d`, `\D`, `\s` and `\S` dependent on Unicode.
- `X`: Verbose (whitespace is ignored).

multi: **`False`** If True, treat the entire file as a single line

Forward slashes and single quotes will be escaped automatically in the `before` and `after` patterns.

CLI Example:

```
salt '*' file.sed /etc/httpd/httpd.conf 'LogLevel warn' 'LogLevel info'
```

salt.modules.file.**readdir**(*path*)

New in version 2014.1.0.

Return a list containing the contents of a directory

CLI Example:

```
salt '*' file.readdir /path/to/dir/
```

salt.modules.file.**readlink**(*path*)

New in version 2014.1.0.

Return the path that a symlink points to

CLI Example:

```
salt '*' file.readlink /path/to/link
```

salt.modules.file.**remove**(*path*)

Remove the named file

CLI Example:

```
salt '*' file.remove /tmp/foo
```

salt.modules.file.**rename**(*src*, *dst*)

Rename a file or directory

CLI Example:

```
salt '*' file.rename /path/to/src /path/to/dst
```

salt.modules.file.**replace**(*path*, *pattern*, *repl*, *count=0*, *flags=0*, *bufsize=1*, *append_if_not_found=False*, *prepend_if_not_found=False*, *not_found_content=None*, *backup='.bak'*, *dry_run=False*, *search_only=False*, *show_changes=True*)

New in version 0.17.0.

Replace occurrences of a pattern in a file

This is a pure Python implementation that wraps Python's `sub()`.

path Filesystem path to the file to be edited

pattern Python's regular expression search https://docs.python.org/2/library/re.html

repl The replacement text

count Maximum number of pattern occurrences to be replaced

flags (list or int) A list of flags defined in the *re module documentation*. Each list item should be a string that will correlate to the human-friendly flag name. E.g., `['IGNORECASE', 'MULTILINE']`. Note: multiline searches must specify `file` as the `bufsize` argument below.

bufsize (int or str) How much of the file to buffer into memory at once. The default value `1` processes one line at a time. The special value `file` may be specified which will read the entire file into memory before processing. Note: multiline searches must specify `file` buffering.

append_if_not_found New in version 2014.7.0.

If pattern is not found and set to `True` then, the content will be appended to the file. Default is `False`

prepend_if_not_found New in version 2014.7.0.

If pattern is not found and set to `True` then, the content will be appended to the file. Default is `False`

not_found_content New in version 2014.7.0.

Content to use for append/prepend if not found. If None (default), uses `repl`. Useful when `repl` uses references to group in pattern.

backup The file extension to use for a backup of the file before editing. Set to `False` to skip making a backup. Default is `.bak`

dry_run Don't make any edits to the file, Default is `False`

search_only Just search for the pattern; ignore the replacement; stop on the first match. Default is `False`

show_changes Output a unified diff of the old file and the new file. If `False` return a boolean if any changes were made. Default is `True`

Note: Using this option will store two copies of the file in-memory (the original version and the edited version) in order to generate the diff.

If an equal sign (`=`) appears in an argument to a Salt command it is interpreted as a keyword argument in the format `key=val`. That processing can be bypassed in order to pass an equal sign through to the remote shell command by manually specifying the kwarg:

```
salt '*' file.replace /path/to/file pattern='=' repl=':'
salt '*' file.replace /path/to/file pattern="bind-address\s*=" repl='bind-address:'
```

CLI Examples:

```
salt '*' file.replace /etc/httpd/httpd.conf pattern='LogLevel warn' repl='LogLevel info'
salt '*' file.replace /some/file pattern='before' repl='after' flags='[MULTILINE, IGNORECASE]'
```

`salt.modules.file.`**`restore_backup`**(*path*, *backup_id*)

New in version 0.17.0.

Restore a previous version of a file that was backed up using Salt's *file state backup* system.

path The path on the minion to check for backups

backup_id The numeric id for the backup you wish to restore, as found using `file.list_backups`

CLI Example:

```
salt '*' file.restore_backup /foo/bar/baz.txt 0
```

`salt.modules.file.`**`restorecon`**(*path*, *recursive=False*)

Reset the SELinux context on a given path

CLI Example:

```
salt '*' file.restorecon /home/user/.ssh/authorized_keys
```

`salt.modules.file.`**`rmdir`**(*path*)

New in version 2014.1.0.

Remove the specified directory. Fails if a directory is not empty.

CLI Example:

```
salt '*' file.rmdir /tmp/foo/
```

salt.modules.file.**search**(*path*, *pattern*, *flags=0*, *bufsize=1*)

New in version 0.17.0.

Search for occurrences of a pattern in a file

Params are identical to `replace()`.

CLI Example:

```
salt '*' file.search /etc/crontab 'mymaintenance.sh'
```

salt.modules.file.**sed**(*path*, *before*, *after*, *limit=''*, *backup='.bak'*, *options='-r -e'*, *flags='g'*, *escape_all=False*, *negate_match=False*)

Deprecated since version 0.17.0: Use `replace()` instead.

Make a simple edit to a file

Equivalent to:

```
sed <backup> <options> "/<limit>/ s/<before>/<after>/<flags> <file>"
```

path The full path to the file to be edited

before A pattern to find in order to replace with `after`

after Text that will replace `before`

limit [`''`] An initial pattern to search for before searching for `before`

backup [`.bak`] The file will be backed up before edit with this file extension; **WARNING**: each time `sed`/`comment`/`uncomment` is called will overwrite this backup

options [`-r -e`] Options to pass to sed

flags [`g`] Flags to modify the sed search; e.g., `i` for case-insensitive pattern matching

negate_match [False] Negate the search command (`!`)

New in version 0.17.0.

Forward slashes and single quotes will be escaped automatically in the `before` and `after` patterns.

CLI Example:

```
salt '*' file.sed /etc/httpd/httpd.conf 'LogLevel warn' 'LogLevel info'
```

salt.modules.file.**sed_contains**(*path*, *text*, *limit=''*, *flags='g'*)

Deprecated since version 0.17.0: Use `search()` instead.

Return True if the file at `path` contains `text`. Utilizes sed to perform the search (line-wise search).

Note: the `p` flag will be added to any flags you pass in.

CLI Example:

```
salt '*' file.contains /etc/crontab 'mymaintenance.sh'
```

salt.modules.file.**seek_read**(*path*, *size*, *offset*)

New in version 2014.1.0.

Seek to a position on a file and read it

path path to file

seek amount to read at once

offset offset to start into the file

CLI Example:

```
salt '*' file.seek_read /path/to/file 4096 0
```

salt.modules.file.**seek_write**(*path*, *data*, *offset*)

New in version 2014.1.0.

Seek to a position on a file and write to it

path path to file

data data to write to file

offset position in file to start writing

CLI Example:

```
salt '*' file.seek_write /path/to/file 'some data' 4096
```

salt.modules.file.**set_mode**(*path*, *mode*)

Set the mode of a file

path file or directory of which to set the mode

mode mode to set the path to

CLI Example:

```
salt '*' file.set_mode /etc/passwd 0644
```

salt.modules.file.**set_selinux_context**(*path*, *user=None*, *role=None*, *type=None*, *range=None*)

Set a specific SELinux label on a given path

CLI Example:

```
salt '*' file.set_selinux_context path <role> <type> <range>
```

salt.modules.file.**source_list**(*source*, *source_hash*, *saltenv*)

Check the source list and return the source to use

CLI Example:

```
salt '*' file.source_list salt://http/httpd.conf '{hash_type: 'md5', 'hsum': <md5sum>}' base
```

salt.modules.file.**stats**(*path*, *hash_type=None*, *follow_symlinks=True*)

Return a dict containing the stats for a given file

CLI Example:

```
salt '*' file.stats /etc/passwd
```

salt.modules.file.**statvfs**(*path*)

New in version 2014.1.0.

Perform a statvfs call against the filesystem that the file resides on

CLI Example:

```
salt '*' file.statvfs /path/to/file
```

salt.modules.file.**symlink**(*src*, *path*)

Create a symbolic link to a file

CLI Example:

```
salt '*' file.symlink /path/to/file /path/to/link
```

salt.modules.file.**touch**(*name*, *atime=None*, *mtime=None*)

New in version 0.9.5.

Just like the `touch` command, create a file if it doesn't exist or simply update the atime and mtime if it already does.

atime: Access time in Unix epoch time

mtime: Last modification in Unix epoch time

CLI Example:

```
salt '*' file.touch /var/log/emptyfile
```

salt.modules.file.**truncate**(*path*, *length*)

New in version 2014.1.0.

Seek to a position on a file and delete everything after that point

path path to file

length offset into file to truncate

CLI Example:

```
salt '*' file.truncate /path/to/file 512
```

salt.modules.file.**uid_to_user**(*uid*)

Convert a uid to a user name

uid uid to convert to a username

CLI Example:

```
salt '*' file.uid_to_user 0
```

salt.modules.file.**uncomment**(*path*, *regex*, *char='#'*, *backup='.bak'*)

Deprecated since version 0.17.0: Use `replace()` instead.

Uncomment specified commented lines in a file

path The full path to the file to be edited

regex A regular expression used to find the lines that are to be uncommented. This regex should not include the comment character. A leading `^` character will be stripped for convenience (for easily switching between comment() and uncomment()).

char [`#`] The character to remove in order to uncomment a line

backup [`.bak`] The file will be backed up before edit with this file extension; **WARNING**: each time `sed`/`comment`/`uncomment` is called will overwrite this backup

CLI Example:

```
salt '*' file.uncomment /etc/hosts.deny 'ALL: PARANOID'
```

salt.modules.file.**user_to_uid**(*user*)

Convert user name to a uid

user user name to convert to its uid

CLI Example:

```
salt '*' file.user_to_uid root
```

salt.modules.file.**write**(*path*, **args*, ***kwargs*)

New in version 2014.7.0.

Write text to a file, overwriting any existing contents.

path path to file

****args*** strings to write to the file

CLI Example:

```
salt '*' file.write /etc/motd \
        "With all thine offerings thou shalt offer salt."
```

Attention

If you need to pass a string to append and that string contains an equal sign, you **must** include the argument name, args. For example:

```
salt '*' file.write /etc/motd args='cheese=spam'

salt '*' file.write /etc/motd args="['cheese=spam','spam=cheese']"
```

22.16.69 salt.modules.firewalld

Support for firewalld

salt.modules.firewalld.**add_service**(*name*, *zone=None*, *permanent=True*)

Add a service for zone. If zone is omitted, default zone will be used.

CLI Example:

```
salt '*' firewalld.add_service ssh
```

To assign a service to a specific zone

```
salt '*' firewalld.add_service ssh my_zone
```

salt.modules.firewalld.**default_zone**()

Print default zone for connections and interfaces

CLI Example:

```
salt '*' firewalld.default_zone
```

salt.modules.firewalld.**delete_service**(*name*, *restart=True*)

Delete an existing service

CLI Example:

```
salt '*' firewalld.delete_service my_service
```

By default firewalld will be reloaded. However, to avoid reloading you need to specify the restart as False

```
salt '*' firewalld.delete_service my_service False
```

salt.modules.firewalld.**delete_zone**(*zone*, *restart=True*)
Delete an existing zone

CLI Example:

```
salt '*' firewalld.delete_zone my_zone
```

By default firewalld will be reloaded. However, to avoid reloading you need to specify the restart as False

```
salt '*' firewalld.delete_zone my_zone False
```

salt.modules.firewalld.**get_icmp_types**()
Print predefined icmptypes

CLI Example:

```
salt '*' firewalld.get_icmp_types
```

salt.modules.firewalld.**get_services**()
Print predefined services

CLI Example:

```
salt '*' firewalld.get_services
```

salt.modules.firewalld.**get_zones**()
Print predefined zones

CLI Example:

```
salt '*' firewalld.get_zones
```

salt.modules.firewalld.**list_all**(*zone=None*)
List everything added for or enabled in a zone

CLI Example:

```
salt '*' firewalld.list_all
```

List a specific zone

```
salt '*' firewalld.list_all my_zone
```

salt.modules.firewalld.**list_services**(*zone=None*)
List services added for zone as a space separated list. If zone is omitted, default zone will be used.

CLI Example:

```
salt '*' firewalld.list_services
```

List a specific zone

```
salt '*' firewalld.list_services my_zone
```

salt.modules.firewalld.**list_zones**()
List everything added for or enabled in all zones

CLI Example:

```
salt '*' firewalld.list_zones
```

salt.modules.firewalld.**new_service**(*name*, *restart=True*)
Add a new service

CLI Example:

```
salt '*' firewalld.new_service my_service
```

By default firewalld will be reloaded. However, to avoid reloading you need to specify the restart as False

```
salt '*' firewalld.new_service my_service False
```

salt.modules.firewalld.**new_zone**(*zone*, *restart=True*)
Add a new zone

CLI Example:

```
salt '*' firewalld.new_zone my_zone
```

By default firewalld will be reloaded. However, to avoid reloading you need to specify the restart as False

```
salt '*' firewalld.new_zone my_zone False
```

salt.modules.firewalld.**remove_service**(*name*, *zone=None*, *permanent=True*)
Remove a service from zone. This option can be specified multiple times. If zone is omitted, default zone will be used.

CLI Example:

```
salt '*' firewalld.remove_service ssh
```

To remove a service from a specific zone

```
salt '*' firewalld.remove_service ssh dmz
```

salt.modules.firewalld.**set_default_zone**(*zone*)
Set default zone

CLI Example:

```
salt '*' firewalld.set_default_zone damian
```

salt.modules.firewalld.**version**()
Return version from firewall-cmd

CLI Example:

```
salt '*' firewalld.version
```

22.16.70 salt.modules.freebsd_sysctl

Module for viewing and modifying sysctl parameters

salt.modules.freebsd_sysctl.**assign**(*name*, *value*)
Assign a single sysctl parameter for this minion

CLI Example:

```
salt '*' sysctl.assign net.inet.icmp.icmplim 50
```

salt.modules.freebsd_sysctl.**get**(*name*)
Return a single sysctl parameter for this minion

CLI Example:

```
salt '*' sysctl.get hw.physmem
```

salt.modules.freebsd_sysctl.**persist**(*name*, *value*, *config='/etc/sysctl.conf'*)
Assign and persist a simple sysctl parameter for this minion

CLI Example:

```
salt '*' sysctl.persist net.inet.icmp.icmplim 50
salt '*' sysctl.persist coretemp_load NO config=/boot/loader.conf
```

salt.modules.freebsd_sysctl.**show**(*config_file=False*)
Return a list of sysctl parameters for this minion

CLI Example:

```
salt '*' sysctl.show
```

22.16.71 salt.modules.freebsdjail

The jail module for FreeBSD

salt.modules.freebsdjail.**fstab**(*jail*)
Display contents of a fstab(5) file defined in specified jail's configuration. If no file is defined, return False.

CLI Example:

```
salt '*' jail.fstab <jail name>
```

salt.modules.freebsdjail.**get_enabled**()
Return which jails are set to be run

CLI Example:

```
salt '*' jail.get_enabled
```

salt.modules.freebsdjail.**is_enabled**()
See if jail service is actually enabled on boot

CLI Example:

```
salt '*' jail.is_enabled <jail name>
```

salt.modules.freebsdjail.**restart**(*jail=''*)
Restart the specified jail or all, if none specified

CLI Example:

```
salt '*' jail.restart [<jail name>]
```

salt.modules.freebsdjail.**show_config**(*jail*)
Display specified jail's configuration

CLI Example:

```
salt '*' jail.show_config <jail name>
```

salt.modules.freebsdjail.**start**(*jail=''*)
Start the specified jail or all, if none specified

CLI Example:

```
salt '*' jail.start [<jail name>]
```

salt.modules.freebsdjail.**status**(*jail*)
See if specified jail is currently running

CLI Example:

```
salt '*' jail.status <jail name>
```

salt.modules.freebsdjail.**stop**(*jail=''*)
Stop the specified jail or all, if none specified

CLI Example:

```
salt '*' jail.stop [<jail name>]
```

salt.modules.freebsdjail.**sysctl**()
Dump all jail related kernel states (sysctl)

CLI Example:

```
salt '*' jail.sysctl
```

22.16.72 salt.modules.freebsdkmod

Module to manage FreeBSD kernel modules

salt.modules.freebsdkmod.**available**()
Return a list of all available kernel modules

CLI Example:

```
salt '*' kmod.available
```

salt.modules.freebsdkmod.**check_available**(*mod*)
Check to see if the specified kernel module is available

CLI Example:

```
salt '*' kmod.check_available vmm
```

salt.modules.freebsdkmod.**is_loaded**(*mod*)
Check to see if the specified kernel module is loaded

CLI Example:

```
salt '*' kmod.is_loaded vmm
```

salt.modules.freebsdkmod.**load**(*mod, persist=False*)
Load the specified kernel module

mod Name of the module to add

persist Write the module to sysrc kld_modules to make it load on system reboot

CLI Example:

```
salt '*' kmod.load bhyve
```

salt.modules.freebsdkmod.**lsmod**()
Return a dict containing information about currently loaded modules

CLI Example:

```
salt '*' kmod.lsmod
```

salt.modules.freebsdkmod.**mod_list**(*only_persist=False*)
Return a list of the loaded module names

CLI Example:

```
salt '*' kmod.mod_list
```

salt.modules.freebsdkmod.**remove**(*mod*, *persist=False*)
Remove the specified kernel module

CLI Example:

```
salt '*' kmod.remove vmm
```

22.16.73 salt.modules.freebsdpkg

Remote package support using `pkg_add(1)`

Warning: This module has been completely rewritten. Up to and including version 0.17.0, it supported `pkg_add(1)`, but checked for the existence of a pkgng local database and, if found, would provide some of pkgng's functionality. The rewrite of this module has removed all pkgng support, and moved it to the `pkgng` execution module. For versions <= 0.17.0, the documentation here should not be considered accurate. If your Minion is running one of these versions, then the documentation for this module can be viewed using the `sys.doc` function:

```
salt bsdminion sys.doc pkg
```

This module acts as the default package provider for FreeBSD 9 and older. If you need to use pkgng on a FreeBSD 9 system, you will need to override the `pkg` provider by setting the `providers` parameter in your Minion config file, in order to use pkgng.

```
providers:
  pkg: pkgng
```

More information on pkgng support can be found in the documentation for the `pkgng` module.

This module will respect the `PACKAGEROOT` and `PACKAGESITE` environment variables, if set, but these values can also be overridden in several ways:

1. **Salt configuration parameters**. The configuration parameters `freebsdpkg.PACKAGEROOT` and `freebsdpkg.PACKAGESITE` are recognized. These config parameters are looked up using `config.get` and can thus be specified in the Master config file, Grains, Pillar, or in the Minion config file. Example:

```
freebsdpkg.PACKAGEROOT: ftp://ftp.freebsd.org/
freebsdpkg.PACKAGESITE: ftp://ftp.freebsd.org/pub/FreeBSD/ports/ia64/packages-9-stable/Latest/
```

2. **CLI arguments**. Both the `packageroot` (used interchangeably with `fromrepo` for API compatibility) and `packagesite` CLI arguments are recognized, and override their config counterparts from section 1 above.

```
        salt -G 'os:FreeBSD' pkg.install zsh fromrepo=ftp://ftp2.freebsd.org/
        salt -G 'os:FreeBSD' pkg.install zsh packageroot=ftp://ftp2.freebsd.org/
        salt -G 'os:FreeBSD' pkg.install zsh packagesite=ftp://ftp2.freebsd.org/pub/FreeBSD/ports/ia

    .. note::

        These arguments can also be passed through in states:

        .. code-block:: yaml

            zsh:
              pkg.installed:
                - fromrepo: ftp://ftp2.freebsd.org/
```

`salt.modules.freebsdpkg.`**`file_dict`**(**packages*)

List the files that belong to a package, grouped by package. Not specifying any packages will return a list of _every_ file on the system's package database (not generally recommended).

CLI Examples:

```
salt '*' pkg.file_list httpd
salt '*' pkg.file_list httpd postfix
salt '*' pkg.file_list
```

`salt.modules.freebsdpkg.`**`file_list`**(**packages*)

List the files that belong to a package. Not specifying any packages will return a list of _every_ file on the system's package database (not generally recommended).

CLI Examples:

```
salt '*' pkg.file_list httpd
salt '*' pkg.file_list httpd postfix
salt '*' pkg.file_list
```

`salt.modules.freebsdpkg.`**`install`**(*name=None*, *refresh=False*, *fromrepo=None*, *pkgs=None*, *sources=None*, ***kwargs*)

Install package(s) using `pkg_add(1)`

name The name of the package to be installed.

refresh Whether or not to refresh the package database before installing.

fromrepo or packageroot Specify a package repository from which to install. Overrides the system default, as well as the PACKAGEROOT environment variable.

packagesite Specify the exact directory from which to install the remote package. Overrides the PACKAGESITE environment variable, if present.

Multiple Package Installation Options:

pkgs A list of packages to install from a software repository. Must be passed as a python list.

CLI Example:

```
        salt '*' pkg.install pkgs='["foo", "bar"]'
```

sources A list of packages to install. Must be passed as a list of dicts, with the keys being package names, and the values being the source URI or local path to the package.

CLI Example:

```
        salt '*' pkg.install sources='[{"foo": "salt://foo.deb"}, {"bar": "salt://bar.deb"}]'
```

Return a dict containing the new package names and versions:

```
{'<package>': {'old': '<old-version>',
               'new': '<new-version>'}}
```

CLI Example:

```
salt '*' pkg.install <package name>
```

`salt.modules.freebsdpkg.`**`latest_version`**`(*names, **kwargs)`

`pkg_add(1)` is not capable of querying for remote packages, so this function will always return results as if there is no package available for install or upgrade.

CLI Example:

```
salt '*' pkg.latest_version <package name>
salt '*' pkg.latest_version <package1> <package2> <package3> ...
```

`salt.modules.freebsdpkg.`**`list_pkgs`**`(versions_as_list=False, with_origin=False, **kwargs)`

List the packages currently installed as a dict:

```
{'<package_name>': '<version>'}
```

with_origin [False] Return a nested dictionary containing both the origin name and version for each installed package.

New in version 2014.1.0.

CLI Example:

```
salt '*' pkg.list_pkgs
```

`salt.modules.freebsdpkg.`**`refresh_db`**`()`

`pkg_add(1)` does not use a local database of available packages, so this function simply returns `True`. it exists merely for API compatibility.

CLI Example:

```
salt '*' pkg.refresh_db
```

`salt.modules.freebsdpkg.`**`remove`**`(name=None, pkgs=None, **kwargs)`

Remove packages using `pkg_delete(1)`

name The name of the package to be deleted.

Multiple Package Options:

pkgs A list of packages to delete. Must be passed as a python list. The `name` parameter will be ignored if this option is passed.

New in version 0.16.0.

Returns a dict containing the changes.

CLI Example:

```
salt '*' pkg.remove <package name>
salt '*' pkg.remove <package1>,<package2>,<package3>
salt '*' pkg.remove pkgs='["foo", "bar"]'
```

salt.modules.freebsdpkg.**upgrade**()
Upgrades are not supported with `pkg_add(1)`. This function is included for API compatibility only and always returns an empty dict.

CLI Example:

```
salt '*' pkg.upgrade
```

salt.modules.freebsdpkg.**version**(**names*, ***kwargs*)
Returns a string representing the package version or an empty string if not installed. If more than one package name is specified, a dict of name/version pairs is returned.

with_origin [False] Return a nested dictionary containing both the origin name and version for each specified package.

New in version 2014.1.0.

CLI Example:

```
salt '*' pkg.version <package name>
salt '*' pkg.version <package1> <package2> <package3> ...
```

22.16.74 salt.modules.freebsdports

Install software from the FreeBSD `ports(7)` system

New in version 2014.1.0.

This module allows you to install ports using `BATCH=yes` to bypass configuration prompts. It is recommended to use the `ports state` to install ports, but it it also possible to use this module exclusively from the command line.

```
salt minion-id ports.config security/nmap IPV6=off
salt minion-id ports.install security/nmap
```

salt.modules.freebsdports.**config**(*name*, *reset=False*, ***kwargs*)
Modify configuration options for a given port. Multiple options can be specified. To see the available options for a port, use `ports.showconfig`.

name The port name, in `category/name` format

reset [False] If `True`, runs a `make rmconfig` for the port, clearing its configuration before setting the desired options

CLI Examples:

```
salt '*' ports.config security/nmap IPV6=off
```

salt.modules.freebsdports.**deinstall**(*name*)
De-install a port.

CLI Example:

```
salt '*' ports.deinstall security/nmap
```

salt.modules.freebsdports.**install**(*name*, *clean=True*)
Install a port from the ports tree. Installs using `BATCH=yes` for non-interactive building. To set config options for a given port, use `ports.config`.

clean [True] If `True`, cleans after installation. Equivalent to running `make install clean BATCH=yes`.

Note: It may be helpful to run this function using the `-t` option to set a higher timeout, since compiling a port may cause the Salt command to exceed the default timeout.

CLI Example:

```
salt -t 1200 '*' ports.install security/nmap
```

`salt.modules.freebsdports.`**`list_all`**`()`

Lists all ports available.

CLI Example:

```
salt '*' ports.list_all
```

Warning: Takes a while to run, and returns a **LOT** of output

`salt.modules.freebsdports.`**`rmconfig`**`(`*name*`)`

Clear the cached options for the specified port; run a `make rmconfig`

name The name of the port to clear

CLI Example:

```
salt '*' ports.rmconfig security/nmap
```

`salt.modules.freebsdports.`**`search`**`(`*name*`)`

Search for matches in the ports tree. Globs are supported, and the category is optional

CLI Examples:

```
salt '*' ports.search 'security/*'
salt '*' ports.search 'security/n*'
salt '*' ports.search nmap
```

Warning: Takes a while to run

`salt.modules.freebsdports.`**`showconfig`**`(`*name*, *default=False*, *dict_return=False*`)`

Show the configuration options for a given port.

default [False] Show the default options for a port (not necessarily the same as the current configuration)

dict_return [False] Instead of returning the output of `make showconfig`, return the data in an dictionary

CLI Example:

```
salt '*' ports.showconfig security/nmap
salt '*' ports.showconfig security/nmap default=True
```

`salt.modules.freebsdports.`**`update`**`(`*extract=False*`)`

Update the ports tree

extract [False] If `True`, runs a `portsnap extract` after fetching, should be used for first-time installation of the ports tree.

CLI Example:

```
salt '*' ports.update
```

22.16.75 salt.modules.freebsdservice

The service module for FreeBSD

salt.modules.freebsdservice.**available**(*name*)
Check that the given service is available.

CLI Example:

```
salt '*' service.available sshd
```

salt.modules.freebsdservice.**disable**(*name*, ***kwargs*)
Disable the named service to start at boot

Arguments the same as for enable()

CLI Example:

```
salt '*' service.disable <service name>
```

salt.modules.freebsdservice.**disabled**(*name*)
Return True if the named service is enabled, false otherwise

CLI Example:

```
salt '*' service.disabled <service name>
```

salt.modules.freebsdservice.**enable**(*name*, ***kwargs*)
Enable the named service to start at boot

name service name

config [/etc/rc.conf] Config file for managing service. If config value is empty string, then /etc/rc.conf.d/<service> used. See man rc.conf(5) for details.

Also service.config variable can be used to change default.

CLI Example:

```
salt '*' service.enable <service name>
```

salt.modules.freebsdservice.**enabled**(*name*, ***kwargs*)
Return True if the named service is enabled, false otherwise

name Service name

CLI Example:

```
salt '*' service.enabled <service name>
```

salt.modules.freebsdservice.**get_all**()
Return a list of all available services

CLI Example:

```
salt '*' service.get_all
```

salt.modules.freebsdservice.**get_disabled**()
Return what services are available but not enabled to start at boot

CLI Example:

```
salt '*' service.get_disabled
```

salt.modules.freebsdservice.**get_enabled**()
Return what services are set to run on boot

CLI Example:

```
salt '*' service.get_enabled
```

salt.modules.freebsdservice.**missing**(*name*)
The inverse of service.available. Returns `True` if the specified service is not available, otherwise returns `False`.

CLI Example:

```
salt '*' service.missing sshd
```

salt.modules.freebsdservice.**reload_**(*name*)
Restart the named service

CLI Example:

```
salt '*' service.reload <service name>
```

salt.modules.freebsdservice.**restart**(*name*)
Restart the named service

CLI Example:

```
salt '*' service.restart <service name>
```

salt.modules.freebsdservice.**start**(*name*)
Start the specified service

CLI Example:

```
salt '*' service.start <service name>
```

salt.modules.freebsdservice.**status**(*name*, *sig=None*)
Return the status for a service (True or False).

name Name of service

CLI Example:

```
salt '*' service.status <service name>
```

salt.modules.freebsdservice.**stop**(*name*)
Stop the specified service

CLI Example:

```
salt '*' service.stop <service name>
```

22.16.76 salt.modules.fsutils

Run-time utilities

22.16.77 salt.modules.gem

Manage ruby gems.

salt.modules.gem.**install**(*gems*, *ruby=None*, *gem_bin=None*, *runas=None*, *version=None*, *rdoc=False*, *ri=False*, *pre_releases=False*, *proxy=None*)

Installs one or several gems.

gems The gems to install

gem_bin [None] Full path to `gem` binary to use.

ruby [None] If RVM or rbenv are installed, the ruby version and gemset to use. Ignored if `gem_bin` is specified.

runas [None] The user to run gem as.

version [None] Specify the version to install for the gem. Doesn't play nice with multiple gems at once

rdoc [False] Generate RDoc documentation for the gem(s).

ri [False] Generate RI documentation for the gem(s).

pre_releases Include pre-releases in the available versions

proxy [None] Use the specified HTTP proxy server for all outgoing traffic. Format: http://hostname[:port]

CLI Example:

```
salt '*' gem.install vagrant

salt '*' gem.install redphone gem_bin=/opt/sensu/embedded/bin/gem
```

salt.modules.gem.**list_**(*prefix=''*, *ruby=None*, *runas=None*, *gem_bin=None*)

List locally installed gems.

prefix : Only list gems when the name matches this prefix.

gem_bin [None] Full path to `gem` binary to use.

ruby [None] If RVM or rbenv are installed, the ruby version and gemset to use. Ignored if `gem_bin` is specified.

runas [None] The user to run gem as.

CLI Example:

```
salt '*' gem.list
```

salt.modules.gem.**sources_add**(*source_uri*, *ruby=None*, *runas=None*, *gem_bin=None*)

Add a gem source.

source_uri The source URI to add.

gem_bin [None] Full path to `gem` binary to use.

ruby [None] If RVM or rbenv are installed, the ruby version and gemset to use. Ignored if `gem_bin` is specified.

runas [None] The user to run gem as.

CLI Example:

```
salt '*' gem.sources_add http://rubygems.org/
```

salt.modules.gem.**sources_list**(*ruby=None*, *runas=None*, *gem_bin=None*)

List the configured gem sources.

gem_bin [None] Full path to `gem` binary to use.

ruby [None] If RVM or rbenv are installed, the ruby version and gemset to use. Ignored if `gem_bin` is specified.

runas [None] The user to run gem as.

CLI Example:

```
salt '*' gem.sources_list
```

`salt.modules.gem.`**`sources_remove`**(*source_uri*, *ruby=None*, *runas=None*, *gem_bin=None*)
Remove a gem source.

source_uri The source URI to remove.

gem_bin [None] Full path to `gem` binary to use.

ruby [None] If RVM or rbenv are installed, the ruby version and gemset to use. Ignored if `gem_bin` is specified.

runas [None] The user to run gem as.

CLI Example:

```
salt '*' gem.sources_remove http://rubygems.org/
```

`salt.modules.gem.`**`uninstall`**(*gems*, *ruby=None*, *runas=None*, *gem_bin=None*)
Uninstall one or several gems.

gems The gems to uninstall.

gem_bin [None] Full path to `gem` binary to use.

ruby [None] If RVM or rbenv are installed, the ruby version and gemset to use. Ignored if `gem_bin` is specified.

runas [None] The user to run gem as.

CLI Example:

```
salt '*' gem.uninstall vagrant
```

`salt.modules.gem.`**`update`**(*gems*, *ruby=None*, *runas=None*, *gem_bin=None*)
Update one or several gems.

gems The gems to update.

gem_bin [None] Full path to `gem` binary to use.

ruby [None] If RVM or rbenv are installed, the ruby version and gemset to use. Ignored if `gem_bin` is specified.

runas [None] The user to run gem as.

CLI Example:

```
salt '*' gem.update vagrant
```

`salt.modules.gem.`**`update_system`**(*version=''*, *ruby=None*, *runas=None*, *gem_bin=None*)
Update rubygems.

version [(newest)] The version of rubygems to install.

gem_bin [None] Full path to `gem` binary to use.

ruby [None] If RVM or rbenv are installed, the ruby version and gemset to use. Ignored if `gem_bin` is specified.

runas [None] The user to run gem as.

CLI Example:

```
salt '*' gem.update_system
```

22.16.78 salt.modules.genesis

Module for managing container and VM images

New in version 2014.7.0.

salt.modules.genesis.**avail_platforms**()
Return which platforms are available

CLI Example:

```
salt myminion genesis.avail_platforms
```

salt.modules.genesis.**bootstrap**(*platform*, *root*, *img_format='dir'*, *fs_format='ext2'*, *arch=None*, *flavor=None*, *repo_url=None*, *static_qemu=None*)
Create an image for a specific platform.

Please note that this function *MUST* be run as root, as images that are created make files belonging to root.

platform Which platform to use to create the image. Currently supported platforms are rpm, deb and pacman.

root Local path to create the root of the image filesystem.

img_format Which format to create the image in. By default, just copies files into a directory on the local filesystem (`dir`). Future support will exist for `sparse`.

fs_format When using a non-`dir` img_format, which filesystem to format the image to. By default, `ext2`.

arch Architecture to install packages for, if supported by the underlying bootstrap tool. Currently only used for deb.

flavor Which flavor of operating system to install. This correlates to a specific directory on the distribution repositories. For instance, `wheezy` on Debian.

repo_url Mainly important for Debian-based repos. Base URL for the mirror to install from. (e.x.: http://ftp.debian.org/debian/)

static_qemu Local path to the static qemu binary required for this arch. (e.x.: /usr/bin/qemu-amd64-static)

pkg_confs The location of the conf files to copy into the image, to point the installer to the right repos and configuration.

CLI Examples:

```
salt myminion genesis.bootstrap pacman /root/arch
salt myminion genesis.bootstrap rpm /root/redhat
salt myminion genesis.bootstrap deb /root/wheezy arch=amd64              flavor=wheezy static_qem
```

salt.modules.genesis.**pack**(*name*, *root*, *path=None*, *pack_format='tar'*, *compress='bzip2'*)
Pack up a directory structure, into a specific format

CLI Examples:

```
salt myminion genesis.pack centos /root/centos
salt myminion genesis.pack centos /root/centos pack_format='tar'
```

salt.modules.genesis.**unpack**(*name*, *dest=None*, *path=None*, *pack_format='tar'*, *compress='bz2'*)
Unpack an image into a directory structure

CLI Example:

```
salt myminion genesis.unpack centos /root/centos
```

22.16.79 salt.modules.gentoo_service

Top level package command wrapper, used to translate the os detected by grains to the correct service manager

salt.modules.gentoo_service.**available**(*name*)
Returns `True` if the specified service is available, otherwise returns `False`.

CLI Example:

```
salt '*' service.available sshd
```

salt.modules.gentoo_service.**disable**(*name*, ***kwargs*)
Disable the named service to start at boot

CLI Example:

```
salt '*' service.disable <service name>
```

salt.modules.gentoo_service.**disabled**(*name*)
Return True if the named service is enabled, false otherwise

CLI Example:

```
salt '*' service.disabled <service name>
```

salt.modules.gentoo_service.**enable**(*name*, ***kwargs*)
Enable the named service to start at boot

CLI Example:

```
salt '*' service.enable <service name>
```

salt.modules.gentoo_service.**enabled**(*name*, ***kwargs*)
Return True if the named service is enabled, false otherwise

CLI Example:

```
salt '*' service.enabled <service name>
```

salt.modules.gentoo_service.**get_all**()
Return all available boot services

CLI Example:

```
salt '*' service.get_all
```

salt.modules.gentoo_service.**get_disabled**()
Return a set of services that are installed but disabled

CLI Example:

```
salt '*' service.get_disabled
```

salt.modules.gentoo_service.**get_enabled**()
Return a list of service that are enabled on boot

CLI Example:

```
salt '*' service.get_enabled
```

salt.modules.gentoo_service.**missing**(*name*)
The inverse of service.available. Returns `True` if the specified service is not available, otherwise returns `False`.

CLI Example:

```
salt '*' service.missing sshd
```

salt.modules.gentoo_service.**restart**(*name*)
Restart the named service

CLI Example:

```
salt '*' service.restart <service name>
```

salt.modules.gentoo_service.**start**(*name*)
Start the specified service

CLI Example:

```
salt '*' service.start <service name>
```

salt.modules.gentoo_service.**status**(*name*, *sig=None*)
Return the status for a service, returns the PID or an empty string if the service is running or not, pass a signature to use to find the service via ps

CLI Example:

```
salt '*' service.status <service name> [service signature]
```

salt.modules.gentoo_service.**stop**(*name*)
Stop the specified service

CLI Example:

```
salt '*' service.stop <service name>
```

22.16.80 salt.modules.gentoolkitmod

Support for Gentoolkit

salt.modules.gentoolkitmod.**eclean_dist**(*destructive=False*, *package_names=False*, *size_limit=0*, *time_limit=0*, *fetch_restricted=False*, *exclude_file='/etc/eclean/distfiles.exclude'*)
Clean obsolete portage sources

destructive Only keep minimum for reinstallation

package_names Protect all versions of installed packages. Only meaningful if used with destructive=True

size_limit <size> Don't delete distfiles bigger than <size>. <size> is a size specification: ``10M'' is ``ten megabytes'', ``200K'' is ``two hundreds kilobytes'', etc. Units are: G, M, K and B.

time_limit <time> Don't delete distfiles files modified since <time> <time> is an amount of time: ``1y'' is ``one year'', ``2w'' is ``two weeks'', etc. Units are: y (years), m (months), w (weeks), d (days) and h (hours).

fetch_restricted Protect fetch-restricted files. Only meaningful if used with destructive=True

exclude_file Path to exclusion file. Default is /etc/eclean/distfiles.exclude This is the same default eclean-dist uses. Use None if this file exists and you want to ignore.

Returns a dict containing the cleaned, saved, and deprecated dists:

```
{'cleaned': {<dist file>: <size>},
 'deprecated': {<package>: <dist file>},
 'saved': {<package>: <dist file>},
 'total_cleaned': <size>}
```

CLI Example:

```
salt '*' gentoolkit.eclean_dist destructive=True
```

salt.modules.gentoolkitmod.**eclean_pkg**(*destructive=False, package_names=False, time_limit=0, exclude_file='/etc/eclean/packages.exclude'*)

Clean obsolete binary packages

destructive Only keep minimum for reinstallation

package_names Protect all versions of installed packages. Only meaningful if used with destructive=True

time_limit <time> Don't delete distfiles files modified since <time> <time> is an amount of time: ``1y'' is ``one year'', ``2w'' is ``two weeks'', etc. Units are: y (years), m (months), w (weeks), d (days) and h (hours).

exclude_file Path to exclusion file. Default is /etc/eclean/packages.exclude This is the same default eclean-pkg uses. Use None if this file exists and you want to ignore.

Returns a dict containing the cleaned binary packages:

```
{'cleaned': {<dist file>: <size>},
 'total_cleaned': <size>}
```

CLI Example:

```
salt '*' gentoolkit.eclean_pkg destructive=True
```

salt.modules.gentoolkitmod.**glsa_check_list**(*glsa_list*)

List the status of Gentoo Linux Security Advisories

glsa_list can contain an arbitrary number of GLSA ids, filenames containing GLSAs or the special identifiers `all' and `affected'

Returns a dict containing glsa ids with a description, status, and CVEs:

```
{<glsa_id>: {'description': <glsa_description>,
 'status': <glsa status>,
 'CVEs': [<list of CVEs>]}}
```

CLI Example:

```
salt '*' gentoolkit.glsa_check_list 'affected'
```

salt.modules.gentoolkitmod.**revdep_rebuild**(*lib=None*)

Fix up broken reverse dependencies

lib Search for reverse dependencies for a particular library rather than every library on the system. It can be a full path to a library or basic regular expression.

CLI Example:

```
salt '*' gentoolkit.revdep_rebuild
```

22.16.81 salt.modules.git

Support for the Git SCM

`salt.modules.git.`**`add`**(*cwd*, *file_name*, *user=None*, *opts=None*)

add a file to git

cwd The path to the Git repository

file_name Path to the file in the cwd

opts [None] Any additional options to add to the command line

user [None] Run git as a user other than what the minion runs as

CLI Example:

```
salt '*' git.add /path/to/git/repo /path/to/file
```

`salt.modules.git.`**`archive`**(*cwd*, *output*, *rev='HEAD'*, *fmt=None*, *prefix=None*, *user=None*)

Export a tarball from the repository

cwd The path to the Git repository

output The path to the archive tarball

rev: **HEAD** The revision to create an archive from

fmt: **None** Format of the resulting archive, zip and tar are commonly used

prefix [None] Prepend <prefix>/ to every filename in the archive

user [None] Run git as a user other than what the minion runs as

If `prefix` is not specified it defaults to the basename of the repo directory.

CLI Example:

```
salt '*' git.archive /path/to/repo /path/to/archive.tar.gz
```

`salt.modules.git.`**`branch`**(*cwd*, *rev*, *opts=None*, *user=None*)

Interacts with branches.

cwd The path to the Git repository

rev The branch/revision to be used in the command.

opts [None] Any additional options to add to the command line

user [None] Run git as a user other than what the minion runs as

CLI Example:

```
salt '*' git.branch mybranch --set-upstream-to=origin/mybranch
```

`salt.modules.git.`**`checkout`**(*cwd*, *rev*, *force=False*, *opts=None*, *user=None*)

Checkout a given revision

cwd The path to the Git repository

rev The remote branch or revision to checkout

force [False] Force a checkout even if there might be overwritten changes

opts [None] Any additional options to add to the command line

user [None] Run git as a user other than what the minion runs as

CLI Examples:

```
salt '*' git.checkout /path/to/repo somebranch user=jeff

salt '*' git.checkout /path/to/repo opts='testbranch -- conf/file1 file2'

salt '*' git.checkout /path/to/repo rev=origin/mybranch opts=--track
```

salt.modules.git.**clone**(*cwd*, *repository*, *opts=None*, *user=None*, *identity=None*, *https_user=None*, *https_pass=None*)

Clone a new repository

cwd The path to the Git repository

repository The git URI of the repository

opts [None] Any additional options to add to the command line

user [None] Run git as a user other than what the minion runs as

identity [None] A path to a private key to use over SSH

https_user [None] HTTP Basic Auth username for HTTPS (only) clones

https_pass [None] HTTP Basic Auth password for HTTPS (only) clones

CLI Example:

```
salt '*' git.clone /path/to/repo git://github.com/saltstack/salt.git

salt '*' git.clone /path/to/repo.git\
        git://github.com/saltstack/salt.git '--bare --origin github'
```

salt.modules.git.**commit**(*cwd*, *message*, *user=None*, *opts=None*)

create a commit

cwd The path to the Git repository

message The commit message

opts [None] Any additional options to add to the command line

user [None] Run git as a user other than what the minion runs as

CLI Example:

```
salt '*' git.commit /path/to/git/repo 'The commit message'
```

salt.modules.git.**config_get**(*cwd=None*, *setting_name=None*, *user=None*)

Get a key or keys from the git configuration file (.git/config).

cwd [None] Optional path to a Git repository

Changed in version 2014.7.0: Made `cwd` optional

setting_name [None] The name of the configuration key to get. Required.

user [None] Run git as a user other than what the minion runs as

CLI Example:

```
salt '*' git.config_get setting_name=user.email
salt '*' git.config_get /path/to/repo user.name arthur
```

salt.modules.git.**config_set**(*cwd=None*, *setting_name=None*, *setting_value=None*, *user=None*, *is_global=False*)

Set a key in the git configuration file (.git/config) of the repository or globally.

cwd [None] Options path to the Git repository

Changed in version 2014.7.0: Made `cwd` optional

setting_name [None] The name of the configuration key to set. Required.

setting_value [None] The (new) value to set. Required.

user [None] Run git as a user other than what the minion runs as

is_global [False] Set to True to use the `--global' flag with `git config'

CLI Example:

```
salt '*' git.config_set /path/to/repo user.email me@example.com
```

salt.modules.git.**current_branch**(*cwd*, *user=None*)

Returns the current branch name, if on a branch.

CLI Example:

```
salt '*' git.current_branch /path/to/repo
```

salt.modules.git.**describe**(*cwd*, *rev='HEAD'*, *user=None*)

Returns the git describe string (or the SHA hash if there are no tags) for the given revision

cwd The path to the Git repository

rev: **HEAD** The revision to describe

user [None] Run git as a user other than what the minion runs as

CLI Examples:

```
salt '*' git.describe /path/to/repo

salt '*' git.describe /path/to/repo develop
```

salt.modules.git.**fetch**(*cwd*, *opts=None*, *user=None*, *identity=None*)

Perform a fetch on the given repository

cwd The path to the Git repository

opts [None] Any additional options to add to the command line

user [None] Run git as a user other than what the minion runs as

identity [None] A path to a private key to use over SSH

CLI Example:

```
salt '*' git.fetch /path/to/repo '--all'

salt '*' git.fetch cwd=/path/to/repo opts='--all' user=johnny
```

salt.modules.git.**init**(*cwd*, *opts=None*, *user=None*)
Initialize a new git repository

cwd The path to the Git repository

opts [None] Any additional options to add to the command line

user [None] Run git as a user other than what the minion runs as

CLI Example:

```
salt '*' git.init /path/to/repo.git opts='--bare'
```

salt.modules.git.**ls_remote**(*cwd*, *repository='origin'*, *branch='master'*, *user=None*, *identity=None*, *https_user=None*, *https_pass=None*)
Returns the upstream hash for any given URL and branch.

cwd The path to the Git repository

repository: **origin** The name of the repository to get the revision from. Can be the name of a remote, an URL, etc.

branch: **master** The name of the branch to get the revision from.

user [none] run git as a user other than what the minion runs as

identity [none] a path to a private key to use over ssh

https_user [None] HTTP Basic Auth username for HTTPS (only) clones

https_pass [None] HTTP Basic Auth password for HTTPS (only) clones

CLI Example:

```
salt '*' git.ls_remote /pat/to/repo origin master
```

salt.modules.git.**merge**(*cwd*, *branch='@{upstream}'*, *opts=None*, *user=None*)
Merge a given branch

cwd The path to the Git repository

branch [@{upstream}] The remote branch or revision to merge into the current branch

opts [None] Any additional options to add to the command line

user [None] Run git as a user other than what the minion runs as

CLI Example:

```
salt '*' git.fetch /path/to/repo
salt '*' git.merge /path/to/repo @{upstream}
```

salt.modules.git.**pull**(*cwd*, *opts=None*, *user=None*, *identity=None*)
Perform a pull on the given repository

cwd The path to the Git repository

opts [None] Any additional options to add to the command line

user [None] Run git as a user other than what the minion runs as

identity [None] A path to a private key to use over SSH

CLI Example:

```
salt '*' git.pull /path/to/repo opts='--rebase origin master'
```

salt.modules.git.**push**(*cwd*, *remote_name*, *branch='master'*, *user=None*, *opts=None*, *identity=None*)
Push to remote

cwd The path to the Git repository

remote_name Name of the remote to push to

branch [master] Name of the branch to push

opts [None] Any additional options to add to the command line

user [None] Run git as a user other than what the minion runs as

identity [None] A path to a private key to use over SSH

CLI Example:

```
salt '*' git.push /path/to/git/repo remote-name
```

salt.modules.git.**rebase**(*cwd*, *rev='master'*, *opts=None*, *user=None*)
Rebase the current branch

cwd The path to the Git repository

rev [master] The revision to rebase onto the current branch

opts [None] Any additional options to add to the command line

user [None] Run git as a user other than what the minion runs as

CLI Example:

```
salt '*' git.rebase /path/to/repo master
salt '*' git.rebase /path/to/repo 'origin master'
```

That is the same as:

```
git rebase master
git rebase origin master
```

salt.modules.git.**remote_get**(*cwd*, *remote='origin'*, *user=None*)
get the fetch and push URL for a specified remote name

remote [origin] the remote name used to define the fetch and push URL

user [None] Run git as a user other than what the minion runs as

CLI Example:

```
salt '*' git.remote_get /path/to/repo
salt '*' git.remote_get /path/to/repo upstream
```

salt.modules.git.**remote_set**(*cwd*, *name='origin'*, *url=None*, *user=None*, *https_user=None*, *https_pass=None*)
sets a remote with name and URL like git remote add <remote_name> <remote_url>

remote_name [origin] defines the remote name

remote_url [None] defines the remote URL; should not be None!

user [None] Run git as a user other than what the minion runs as

https_user [None] HTTP Basic Auth username for HTTPS (only) clones

https_pass [None] HTTP Basic Auth password for HTTPS (only) clones

CLI Example:

```
salt '*' git.remote_set /path/to/repo remote_url=git@github.com:saltstack/salt.git
salt '*' git.remote_set /path/to/repo origin git@github.com:saltstack/salt.git
```

salt.modules.git.**remotes**(*cwd*, *user=None*)

Get remotes like git remote -v

cwd The path to the Git repository

user [None] Run git as a user other than what the minion runs as

CLI Example:

```
salt '*' git.remotes /path/to/repo
```

salt.modules.git.**reset**(*cwd*, *opts=None*, *user=None*)

Reset the repository checkout

cwd The path to the Git repository

opts [None] Any additional options to add to the command line

user [None] Run git as a user other than what the minion runs as

CLI Example:

```
salt '*' git.reset /path/to/repo master
```

salt.modules.git.**revision**(*cwd*, *rev='HEAD'*, *short=False*, *user=None*)

Returns the long hash of a given identifier (hash, branch, tag, HEAD, etc)

cwd The path to the Git repository

rev: **HEAD** The revision

short: **False** Return an abbreviated SHA1 git hash

user [None] Run git as a user other than what the minion runs as

CLI Example:

```
salt '*' git.revision /path/to/repo mybranch
```

salt.modules.git.**rm**(*cwd*, *file_name*, *user=None*, *opts=None*)

Remove a file from git

cwd The path to the Git repository

file_name Path to the file in the cwd

opts [None] Any additional options to add to the command line

user [None] Run git as a user other than what the minion runs as

CLI Example:

```
salt '*' git.rm /path/to/git/repo /path/to/file
```

salt.modules.git.**stash**(*cwd*, *opts=None*, *user=None*)

Stash changes in the repository checkout

cwd The path to the Git repository

opts [None] Any additional options to add to the command line

user [None] Run git as a user other than what the minion runs as

CLI Example:

```
salt '*' git.stash /path/to/repo master
```

salt.modules.git.**status**(*cwd*, *user=None*)

Return the status of the repository. The returned format uses the status codes of git's `porcelain' output mode

cwd The path to the Git repository

user [None] Run git as a user other than what the minion runs as

CLI Example:

```
salt '*' git.status /path/to/git/repo
```

salt.modules.git.**submodule**(*cwd*, *init=True*, *opts=None*, *user=None*, *identity=None*)

Initialize git submodules

cwd The path to the Git repository

init [True] Ensure that new submodules are initialized

opts [None] Any additional options to add to the command line

user [None] Run git as a user other than what the minion runs as

identity [None] A path to a private key to use over SSH

CLI Example:

```
salt '*' git.submodule /path/to/repo.git/sub/repo
```

22.16.82 salt.modules.glance

Module for handling openstack glance calls.

optdepends

- glanceclient Python adapter

configuration This module is not usable until the following are specified either in a pillar or in the minion's config file:

```
keystone.user: admin
keystone.password: verybadpass
keystone.tenant: admin
keystone.tenant_id: f80919baedab48ec8931f200c65a50df
keystone.insecure: False   #(optional)
keystone.auth_url: 'http://127.0.0.1:5000/v2.0/'
```

If configuration for multiple openstack accounts is required, they can be set up as different configuration profiles: For example:

```
openstack1:
  keystone.user: admin
  keystone.password: verybadpass
  keystone.tenant: admin
  keystone.tenant_id: f80919baedab48ec8931f200c65a50df
  keystone.auth_url: 'http://127.0.0.1:5000/v2.0/'
```

```
        openstack2:
          keystone.user: admin
          keystone.password: verybadpass
          keystone.tenant: admin
          keystone.tenant_id: f80919baedab48ec8931f200c65a50df
          keystone.auth_url: 'http://127.0.0.2:5000/v2.0/'
```

With this configuration in place, any of the keystone functions can make use of a configuration profile by declaring it explicitly. For example:

```
salt '*' glance.image_list profile=openstack1
```

salt.modules.glance.**image_create**(*profile=None*, ***kwargs*)

Create an image (glance image-create)

CLI Example:

```
salt '*' glance.image_create name=f16-jeos is_public=true \
         disk_format=qcow2 container_format=ovf \
         copy_from=http://berrange.fedorapeople.org/images/2012-02-29/f16-x86_64-openstack-sda.q
```

For all possible values, run `glance help image-create` on the minion.

salt.modules.glance.**image_delete**(*id=None*, *name=None*, *profile=None*)

Delete an image (glance image-delete)

CLI Examples:

```
salt '*' glance.image_delete c2eb2eb0-53e1-4a80-b990-8ec887eae7df
salt '*' glance.image_delete id=c2eb2eb0-53e1-4a80-b990-8ec887eae7df
salt '*' glance.image_delete name=f16-jeos
```

salt.modules.glance.**image_list**(*id=None*, *profile=None*)

Return a list of available images (glance image-list)

CLI Example:

```
salt '*' glance.image_list
```

salt.modules.glance.**image_show**(*id=None*, *name=None*, *profile=None*)

Return details about a specific image (glance image-show)

CLI Example:

```
salt '*' glance.image_get
```

22.16.83 salt.modules.glusterfs

Manage a glusterfs pool

salt.modules.glusterfs.**add_volume_bricks**(*name*, *bricks*)

Add brick(s) to an existing volume

name Volume name

bricks List of bricks to add to the volume

salt.modules.glusterfs.**create**(*name*, *bricks*, *stripe=False*, *replica=False*, *device_vg=False*, *transport='tcp'*, *start=False*)

Create a glusterfs volume.

name Name of the gluster volume

bricks Bricks to create volume from, in <peer>:<brick path> format. For multiple bricks use list format: `["<peer1>:<brick1>", ``<peer2>:<brick2>"]'

stripe Stripe count, the number of bricks should be a multiple of the stripe count for a distributed striped volume

replica Replica count, the number of bricks should be a multiple of the replica count for a distributed replicated volume

device_vg If true, specifies volume should use block backend instead of regular posix backend. Block device backend volume does not support multiple bricks

transport Transport protocol to use, can be `tcp', `rdma' or `tcp,rdma'

start Start the volume after creation

CLI Example:

```
salt host1 glusterfs.create newvolume host1:/brick

salt gluster1 glusterfs.create vol2 '["gluster1:/export/vol2/brick",         "gluster2:/export/v
```

salt.modules.glusterfs.**delete**(*target*, *stop=True*)

Deletes a gluster volume

target Volume to delete

stop Stop volume before delete if it is started, True by default

salt.modules.glusterfs.**list_peers**()

Return a list of gluster peers

CLI Example:

```
salt '*' glusterfs.list_peers
```

GLUSTER direct CLI example (to show what salt is sending to gluster):

$ gluster peer status

GLUSTER CLI 3.4.4 return example (so we know what we are parsing):

Number of Peers: 2

Hostname: ftp2 Port: 24007 Uuid: cbcb256b-e66e-4ec7-a718-21082d396c24 State: Peer in Cluster (Connected)

Hostname: ftp3 Uuid: 5ea10457-6cb2-427b-a770-7897509625e9 State: Peer in Cluster (Connected)

salt.modules.glusterfs.**list_volumes**()

List configured volumes

CLI Example:

```
salt '*' glusterfs.list_volumes
```

salt.modules.glusterfs.**peer**(*name*)

Add another node into the peer list.

name The remote host to probe.

CLI Example:

```
salt 'one.gluster.*' glusterfs.peer two
```

GLUSTER direct CLI example (to show what salt is sending to gluster):

$ gluster peer probe ftp2

GLUSTER CLI 3.4.4 return example (so we know what we are parsing): #if the ``peer'' is the local host: peer probe: success: on localhost not needed

#if the peer was just added: peer probe: success

#if the peer was already part of the cluster: peer probe: success: host ftp2 port 24007 already in peer list

salt.modules.glusterfs.**start_volume**(*name*)

Start a gluster volume.

name Volume name

CLI Example:

```
salt '*' glusterfs.start mycluster
```

salt.modules.glusterfs.**status**(*name*)

Check the status of a gluster volume.

name Volume name

CLI Example:

```
salt '*' glusterfs.status myvolume
```

salt.modules.glusterfs.**stop_volume**(*name*)

Stop a gluster volume.

name Volume name

CLI Example:

```
salt '*' glusterfs.stop_volume mycluster
```

22.16.84 salt.modules.gnomedesktop

GNOME implementations

salt.modules.gnomedesktop.**get**(*schema=None*, *key=None*, *user=None*, ***kwargs*)

Get key in a particular GNOME schema

CLI Example:

```
salt '*' gnome.get user=<username> schema=org.gnome.desktop.screensaver key=idle-activation-enab
```

salt.modules.gnomedesktop.**getClockFormat**(***kwargs*)

Return the current clock format, either 12h or 24h format.

CLI Example:

```
salt '*' gnome.getClockFormat user=<username>
```

salt.modules.gnomedesktop.**getClockShowDate**(***kwargs*)
Return the current setting, if the date is shown in the clock

CLI Example:

```
salt '*' gnome.getClockShowDate user=<username>
```

salt.modules.gnomedesktop.**getIdleActivation**(***kwargs*)
Get whether the idle activation is enabled

CLI Example:

```
salt '*' gnome.getIdleActivation user=<username>
```

salt.modules.gnomedesktop.**getIdleDelay**(***kwargs*)
Return the current idle delay setting in seconds

CLI Example:

```
salt '*' gnome.getIdleDelay user=<username>
```

salt.modules.gnomedesktop.**ping**(***kwargs*)
A test to ensure the GNOME module is loaded

CLI Example:

```
salt '*' gnome.ping user=<username>
```

salt.modules.gnomedesktop.**setClockFormat**(*clockFormat*, ***kwargs*)
Set the clock format, either 12h or 24h format.

CLI Example:

```
salt '*' gnome.setClockFormat <12h|24h> user=<username>
```

salt.modules.gnomedesktop.**setClockShowDate**(*kvalue*, ***kwargs*)
Set whether the date is visible in the clock

CLI Example:

```
salt '*' gnome.setClockShowDate <True|False> user=<username>
```

salt.modules.gnomedesktop.**setIdleActivation**(*kvalue*, ***kwargs*)
Set whether the idle activation is enabled

CLI Example:

```
salt '*' gnome.setIdleActivation <True|False> user=<username>
```

salt.modules.gnomedesktop.**setIdleDelay**(*delaySeconds*, ***kwargs*)
Set the current idle delay setting in seconds

CLI Example:

```
salt '*' gnome.setIdleDelay <seconds> user=<username>
```

salt.modules.gnomedesktop.**set_**(*schema=None*, *key=None*, *user=None*, *value=None*, ***kwargs*)
Set key in a particular GNOME schema

CLI Example:

```
salt '*' gnome.set user=<username> schema=org.gnome.desktop.screensaver key=idle-activation-enab
```

22.16.85 salt.modules.gpg

Manage a GPG keychains, add keys, create keys, retrieve keys from keyservers. Sign, encrypt and sign & encrypt text and files.

New in version 2015.5.0.

Note: The `python-gnupg` library and gpg binary are required to be installed.

`salt.modules.gpg.`**`create_key`**(*key_type='RSA'*, *key_length=1024*, *name_real='Autogenerated Key'*, *name_comment='Generated by SaltStack'*, *name_email=None*, *subkey_type=None*, *subkey_length=None*, *expire_date=None*, *use_passphrase=False*, *user=None*)

Create a key in the GPG keychain

Note: GPG key generation requires *a lot* of entropy and randomness. Difficult to do over a remote connection, consider having another process available which is generating randomness for the machine. Also especially difficult on virtual machines, consider the rpg-tools package.

The create_key process takes awhile so increasing the timeout may be necessary, e.g. -t 15.

---

key_type The type of the primary key to generate. It must be capable of signing. `RSA' or `DSA'.

key_length The length of the primary key in bits.

name_real The real name of the user identity which is represented by the key.

name_comment A comment to attach to the user id.

name_email An email address for the user.

subkey_type The type of the secondary key to generate.

subkey_length The length of the secondary key in bits.

expire_date The expiration date for the primary and any secondary key. You can specify an ISO date, A number of days/weeks/months/years, an epoch value, or 0 for a non-expiring key.

use_passphrase Whether to use a passphrase with the signing key. Passphrase is received from pillar.

user Which user's keychain to access, defaults to user Salt is running as. Passing the user as `salt' will set the GPG home directory to /etc/salt/gpgkeys.

CLI Example:

```
salt -t 15 '*' gpg.create_key
```

`salt.modules.gpg.`**`decrypt`**(*user=None*, *text=None*, *filename=None*, *output=None*, *use_passphrase=False*)

Decrypt a message or file

user Which user's keychain to access, defaults to user Salt is running as. Passing the user as `salt' will set the GPG home directory to /etc/salt/gpg.

text The encrypted text to decrypt.

filename The encrypted filename to decrypt.

output The filename where the decrypted data will be written, default is standard out.

use_passphrase Whether to use a passphrase with the signing key. Passphrase is received from pillar.

CLI Example:

```
salt '*' gpg.decrypt filename='/path/to/important.file.gpg'

salt '*' gpg.decrypt filename='/path/to/important.file.gpg' use_pasphrase=True
```

salt.modules.gpg.**delete_key**(*keyid=None*, *fingerprint=None*, *delete_secret=False*, *user=None*)

Get a key from the GPG keychain

keyid The keyid of the key to be deleted.

fingerprint The fingerprint of the key to be deleted.

delete_secret Whether to delete a corresponding secret key prior to deleting the public key. Secret keys must be deleted before deleting any corresponding public keys.

user Which user's keychain to access, defaults to user Salt is running as. Passing the user as `salt' will set the GPG home directory to /etc/salt/gpgkeys.

CLI Example:

```
salt '*' gpg.delete_key keyid=3FAD9F1E

salt '*' gpg.delete_key fingerprint=53C96788253E58416D20BCD352952C84C3252192

salt '*' gpg.delete_key keyid=3FAD9F1E user=username

salt '*' gpg.delete_key keyid=3FAD9F1E user=username delete_secret=True
```

salt.modules.gpg.**encrypt**(*user=None*, *recipients=None*, *text=None*, *filename=None*, *output=None*, *sign=None*, *use_passphrase=False*)

Encrypt a message or file

user Which user's keychain to access, defaults to user Salt is running as. Passing the user as `salt' will set the GPG home directory to /etc/salt/gpg.

recipients The fingerprints for those recipient whom the data is being encrypted for.

text The text to encrypt.

filename The filename to encrypt.

output The filename where the signed file will be written, default is standard out.

sign Whether to sign, in addition to encrypt, the data. True to use default key or fingerprint to specify a different key to sign with.

use_passphrase Whether to use a passphrase with the signing key. Passphrase is received from pillar.

CLI Example:

```
salt '*' gpg.encrypt text='Hello there.  How are you?'

salt '*' gpg.encrypt filename='/path/to/important.file'

salt '*' gpg.encrypt filename='/path/to/important.file' use_pasphrase=True
```

salt.modules.gpg.**export_key**(*keyids=None*, *secret=False*, *user=None*)

Export a key from the GPG keychain

keyids The keyid(s) of the key(s) to be exported. Can be specified as a comma separated string or a list. Anything which GnuPG itself accepts to identify a key - for example, the keyid or the fingerprint could be used.

secret Export the secret key identified by the keyid information passed.

user Which user's keychain to access, defaults to user Salt is running as. Passing the user as `salt' will set the GPG home directory to /etc/salt/gpg.

CLI Example:

```
salt '*' gpg.export_key keyids=3FAD9F1E

salt '*' gpg.export_key keyids=3FAD9F1E secret=True

salt '*' gpg.export_key keyid="['3FAD9F1E','3FBD8F1E']" user=username
```

salt.modules.gpg.**get_key**(*keyid=None*, *fingerprint=None*, *user=None*)

Get a key from the GPG keychain

keyid The keyid of the key to be retrieved.

fingerprint The fingerprint of the key to be retrieved.

user Which user's keychain to access, defaults to user Salt is running as. Passing the user as `salt' will set the GPG home directory to /etc/salt/gpg.

CLI Example:

```
salt '*' gpg.get_key keyid=3FAD9F1E

salt '*' gpg.get_key fingerprint=53C96788253E58416D20BCD352952C84C3252192

salt '*' gpg.get_key keyid=3FAD9F1E user=username
```

salt.modules.gpg.**get_secret_key**(*keyid=None*, *fingerprint=None*, *user=None*)

Get a key from the GPG keychain

keyid The keyid of the key to be retrieved.

fingerprint The fingerprint of the key to be retrieved.

user Which user's keychain to access, defaults to user Salt is running as. Passing the user as `salt' will set the GPG home directory to /etc/salt/gpg.

CLI Example:

```
salt '*' gpg.get_secret_key keyid=3FAD9F1E

salt '*' gpg.get_secret_key fingerprint=53C96788253E58416D20BCD352952C84C3252192

salt '*' gpg.get_secret_key keyid=3FAD9F1E user=username
```

salt.modules.gpg.**import_key**(*user=None*, *text=None*, *filename=None*)

Import a key from text or file

user Which user's keychain to access, defaults to user Salt is running as. Passing the user as `salt' will set the GPG home directory to /etc/salt/gpg.

text The text containing to import.

filename The filename containing the key to import.

CLI Example:

```
salt '*' gpg.import_key text='-----BEGIN PGP PUBLIC KEY BLOCK-----
```

... -----END PGP PUBLIC KEY BLOCK-----`

salt `*' gpg.import_key filename='/path/to/public-key-file'

`salt.modules.gpg.`**`list_keys`**(*user=None*)

List keys in GPG keychain

user Which user's keychain to access, defaults to user Salt is running as. Passing the user as `salt' will set the GPG home directory to /etc/salt/gpgkeys.

CLI Example:

```
salt '*' gpg.list_keys
```

`salt.modules.gpg.`**`list_secret_keys`**(*user=None*)

List secret keys in GPG keychain

user Which user's keychain to access, defaults to user Salt is running as. Passing the user as `salt' will set the GPG home directory to /etc/salt/gpgkeys.

CLI Example:

```
salt '*' gpg.list_secret_keys
```

`salt.modules.gpg.`**`receive_keys`**(*keyserver=None, keys=None, user=None*)

Receive key(s) from keyserver and add them to keychain

keyserver Keyserver to use for searching for GPG keys, defaults to pgp.mit.edu

keys The keyID(s) to retrieve from the keyserver. Can be specified as a comma separated string or a list.

user Which user's keychain to access, defaults to user Salt is running as. Passing the user as `salt' will set the GPG home directory to /etc/salt/gpg.

CLI Example:

```
salt '*' gpg.receive_key keys='3FAD9F1E'

salt '*' gpg.receive_key keys="['3FAD9F1E','3FBD9F2E']"

salt '*' gpg.receive_key keys=3FAD9F1E user=username
```

`salt.modules.gpg.`**`search_keys`**(*text, keyserver=None, user=None*)

Search keys from keyserver

text Text to search the keyserver for, e.g. email address, keyID or fingerprint.

keyserver Keyserver to use for searching for GPG keys, defaults to pgp.mit.edu

user Which user's keychain to access, defaults to user Salt is running as. Passing the user as `salt' will set the GPG home directory to /etc/salt/gpgkeys.

CLI Example:

```
salt '*' gpg.search_keys user@example.com

salt '*' gpg.search_keys user@example.com keyserver=keyserver.ubuntu.com

salt '*' gpg.search_keys user@example.com keyserver=keyserver.ubuntu.com user=username
```

salt.modules.gpg.**sign**(*user=None*, *keyid=None*, *text=None*, *filename=None*, *output=None*, *use_passphrase=False*)

Sign message or file

user Which user's keychain to access, defaults to user Salt is running as. Passing the user as `salt' will set the GPG home directory to /etc/salt/gpg.

keyid The keyid of the key to set the trust level for, defaults to first key in the secret keyring.

text The text to sign.

filename The filename to sign.

output The filename where the signed file will be written, default is standard out.

use_passphrase Whether to use a passphrase with the signing key. Passphrase is received from pillar.

CLI Example:

```
salt '*' gpg.sign text='Hello there.  How are you?'

salt '*' gpg.sign filename='/path/to/important.file'

salt '*' gpg.sign filename='/path/to/important.file' use_pasphrase=True
```

salt.modules.gpg.**trust_key**(*keyid=None*, *fingerprint=None*, *trust_level=None*, *user=None*)

Set the trust level for a key in GPG keychain

keyid The keyid of the key to set the trust level for.

fingerprint The fingerprint of the key to set the trust level for.

trust_level The trust level to set for the specified key, must be one of the following:

expired, unknown, not_trusted, marginally, fully, ultimately

user Which user's keychain to access, defaults to user Salt is running as. Passing the user as `salt' will set the GPG home directory to /etc/salt/gpg.

CLI Example:

```
salt '*' gpg.trust_key keyid='3FAD9F1E' trust_level='marginally'

salt '*' gpg.trust_key fingerprint='53C96788253E58416D20BCD352952C84C3252192' trust_level='not_t

salt '*' gpg.trust_key keys=3FAD9F1E trust_level='ultimately' user='username'
```

salt.modules.gpg.**verify**(*text=None*, *user=None*, *filename=None*)

Verify a message or file

text The text to verify.

filename The filename to verify.

user Which user's keychain to access, defaults to user Salt is running as. Passing the user as `salt' will set the GPG home directory to /etc/salt/gpg.

CLI Example:

```
salt '*' gpg.verify text='Hello there.  How are you?'

salt '*' gpg.verify filename='/path/to/important.file'

salt '*' gpg.verify filename='/path/to/important.file' use_pasphrase=True
```

22.16.86 salt.modules.grains

Return/control aspects of the grains data

salt.modules.grains.**append**(*key*, *val*, *convert=False*)

New in version 0.17.0.

Append a value to a list in the grains config file. If the grain doesn't exist, the grain key is added and the value is appended to the new grain as a list item.

key The grain key to be appended to

val The value to append to the grain key

Parameters **convert** -- If convert is True, convert non-list contents into a list. If convert is False and the grain contains non-list contents, an error is given. Defaults to False.

CLI Example:

```
salt '*' grains.append key val
```

salt.modules.grains.**delval**(*key*, *destructive=False*)

New in version 0.17.0.

Delete a grain from the grains config file

Parameters **destructive** -- Delete the key, too. Defaults to False.

CLI Example:

```
salt '*' grains.delval key
```

salt.modules.grains.**filter_by**(*lookup_dict*, *grain='os_family'*, *merge=None*, *default='default'*, *base=None*)

New in version 0.17.0.

Look up the given grain in a given dictionary for the current OS and return the result

Although this may occasionally be useful at the CLI, the primary intent of this function is for use in Jinja to make short work of creating lookup tables for OS-specific data. For example:

```
{% set apache = salt['grains.filter_by']({
    'Debian': {'pkg': 'apache2', 'srv': 'apache2'},
    'RedHat': {'pkg': 'httpd', 'srv': 'httpd'},
}, default='Debian') %}

myapache:
  pkg.installed:
    - name: {{ apache.pkg }}
  service.running:
    - name: {{ apache.srv }}
```

Values in the lookup table may be overridden by values in Pillar. An example Pillar to override values in the example above could be as follows:

```
apache:
  lookup:
    pkg: apache_13
    srv: apache
```

The call to `filter_by()` would be modified as follows to reference those Pillar values:

```
{% set apache = salt['grains.filter_by']({
    ...
}, merge=salt['pillar.get']('apache:lookup')) %}
```

Parameters

- **lookup_dict** -- A dictionary, keyed by a grain, containing a value or values relevant to systems matching that grain. For example, a key could be the grain for an OS and the value could the name of a package on that particular OS.
- **grain** -- The name of a grain to match with the current system's grains. For example, the value of the ``os_family'' grain for the current system could be used to pull values from the `lookup_dict` dictionary.
- **merge** -- A dictionary to merge with the results of the grain selection from `lookup_dict`. This allows Pillar to override the values in the `lookup_dict`. This could be useful, for example, to override the values for non-standard package names such as when using a different Python version from the default Python version provided by the OS (e.g., `python26-mysql` instead of `python-mysql`).
- **default** -- default lookup_dict's key used if the grain does not exists or if the grain value has no match on lookup_dict. If unspecified the value is ``default''.

  New in version 2014.1.0.
- **base** -- A lookup_dict key to use for a base dictionary. The grain-selected `lookup_dict` is merged over this and then finally the `merge` dictionary is merged. This allows common values for each case to be collected in the base and overridden by the grain selection dictionary and the merge dictionary. Default is unset.

  New in version 2015.5.0.

CLI Example:

```
salt '*' grains.filter_by '{Debian: Debheads rule, RedHat: I love my hat}'
# this one will render {D: {E: I, G: H}, J: K}
salt '*' grains.filter_by '{A: B, C: {D: {E: F,G: H}}}' 'xxx' '{D: {E: I},J: K}' 'C'
# next one renders {A: {B: G}, D: J}
salt '*' grains.filter_by '{default: {A: {B: C}, D: E}, F: {A: {B: G}}, H: {D: I}}' 'xxx' '{D: J
# next same as above when default='H' instead of 'F' renders {A: {B: C}, D: J}
```

`salt.modules.grains.get`(*key*, *default=''*, *delimiter=':'*)

Attempt to retrieve the named value from grains, if the named value is not available return the passed default. The default return is an empty string.

The value can also represent a value in a nested dict using a ":" delimiter for the dict. This means that if a dict in grains looks like this:

```
{'pkg': {'apache': 'httpd'}}
```

To retrieve the value associated with the apache key in the pkg dict this key can be passed:

```
pkg:apache
```

delimiter Specify an alternate delimiter to use when traversing a nested dict

New in version 2014.7.0.

CLI Example:

```
salt '*' grains.get pkg:apache
```

salt.modules.grains.**get_or_set_hash**(*name, length=8, chars='abcdefghijklmnopqrstuvwxyz0123456789!@#$%^&*(-_=+)'*)

Perform a one-time generation of a hash and write it to the local grains. If that grain has already been set return the value instead.

This is useful for generating passwords or keys that are specific to a single minion that don't need to be stored somewhere centrally.

State Example:

```
some_mysql_user:
  mysql_user:
    - present
    - host: localhost
    - password: {{ salt['grains.get_or_set_hash']('mysql:some_mysql_user') }}
```

CLI Example:

```
salt '*' grains.get_or_set_hash 'django:SECRET_KEY' 50
```

salt.modules.grains.**has_value**(*key*)

Determine whether a named value exists in the grains dictionary.

Given a grains dictionary that contains the following structure:

```
{'pkg': {'apache': 'httpd'}}
```

One would determine if the apache key in the pkg dict exists by:

```
pkg:apache
```

CLI Example:

```
salt '*' grains.has_value pkg:apache
```

salt.modules.grains.**item**(**args, **kwargs*)

Return one or more grains

CLI Example:

```
salt '*' grains.item os
salt '*' grains.item os osrelease oscodename
```

Sanitized CLI Example:

```
salt '*' grains.item host sanitize=True
```

salt.modules.grains.**items**(*sanitize=False*)

Return all of the minion's grains

CLI Example:

```
salt '*' grains.items
```

Sanitized CLI Example:

```
salt '*' grains.items sanitize=True
```

salt.modules.grains.**ls**()
Return a list of all available grains

CLI Example:

```
salt '*' grains.ls
```

salt.modules.grains.**remove**(*key*, *val*)
New in version 0.17.0.

Remove a value from a list in the grains config file

CLI Example:

```
salt '*' grains.remove key val
```

salt.modules.grains.**setval**(*key*, *val*, *destructive=False*)
Set a grains value in the grains config file

Parameters **Destructive** -- If an operation results in a key being removed, delete the key, too. Defaults to False.

CLI Example:

```
salt '*' grains.setval key val
salt '*' grains.setval key "{'sub-key': 'val', 'sub-key2': 'val2'}"
```

salt.modules.grains.**setvals**(*grains*, *destructive=False*)
Set new grains values in the grains config file

Parameters **Destructive** -- If an operation results in a key being removed, delete the key, too. Defaults to False.

CLI Example:

```
salt '*' grains.setvals "{'key1': 'val1', 'key2': 'val2'}"
```

22.16.87 salt.modules.groupadd

Manage groups on Linux, OpenBSD and NetBSD

salt.modules.groupadd.**add**(*name*, *gid=None*, *system=False*)
Add the specified group

CLI Example:

```
salt '*' group.add foo 3456
```

salt.modules.groupadd.**adduser**(*name*, *username*)
Add a user in the group.

CLI Example:

```
salt '*' group.adduser foo bar
```

Verifies if a valid username `bar' as a member of an existing group `foo', if not then adds it.

salt.modules.groupadd.**chgid**(*name*, *gid*)
Change the gid for a named group

CLI Example:

```
salt '*' group.chgid foo 4376
```

salt.modules.groupadd.**delete**(*name*)
Remove the named group

CLI Example:

```
salt '*' group.delete foo
```

salt.modules.groupadd.**deluser**(*name*, *username*)
Remove a user from the group.

CLI Example:

```
salt '*' group.deluser foo bar
```

Removes a member user `bar' from a group `foo'. If group is not present then returns True.

salt.modules.groupadd.**getent**(*refresh=False*)
Return info on all groups

CLI Example:

```
salt '*' group.getent
```

salt.modules.groupadd.**info**(*name*)
Return information about a group

CLI Example:

```
salt '*' group.info foo
```

salt.modules.groupadd.**members**(*name*, *members_list*)
Replaces members of the group with a provided list.

CLI Example:

salt `*' group.members foo `user1,user2,user3,...'

Replaces a membership list for a local group `foo'. foo:x:1234:user1,user2,user3,...

22.16.88 salt.modules.grub_legacy

Support for GRUB Legacy

salt.modules.grub_legacy.**conf**()
Parse GRUB conf file

CLI Example:

```
salt '*' grub.conf
```

salt.modules.grub_legacy.**version**()
Return server version from grub --version

CLI Example:

```
salt '*' grub.version
```

22.16.89 salt.modules.guestfs

Interact with virtual machine images via libguestfs

depends

- libguestfs

salt.modules.guestfs.**mount**(*location*, *access='rw'*)
Mount an image

CLI Example:

```
salt '*' guest.mount /srv/images/fedora.qcow
```

22.16.90 salt.modules.hadoop

Support for hadoop

maintainer Yann Jouanin <yann.jouanin@intelunix.fr>

maturity new

depends

platform linux

salt.modules.hadoop.**dfs**(*command=None*, **args*)
Execute a command on DFS

CLI Example:

```
salt '*' hadoop.dfs ls /
```

salt.modules.hadoop.**dfs_absent**(*path*)
Check if a file or directory is absent on the distributed FS.

CLI Example:

```
salt '*' hadoop.dfs_absent /some_random_file
```

Returns True if the file is absent

salt.modules.hadoop.**dfs_present**(*path*)
Check if a file or directory is present on the distributed FS.

CLI Example:

```
salt '*' hadoop.dfs_present /some_random_file
```

Returns True if the file is present

salt.modules.hadoop.**namenode_format**(*force=None*)
Format a name node

```
salt '*' hadoop.namenode_format force=True
```

salt.modules.hadoop.**version**()
Return version from hadoop version

CLI Example:

```
salt '*' hadoop.version
```

22.16.91 salt.modules.haproxyconn

Support for haproxy

New in version 2014.7.0.

salt.modules.haproxyconn.**disable_server**(*name*, *backend*, *socket='/var/run/haproxy.sock'*)
Disable server in haproxy.

name Server to disable

backend haproxy backend

socket haproxy stats socket

```
salt '*' haproxy.disable_server db1.example.com mysql
```

salt.modules.haproxyconn.**enable_server**(*name*, *backend*, *socket='/var/run/haproxy.sock'*)
Enable Server in haproxy

name Server to enable

backend haproxy backend

socket haproxy stats socket

```
salt '*' haproxy.enable_server web1.example.com www
```

salt.modules.haproxyconn.**get_weight**(*name*, *backend*, *socket='/var/run/haproxy.sock'*)
Get server weight

name Server name

backend haproxy backend

socket haproxy stats socket

```
salt '*' haproxy.get_weight web1.example.com www
```

salt.modules.haproxyconn.**list_servers**(*backend*, *socket='/var/run/haproxy.sock'*)
List servers in haproxy backend.

backend haproxy backend

socket haproxy stats socket

```
salt '*' haproxy.list_servers mysql
```

salt.modules.haproxyconn.**set_weight**(*name*, *backend*, *weight=0*, *socket='/var/run/haproxy.sock'*)
Set server weight

name Server name

backend haproxy backend

weight Server Weight

socket haproxy stats socket

```
salt '*' haproxy.set_weight web1.example.com www 13
```

salt.modules.haproxyconn.**show_backends**(*socket='/var/run/haproxy.sock'*)
Show HaProxy Backends

socket haproxy stats socket

```
salt '*' haproxy.show_backends
```

salt.modules.haproxyconn.**show_frontends**(*socket='/var/run/haproxy.sock'*)
Show HaProxy frontends

socket haproxy stats socket

```
salt '*' haproxy.show_frontends
```

22.16.92 salt.modules.hashutil

A collection of hashing and encoding functions

salt.modules.hashutil.**base64_decodestring**(*instr*)
Decode a base64-encoded string

New in version 2014.7.0.

CLI Example:

```
salt '*' hashutil.base64_decodestring 'Z2V0IHNhbHRlZA==\n'
```

salt.modules.hashutil.**base64_encodestring**(*instr*)
Encode a string as base64

New in version 2014.7.0.

CLI Example:

```
salt '*' hashutil.base64_encodestring 'get salted'
```

salt.modules.hashutil.**hmac_signature**(*string*, *shared_secret*, *challenge_hmac*)
Verify a challenging hmac signature against a string / shared-secret

New in version 2014.7.0.

Returns a boolean if the verification succeeded or failed.

CLI Example:

```
salt '*' hashutil.hmac_signature 'get salted' 'shared secret' 'NS2BvKxFRk+rndAlFbCYIFNVkPtI/3KiI
```

salt.modules.hashutil.**md5_digest**(*instr*)
Generate an md5 hash of a given string

New in version 2014.7.0.

CLI Example:

```
salt '*' hashutil.md5_digest 'get salted'
```

salt.modules.hashutil.**sha256_digest**(*instr*)
Generate an sha256 hash of a given string

New in version 2014.7.0.

CLI Example:

```
salt '*' hashutil.sha256_digest 'get salted'
```

salt.modules.hashutil.**sha512_digest**(*instr*)

Generate an sha512 hash of a given string

New in version 2014.7.0.

CLI Example:

```
salt '*' hashutil.sha512_digest 'get salted'
```

22.16.93 salt.modules.hg

Support for the Mercurial SCM

salt.modules.hg.**archive**(*cwd*, *output*, *rev='tip'*, *fmt=None*, *prefix=None*, *user=None*)

Export a tarball from the repository

cwd The path to the Mercurial repository

output The path to the archive tarball

rev: **tip** The revision to create an archive from

fmt: **None** Format of the resulting archive. Mercurial supports: tar, tbz2, tgz, zip, uzip, and files formats.

prefix [None] Prepend <prefix>/ to every filename in the archive

user [None] Run hg as a user other than what the minion runs as

If `prefix` is not specified it defaults to the basename of the repo directory.

CLI Example:

```
salt '*' hg.archive /path/to/repo output=/tmp/archive.tgz fmt=tgz
```

salt.modules.hg.**clone**(*cwd*, *repository*, *opts=None*, *user=None*, *identity=None*)

Clone a new repository

cwd The path to the Mercurial repository

repository The hg URI of the repository

opts [None] Any additional options to add to the command line

user [None] Run hg as a user other than what the minion runs as

identity [None] Private SSH key on the minion server for authentication (ssh://)

New in version 2015.5.0.

CLI Example:

```
salt '*' hg.clone /path/to/repo https://bitbucket.org/birkenfeld/sphinx
```

salt.modules.hg.**describe**(*cwd*, *rev='tip'*, *user=None*)

Mimic git describe and return an identifier for the given revision

cwd The path to the Mercurial repository

rev: **tip** The path to the archive tarball

user [None] Run hg as a user other than what the minion runs as

CLI Example:

```
salt '*' hg.describe /path/to/repo
```

salt.modules.hg.**pull**(*cwd*, *opts=None*, *user=None*, *identity=None*, *repository=None*)
Perform a pull on the given repository

cwd The path to the Mercurial repository

repository [None] Perform pull from the repository different from .hg/hgrc:[paths]:default

opts [None] Any additional options to add to the command line

user [None] Run hg as a user other than what the minion runs as

identity [None] Private SSH key on the minion server for authentication (ssh://)

New in version 2015.5.0.

CLI Example:

```
salt '*' hg.pull /path/to/repo opts=-u
```

salt.modules.hg.**revision**(*cwd*, *rev='tip'*, *short=False*, *user=None*)
Returns the long hash of a given identifier (hash, branch, tag, HEAD, etc)

cwd The path to the Mercurial repository

rev: **tip** The revision

short: **False** Return an abbreviated commit hash

user [None] Run hg as a user other than what the minion runs as

CLI Example:

```
salt '*' hg.revision /path/to/repo mybranch
```

salt.modules.hg.**update**(*cwd*, *rev*, *force=False*, *user=None*)
Update to a given revision

cwd The path to the Mercurial repository

rev The revision to update to

force [False] Force an update

user [None] Run hg as a user other than what the minion runs as

CLI Example:

```
salt devserver1 hg.update /path/to/repo somebranch
```

22.16.94 salt.modules.hosts

Manage the information in the hosts file

salt.modules.hosts.**add_host**(*ip*, *alias*)
Add a host to an existing entry, if the entry is not in place then create it with the given host

CLI Example:

```
salt '*' hosts.add_host <ip> <alias>
```

salt.modules.hosts.**get_alias**(*ip*)
Return the list of aliases associated with an ip

CLI Example:

```
salt '*' hosts.get_alias <ip addr>
```

salt.modules.hosts.**get_ip**(*host*)
Return the ip associated with the named host

CLI Example:

```
salt '*' hosts.get_ip <hostname>
```

salt.modules.hosts.**has_pair**(*ip*, *alias*)
Return true if the alias is set

CLI Example:

```
salt '*' hosts.has_pair <ip> <alias>
```

salt.modules.hosts.**list_hosts**()
Return the hosts found in the hosts file in this format:

```
{'<ip addr>': ['alias1', 'alias2', ...]}
```

CLI Example:

```
salt '*' hosts.list_hosts
```

salt.modules.hosts.**rm_host**(*ip*, *alias*)
Remove a host entry from the hosts file

CLI Example:

```
salt '*' hosts.rm_host <ip> <alias>
```

salt.modules.hosts.**set_host**(*ip*, *alias*)
Set the host entry in the hosts file for the given ip, this will overwrite any previous entry for the given ip

CLI Example:

```
salt '*' hosts.set_host <ip> <alias>
```

22.16.95 salt.modules.htpasswd

Support for htpasswd command

New in version 2014.1.0.

The functions here will load inside the webutil module. This allows other functions that don't use htpasswd to use the webutil module name.

salt.modules.htpasswd.**useradd**(*pwfile*, *user*, *password*, *opts=''*)
Add an HTTP user using the htpasswd command. If the htpasswd file does not exist, it will be created. Valid options that can be passed are:

n Don't update file; display results on stdout. m Force MD5 encryption of the password (default). d Force CRYPT encryption of the password. p Do not encrypt the password (plaintext). s Force SHA encryption of the password.

CLI Examples:

```
salt '*' webutil.useradd /etc/httpd/htpasswd larry badpassword
salt '*' webutil.useradd /etc/httpd/htpasswd larry badpass opts=ns
```

salt.modules.htpasswd.**useradd_all**(*pwfile*, *user*, *password*, *opts=''*)

Add an HTTP user using the htpasswd command. If the htpasswd file does not exist, it will be created. Valid options that can be passed are:

n Don't update file; display results on stdout. m Force MD5 encryption of the password (default). d Force CRYPT encryption of the password. p Do not encrypt the password (plaintext). s Force SHA encryption of the password.

CLI Examples:

```
salt '*' webutil.useradd /etc/httpd/htpasswd larry badpassword
salt '*' webutil.useradd /etc/httpd/htpasswd larry badpass opts=ns
```

salt.modules.htpasswd.**userdel**(*pwfile*, *user*)

Delete an HTTP user from the specified htpasswd file.

CLI Examples:

```
salt '*' webutil.userdel /etc/httpd/htpasswd larry
```

22.16.96 salt.modules.http

Module for making various web calls. Primarily designed for webhooks and the like, but also useful for basic http testing.

New in version 2015.5.0.

salt.modules.http.**query**(*url*, ***kwargs*)

Query a resource, and decode the return data

CLI Example:

```
salt '*' http.query http://somelink.com/
salt '*' http.query http://somelink.com/ method=POST             params='key1=val1&key2=val2'
salt '*' http.query http://somelink.com/ method=POST             data='<xml>somecontent</xml>'
```

salt.modules.http.**update_ca_bundle**(*target=None*, *source=None*, *merge_files=None*)

Update the local CA bundle file from a URL

CLI Example:

```
salt '*' http.update_ca_bundle
salt '*' http.update_ca_bundle target=/path/to/cacerts.pem
salt '*' http.update_ca_bundle source=https://example.com/cacerts.pem
```

If the `target` is not specified, it will be pulled from the `ca_cert` configuration variable available to the minion. If it cannot be found there, it will be placed at `<<FILE_ROOTS>>/cacerts.pem`.

If the `source` is not specified, it will be pulled from the `ca_cert_url` configuration variable available to the minion. If it cannot be found, it will be downloaded from the cURL website, using an http (not https) URL. USING THE DEFAULT URL SHOULD BE AVOIDED!

`merge_files` may also be specified, which includes a string or list of strings representing a file or files to be appended to the end of the CA bundle, once it is downloaded.

CLI Example:

```
salt '*' http.update_ca_bundle merge_files=/path/to/mycert.pem
```

22.16.97 salt.modules.ilo

Manage HP ILO

depends hponcfg (SmartStart Scripting Toolkit Linux Edition)

salt.modules.ilo.**change_password**(*username*, *password*)
Reset a users password

CLI Example:

```
salt '*' ilo.change_password damianMyerscough
```

salt.modules.ilo.**change_username**(*old_username*, *new_username*)
Change a username

CLI Example:

```
salt '*' ilo.change_username damian diana
```

salt.modules.ilo.**configure_network**(*ip*, *netmask*, *gateway*)
Configure Network Interface

CLI Example:

```
salt '*' ilo.configure_network [IP ADDRESS] [NETMASK] [GATEWAY]
```

salt.modules.ilo.**configure_snmp**(*community*, *snmp_port=161*, *snmp_trapport=161*)
Configure SNMP

CLI Example:

salt.modules.ilo.**create_user**(*name*, *password*, **privileges*)
Create user

CLI Example:

```
salt '*' ilo.create_user damian secretagent VIRTUAL_MEDIA_PRIV
```

If no permissions are specify the user will only have a read-only account.

Supported privelges:

- ADMIN_PRIV Enables the user to administer user accounts.
- REMOTE_CONS_PRIV Enables the user to access the Remote Console functionality.
- RESET_SERVER_PRIV Enables the user to remotely manipulate the server power setting.
- VIRTUAL_MEDIA_PRIV Enables the user permission to access the virtual media functionality.
- CONFIG_ILO_PRIV Enables the user to configure iLO settings.

salt.modules.ilo.**delete_ssh_key**(*username*)
Delete a users SSH key from the ILO

CLI Example:

```
salt '*' ilo.delete_user_sshkey damian
```

salt.modules.ilo.**delete_user**(*username*)
Delete a user

CLI Example:

```
salt '*' ilo.delete_user damian
```

salt.modules.ilo.**disable_dhcp**()
Disable DHCP

CLI Example:

```
salt '*' ilo.disable_dhcp
```

salt.modules.ilo.**disable_ssh**()
Disable the SSH daemon

CLI Example:

```
salt '*' ilo.disable_ssh
```

salt.modules.ilo.**enable_dhcp**()
Enable DHCP

CLI Example:

```
salt '*' ilo.enable_dhcp
```

salt.modules.ilo.**enable_ssh**()
Enable the SSH daemon

CLI Example:

```
salt '*' ilo.enable_ssh
```

salt.modules.ilo.**get_user**(*username*)
Returns local user information, excluding the password

CLI Example:

```
salt '*' ilo.get_user damian
```

salt.modules.ilo.**global_settings**()
Show global settings

CLI Example:

```
salt '*' ilo.global_settings
```

salt.modules.ilo.**list_users**()
List all users

CLI Example:

```
salt '*' ilo.list_users
```

salt.modules.ilo.**list_users_info**()
List all users in detail

CLI Example:

```
salt '*' ilo.list_users_info
```

salt.modules.ilo.**network**()
Grab the current network settings

CLI Example:

```
salt '*' ilo.network
```

salt.modules.ilo.**set_http_port**(*port=80*)
Configure the port HTTP should listen on

CLI Example:

salt.modules.ilo.**set_https_port**(*port=443*)
Configure the port HTTPS should listen on

CLI Example:

```
salt '*' ilo.set_https_port 4334
```

salt.modules.ilo.**set_ssh_key**(*public_key*)
Configure SSH public keys for specific users

CLI Example:

```
salt '*' ilo.set_ssh_key "ssh-dss AAAAB3NzaC1kc3MAAACBA... damian"
```

The SSH public key needs to be DSA and the last argument in the key needs to be the username (case-senstive) of the ILO username.

salt.modules.ilo.**set_ssh_port**(*port=22*)
Enable SSH on a user defined port

CLI Example:

```
salt '*' ilo.set_ssh_port 2222
```

22.16.98 salt.modules.img

Virtual machine image management tools

salt.modules.img.**bootstrap**(*location*, *size*, *fmt*)
HIGHLY EXPERIMENTAL Bootstrap a virtual machine image

location: The location to create the image

size: The size of the image to create in megabytes

fmt: The image format, raw or qcow2

CLI Example:

```
salt '*' img.bootstrap /srv/salt-images/host.qcow 4096 qcow2
```

salt.modules.img.**mount_image**(*location*)
Mount the named image and return the mount point

CLI Example:

```
salt '*' img.mount_image /tmp/foo
```

salt.modules.img.**umount_image**(*mnt*)
Unmount an image mountpoint

CLI Example:

```
salt '*' img.umount_image /mnt/foo
```

22.16.99 salt.modules.incron

Work with incron

salt.modules.incron.**list_tab**(*user*)
Return the contents of the specified user's incrontab

CLI Example:

```
salt '*' incron.list_tab root
```

salt.modules.incron.**ls**(*user*)
Return the contents of the specified user's incrontab

CLI Example:

```
salt '*' incron.list_tab root
```

salt.modules.incron.**raw_incron**(*user*)
Return the contents of the user's incrontab

CLI Example:

```
salt '*' incron.raw_cron root
```

salt.modules.incron.**raw_system_incron**()
Return the contents of the system wide incrontab

CLI Example:

```
salt '*' incron.raw_system_cron
```

salt.modules.incron.**rm**(*user*, *path*, *mask*, *cmd*)
Remove a cron job for a specified user. If any of the day/time params are specified, the job will only be removed if the specified params match.

CLI Example:

```
salt '*' incron.rm_job root /path
```

salt.modules.incron.**rm_job**(*user*, *path*, *mask*, *cmd*)
Remove a cron job for a specified user. If any of the day/time params are specified, the job will only be removed if the specified params match.

CLI Example:

```
salt '*' incron.rm_job root /path
```

salt.modules.incron.**set_job**(*user*, *path*, *mask*, *cmd*)
Sets a cron job up for a specified user.

CLI Example:

```
salt '*' incron.set_job root '/root' 'IN_MODIFY' 'echo "$$ $@ $# $% $&"'
```

salt.modules.incron.**write_cron_file_verbose**(*user*, *path*)
Writes the contents of a file to a user's crontab and return error message on error

CLI Example:

```
salt '*' incron.write_incron_file_verbose root /tmp/new_cron
```

salt.modules.incron.**write_incron_file**(*user*, *path*)
Writes the contents of a file to a user's crontab

CLI Example:

```
salt '*' incron.write_cron_file root /tmp/new_cron
```

22.16.100 salt.modules.influx

InfluxDB - A distributed time series database

Module to provide InfluxDB compatibility to Salt (compatible with InfluxDB version 0.5+)

New in version 2014.7.0.

depends

- influxdb Python module

configuration This module accepts connection configuration details either as parameters or as configuration settings in /etc/salt/minion on the relevant minions:

```
influxdb.host: 'localhost'
influxdb.port: 8086
influxdb.user: 'root'
influxdb.password: 'root'
```

This data can also be passed into pillar. Options passed into opts will overwrite options passed into pillar.

salt.modules.influx.**db_create**(*name*, *user=None*, *password=None*, *host=None*, *port=None*)
Create a database

name Database name to create

user The user to connect as

password The password of the user

host The host to connect to

port The port to connect to

CLI Example:

```
salt '*' influxdb.db_create <name>
salt '*' influxdb.db_create <name> <user> <password> <host> <port>
```

salt.modules.influx.**db_exists**(*name*, *user=None*, *password=None*, *host=None*, *port=None*)
Checks if a database exists in InfluxDB

name Database name to create

user The user to connect as

password The password of the user

host The host to connect to

port The port to connect to

CLI Example:

```
salt '*' influxdb.db_exists <name>
salt '*' influxdb.db_exists <name> <user> <password> <host> <port>
```

salt.modules.influx.**db_list**(*user=None*, *password=None*, *host=None*, *port=None*)
List all InfluxDB databases

user The user to connect as

password The password of the user

host The host to connect to

port The port to connect to

CLI Example:

```
salt '*' influxdb.db_list
salt '*' influxdb.db_list <user> <password> <host> <port>
```

salt.modules.influx.**db_remove**(*name*, *user=None*, *password=None*, *host=None*, *port=None*)
Remove a database

name Database name to remove

user The user to connect as

password The password of the user

host The host to connect to

port The port to connect to

CLI Example:

```
salt '*' influxdb.db_remove <name>
salt '*' influxdb.db_remove <name> <user> <password> <host> <port>
```

salt.modules.influx.**query**(*database*, *query*, *time_precision='s'*, *chunked=False*, *user=None*, *password=None*, *host=None*, *port=None*)
Querying data

database The database to query

query Query to be executed

time_precision Time precision to use (`s', `m', or `u')

chunked Whether is chunked or not

user The user to connect as

password The password of the user

host The host to connect to

port The port to connect to

CLI Example:

```
    salt '*' influxdb.query <database> <query>
    salt '*' influxdb.query <database> <query> <time_precision> <chunked> <user> <password> <host> <
```

salt.modules.influx.**user_chpass**(*name*, *passwd*, *database=None*, *user=None*, *password=None*, *host=None*, *port=None*)

Change password for a cluster admin or a database user.

If a database is specified: it will update database user password. If a database is not specified: it will update cluster admin password.

name User name for whom to change the password

passwd New password

database The database on which to operate

user The user to connect as

password The password of the user

host The host to connect to

port The port to connect to

CLI Example:

```
    salt '*' influxdb.user_chpass <name> <passwd>
    salt '*' influxdb.user_chpass <name> <passwd> <database>
    salt '*' influxdb.user_chpass <name> <passwd> <database> <user> <password> <host> <port>
```

salt.modules.influx.**user_create**(*name*, *passwd*, *database=None*, *user=None*, *password=None*, *host=None*, *port=None*)

Create a cluster admin or a database user.

If a database is specified: it will create database user. If a database is not specified: it will create a cluster admin.

name User name for the new user to create

passwd Password for the new user to create

database The database to create the user in

user The user to connect as

password The password of the user

host The host to connect to

port The port to connect to

CLI Example:

```
    salt '*' influxdb.user_create <name> <passwd>
    salt '*' influxdb.user_create <name> <passwd> <database>
    salt '*' influxdb.user_create <name> <passwd> <database> <user> <password> <host> <port>
```

salt.modules.influx.**user_exists**(*name*, *database=None*, *user=None*, *password=None*, *host=None*, *port=None*)

Checks if a cluster admin or database user exists.

If a database is specified: it will check for database user existence. If a database is not specified: it will check for cluster admin existence.

name User name

database The database to check for the user to exist

user The user to connect as

password The password of the user

host The host to connect to

port The port to connect to

CLI Example:

```
salt '*' influxdb.user_exists <name>
salt '*' influxdb.user_exists <name> <database>
salt '*' influxdb.user_exists <name> <database> <user> <password> <host> <port>
```

salt.modules.influx.**user_list**(*database=None*, *user=None*, *password=None*, *host=None*, *port=None*)

List cluster admins or database users.

If a database is specified: it will return database users list. If a database is not specified: it will return cluster admins list.

database The database to list the users from

user The user to connect as

password The password of the user

host The host to connect to

port The port to connect to

CLI Example:

```
salt '*' influxdb.user_list
salt '*' influxdb.user_list <database>
salt '*' influxdb.user_list <database> <user> <password> <host> <port>
```

salt.modules.influx.**user_remove**(*name*, *database=None*, *user=None*, *password=None*, *host=None*, *port=None*)

Remove a cluster admin or a database user.

If a database is specified: it will remove the database user. If a database is not specified: it will remove the cluster admin.

name User name to remove

database The database to remove the user from

user User name for the new user to delete

user The user to connect as

password The password of the user

host The host to connect to

port The port to connect to

CLI Example:

```
salt '*' influxdb.user_remove <name>
salt '*' influxdb.user_remove <name> <database>
salt '*' influxdb.user_remove <name> <database> <user> <password> <host> <port>
```

22.16.101 salt.modules.ini_manage

Edit ini files

maintainer <akilesh1597@gmail.com>

maturity new

depends re

platform all

Use section as DEFAULT_IMPLICIT if your ini file does not have any section (for example /etc/sysctl.conf)

`salt.modules.ini_manage.`**`get_option`**(*file_name*, *section*, *option*)

Get value of a key from a section in an ini file. Returns `None` if no matching key was found.

API Example:

```
import salt
sc = salt.client.get_local_client()
sc.cmd('target', 'ini.get_option',
       [path_to_ini_file, section_name, option])
```

CLI Example:

```
salt '*' ini.get_option /path/to/ini section_name option_name
```

`salt.modules.ini_manage.`**`get_section`**(*file_name*, *section*)

Retrieve a section from an ini file. Returns the section as dictionary. If the section is not found, an empty dictionary is returned.

API Example:

```
import salt
sc = salt.client.get_local_client()
sc.cmd('target', 'ini.get_section',
       [path_to_ini_file, section_name])
```

CLI Example:

```
salt '*' ini.get_section /path/to/ini section_name
```

`salt.modules.ini_manage.`**`remove_option`**(*file_name*, *section*, *option*)

Remove a key/value pair from a section in an ini file. Returns the value of the removed key, or `None` if nothing was removed.

API Example:

```
import salt
sc = salt.client.get_local_client()
sc.cmd('target', 'ini.remove_option',
       [path_to_ini_file, section_name, option])
```

CLI Example:

```
salt '*' ini.remove_option /path/to/ini section_name option_name
```

`salt.modules.ini_manage.`**`remove_section`**(*file_name*, *section*)

Remove a section in an ini file. Returns the removed section as dictionary, or `None` if nothing was removed.

API Example:

```
import salt
sc = salt.client.get_local_client()
sc.cmd('target', 'ini.remove_section',
       [path_to_ini_file, section_name])
```

CLI Example:

```
salt '*' ini.remove_section /path/to/ini section_name
```

salt.modules.ini_manage.**set_option**(*file_name*, *sections=None*, *summary=True*)

Edit an ini file, replacing one or more sections. Returns a dictionary containing the changes made.

file_name path of ini_file

sections [None] A dictionary representing the sections to be edited ini file

Set `summary=False` if return data need not have previous option value

API Example:

```
import salt
sc = salt.client.get_local_client()
sc.cmd('target', 'ini.set_option',
       ['path_to_ini_file', '{"section_to_change": {"key": "value"}}'])
```

CLI Example:

```
salt '*' ini.set_option /path/to/ini '{section_foo: {key: value}}'
```

22.16.102 salt.modules.introspect

Functions to perform introspection on a minion, and return data in a format usable by Salt States

salt.modules.introspect.**enabled_service_owners**()

Return which packages own each of the services that are currently enabled.

CLI Example:

salt myminion introspect.enabled_service_owners

salt.modules.introspect.**running_service_owners**(*exclude=('/dev', '/home', '/media', '/proc', '/run', '/sys/', '/tmp', '/var')*)

Determine which packages own the currently running services. By default, excludes files whose full path starts with `/dev`, `/home`, `/media`, `/proc`, `/run`, `/sys`, `/tmp` and `/var`. This can be overridden by passing in a new list to `exclude`.

CLI Example:

salt myminion introspect.running_service_owners

salt.modules.introspect.**service_highstate**(*requires=True*)

Return running and enabled services in a highstate structure. By default also returns package dependencies for those services, which means that package definitions must be created outside this function. To drop the package dependencies, set `requires` to False.

CLI Example:

salt myminion introspect.service_highstate salt myminion introspect.service_highstate requires=False

22.16.103 salt.modules.ipset

Support for ipset

salt.modules.ipset.**add**(*set=None*, *entry=None*, *family='ipv4'*, ***kwargs*)
Append an entry to the specified set.

CLI Example:

```
salt '*' ipset.add setname 192.168.1.26

salt '*' ipset.add setname 192.168.0.3,AA:BB:CC:DD:EE:FF
```

salt.modules.ipset.**check**(*set=None*, *entry=None*, *family='ipv4'*)
Check that an entry exists in the specified set.

CLI Example:

```
salt '*' ipset.check setname '192.168.0.1 comment "Hello"'
```

salt.modules.ipset.**check_set**(*set=None*, *family='ipv4'*)
New in version 2014.7.0.

Check that given ipset set exists.

CLI Example:

```
salt '*' ipset.check_set setname
```

salt.modules.ipset.**delete**(*set=None*, *entry=None*, *family='ipv4'*, ***kwargs*)
Delete an entry from the specified set.

CLI Example:

```
salt '*' ipset.delete setname 192.168.0.3,AA:BB:CC:DD:EE:FF
```

salt.modules.ipset.**delete_set**(*set=None*, *family='ipv4'*)
New in version 2014.7.0.

Delete ipset set.

CLI Example:

```
salt '*' ipset.delete_set custom_set

IPv6:
salt '*' ipset.delete_set custom_set family=ipv6
```

salt.modules.ipset.**flush**(*set=None*, *family='ipv4'*)
Flush entries in the specified set, Flush all sets if set is not specified.

CLI Example:

```
salt '*' ipset.flush

salt '*' ipset.flush set

IPv6:
salt '*' ipset.flush

salt '*' ipset.flush set
```

salt.modules.ipset.**list_sets**(*family='ipv4'*)
New in version 2014.7.0.

List all ipset sets.

CLI Example:

```
salt '*' ipset.list_sets
```

salt.modules.ipset.**new_set**(*set=None*, *set_type=None*, *family='ipv4'*, *comment=False*, ***kwargs*)
New in version 2014.7.0.

Create new custom set

CLI Example:

```
salt '*' ipset.new_set custom_set list:set

salt '*' ipset.new_set custom_set list:set comment=True

IPv6:
salt '*' ipset.new_set custom_set list:set family=ipv6
```

salt.modules.ipset.**rename_set**(*set=None*, *new_set=None*, *family='ipv4'*)
New in version 2014.7.0.

Delete ipset set.

CLI Example:

```
salt '*' ipset.rename_set custom_set new_set=new_set_name

IPv6:
salt '*' ipset.rename_set custom_set new_set=new_set_name family=ipv6
```

salt.modules.ipset.**test**(*set=None*, *entry=None*, *family='ipv4'*, ***kwargs*)
Test if an entry is in the specified set.

CLI Example:

```
salt '*' ipset.test setname 192.168.0.2

IPv6:
salt '*' ipset.test setname fd81:fc56:9ac7::/48
```

salt.modules.ipset.**version**()
Return version from ipset --version

CLI Example:

```
salt '*' ipset.version
```

22.16.104 salt.modules.iptables

Support for iptables

salt.modules.iptables.**append**(*table='filter'*, *chain=None*, *rule=None*, *family='ipv4'*)
Append a rule to the specified table/chain.

This function accepts a rule in a standard iptables command format, starting with the chain. Trying to force users to adapt to a new method of creating rules would be irritating at best, and we already have a parser that can handle it.

CLI Example:

```
salt '*' iptables.append filter INPUT \
    rule='-m state --state RELATED,ESTABLISHED -j ACCEPT'

IPv6:
salt '*' iptables.append filter INPUT \
    rule='-m state --state RELATED,ESTABLISHED -j ACCEPT' \
    family=ipv6
```

salt.modules.iptables.**build_rule**(*table=None*, *chain=None*, *command=None*, *position=''*, *full=None*, *family='ipv4'*, ***kwargs*)

Build a well-formatted iptables rule based on kwargs. Long options must be used (*--jump* instead of *-j*) because they will have the -- added to them. A *table* and *chain* are not required, unless *full* is True.

If *full* is *True*, then *table*, *chain* and *command* are required. *command* may be specified as either a short option (`I') or a long option (*--insert*). This will return the iptables command, exactly as it would be used from the command line.

If a position is required (as with *-I* or *-D*), it may be specified as *position*. This will only be useful if *full* is True.

If *connstate* is passed in, it will automatically be changed to *state*.

CLI Examples:

```
salt '*' iptables.build_rule match=state \
    connstate=RELATED,ESTABLISHED jump=ACCEPT

salt '*' iptables.build_rule filter INPUT command=I position=3 \
    full=True match=state state=RELATED,ESTABLISHED jump=ACCEPT

salt '*' iptables.build_rule filter INPUT command=A \
    full=True match=state state=RELATED,ESTABLISHED \
    source='127.0.0.1' jump=ACCEPT

.. Invert Rules
salt '*' iptables.build_rule filter INPUT command=A \
    full=True match=state state=RELATED,ESTABLISHED \
    source='! 127.0.0.1' jump=ACCEPT

salt '*' iptables.build_rule filter INPUT command=A \
    full=True match=state state=RELATED,ESTABLISHED \
    destination='not 127.0.0.1' jump=ACCEPT

IPv6:
salt '*' iptables.build_rule match=state \
    connstate=RELATED,ESTABLISHED jump=ACCEPT \
    family=ipv6
salt '*' iptables.build_rule filter INPUT command=I position=3 \
    full=True match=state state=RELATED,ESTABLISHED jump=ACCEPT \
    family=ipv6
```

salt.modules.iptables.**check**(*table='filter'*, *chain=None*, *rule=None*, *family='ipv4'*)

Check for the existence of a rule in the table and chain

This function accepts a rule in a standard iptables command format, starting with the chain. Trying to force users to adapt to a new method of creating rules would be irritating at best, and we already have a

parser that can handle it.

CLI Example:

```
salt '*' iptables.check filter INPUT \
    rule='-m state --state RELATED,ESTABLISHED -j ACCEPT'

IPv6:
salt '*' iptables.check filter INPUT \
    rule='-m state --state RELATED,ESTABLISHED -j ACCEPT' \
    family=ipv6
```

salt.modules.iptables.**check_chain**(*table='filter'*, *chain=None*, *family='ipv4'*)

New in version 2014.1.0.

Check for the existence of a chain in the table

CLI Example:

```
salt '*' iptables.check_chain filter INPUT

IPv6:
salt '*' iptables.check_chain filter INPUT family=ipv6
```

salt.modules.iptables.**delete**(*table*, *chain=None*, *position=None*, *rule=None*, *family='ipv4'*)

Delete a rule from the specified table/chain, specifying either the rule in its entirety, or the rule's position in the chain.

This function accepts a rule in a standard iptables command format, starting with the chain. Trying to force users to adapt to a new method of creating rules would be irritating at best, and we already have a parser that can handle it.

CLI Examples:

```
salt '*' iptables.delete filter INPUT position=3
salt '*' iptables.delete filter INPUT \
    rule='-m state --state RELATED,ESTABLISHED -j ACCEPT'

IPv6:
salt '*' iptables.delete filter INPUT position=3 family=ipv6
salt '*' iptables.delete filter INPUT \
    rule='-m state --state RELATED,ESTABLISHED -j ACCEPT' \
    family=ipv6
```

salt.modules.iptables.**delete_chain**(*table='filter'*, *chain=None*, *family='ipv4'*)

New in version 2014.1.0.

Delete custom chain to the specified table.

CLI Example:

```
salt '*' iptables.delete_chain filter CUSTOM_CHAIN

IPv6:
salt '*' iptables.delete_chain filter CUSTOM_CHAIN family=ipv6
```

salt.modules.iptables.**flush**(*table='filter'*, *chain=''*, *family='ipv4'*)

Flush the chain in the specified table, flush all chains in the specified table if not specified chain.

CLI Example:

```
    salt '*' iptables.flush filter INPUT

    IPv6:
    salt '*' iptables.flush filter INPUT family=ipv6
```

salt.modules.iptables.**get_policy**(*table='filter'*, *chain=None*, *family='ipv4'*)

Return the current policy for the specified table/chain

CLI Example:

```
    salt '*' iptables.get_policy filter INPUT

    IPv6:
    salt '*' iptables.get_policy filter INPUT family=ipv6
```

salt.modules.iptables.**get_rules**(*family='ipv4'*)

Return a data structure of the current, in-memory rules

CLI Example:

```
    salt '*' iptables.get_rules

    IPv6:
    salt '*' iptables.get_rules family=ipv6
```

salt.modules.iptables.**get_saved_policy**(*table='filter'*, *chain=None*, *conf_file=None*, *family='ipv4'*)

Return the current policy for the specified table/chain

CLI Examples:

```
    salt '*' iptables.get_saved_policy filter INPUT
    salt '*' iptables.get_saved_policy filter INPUT \
        conf_file=/etc/iptables.saved

    IPv6:
    salt '*' iptables.get_saved_policy filter INPUT family=ipv6
    salt '*' iptables.get_saved_policy filter INPUT \
        conf_file=/etc/iptables.saved family=ipv6
```

salt.modules.iptables.**get_saved_rules**(*conf_file=None*, *family='ipv4'*)

Return a data structure of the rules in the conf file

CLI Example:

```
    salt '*' iptables.get_saved_rules

    IPv6:
    salt '*' iptables.get_saved_rules family=ipv6
```

salt.modules.iptables.**insert**(*table='filter'*, *chain=None*, *position=None*, *rule=None*, *family='ipv4'*)

Insert a rule into the specified table/chain, at the specified position.

This function accepts a rule in a standard iptables command format, starting with the chain. Trying to force users to adapt to a new method of creating rules would be irritating at best, and we already have a parser that can handle it.

If the position specified is a negative number, then the insert will be performed counting from the end of the list. For instance, a position of -1 will insert the rule as the second to last rule. To insert a rule in the last position, use the append function instead.

CLI Examples:

```
salt '*' iptables.insert filter INPUT position=3 \
    rule='-m state --state RELATED,ESTABLISHED -j ACCEPT'

IPv6:
salt '*' iptables.insert filter INPUT position=3 \
    rule='-m state --state RELATED,ESTABLISHED -j ACCEPT' \
    family=ipv6
```

salt.modules.iptables.**new_chain**(*table='filter'*, *chain=None*, *family='ipv4'*)

New in version 2014.1.0.

Create new custom chain to the specified table.

CLI Example:

```
salt '*' iptables.new_chain filter CUSTOM_CHAIN

IPv6:
salt '*' iptables.new_chain filter CUSTOM_CHAIN family=ipv6
```

salt.modules.iptables.**save**(*filename=None*, *family='ipv4'*)

Save the current in-memory rules to disk

CLI Example:

```
salt '*' iptables.save /etc/sysconfig/iptables

IPv6:
salt '*' iptables.save /etc/sysconfig/iptables family=ipv6
```

salt.modules.iptables.**set_policy**(*table='filter'*, *chain=None*, *policy=None*, *family='ipv4'*)

Set the current policy for the specified table/chain

CLI Example:

```
salt '*' iptables.set_policy filter INPUT ACCEPT

IPv6:
salt '*' iptables.set_policy filter INPUT ACCEPT family=ipv6
```

salt.modules.iptables.**version**(*family='ipv4'*)

Return version from iptables --version

CLI Example:

```
salt '*' iptables.version

IPv6:
salt '*' iptables.version family=ipv6
```

22.16.105 salt.modules.jboss7

Module for managing JBoss AS 7 through the CLI interface.

In order to run each function, jboss_config dictionary with the following properties must be passed:

- cli_path: the path to jboss-cli script, for example: `/opt/jboss/jboss-7.0/bin/jboss-cli.sh'
- controller: the ip addres and port of controller, for example: 10.11.12.13:9999

- cli_user: username to connect to jboss administration console if necessary
- cli_password: password to connect to jboss administration console if necessary

Example:

```
jboss_config:
   cli_path: '/opt/jboss/jboss-7.0/bin/jboss-cli.sh'
   controller: 10.11.12.13:9999
   cli_user: 'jbossadm'
   cli_password: 'jbossadm'
```

salt.modules.jboss7.**create_datasource**(*jboss_config*, *name*, *datasource_properties*)

Create datasource in running jboss instance

jboss_config Configuration dictionary with properties specified above.

name Datasource name

datasource_properties

A dictionary of datasource properties to be created:

- driver-name: mysql
- connection-url: `jdbc:mysql://localhost:3306/sampleDatabase`
- jndi-name: `java:jboss/datasources/sampleDS'
- user-name: sampleuser
- password: secret
- min-pool-size: 3
- use-java-context: True

CLI Example:

```
salt '*' jboss7.create_datasource '{"cli_path": "integration.modules.sysmod.SysModuleTest.test_v
```

salt.modules.jboss7.**create_simple_binding**(*jboss_config*, *binding_name*, *value*)

Create a simple jndi binding in the running jboss instance

jboss_config Configuration dictionary with properties specified above.

binding_name Binding name to be created

value Binding value

CLI Example:

```
salt '*' jboss7.create_simple_binding '{"cli_path": "integration.modules.sysmod.SysModuleTest.te
```

salt.modules.jboss7.**deploy**(*jboss_config*, *source_file*)

Deploy the application on the jboss instance from the local file system where minion is running.

jboss_config Configuration dictionary with properties specified above.

source_file Source file to deploy from

CLI Example:

```
salt '*' jboss7.deploy '{"cli_path": "integration.modules.sysmod.SysModuleTest.test_valid_docs",
```

salt.modules.jboss7.**list_deployments**(*jboss_config*)

List all deployments on the jboss instance

jboss_config

Configuration dictionary with properties specified above.

CLI Example:

```
salt '*' jboss7.list_deployments '{"cli_path": "integration.modules.sysmod.SysModuleTest.tes
```

salt.modules.jboss7.**read_datasource**(*jboss_config*, *name*)

Read datasource properties in the running jboss instance.

jboss_config Configuration dictionary with properties specified above.

name Datasource name

CLI Example:

```
salt '*' jboss7.read_datasource '{"cli_path": "integration.modules.sysmod.SysModuleTest.test_val
```

salt.modules.jboss7.**read_simple_binding**(*jboss_config*, *binding_name*)

Read jndi binding in the running jboss instance

jboss_config Configuration dictionary with properties specified above.

binding_name Binding name to be created

CLI Example:

```

```

salt `*' jboss7.read_simple_binding `{``cli_path'': ``integration.modules.sysmod.SysModuleTest.test_valid_docs'', ``controller'': ``10.11.12.13:9999'', ``cli_user'': ``jbossadm'', ``cli_password'': ``jbossadm''}' my_binding_name

salt.modules.jboss7.**reload**(*jboss_config*)

Reload running jboss instance

jboss_config Configuration dictionary with properties specified above.

CLI Example:

```
salt '*' jboss7.reload '{"cli_path": "integration.modules.sysmod.SysModuleTest.test_valid_docs",
```

salt.modules.jboss7.**remove_datasource**(*jboss_config*, *name*)

Remove an existing datasource from the running jboss instance.

jboss_config Configuration dictionary with properties specified above.

name Datasource name

CLI Example:

```
salt '*' jboss7.remove_datasource '{"cli_path": "integration.modules.sysmod.SysModuleTest.test_v
```

salt.modules.jboss7.**status**(*jboss_config*, *timeout=5*)

Get status of running jboss instance.

jboss_config Configuration dictionary with properties specified above.

CLI Example:

```
salt '*' jboss7.status '{"cli_path": "integration.modules.sysmod.SysModuleTest.test_valid_docs",
```

salt.modules.jboss7.**stop_server**(*jboss_config*)

Stop running jboss instance

jboss_config Configuration dictionary with properties specified above.

CLI Example:

```
salt '*' jboss7.stop_server '{"cli_path": "integration.modules.sysmod.SysModuleTest.test_valid_d
```

salt.modules.jboss7.**undeploy**(*jboss_config*, *deployment*)

Undeploy the application from jboss instance

jboss_config Configuration dictionary with properties specified above.

deployment Deployment name to undeploy

CLI Example:

```
salt '*' jboss7.undeploy '{"cli_path": "integration.modules.sysmod.SysModuleTest.test_valid_docs
```

salt.modules.jboss7.**update_datasource**(*jboss_config*, *name*, *new_properties*)

Update an existing datasource in running jboss instance. If the property doesn't exist if will be created, if it does, it will be updated with the new value

jboss_config Configuration dictionary with properties specified above.

name Datasource name

new_properties

A dictionary of datasource properties to be updated. For example:

- driver-name: mysql
- connection-url: `jdbc:mysql://localhost:3306/sampleDatabase`
- jndi-name: `java:jboss/datasources/sampleDS'
- user-name: sampleuser
- password: secret
- min-pool-size: 3
- use-java-context: True

CLI Example:

```
salt '*' jboss7.update_datasource '{"cli_path": "integration.modules.sysmod.SysModuleTest.test_v
```

salt.modules.jboss7.**update_simple_binding**(*jboss_config*, *binding_name*, *value*)

Update the simple jndi binding in the running jboss instance

jboss_config Configuration dictionary with properties specified above.

binding_name Binding name to be updated

value New binding value

CLI Example:

```
salt '*' jboss7.update_simple_binding '{"cli_path": "integration.modules.sysmod.SysModuleTest.te
```

22.16.106 salt.modules.jboss7_cli

Module for low-level interaction with JbossAS7 through CLI.

This module exposes two ways of interaction with the CLI, either through commands or operations.

Note: Following JBoss documentation (https://developer.jboss.org/wiki/CommandLineInterface): ``Operations are considered a low level but comprehensive way to manage the AS controller, i.e. if it can't be done with operations it can't be done in any other way. Commands, on the other hand, are more user-friendly in syntax, although most of them still translate into operation requests and some of them even into a few composite operation requests, i.e. commands also simplify some management operations from the user's point of view.''

The difference between calling a command or operation is in handling the result. Commands return a zero return code if operation is successful or return non-zero return code and print an error to standard output in plain text, in case of an error.

Operations return a json-like structure, that contain more information about the result. In case of a failure, they also return a specific return code. This module parses the output from the operations and returns it as a dictionary so that an execution of an operation can then be verified against specific errors.

In order to run each function, jboss_config dictionary with the following properties must be passed:

- cli_path: the path to jboss-cli script, for example: `/opt/jboss/jboss-7.0/bin/jboss-cli.sh'
- controller: the ip addres and port of controller, for example: 10.11.12.13:9999
- cli_user: username to connect to jboss administration console if necessary
- cli_password: password to connect to jboss administration console if necessary

Example:

```
jboss_config:
   cli_path: '/opt/jboss/jboss-7.0/bin/jboss-cli.sh'
   controller: 10.11.12.13:9999
   cli_user: 'jbossadm'
   cli_password: 'jbossadm'
```

`salt.modules.jboss7_cli.`**`run_command`**(*jboss_config*, *command*, *fail_on_error=True*)

Execute a command against jboss instance through the CLI interface.

jboss_config Configuration dictionary with properties specified above.

command Command to execute against jboss instance

fail_on_error (default=True) Is true, raise CommandExecutionException exception if execution fails. If false, `success' property of the returned dictionary is set to False

CLI Example:

```
salt '*' jboss7_cli.run_command '{"cli_path": "integration.modules.sysmod.SysModuleTest.test_val
```

`salt.modules.jboss7_cli.`**`run_operation`**(*jboss_config*, *operation*, *fail_on_error=True*, *retries=1*)

Execute an operation against jboss instance through the CLI interface.

jboss_config Configuration dictionary with properties specified above.

operation An operation to execute against jboss instance

fail_on_error (default=True) Is true, raise CommandExecutionException exception if execution fails. If false, `success' property of the returned dictionary is set to False

retries: Number of retries in case of ``JBAS012144: Could not connect to remote'' error.

CLI Example:

```
salt '*' jboss7_cli.run_operation '{"cli_path": "integration.modules.sysmod.SysModuleTest.test_v
```

22.16.107 salt.modules.junos

Module for interfacing to Junos devices

ALPHA QUALITY code.

salt.modules.junos.**commit**()

salt.modules.junos.**diff**()

salt.modules.junos.**facts_refresh**()
Reload the facts dictionary from the device. Usually only needed if the device configuration is changed by some other actor.

salt.modules.junos.**ping**()

salt.modules.junos.**rollback**()

salt.modules.junos.**set_hostname**(*hostname=None*, *commit_change=True*)

22.16.108 salt.modules.key

Functions to view the minion's public key information

salt.modules.key.**finger**()
Return the minion's public key fingerprint

CLI Example:

```
salt '*' key.finger
```

salt.modules.key.**finger_master**()
Return the fingerprint of the master's public key on the minion.

CLI Example:

```
salt '*' key.finger_master
```

22.16.109 salt.modules.keyboard

Module for managing keyboards on supported POSIX-like systems using systemd, or such as Redhat, Debian and Gentoo.

salt.modules.keyboard.**get_sys**()
Get current system keyboard setting

CLI Example:

```
salt '*' keyboard.get_sys
```

salt.modules.keyboard.**get_x**()
Get current X keyboard setting

CLI Example:

```
salt '*' keyboard.get_x
```

salt.modules.keyboard.**set_sys**(*layout*)
Set current system keyboard setting

CLI Example:

```
salt '*' keyboard.set_sys dvorak
```

salt.modules.keyboard.**set_x**(*layout*)
Set current X keyboard setting

CLI Example:

```
salt '*' keyboard.set_x dvorak
```

22.16.110 salt.modules.keystone

Module for handling openstack keystone calls.

optdepends

- keystoneclient Python adapter

configuration This module is not usable until the following are specified either in a pillar or in the minion's config file:

```
keystone.user: admin
keystone.password: verybadpass
keystone.tenant: admin
keystone.tenant_id: f80919baedab48ec8931f200c65a50df
keystone.auth_url: 'http://127.0.0.1:5000/v2.0/'
```

OR (for token based authentication)

```
keystone.token: 'ADMIN'
keystone.endpoint: 'http://127.0.0.1:35357/v2.0'
```

If configuration for multiple openstack accounts is required, they can be set up as different configuration profiles. For example:

```
openstack1:
  keystone.user: admin
  keystone.password: verybadpass
  keystone.tenant: admin
  keystone.tenant_id: f80919baedab48ec8931f200c65a50df
  keystone.auth_url: 'http://127.0.0.1:5000/v2.0/'

openstack2:
  keystone.user: admin
  keystone.password: verybadpass
  keystone.tenant: admin
  keystone.tenant_id: f80919baedab48ec8931f200c65a50df
  keystone.auth_url: 'http://127.0.0.2:5000/v2.0/'
```

With this configuration in place, any of the keystone functions can make use of a configuration profile by declaring it explicitly. For example:

```
salt '*' keystone.tenant_list profile=openstack1
```

salt.modules.keystone.**auth**(*profile=None*, ***connection_args*)
Set up keystone credentials

Only intended to be used within Keystone-enabled modules

salt.modules.keystone.**ec2_credentials_create**(*user_id=None, name=None, tenant_id=None, tenant=None, profile=None, **connection_args*)

Create EC2-compatible credentials for user per tenant

CLI Examples:

```
salt '*' keystone.ec2_credentials_create name=admin tenant=admin
salt '*' keystone.ec2_credentials_create user_id=c965f79c4f864eaaa9c3b41904e67082 tenant_id=7227
```

salt.modules.keystone.**ec2_credentials_delete**(*user_id=None, name=None, access_key=None, profile=None, **connection_args*)

Delete EC2-compatible credentials

CLI Examples:

```
salt '*' keystone.ec2_credentials_delete 860f8c2c38ca4fab989f9bc56a061a64 access_key=5f66d2f24f6
salt '*' keystone.ec2_credentials_delete name=admin access_key=5f66d2f24f604b8bb9cd28886106f442
```

salt.modules.keystone.**ec2_credentials_get**(*user_id=None, name=None, access=None, profile=None, **connection_args*)

Return ec2_credentials for a user (keystone ec2-credentials-get)

CLI Examples:

```
salt '*' keystone.ec2_credentials_get c965f79c4f864eaaa9c3b41904e67082 access=722787eb540849158
salt '*' keystone.ec2_credentials_get user_id=c965f79c4f864eaaa9c3b41904e67082 access=722787eb54
salt '*' keystone.ec2_credentials_get name=nova access=722787eb540849158668370dc627ec5f
```

salt.modules.keystone.**ec2_credentials_list**(*user_id=None, name=None, profile=None, **connection_args*)

Return a list of ec2_credentials for a specific user (keystone ec2-credentials-list)

CLI Examples:

```
salt '*' keystone.ec2_credentials_list 298ce377245c4ec9b70e1c639c89e654
salt '*' keystone.ec2_credentials_list user_id=298ce377245c4ec9b70e1c639c89e654
salt '*' keystone.ec2_credentials_list name=jack
```

salt.modules.keystone.**endpoint_create**(*service, publicurl=None, internalurl=None, adminurl=None, region=None, profile=None, **connection_args*)

Create an endpoint for an Openstack service

CLI Examples:

```
salt '*' keystone.endpoint_create nova 'http://public/url'
    'http://internal/url' 'http://adminurl/url' region
```

salt.modules.keystone.**endpoint_delete**(*service, profile=None, **connection_args*)

Delete endpoints of an Openstack service

CLI Examples:

```
salt '*' keystone.endpoint_delete nova
```

salt.modules.keystone.**endpoint_get**(*service, profile=None, **connection_args*)

Return a specific endpoint (keystone endpoint-get)

CLI Example:

```
salt '*' keystone.endpoint_get nova
```

salt.modules.keystone.**endpoint_list**(*profile=None*, ***connection_args*)
Return a list of available endpoints (keystone endpoints-list)

CLI Example:

```
salt '*' keystone.endpoint_list
```

salt.modules.keystone.**role_create**(*name*, *profile=None*, ***connection_args*)
Create named role

```
salt '*' keystone.role_create admin
```

salt.modules.keystone.**role_delete**(*role_id=None*, *name=None*, *profile=None*, ***connection_args*)
Delete a role (keystone role-delete)

CLI Examples:

```
salt '*' keystone.role_delete c965f79c4f864eaaa9c3b41904e67082
salt '*' keystone.role_delete role_id=c965f79c4f864eaaa9c3b41904e67082
salt '*' keystone.role_delete name=admin
```

salt.modules.keystone.**role_get**(*role_id=None*, *name=None*, *profile=None*, ***connection_args*)
Return a specific roles (keystone role-get)

CLI Examples:

```
salt '*' keystone.role_get c965f79c4f864eaaa9c3b41904e67082
salt '*' keystone.role_get role_id=c965f79c4f864eaaa9c3b41904e67082
salt '*' keystone.role_get name=nova
```

salt.modules.keystone.**role_list**(*profile=None*, ***connection_args*)
Return a list of available roles (keystone role-list)

CLI Example:

```
salt '*' keystone.role_list
```

salt.modules.keystone.**service_create**(*name*, *service_type*, *description=None*, *profile=None*, ***connection_args*)
Add service to Keystone service catalog

CLI Examples:

```
salt '*' keystone.service_create nova compute 'OpenStack Compute Service'
```

salt.modules.keystone.**service_delete**(*service_id=None*, *name=None*, *profile=None*, ***connection_args*)
Delete a service from Keystone service catalog

CLI Examples:

```
salt '*' keystone.service_delete c965f79c4f864eaaa9c3b41904e67082
salt '*' keystone.service_delete name=nova
```

salt.modules.keystone.**service_get**(*service_id=None*, *name=None*, *profile=None*, ***connection_args*)
Return a specific services (keystone service-get)

CLI Examples:

```
salt '*' keystone.service_get c965f79c4f864eaaa9c3b41904e67082
salt '*' keystone.service_get service_id=c965f79c4f864eaaa9c3b41904e67082
salt '*' keystone.service_get name=nova
```

salt.modules.keystone.**service_list**(*profile=None*, ***connection_args*)

Return a list of available services (keystone services-list)

CLI Example:

```
salt '*' keystone.service_list
```

salt.modules.keystone.**tenant_create**(*name*, *description=None*, *enabled=True*, *profile=None*, ***connection_args*)

Create a keystone tenant

CLI Examples:

```
salt '*' keystone.tenant_create nova description='nova tenant'
salt '*' keystone.tenant_create test enabled=False
```

salt.modules.keystone.**tenant_delete**(*tenant_id=None*, *name=None*, *profile=None*, ***connection_args*)

Delete a tenant (keystone tenant-delete)

CLI Examples:

```
salt '*' keystone.tenant_delete c965f79c4f864eaaa9c3b41904e67082
salt '*' keystone.tenant_delete tenant_id=c965f79c4f864eaaa9c3b41904e67082
salt '*' keystone.tenant_delete name=demo
```

salt.modules.keystone.**tenant_get**(*tenant_id=None*, *name=None*, *profile=None*, ***connection_args*)

Return a specific tenants (keystone tenant-get)

CLI Examples:

```
salt '*' keystone.tenant_get c965f79c4f864eaaa9c3b41904e67082
salt '*' keystone.tenant_get tenant_id=c965f79c4f864eaaa9c3b41904e67082
salt '*' keystone.tenant_get name=nova
```

salt.modules.keystone.**tenant_list**(*profile=None*, ***connection_args*)

Return a list of available tenants (keystone tenants-list)

CLI Example:

```
salt '*' keystone.tenant_list
```

salt.modules.keystone.**tenant_update**(*tenant_id=None*, *name=None*, *description=None*, *enabled=None*, *profile=None*, ***connection_args*)

Update a tenant's information (keystone tenant-update) The following fields may be updated: name, email, enabled. Can only update name if targeting by ID

CLI Examples:

```
salt '*' keystone.tenant_update name=admin enabled=True
salt '*' keystone.tenant_update c965f79c4f864eaaa9c3b41904e67082 name=admin email=admin@domain.c
```

salt.modules.keystone.**token_get**(*profile=None*, ***connection_args*)

Return the configured tokens (keystone token-get)

CLI Example:

```
salt '*' keystone.token_get c965f79c4f864eaaa9c3b41904e67082
```

salt.modules.keystone.**user_create**(*name*, *password*, *email*, *tenant_id=None*, *enabled=True*, *profile=None*, ***connection_args*)

Create a user (keystone user-create)

CLI Examples:

```
salt '*' keystone.user_create name=jack password=zero email=jack@halloweentown.org tenant_id=a28
```

salt.modules.keystone.**user_delete**(*user_id=None*, *name=None*, *profile=None*, ***connection_args*)

Delete a user (keystone user-delete)

CLI Examples:

```
salt '*' keystone.user_delete c965f79c4f864eaaa9c3b41904e67082
salt '*' keystone.user_delete user_id=c965f79c4f864eaaa9c3b41904e67082
salt '*' keystone.user_delete name=nova
```

salt.modules.keystone.**user_get**(*user_id=None*, *name=None*, *profile=None*, ***connection_args*)

Return a specific users (keystone user-get)

CLI Examples:

```
salt '*' keystone.user_get c965f79c4f864eaaa9c3b41904e67082
salt '*' keystone.user_get user_id=c965f79c4f864eaaa9c3b41904e67082
salt '*' keystone.user_get name=nova
```

salt.modules.keystone.**user_list**(*profile=None*, ***connection_args*)

Return a list of available users (keystone user-list)

CLI Example:

```
salt '*' keystone.user_list
```

salt.modules.keystone.**user_password_update**(*user_id=None*, *name=None*, *password=None*, *profile=None*, ***connection_args*)

Update a user's password (keystone user-password-update)

CLI Examples:

```
salt '*' keystone.user_password_update c965f79c4f864eaaa9c3b41904e67082 password=12345
salt '*' keystone.user_password_update user_id=c965f79c4f864eaaa9c3b41904e67082 password=12345
salt '*' keystone.user_password_update name=nova password=12345
```

salt.modules.keystone.**user_role_add**(*user_id=None*, *user=None*, *tenant_id=None*, *tenant=None*, *role_id=None*, *role=None*, *profile=None*, ***connection_args*)

Add role for user in tenant (keystone user-role-add)

CLI Examples:

```
salt '*' keystone.user_role_add user_id=298ce377245c4ec9b70e1c639c89e654 tenant_id=7167a092ece84
salt '*' keystone.user_role_add user=admin tenant=admin role=admin
```

salt.modules.keystone.**user_role_list**(*user_id=None*, *tenant_id=None*, *user_name=None*, *tenant_name=None*, *profile=None*, ***connection_args*)

Return a list of available user_roles (keystone user-roles-list)

CLI Examples:

```
    salt '*' keystone.user_role_list user_id=298ce377245c4ec9b70e1c639c89e654 tenant_id=7167a092ece8
    salt '*' keystone.user_role_list user_name=admin tenant_name=admin
```

salt.modules.keystone.**user_role_remove**(*user_id=None, user=None, tenant_id=None, tenant=None, role_id=None, role=None, profile=None, **connection_args*)

Remove role for user in tenant (keystone user-role-remove)

CLI Examples:

```
    salt '*' keystone.user_role_remove user_id=298ce377245c4ec9b70e1c639c89e654 tenant_id=7167a092ec
    salt '*' keystone.user_role_remove user=admin tenant=admin role=admin
```

salt.modules.keystone.**user_update**(*user_id=None, name=None, email=None, enabled=None, tenant=None, profile=None, **connection_args*)

Update a user's information (keystone user-update) The following fields may be updated: name, email, enabled, tenant. Because the name is one of the fields, a valid user id is required.

CLI Examples:

```
    salt '*' keystone.user_update user_id=c965f79c4f864eaaa9c3b41904e67082 name=newname
    salt '*' keystone.user_update c965f79c4f864eaaa9c3b41904e67082 name=newname email=newemail@domai
```

salt.modules.keystone.**user_verify_password**(*user_id=None, name=None, password=None, profile=None, **connection_args*)

Verify a user's password

CLI Examples:

```
    salt '*' keystone.user_verify_password name=test password=foobar
    salt '*' keystone.user_verify_password user_id=c965f79c4f864eaaa9c3b41904e67082 password=foobar
```

22.16.111 salt.modules.kmod

Module to manage Linux kernel modules

salt.modules.kmod.**available**()

Return a list of all available kernel modules

CLI Example:

```
    salt '*' kmod.available
```

salt.modules.kmod.**check_available**(*mod*)

Check to see if the specified kernel module is available

CLI Example:

```
    salt '*' kmod.check_available kvm
```

salt.modules.kmod.**is_loaded**(*mod*)

Check to see if the specified kernel module is loaded

CLI Example:

```
    salt '*' kmod.is_loaded kvm
```

salt.modules.kmod.**load**(*mod, persist=False*)

Load the specified kernel module

mod Name of module to add

persist Write module to /etc/modules to make it load on system reboot

CLI Example:

```
salt '*' kmod.load kvm
```

salt.modules.kmod.**lsmod**()
Return a dict containing information about currently loaded modules

CLI Example:

```
salt '*' kmod.lsmod
```

salt.modules.kmod.**mod_list**(*only_persist=False*)
Return a list of the loaded module names

CLI Example:

```
salt '*' kmod.mod_list
```

salt.modules.kmod.**remove**(*mod, persist=False, comment=True*)
Remove the specified kernel module

mod Name of module to remove

persist Also remove module from /etc/modules

comment If persist is set don't remove line from /etc/modules but only comment it

CLI Example:

```
salt '*' kmod.remove kvm
```

22.16.112 salt.modules.launchctl

Module for the management of MacOS systems that use launchd/launchctl

depends

- plistlib Python module

salt.modules.launchctl.**available**(*job_label*)
Check that the given service is available.

CLI Example:

```
salt '*' service.available com.openssh.sshd
```

salt.modules.launchctl.**get_all**()
Return all installed services

CLI Example:

```
salt '*' service.get_all
```

salt.modules.launchctl.**missing**(*job_label*)
The inverse of service.available Check that the given service is not available.

CLI Example:

```
salt '*' service.missing com.openssh.sshd
```

salt.modules.launchctl.**restart**(*job_label*, *runas=None*)
Restart the named service

CLI Example:

```
salt '*' service.restart <service label>
```

salt.modules.launchctl.**start**(*job_label*, *runas=None*)
Start the specified service

CLI Example:

```
salt '*' service.start <service label>
salt '*' service.start org.ntp.ntpd
salt '*' service.start /System/Library/LaunchDaemons/org.ntp.ntpd.plist
```

salt.modules.launchctl.**status**(*job_label*, *runas=None*)
Return the status for a service, returns a bool whether the service is running.

CLI Example:

```
salt '*' service.status <service label>
```

salt.modules.launchctl.**stop**(*job_label*, *runas=None*)
Stop the specified service

CLI Example:

```
salt '*' service.stop <service label>
salt '*' service.stop org.ntp.ntpd
salt '*' service.stop /System/Library/LaunchDaemons/org.ntp.ntpd.plist
```

22.16.113 salt.modules.layman

Support for Layman

salt.modules.layman.**add**(*overlay*)
Add the given overlay from the cached remote list to your locally installed overlays. Specify `ALL' to add all overlays from the remote list.

Return a list of the new overlay(s) added:

CLI Example:

```
salt '*' layman.add <overlay name>
```

salt.modules.layman.**delete**(*overlay*)
Remove the given overlay from the your locally installed overlays. Specify `ALL' to remove all overlays.

Return a list of the overlays(s) that were removed:

CLI Example:

```
salt '*' layman.delete <overlay name>
```

salt.modules.layman.**list_local**()
List the locally installed overlays.

Return a list of installed overlays:

CLI Example:

```
salt '*' layman.list_local
```

salt.modules.layman.**sync**(*overlay='ALL'*)

Update the specified overlay. Use `ALL' to synchronize all overlays. This is the default if no overlay is specified.

overlay Name of the overlay to sync. (Defaults to `ALL')

CLI Example:

```
salt '*' layman.sync
```

22.16.114 salt.modules.ldapmod

Salt interface to LDAP commands

depends

- ldap Python module

configuration In order to connect to LDAP, certain configuration is required in the minion config on the LDAP server. The minimum configuration items that must be set are:

```
ldap.basedn: dc=acme,dc=com (example values, adjust to suit)
```

If your LDAP server requires authentication then you must also set:

```
ldap.anonymous: False
ldap.binddn: admin
ldap.bindpw: password
```

In addition, the following optional values may be set:

```
ldap.server: localhost (default=localhost, see warning below)
ldap.port: 389 (default=389, standard port)
ldap.tls: False (default=False, no TLS)
ldap.no_verify: False (default=False, verify TLS)
ldap.anonymous: True (default=True, bind anonymous)
ldap.scope: 2 (default=2, ldap.SCOPE_SUBTREE)
ldap.attrs: [saltAttr] (default=None, return all attributes)
```

Warning: At the moment this module only recommends connection to LDAP services listening on `localhost`. This is deliberate to avoid the potentially dangerous situation of multiple minions sending identical update commands to the same LDAP server. It's easy enough to override this behavior, but badness may ensue - you have been warned.

salt.modules.ldapmod.**search**(*filter*, *dn=None*, *scope=None*, *attrs=None*, ***kwargs*)

Run an arbitrary LDAP query and return the results.

CLI Example:

```
salt 'ldaphost' ldap.search "filter=cn=myhost"
```

Return data:

```
{'myhost': {'count': 1,
            'results': [['cn=myhost,ou=hosts,o=acme,c=gb',
                         {'saltKeyValue': ['ntpserver=ntp.acme.local',
```

```
                                            'foo=myfoo'],
                                'saltState': ['foo', 'bar']}]],
                    'time': {'human': '1.2ms', 'raw': '0.00123'}}}
```

Search and connection options can be overridden by specifying the relevant option as key=value pairs, for example:

```
salt 'ldaphost' ldap.search filter=cn=myhost dn=ou=hosts,o=acme,c=gb
scope=1 attrs='' server='localhost' port='7393' tls=True bindpw='ssh'
```

22.16.115 salt.modules.linux_acl

Support for Linux File Access Control Lists

salt.modules.linux_acl.**delfacl**(*acl_type*, *acl_name*, **args*, ***kwargs*)

Remove specific FACL from the specified file(s)

CLI Examples:

```
salt '*' acl.delfacl user myuser /tmp/house/kitchen
salt '*' acl.delfacl default:group mygroup /tmp/house/kitchen
salt '*' acl.delfacl d:u myuser /tmp/house/kitchen
salt '*' acl.delfacl g myuser /tmp/house/kitchen /tmp/house/livingroom
salt '*' acl.delfacl user myuser /tmp/house/kitchen recursive=True
```

salt.modules.linux_acl.**getfacl**(**args*, ***kwargs*)

Return (extremely verbose) map of FACLs on specified file(s)

CLI Examples:

```
salt '*' acl.getfacl /tmp/house/kitchen
salt '*' acl.getfacl /tmp/house/kitchen /tmp/house/livingroom
salt '*' acl.getfacl /tmp/house/kitchen /tmp/house/livingroom recursive=True
```

salt.modules.linux_acl.**modfacl**(*acl_type*, *acl_name*, *perms*, **args*, ***kwargs*)

Add or modify a FACL for the specified file(s)

CLI Examples:

```
salt '*' acl.modfacl user myuser rwx /tmp/house/kitchen
salt '*' acl.modfacl default:group mygroup rx /tmp/house/kitchen
salt '*' acl.modfacl d:u myuser 7 /tmp/house/kitchen
salt '*' acl.modfacl g mygroup 0 /tmp/house/kitchen /tmp/house/livingroom
salt '*' acl.modfacl user myuser rwx /tmp/house/kitchen recursive=True
```

salt.modules.linux_acl.**version**()

Return facl version from getfacl --version

CLI Example:

```
salt '*' acl.version
```

salt.modules.linux_acl.**wipefacls**(**args*, ***kwargs*)

Remove all FACLs from the specified file(s)

CLI Examples:

```
salt '*' acl.wipefacls /tmp/house/kitchen
salt '*' acl.wipefacls /tmp/house/kitchen /tmp/house/livingroom
salt '*' acl.wipefacls /tmp/house/kitchen /tmp/house/livingroom recursive=True
```

22.16.116 salt.modules.linux_lvm

Support for Linux LVM2

salt.modules.linux_lvm.**fullversion**()
Return all version info from lvm version

CLI Example:

```
salt '*' lvm.fullversion
```

salt.modules.linux_lvm.**lvcreate**(*lvname*, *vgname*, *size=None*, *extents=None*, *snapshot=None*, *pv=None*, ***kwargs*)
Create a new logical volume, with option for which physical volume to be used

CLI Examples:

```
salt '*' lvm.lvcreate new_volume_name vg_name size=10G
salt '*' lvm.lvcreate new_volume_name vg_name extents=100 pv=/dev/sdb
salt '*' lvm.lvcreate new_snapshot    vg_name snapshot=volume_name size=3G
```

salt.modules.linux_lvm.**lvdisplay**(*lvname=''*)
Return information about the logical volume(s)

CLI Examples:

```
salt '*' lvm.lvdisplay
salt '*' lvm.lvdisplay /dev/vg_myserver/root
```

salt.modules.linux_lvm.**lvremove**(*lvname*, *vgname*)
Remove a given existing logical volume from a named existing volume group

CLI Example:

```
salt '*' lvm.lvremove lvname vgname force=True
```

salt.modules.linux_lvm.**lvresize**(*size*, *lvpath*)
Return information about the logical volume(s)

CLI Examples:

```
salt '*' lvm.lvresize +12M /dev/mapper/vg1-test
```

salt.modules.linux_lvm.**pvcreate**(*devices*, ***kwargs*)
Set a physical device to be used as an LVM physical volume

CLI Examples:

```
salt mymachine lvm.pvcreate /dev/sdb1,/dev/sdb2
salt mymachine lvm.pvcreate /dev/sdb1 dataalignmentoffset=7s
```

salt.modules.linux_lvm.**pvdisplay**(*pvname=''*)
Return information about the physical volume(s)

CLI Examples:

```
salt '*' lvm.pvdisplay
salt '*' lvm.pvdisplay /dev/md0
```

salt.modules.linux_lvm.**pvremove**(*devices*)
Remove a physical device being used as an LVM physical volume

CLI Examples:

```
salt mymachine lvm.pvremove /dev/sdb1,/dev/sdb2
```

salt.modules.linux_lvm.**version**()
Return LVM version from lvm version

CLI Example:

```
salt '*' lvm.version
```

salt.modules.linux_lvm.**vgcreate**(*vgname*, *devices*, ***kwargs*)
Create an LVM volume group

CLI Examples:

```
salt mymachine lvm.vgcreate my_vg /dev/sdb1,/dev/sdb2
salt mymachine lvm.vgcreate my_vg /dev/sdb1 clustered=y
```

salt.modules.linux_lvm.**vgdisplay**(*vgname=''*)
Return information about the volume group(s)

CLI Examples:

```
salt '*' lvm.vgdisplay
salt '*' lvm.vgdisplay nova-volumes
```

salt.modules.linux_lvm.**vgextend**(*vgname*, *devices*)
Add physical volumes to an LVM volume group

CLI Examples:

```
salt mymachine lvm.vgextend my_vg /dev/sdb1,/dev/sdb2
salt mymachine lvm.vgextend my_vg /dev/sdb1
```

salt.modules.linux_lvm.**vgremove**(*vgname*)
Remove an LVM volume group

CLI Examples:

```
salt mymachine lvm.vgremove vgname
salt mymachine lvm.vgremove vgname force=True
```

22.16.117 salt.modules.linux_sysctl

Module for viewing and modifying sysctl parameters

salt.modules.linux_sysctl.**assign**(*name*, *value*)
Assign a single sysctl parameter for this minion

CLI Example:

```
salt '*' sysctl.assign net.ipv4.ip_forward 1
```

salt.modules.linux_sysctl.**default_config**()
Linux hosts using systemd 207 or later ignore `/etc/sysctl.conf` and only load from `/etc/sysctl.d/*.conf`. This function will do the proper checks and return a default config file which will be valid for the Minion. Hosts running systemd >= 207 will use `/etc/sysctl.d/99-salt.conf`.

CLI Example:

```
salt -G 'kernel:Linux' sysctl.default_config
```

salt.modules.linux_sysctl.**get**(*name*)
Return a single sysctl parameter for this minion

CLI Example:

```
salt '*' sysctl.get net.ipv4.ip_forward
```

salt.modules.linux_sysctl.**persist**(*name*, *value*, *config=None*)
Assign and persist a simple sysctl parameter for this minion. If `config` is not specified, a sensible default will be chosen using `sysctl.default_config`.

CLI Example:

```
salt '*' sysctl.persist net.ipv4.ip_forward 1
```

salt.modules.linux_sysctl.**show**(*config_file=False*)
Return a list of sysctl parameters for this minion

config: **Pull the data from the system configuration file** instead of the live data.

CLI Example:

```
salt '*' sysctl.show
```

22.16.118 salt.modules.localemod

Module for managing locales on POSIX-like systems.

salt.modules.localemod.**avail**(*locale*)
Check if a locale is available.

New in version 2014.7.0.

CLI Example:

```
salt '*' locale.avail 'en_US.UTF-8'
```

salt.modules.localemod.**gen_locale**(*locale*, ***kwargs*)
Generate a locale. Options:

verbose Show extra warnings about errors that are normally ignored.

New in version 2014.7.0.

Parameters **locale** -- Any locale listed in /usr/share/i18n/locales or /usr/share/i18n/SUPPORTED for debian and gentoo based distros

CLI Example:

```
salt '*' locale.gen_locale en_US.UTF-8
salt '*' locale.gen_locale 'en_IE@euro ISO-8859-15'
```

salt.modules.localemod.**get_locale**()
Get the current system locale

CLI Example:

```
salt '*' locale.get_locale
```

salt.modules.localemod.**list_avail**()
Lists available (compiled) locales

CLI Example:

```
salt '*' locale.list_avail
```

salt.modules.localemod.**set_locale**(*locale*)
Sets the current system locale

CLI Example:

```
salt '*' locale.set_locale 'en_US.UTF-8'
```

22.16.119 salt.modules.locate

Module for using the locate utilities

salt.modules.locate.**locate**(*pattern*, *database=''*, *limit=0*, ***kwargs*)
Performs a file lookup. Valid options (and their defaults) are:

```
basename=False
count=False
existing=False
follow=True
ignore=False
nofollow=False
wholename=True
regex=False
database=<locate's default database>
limit=<integer, not set by default>
```

See the manpage for `locate(1)` for further explanation of these options.

CLI Example:

```
salt '*' locate.locate
```

salt.modules.locate.**stats**()
Returns statistics about the locate database

CLI Example:

```
salt '*' locate.stats
```

salt.modules.locate.**updatedb**()
Updates the locate database

CLI Example:

```
salt '*' locate.updatedb
```

salt.modules.locate.**version**()
Returns the version of locate

CLI Example:

```
salt '*' locate.version
```

22.16.120 salt.modules.logadm

Module for managing Solaris logadm based log rotations.

salt.modules.logadm.**remove**(*name*, *conf_file='/etc/logadm.conf'*)
Remove log pattern from logadm

CLI Example:

```
salt '*' logadm.remove myapplog
```

salt.modules.logadm.**rotate**(*name*, *pattern=False*, *count=False*, *age=False*, *size=False*, *copy=True*, *conf_file='/etc/logadm.conf'*)
Set up pattern for logging.

CLI Example:

```
salt '*' logadm.rotate myapplog pattern='/var/log/myapp/*.log' count=7
```

salt.modules.logadm.**show_conf**(*conf_file='/etc/logadm.conf'*)
Show parsed configuration

CLI Example:

```
salt '*' logadm.show_conf
```

22.16.121 salt.modules.logrotate

Module for managing logrotate.

salt.modules.logrotate.**set_**(*key*, *value*, *setting=None*, *conf_file='/etc/logrotate.conf'*)
Set a new value for a specific configuration line

CLI Example:

```
salt '*' logrotate.set rotate 2
```

Can also be used to set a single value inside a multiline configuration block. For instance, to change rotate in the following block:

```
/var/log/wtmp {
    monthly
    create 0664 root root
    rotate 1
}
```

Use the following command:

```
salt '*' logrotate.set /var/log/wtmp rotate 2
```

This module also has the ability to scan files inside an include directory, and make changes in the appropriate file.

salt.modules.logrotate.**show_conf**(*conf_file='/etc/logrotate.conf'*)
Show parsed configuration

CLI Example:

```
salt '*' logrotate.show_conf
```

22.16.122 salt.modules.lvs

Support for LVS (Linux Virtual Server)

salt.modules.lvs.**add_server**(*protocol=None, service_address=None, server_address=None, packet_forward_method='dr', weight=1, **kwargs*)

Add a real server to a virtual service.

protocol The service protocol(only support `tcp`, `udp` and `fwmark` service).

service_address The LVS service address.

server_address The real server address.

packet_forward_method The LVS packet forwarding method(`dr` for direct routing, `tunnel` for tunneling, `nat` for network access translation).

weight The capacity of a server relative to the others in the pool.

CLI Example:

```
salt '*' lvs.add_server tcp 1.1.1.1:80 192.168.0.11:8080 nat 1
```

salt.modules.lvs.**add_service**(*protocol=None, service_address=None, scheduler='wlc'*)

Add a virtual service.

protocol The service protocol(only support tcp, udp and fwmark service).

service_address The LVS service address.

scheduler Algorithm for allocating TCP connections and UDP datagrams to real servers.

CLI Example:

```
salt '*' lvs.add_service tcp 1.1.1.1:80 rr
```

salt.modules.lvs.**check_server**(*protocol=None, service_address=None, server_address=None, **kwargs*)

Check the real server exists in the specified service.

CLI Example:

```
salt '*' lvs.check_server tcp 1.1.1.1:80 192.168.0.11:8080
```

salt.modules.lvs.**check_service**(*protocol=None, service_address=None, **kwargs*)

Check the virtual service exists.

CLI Example:

```
salt '*' lvs.check_service tcp 1.1.1.1:80
```

salt.modules.lvs.**clear**()

Clear the virtual server table

CLI Example:

```
salt '*' lvs.clear
```

salt.modules.lvs.**delete_server**(*protocol=None, service_address=None, server_address=None*)

Delete the realserver from the virtual service.

protocol The service protocol(only support `tcp`, `udp` and `fwmark` service).

service_address The LVS service address.

server_address The real server address.

CLI Example:

```
salt '*' lvs.delete_server tcp 1.1.1.1:80 192.168.0.11:8080
```

`salt.modules.lvs.`**`delete_service`**(*protocol=None*, *service_address=None*)
Delete the virtual service.

protocol The service protocol(only support tcp, udp and fwmark service).

service_address The LVS service address.

CLI Example:

```
salt '*' lvs.delete_service tcp 1.1.1.1:80
```

`salt.modules.lvs.`**`edit_server`**(*protocol=None*, *service_address=None*, *server_address=None*, *packet_forward_method=None*, *weight=None*, ***kwargs*)
Edit a real server to a virtual service.

protocol The service protocol(only support `tcp`, `udp` and `fwmark` service).

service_address The LVS service address.

server_address The real server address.

packet_forward_method The LVS packet forwarding method(`dr` for direct routing, `tunnel` for tunneling, `nat` for network access translation).

weight The capacity of a server relative to the others in the pool.

CLI Example:

```
salt '*' lvs.edit_server tcp 1.1.1.1:80 192.168.0.11:8080 nat 1
```

`salt.modules.lvs.`**`edit_service`**(*protocol=None*, *service_address=None*, *scheduler=None*)
Edit the virtual service.

protocol The service protocol(only support tcp, udp and fwmark service).

service_address The LVS service address.

scheduler Algorithm for allocating TCP connections and UDP datagrams to real servers.

CLI Example:

```
salt '*' lvs.edit_service tcp 1.1.1.1:80 rr
```

`salt.modules.lvs.`**`get_rules`**()
Get the virtual server rules

CLI Example:

```
salt '*' lvs.get_rules
```

`salt.modules.lvs.`**`list_`**(*protocol=None*, *service_address=None*)
List the virtual server table if service_address is not specified. If a service_address is selected, list this service only.

CLI Example:

```
salt '*' lvs.list
```

salt.modules.lvs.**zero**(*protocol=None*, *service_address=None*)
Zero the packet, byte and rate counters in a service or all services.

CLI Example:

```
salt '*' lvs.zero
```

22.16.123 salt.modules.lxc

Control Linux Containers via Salt

depends lxc package for distribution

lxc >= 1.0 (even beta alpha) is required

salt.modules.lxc.**apply_network_profile**(*name*, *network_profile*, *nic_opts=None*)
New in version 2015.5.0.

Apply a network profile to a container

network_profile profile name or default values (dict)

nic_opts values to override in defaults (dict) indexed by nic card names

CLI Examples:

```
salt 'minion' lxc.apply_network_profile web1 centos
salt 'minion' lxc.apply_network_profile web1 centos \
        nic_opts="{'eth0': {'mac': 'xx:xx:xx:xx:xx:xx'}}"
salt 'minion' lxc.apply_network_profile web1 \
        "{'eth0': {'mac': 'xx:xx:xx:xx:xx:yy'}}"
        nic_opts="{'eth0': {'mac': 'xx:xx:xx:xx:xx:xx'}}"
```

salt.modules.lxc.**attachable**(*name*)
Return True if the named container can be attached to via the lxc-attach command

CLI Example:

```
salt 'minion' lxc.attachable ubuntu
```

salt.modules.lxc.**bootstrap**(*name*, *config=None*, *approve_key=True*, *install=True*, *pub_key=None*, *priv_key=None*, *bootstrap_url=None*, *force_install=False*, *unconditional_install=False*, *bootstrap_delay=None*, *bootstrap_args=None*, *bootstrap_shell=None*)
Install and configure salt in a container.

config Minion configuration options. By default, the `master` option is set to the target host's master.

approve_key Request a pre-approval of the generated minion key. Requires that the salt-master be configured to either auto-accept all keys or expect a signing request from the target host. Default: `True`

pub_key Explicit public key to pressed the minion with (optional). This can be either a filepath or a string representing the key

priv_key Explicit private key to pressed the minion with (optional). This can be either a filepath or a string representing the key

bootstrap_delay Delay in seconds between end of container creation and bootstrapping. Useful when waiting for container to obtain a DHCP lease.

New in version 2015.5.0.

bootstrap_url url, content or filepath to the salt bootstrap script

bootstrap_args salt bootstrap script arguments

bootstrap_shell shell to execute the script into

install Whether to attempt a full installation of salt-minion if needed.

force_install Force installation even if salt-minion is detected, this is the way to run vendor bootstrap scripts even if a salt minion is already present in the container

unconditional_install Run the script even if the container seems seeded

CLI Examples:

```
salt 'minion' lxc.bootstrap container_name [config=config_data] \
        [approve_key=(True|False)] [install=(True|False)]
```

salt.modules.lxc.**clone**(*name*, *orig*, *profile=None*, ***kwargs*)

Create a new container as a clone of another container

name Name of the container

orig Name of the original container to be cloned

profile Profile to use in container cloning (see `lxc.get_container_profile`). Values in a profile will be overridden by the **Container Cloning Arguments** listed below.

Container Cloning Arguments

snapshot Use Copy On Write snapshots (LVM)

size [1G] Size of the volume to create. Only applicable if `backing=lvm`.

backing The type of storage to use. Set to `lvm` to use an LVM group. Defaults to filesystem within /var/lib/lxc.

CLI Examples:

```
salt '*' lxc.clone myclone orig=orig_container
salt '*' lxc.clone myclone orig=orig_container snapshot=True
```

salt.modules.lxc.**cloud_init**(*name*, *vm_=None*, ***kwargs*)

Thin wrapper to lxc.init to be used from the saltcloud lxc driver

name Name of the container may be None and then guessed from saltcloud mapping

vm_ saltcloud mapping defaults for the vm

CLI Example:

```
salt '*' lxc.cloud_init foo
```

salt.modules.lxc.**cloud_init_interface**(*name*, *vm_=None*, ***kwargs*)

Interface between salt.cloud.lxc driver and lxc.init `vm_` is a mapping of vm opts in the salt.cloud format as documented for the lxc driver.

This can be used either:

- from the salt cloud driver
- because you find the argument to give easier here than using directly lxc.init

Warning: BE REALLY CAREFUL CHANGING DEFAULTS !!! IT'S A RETRO COMPATIBLE INTERFACE WITH THE SALT CLOUD DRIVER (ask kiorky).

name name of the lxc container to create

from_container which container we use as a template when running lxc.clone

image which template do we use when we are using lxc.create. This is the default mode unless you specify something in from_container

backing which backing store to use. Values can be: overlayfs, dir(default), lvm, zfs, brtfs

fstype When using a blockdevice level backing store, which filesystem to use on

size When using a blockdevice level backing store, which size for the filesystem to use on

snapshot Use snapshot when cloning the container source

vgname if using LVM: vgname

lgname if using LVM: lvname

pub_key public key to preseed the minion with. Can be the keycontent or a filepath

priv_key private key to preseed the minion with. Can be the keycontent or a filepath

ip ip for the primary nic

mac mac address for the primary nic

netmask netmask for the primary nic (24) = `vm_.get('netmask', '24')`

bridge bridge for the primary nic (lxcbr0)

gateway network gateway for the container

additional_ips additional ips which will be wired on the main bridge (br0) which is connected to internet. Be aware that you may use manual virtual mac addresses providen by you provider (online, ovh, etc). This is a list of mappings {ip: `', mac: `', netmask:'`} Set gateway to None and an interface with a gateway to escape from another interface that eth0. eg:

```
- {'mac': '00:16:3e:01:29:40',
   'gateway': None, (default)
   'link': 'br0', (default)
   'netmask': '', (default)
   'ip': '22.1.4.25'}
```

unconditional_install given to lxc.bootstrap (see relative doc)

force_install given to lxc.bootstrap (see relative doc)

config any extra argument for the salt minion config

dnsservers dns servers to set inside the container

autostart autostart the container at boot time

password administrative password for the container

users administrative users for the container default: [root] and [root, ubuntu] on ubuntu

CLI Example:

```
salt '*' lxc.cloud_init_interface foo
```

`salt.modules.lxc.cp`(*name*, *source*, *dest*, *makedirs=False*)

Copy a file or directory from the host into a container.

name Container name

source File to be copied to the container

dest Destination on the container. Must be an absolute path.

Changed in version 2015.5.0: If the destination is a directory, the file will be copied into that directory.

makedirs : False

Create the parent directory on the container if it does not already exist.

New in version 2015.5.0.

CLI Example:

```
salt 'minion' lxc.cp /tmp/foo /root/foo
```

salt.modules.lxc.**create**(*name*, *config=None*, *profile=None*, *network_profile=None*, *nic_opts=None*, ***kwargs*)

Create a new container.

name Name of the container

config The config file to use for the container. Defaults to system-wide config (usually in /etc/lxc/lxc.conf).

profile Profile to use in container creation (see `lxc.get_container_profile`). Values in a profile will be overridden by the **Container Creation Arguments** listed below.

network_profile Network profile to use for container

New in version 2015.5.0.

Container Creation Arguments

template The template to use. For example, `ubuntu` or `fedora`. Conflicts with the `image` argument.

Note: The `download` template requires the following three parameters to be defined in `options`:

- **dist** - The name of the distribution
- **release** - Release name/version
- **arch** - Architecture of the container

The available images can be listed using the `lxc.images` function.

---

options Template-specific options to pass to the lxc-create command. These correspond to the long options (ones beginning with two dashes) that the template script accepts. For example:

```
options='{"dist": "centos", "release": "6", "arch": "amd64"}'
```

image A tar archive to use as the rootfs for the container. Conflicts with the `template` argument.

backing The type of storage to use. Set to `lvm` to use an LVM group. Defaults to filesystem within /var/lib/lxc.

fstype Filesystem type to use on LVM logical volume

size [1G] Size of the volume to create. Only applicable if `backing=lvm`.

vgname [lxc] Name of the LVM volume group in which to create the volume for this container. Only applicable if `backing=lvm`.

lvname Name of the LVM logical volume in which to create the volume for this container. Only applicable if `backing=lvm`.

nic_opts give extra opts overriding network profile values

salt.modules.lxc.**destroy**(*name*, *stop=False*)

Destroy the named container.

> **Warning**: Destroys all data associated with the container.

stop [False] If `True`, the container will be destroyed even if it is running/frozen.

Changed in version 2015.5.0: Default value changed to `False`. This more closely matches the behavior of `lxc-destroy(1)`, and also makes it less likely that an accidental command will destroy a running container that was being used for important things.

CLI Examples:

```
salt '*' lxc.destroy foo
salt '*' lxc.destroy foo stop=True
```

salt.modules.lxc.**edit_conf**(*conf_file*, *out_format='simple'*, *read_only=False*, *lxc_config=None*, ***kwargs*)

Edit an LXC configuration file. If a setting is already present inside the file, its value will be replaced. If it does not exist, it will be appended to the end of the file. Comments and blank lines will be kept in-tact if they already exist in the file.

out_format: Set to simple if you need backward compatbility (multiple items for a simple key is not supported)

read_only: return only the edited configuration without applying it to the underlying lxc configuration file

lxc_config: List of dict containning lxc configuration items For network configuration, you also need to add the device it belongs to, otherwise it will default to eth0. Also, any change to a network parameter will result in the whole network reconfiguration to avoid mismatchs, be aware of that !

After the file is edited, its contents will be returned. By default, it will be returned in `simple` format, meaning an unordered dict (which may not represent the actual file order). Passing in an `out_format` of `commented` will return a data structure which accurately represents the order and content of the file.

CLI Example:

```
salt 'minion' lxc.edit_conf /etc/lxc/mycontainer.conf \
    out_format=commented lxc.network.type=veth
salt 'minion' lxc.edit_conf /etc/lxc/mycontainer.conf \
    out_format=commented \
    lxc_config="[{'lxc.network.name': 'eth0', \
                  'lxc.network.ipv4': '1.2.3.4'},
                 {'lxc.network.name': 'eth2', \
                  'lxc.network.ipv4': '1.2.3.5',\
                  'lxc.network.gateway': '1.2.3.1'}]"
```

salt.modules.lxc.**exists**(*name*)

Returns whether the named container exists.

CLI Example:

```
salt '*' lxc.exists name
```

salt.modules.lxc.**freeze**(*name*, ***kwargs*)

Freeze the named container

start [False] If `True` and the container is stopped, the container will be started before attempting to freeze.

New in version 2015.5.0.

CLI Example:

```
salt '*' lxc.freeze name
```

salt.modules.lxc.**get_container_profile**(*name=None*, ***kwargs*)

New in version 2015.5.0.

Gather a pre-configured set of container configuration parameters. If no arguments are passed, an empty profile is returned.

Profiles can be defined in the minion or master config files, or in pillar or grains, and are loaded using `config.get`. The key under which LXC profiles must be configured is `lxc.container_profile.profile_name`. An example container profile would be as follows:

```
lxc.container_profile:
  ubuntu:
    template: ubuntu
    backing: lvm
    vgname: lxc
    size: 1G
```

Parameters set in a profile can be overridden by passing additional container creation arguments (such as the ones passed to `lxc.create`) to this function.

A profile can be defined either as the name of the profile, or a dictionary of variable names and values. See the *LXC Tutorial* for more information on how to use LXC profiles.

CLI Example:

```
.. code-block:: bash
```

salt-call lxc.get_container_profile centos salt-call lxc.get_container_profile ubuntu template=ubuntu backing=overlayfs

salt.modules.lxc.**get_network_profile**(*name=None*)

New in version 2015.5.0.

Gather a pre-configured set of network configuration parameters. If no arguments are passed, the following default profile is returned:

```
{'eth0': {'link': 'br0', 'type': 'veth', 'flags': 'up'}}
```

Profiles can be defined in the minion or master config files, or in pillar or grains, and are loaded using `config.get`. The key under which LXC profiles must be configured is `lxc.network_profile`. An example network profile would be as follows:

```
lxc.network_profile.centos:
  eth0:
    link: br0
    type: veth
    flags: up
```

Parameters set in a profile can be overridden by passing additional arguments to this function.

A profile can be passed either as the name of the profile, or a dictionary of variable names and values. See the *LXC Tutorial* for more information on how to use network profiles.

Warning: The `ipv4`, `ipv6`, `gateway`, and `link` (bridge) settings in network profiles will only work if the container doesnt redefine the network configuration (for example in `/etc/sysconfig/network-scripts/ifcfg-<interface_name>` on RHEL/CentOS, or `/etc/network/interfaces` on Debian/Ubuntu/etc.)

CLI Example:

```
.. code-block:: bash
```

salt-call lxc.get_network_profile default

salt.modules.lxc.**get_parameter**(*name*, *parameter*)

Returns the value of a cgroup parameter for a container

CLI Example:

```
salt '*' lxc.get_parameter container_name memory.limit_in_bytes
```

salt.modules.lxc.**images**(*dist=None*)

New in version 2015.5.0.

List the available images for LXC's `download` template.

dist [None] Filter results to a single Linux distribution

CLI Examples:

```
salt myminion lxc.images
salt myminion lxc.images dist=centos
```

salt.modules.lxc.**info**(*name*)

Returns information about a container

CLI Example:

```
salt '*' lxc.info name
```

salt.modules.lxc.**init**(*name*, *config=None*, *cpuset=None*, *cpushare=None*, *memory=None*, *profile=None*, *network_profile=<object object>*, *nic=<object object>*, *nic_opts=None*, *cpu=None*, *autostart=True*, *password=None*, *password_encrypted=None*, *users=None*, *dnsservers=None*, *searchdomains=None*, *bridge=None*, *gateway=None*, *pub_key=None*, *priv_key=None*, *force_install=False*, *unconditional_install=False*, *bootstrap_delay=None*, *bootstrap_args=None*, *bootstrap_shell=None*, *bootstrap_url=None*, ***kwargs*)

Initialize a new container.

This is a partial idempotent function as if it is already provisioned, we will reset a bit the lxc configuration file but much of the hard work will be escaped as markers will prevent re-execution of harmful tasks.

name Name of the container

image A tar archive to use as the rootfs for the container. Conflicts with the `template` argument.

cpus Select a random number of cpu cores and assign it to the cpuset, if the cpuset option is set then this option will be ignored

cpuset Explicitly define the cpus this container will be bound to

cpushare cgroups cpu shares

autostart autostart container on reboot

memory cgroups memory limit, in MB

Changed in version 2015.5.0: If no value is passed, no limit is set. In earlier Salt versions, not passing this value causes a 1024MB memory limit to be set, and it was necessary to pass `memory=0` to set no limit.

gateway the ipv4 gateway to use the default does nothing more than lxcutils does

bridge the bridge to use the default does nothing more than lxcutils does

network_profile Network profile to use for the container

New in version 2015.5.0.

nic Deprecated since version 2015.5.0: Use `network_profile` instead

nic_opts Extra options for network interfaces, will override

```
{"eth0": {"hwaddr": "aa:bb:cc:dd:ee:ff", "ipv4": "10.1.1.1", "ipv6":
"2001:db8::ff00:42:8329"}}
```

or

```
{"eth0": {"hwaddr": "aa:bb:cc:dd:ee:ff", "ipv4": "10.1.1.1/24",
"ipv6": "2001:db8::ff00:42:8329"}}
```

users Users for which the password defined in the `password` param should be set. Can be passed as a comma separated list or a python list. Defaults to just the `root` user.

password Set the initial password for the users defined in the `users` parameter

password_encrypted [False] Set to `True` to denote a password hash instead of a plaintext password

New in version 2015.5.0.

profile A LXC profile (defined in config or pillar). This can be either a real profile mapping or a string to retrieve it in configuration

start Start the newly-created container

dnsservers list of dns servers to set in the container, default [] (no setting)

seed Seed the container with the minion config. Default: `True`

install If salt-minion is not already installed, install it. Default: `True`

config Optional config parameters. By default, the id is set to the name of the container.

pub_key Explicit public key to preseed the minion with (optional). This can be either a filepath or a string representing the key

priv_key Explicit private key to preseed the minion with (optional). This can be either a filepath or a string representing the key

approve_key If explicit preseeding is not used; Attempt to request key approval from the master. Default: `True`

clone Deprecated since version 2015.5.0: Use `clone_from` instead

clone_from Original from which to use a clone operation to create the container. Default: `None`

bootstrap_delay Delay in seconds between end of container creation and bootstrapping. Useful when waiting for container to obtain a DHCP lease.

New in version 2015.5.0.

bootstrap_url See lxc.bootstrap

bootstrap_shell See lxc.bootstrap

bootstrap_args See lxc.bootstrap

force_install Force installation even if salt-minion is detected, this is the way to run vendor bootstrap scripts even if a salt minion is already present in the container

unconditional_install Run the script even if the container seems seeded

CLI Example:

```
salt 'minion' lxc.init name [cpuset=cgroups_cpuset] \
        [cpushare=cgroups_cpushare] [memory=cgroups_memory] \
        [nic=nic_profile] [profile=lxc_profile] \
        [nic_opts=nic_opts] [start=(True|False)] \
        [seed=(True|False)] [install=(True|False)] \
        [config=minion_config] [approve_key=(True|False) \
        [clone_from=original] [autostart=True] \
        [priv_key=/path_or_content] [pub_key=/path_or_content] \
        [bridge=lxcbr0] [gateway=10.0.3.1] \
        [dnsservers[dns1,dns2]] \
        [users=[foo]] [password='secret'] \
        [password_encrypted=(True|False)]
```

salt.modules.lxc.**list_**(*extra=False*, *limit=None*)

List containers classified by state

extra Also get per-container specific info. This will change the return data. Instead of returning a list of containers, a dictionary of containers and each container's output from `lxc.info`.

limit Return output matching a specific state (**frozen**, **running**, or **stopped**).

New in version 2015.5.0.

CLI Examples:

```
salt '*' lxc.list
salt '*' lxc.list extra=True
salt '*' lxc.list limit=running
```

salt.modules.lxc.**ls_**(*active=None*, *cache=True*)

Return a list of the containers available on the minion

active If `True`, return only active (i.e. running) containers

New in version 2015.5.0.

CLI Example:

```
salt '*' lxc.ls
salt '*' lxc.ls active=True
```

salt.modules.lxc.**read_conf**(*conf_file*, *out_format='simple'*)

Read in an LXC configuration file. By default returns a simple, unsorted dict, but can also return a more detailed structure including blank lines and comments.

out_format: set to `simple' if you need the old and unsupported behavior. This wont support the multiple lxc values (eg: multiple network nics)

CLI Examples:

```
salt 'minion' lxc.read_conf /etc/lxc/mycontainer.conf
salt 'minion' lxc.read_conf /etc/lxc/mycontainer.conf out_format=commented
```

salt.modules.lxc.**restart**(*name*, *force=False*)

New in version 2015.5.0.

Restart the named container. If the container was not running, the container will merely be started.

name The name of the container

force [False] If `True`, the container will be force-stopped instead of gracefully shut down

CLI Example:

```
salt myminion lxc.restart name
```

salt.modules.lxc.**retcode**(*name*, *cmd*, *no_start=False*, *preserve_state=True*, *stdin=None*, *python_shell=True*, *output_loglevel='debug'*, *use_vt=False*, *ignore_retcode=False*, *keep_env='http_proxy, https_proxy, no_proxy'*)

New in version 2015.5.0.

Run `cmd.retcode` within a container

Warning: Many shell builtins do not work, failing with stderr similar to the following:
`lxc_container: No such file or directory - failed to exec 'command'`
The same error will be displayed in stderr if the command being run does not exist. If the retcode is nonzero and not what was expected, try using `lxc.run_stderr` or `lxc.run_all`.

name Name of the container in which to run the command

cmd Command to run

no_start [False] If the container is not running, don't start it

preserve_state [True] After running the command, return the container to its previous state

stdin [None] Standard input to be used for the command

output_loglevel [debug] Level at which to log the output from the command. Set to `quiet` to suppress logging.

use_vt [False] Use SaltStack's utils.vt to stream output to console `output=all`.

keep_env [http_proxy,https_proxy,no_proxy] A list of env vars to preserve. May be passed as commma-delimited list.

CLI Example:

```
salt myminion lxc.retcode mycontainer 'ip addr show'
```

salt.modules.lxc.**run**(*name*, *cmd*, *no_start=False*, *preserve_state=True*, *stdin=None*, *python_shell=True*, *output_loglevel='debug'*, *use_vt=False*, *ignore_retcode=False*, *keep_env='http_proxy, https_proxy, no_proxy'*)

New in version 2015.5.0.

Run `cmd.run` within a container

Warning: Many shell builtins do not work, failing with stderr similar to the following:
`lxc_container: No such file or directory - failed to exec 'command'`
The same error will be displayed in stderr if the command being run does not exist. If no output is returned using this function, try using `lxc.run_stderr` or `lxc.run_all`.

name Name of the container in which to run the command

cmd Command to run

no_start [False] If the container is not running, don't start it

preserve_state [True] After running the command, return the container to its previous state

stdin [None] Standard input to be used for the command

output_loglevel [debug] Level at which to log the output from the command. Set to `quiet` to suppress logging.

use_vt [False] Use SaltStack's utils.vt to stream output to console. Assumes `output=all`.

keep_env [http_proxy,https_proxy,no_proxy] A list of env vars to preserve. May be passed as commma-delimited list.

CLI Example:

```
salt myminion lxc.run mycontainer 'ifconfig -a'
```

`salt.modules.lxc.`**`run_all`**(*name*, *cmd*, *no_start=False*, *preserve_state=True*, *stdin=None*, *python_shell=True*, *output_loglevel='debug'*, *use_vt=False*, *ignore_retcode=False*, *keep_env='http_proxy, https_proxy, no_proxy'*)

New in version 2015.5.0.

Run `cmd.run_all` within a container

Warning: Many shell builtins do not work, failing with stderr similar to the following:
`lxc_container: No such file or directory - failed to exec 'command'`
The same error will be displayed in stderr if the command being run does not exist.

name Name of the container in which to run the command

cmd Command to run

no_start [False] If the container is not running, don't start it

preserve_state [True] After running the command, return the container to its previous state

stdin [None] Standard input to be used for the command

output_loglevel [debug] Level at which to log the output from the command. Set to `quiet` to suppress logging.

use_vt [False] Use SaltStack's utils.vt to stream output to console `output=all`.

keep_env [http_proxy,https_proxy,no_proxy] A list of env vars to preserve. May be passed as commma-delimited list.

CLI Example:

```
salt myminion lxc.run_all mycontainer 'ip addr show'
```

`salt.modules.lxc.`**`run_cmd`**(*name*, *cmd*, *no_start=False*, *preserve_state=True*, *stdin=None*, *stdout=True*, *stderr=False*, *python_shell=True*, *output_loglevel='debug'*, *use_vt=False*, *ignore_retcode=False*, *keep_env='http_proxy, https_proxy, no_proxy'*)

Deprecated since version 2015.5.0: Use `lxc.run` instead

`salt.modules.lxc.`**`run_stderr`**(*name*, *cmd*, *no_start=False*, *preserve_state=True*, *stdin=None*, *python_shell=True*, *output_loglevel='debug'*, *use_vt=False*, *ignore_retcode=False*, *keep_env='http_proxy, https_proxy, no_proxy'*)

New in version 2015.5.0.

Run `cmd.run_stderr` within a container

Warning: Many shell builtins do not work, failing with stderr similar to the following:
`lxc_container: No such file or directory - failed to exec 'command'`
The same error will be displayed if the command being run does not exist.

name Name of the container in which to run the command

cmd Command to run

no_start [False] If the container is not running, don't start it

preserve_state [True] After running the command, return the container to its previous state

stdin [None] Standard input to be used for the command

output_loglevel [debug] Level at which to log the output from the command. Set to `quiet` to suppress logging.

use_vt [False] Use SaltStack's utils.vt to stream output to console `output=all`.

keep_env [http_proxy,https_proxy,no_proxy] A list of env vars to preserve. May be passed as commma-delimited list.

CLI Example:

```
salt myminion lxc.run_stderr mycontainer 'ip addr show'
```

`salt.modules.lxc.`**`run_stdout`**(*name*, *cmd*, *no_start=False*, *preserve_state=True*, *stdin=None*, *python_shell=True*, *output_loglevel='debug'*, *use_vt=False*, *ignore_retcode=False*, *keep_env='http_proxy*, *https_proxy*, *no_proxy'*)

New in version 2015.5.0.

Run `cmd.run_stdout` within a container

Warning: Many shell builtins do not work, failing with stderr similar to the following:
`lxc_container: No such file or directory - failed to exec 'command'`
The same error will be displayed in stderr if the command being run does not exist. If no output is returned using this function, try using `lxc.run_stderr` or `lxc.run_all`.

name Name of the container in which to run the command

cmd Command to run

no_start [False] If the container is not running, don't start it

preserve_state [True] After running the command, return the container to its previous state

stdin [None] Standard input to be used for the command

output_loglevel [debug] Level at which to log the output from the command. Set to `quiet` to suppress logging.

use_vt [False] Use SaltStack's utils.vt to stream output to console `output=all`.

keep_env [http_proxy,https_proxy,no_proxy] A list of env vars to preserve. May be passed as commma-delimited list.

CLI Example:

```
salt myminion lxc.run_stdout mycontainer 'ifconfig -a'
```

salt.modules.lxc.**set_dns**(*name*, *dnsservers=None*, *searchdomains=None*)

Changed in version 2015.5.0: The `dnsservers` and `searchdomains` parameters can now be passed as a comma-separated list.

Update /etc/resolv.conf

CLI Example:

```
salt myminion lxc.set_dns ubuntu "['8.8.8.8', '4.4.4.4']"
```

salt.modules.lxc.**set_parameter**(*name*, *parameter*, *value*)

Set the value of a cgroup parameter for a container.

CLI Example:

```
salt '*' lxc.set_parameter name parameter value
```

salt.modules.lxc.**set_password**(*name*, *users*, *password*, *encrypted=True*)

Changed in version 2015.5.0: Function renamed from `set_pass` to `set_password`. Additionally, this function now supports (and defaults to using) a password hash instead of a plaintext password.

Set the password of one or more system users inside containers

users Comma-separated list (or python list) of users to change password

password Password to set for the specified user(s)

encrypted [True] If true, `password` must be a password hash. Set to `False` to set a plaintext password (not recommended).

New in version 2015.5.0.

CLI Example:

```
salt '*' lxc.set_pass container-name root '$6$uJ2uAyLU$KoI67t8As/0fXtJOPcHKGXmUpcoYUcVR2K6x93wal
salt '*' lxc.set_pass container-name root foo encrypted=False
```

salt.modules.lxc.**start**(*name*, ***kwargs*)

Start the named container

restart [False] Deprecated since version 2015.5.0: Use `lxc.restart`

Restart the container if it is already running

CLI Example:

```
salt myminion lxc.start name
```

salt.modules.lxc.**state**(*name*)

Returns the state of a container.

CLI Example:

```
salt '*' lxc.state name
```

salt.modules.lxc.**stop**(*name*, *kill=False*)

Stop the named container

kill [False] Do not wait for the container to stop, kill all tasks in the container. Older LXC versions will stop containers like this irrespective of this argument.

Changed in version 2015.5.0: Default value changed to `False`

CLI Example:

```
salt myminion lxc.stop name
```

salt.modules.lxc.**templates**()
New in version 2015.5.0.

List the available LXC template scripts installed on the minion

CLI Examples:

```
salt myminion lxc.templates
```

salt.modules.lxc.**unfreeze**(*name*)
Unfreeze the named container.

CLI Example:

```
salt '*' lxc.unfreeze name
```

salt.modules.lxc.**update_lxc_conf**(*name*, *lxc_conf*, *lxc_conf_unset*)
Edit LXC configuration options

CLI Example:

```
salt myminion lxc.update_lxc_conf ubuntu \
        lxc_conf="[{'network.ipv4.ip':'10.0.3.5'}]" \
        lxc_conf_unset="['lxc.utsname']"
```

salt.modules.lxc.**write_conf**(*conf_file*, *conf*)
Write out an LXC configuration file

This is normally only used internally. The format of the data structure must match that which is returned from `lxc.read_conf()`, with `out_format` set to `commented`.

An example might look like:

```
[
    {'lxc.utsname': '$CONTAINER_NAME'},
    '# This is a commented line\n',
    '\n',
    {'lxc.mount': '$CONTAINER_FSTAB'},
    {'lxc.rootfs': {'comment': 'This is another test',
                    'value': 'This is another test'}},
    '\n',
    {'lxc.network.type': 'veth'},
    {'lxc.network.flags': 'up'},
    {'lxc.network.link': 'br0'},
    {'lxc.network.mac': '$CONTAINER_MACADDR'},
    {'lxc.network.ipv4': '$CONTAINER_IPADDR'},
    {'lxc.network.name': '$CONTAINER_DEVICENAME'},
]
```

CLI Example:

```
salt 'minion' lxc.write_conf /etc/lxc/mycontainer.conf \
    out_format=commented
```

22.16.124 salt.modules.mac_group

Manage groups on Mac OS 10.7+

salt.modules.mac_group.**add**(*name*, *gid=None*, ***kwargs*)
Add the specified group

CLI Example:

```
salt '*' group.add foo 3456
```

salt.modules.mac_group.**chgid**(*name*, *gid*)
Change the gid for a named group

CLI Example:

```
salt '*' group.chgid foo 4376
```

salt.modules.mac_group.**delete**(*name*)
Remove the named group

CLI Example:

```
salt '*' group.delete foo
```

salt.modules.mac_group.**getent**(*refresh=False*)
Return info on all groups

CLI Example:

```
salt '*' group.getent
```

salt.modules.mac_group.**info**(*name*)
Return information about a group

CLI Example:

```
salt '*' group.info foo
```

22.16.125 salt.modules.mac_user

Manage users on Mac OS 10.7+

salt.modules.mac_user.**add**(*name*, *uid=None*, *gid=None*, *groups=None*, *home=None*, *shell=None*, *fullname=None*, *createhome=True*, ***kwargs*)
Add a user to the minion

CLI Example:

```
salt '*' user.add name <uid> <gid> <groups> <home> <shell>
```

salt.modules.mac_user.**chfullname**(*name*, *fullname*)
Change the user's Full Name

CLI Example:

```
salt '*' user.chfullname foo 'Foo Bar'
```

salt.modules.mac_user.**chgid**(*name*, *gid*)
Change the default group of the user

CLI Example:

```
salt '*' user.chgid foo 4376
```

salt.modules.mac_user.**chgroups**(*name*, *groups*, *append=False*)

Change the groups to which the user belongs. Note that the user's primary group does not have to be one of the groups passed, membership in the user's primary group is automatically assumed.

groups Groups to which the user should belong, can be passed either as a python list or a comma-separated string

append Instead of removing user from groups not included in the `groups` parameter, just add user to any groups for which they are not members

CLI Example:

```
salt '*' user.chgroups foo wheel,root
```

salt.modules.mac_user.**chhome**(*name*, *home*)

Change the home directory of the user

CLI Example:

```
salt '*' user.chhome foo /Users/foo
```

salt.modules.mac_user.**chshell**(*name*, *shell*)

Change the default shell of the user

CLI Example:

```
salt '*' user.chshell foo /bin/zsh
```

salt.modules.mac_user.**chuid**(*name*, *uid*)

Change the uid for a named user

CLI Example:

```
salt '*' user.chuid foo 4376
```

salt.modules.mac_user.**delete**(*name*, **args*)

Remove a user from the minion

CLI Example:

```
salt '*' user.delete foo
```

salt.modules.mac_user.**getent**(*refresh=False*)

Return the list of all info for all users

CLI Example:

```
salt '*' user.getent
```

salt.modules.mac_user.**info**(*name*)

Return user information

CLI Example:

```
salt '*' user.info root
```

salt.modules.mac_user.**list_groups**(*name*)

Return a list of groups the named user belongs to

CLI Example:

```
salt '*' user.list_groups foo
```

salt.modules.mac_user.**list_users**()
Return a list of all users

CLI Example:

```
salt '*' user.list_users
```

salt.modules.mac_user.**rename**(*name*, *new_name*)
Change the username for a named user

CLI Example:

```
salt '*' user.rename name new_name
```

22.16.126 salt.modules.macports

Support for MacPorts under Mac OSX.

This module has some caveats.

1. Updating the database of available ports is quite resource-intensive. However, *refresh=True* is the default for all operations that need an up-to-date copy of available ports. Consider *refresh=False* when you are sure no db update is needed.

2. In some cases MacPorts doesn't always realize when another copy of itself is running and will gleefully tromp all over the available ports database. This makes MacPorts behave in undefined ways until a fresh complete copy is retrieved.

Because of 1 and 2 it is possible to get the salt-minion into a state where *salt mac-machine pkg./something/* won't want to return. Use

salt-run jobs.active

on the master to check for potentially long-running calls to *port*.

Finally, ports database updates are always handled with *port selfupdate* as opposed to *port sync*. This makes sense in the MacPorts user commmunity but may confuse experienced Linux admins as Linux package managers don't upgrade the packaging software when doing a package database update. In other words *salt mac-machine pkg.refresh_db* is more like *apt-get update; apt-get upgrade dpkg apt-get* than simply *apt-get update*.

salt.modules.macports.**available_version**(**names*, ***kwargs*)
Return the latest version of the named package available for upgrade or installation

Options:

refresh Update ports with `port selfupdate`

CLI Example:

```
salt '*' pkg.latest_version <package name>
salt '*' pkg.latest_version <package1> <package2> <package3>
```

salt.modules.macports.**install**(*name=None*, *refresh=False*, *pkgs=None*, ***kwargs*)
Install the passed package(s) with `port install`

name The name of the formula to be installed. Note that this parameter is ignored if ``pkgs'' is passed.

CLI Example:

```
salt '*' pkg.install <package name>
```

version Specify a version to pkg to install. Ignored if pkgs is specified.

CLI Example:

```
salt '*' pkg.install <package name>
salt '*' pkg.install git-core version='1.8.5.5'
```

variant Specify a variant to pkg to install. Ignored if pkgs is specified.

CLI Example:

```
salt '*' pkg.install <package name>
salt '*' pkg.install git-core version='1.8.5.5' variant='+credential_osxkeychain+doc+pcre'
```

Multiple Package Installation Options:

pkgs A list of formulas to install. Must be passed as a python list.

CLI Example:

```
salt '*' pkg.install pkgs='["foo","bar"]'
salt '*' pkg.install pkgs='["foo@1.2","bar"]'
salt '*' pkg.install pkgs='["foo@1.2+ssl","bar@2.3"]'
```

Returns a dict containing the new package names and versions:

```
{'<package>': {'old': '<old-version>',
               'new': '<new-version>'}}
```

CLI Example:

```
salt '*' pkg.install 'package package package'
```

`salt.modules.macports.`**`latest_version`**(**names*, ***kwargs*)

Return the latest version of the named package available for upgrade or installation

Options:

refresh Update ports with `port selfupdate`

CLI Example:

```
salt '*' pkg.latest_version <package name>
salt '*' pkg.latest_version <package1> <package2> <package3>
```

`salt.modules.macports.`**`list_pkgs`**(*versions_as_list=False*, ***kwargs*)

List the packages currently installed in a dict:

```
{'<package_name>': '<version>'}
```

CLI Example:

```
salt '*' pkg.list_pkgs
```

`salt.modules.macports.`**`list_upgrades`**(*refresh=True*)

Check whether or not an upgrade is available for all packages

Options:

refresh Update ports with `port selfupdate`

CLI Example:

```
    salt '*' pkg.list_upgrades
```

salt.modules.macports.**refresh_db**()
Update ports with `port selfupdate`

salt.modules.macports.**remove**(*name=None*, *pkgs=None*, ***kwargs*)
Removes packages with `port uninstall`.

name The name of the package to be deleted.

Multiple Package Options:

pkgs A list of packages to delete. Must be passed as a python list. The `name` parameter will be ignored if this option is passed.

New in version 0.16.0.

Returns a dict containing the changes.

CLI Example:

```
    salt '*' pkg.remove <package name>
    salt '*' pkg.remove <package1>,<package2>,<package3>
    salt '*' pkg.remove pkgs='["foo", "bar"]'
```

salt.modules.macports.**upgrade**(*refresh=True*)
Run a full upgrade using MacPorts `port upgrade outdated'

Options:

refresh Update ports with `port selfupdate`

Return a dict containing the new package names and versions:

```
    {'<package>': {'old': '<old-version>',
                   'new': '<new-version>'}}
```

CLI Example:

```
    salt '*' pkg.upgrade
```

salt.modules.macports.**upgrade_available**(*pkg*, *refresh=True*)
Check whether or not an upgrade is available for a given package

CLI Example:

```
    salt '*' pkg.upgrade_available <package name>
```

salt.modules.macports.**version**(**names*, ***kwargs*)
Returns a string representing the package version or an empty string if not installed. If more than one package name is specified, a dict of name/version pairs is returned.

CLI Example:

```
    salt '*' pkg.version <package name>
    salt '*' pkg.version <package1> <package2> <package3>
```

22.16.127 salt.modules.makeconf

Support for modifying make.conf under Gentoo

salt.modules.makeconf.**append_cflags**(*value*)
Add to or create a new CFLAGS in the make.conf

Return a dict containing the new value for variable:

```
{'<variable>': {'old': '<old-value>',
                'new': '<new-value>'}}
```

CLI Example:

```
salt '*' makeconf.append_cflags '-pipe'
```

salt.modules.makeconf.**append_cxxflags**(*value*)
Add to or create a new CXXFLAGS in the make.conf

Return a dict containing the new value for variable:

```
{'<variable>': {'old': '<old-value>',
                'new': '<new-value>'}}
```

CLI Example:

```
salt '*' makeconf.append_cxxflags '-pipe'
```

salt.modules.makeconf.**append_emerge_default_opts**(*value*)
Add to or create a new EMERGE_DEFAULT_OPTS in the make.conf

Return a dict containing the new value for variable:

```
{'<variable>': {'old': '<old-value>',
                'new': '<new-value>'}}
```

CLI Example:

```
salt '*' makeconf.append_emerge_default_opts '--jobs'
```

salt.modules.makeconf.**append_features**(*value*)
Add to or create a new FEATURES in the make.conf

Return a dict containing the new value for variable:

```
{'<variable>': {'old': '<old-value>',
                'new': '<new-value>'}}
```

CLI Example:

```
salt '*' makeconf.append_features 'webrsync-gpg'
```

salt.modules.makeconf.**append_gentoo_mirrors**(*value*)
Add to or create a new GENTOO_MIRRORS in the make.conf

Return a dict containing the new value for variable:

```
{'<variable>': {'old': '<old-value>',
                'new': '<new-value>'}}
```

CLI Example:

```
salt '*' makeconf.append_gentoo_mirrors 'http://distfiles.gentoo.org'
```

salt.modules.makeconf.**append_makeopts**(*value*)
Add to or create a new MAKEOPTS in the make.conf

Return a dict containing the new value for variable:

```
{'<variable>': {'old': '<old-value>',
                'new': '<new-value>'}}
```

CLI Example:

```
salt '*' makeconf.append_makeopts '-j3'
```

salt.modules.makeconf.**append_var**(*var*, *value*)
Add to or create a new variable in the make.conf

Return a dict containing the new value for variable:

```
{'<variable>': {'old': '<old-value>',
                'new': '<new-value>'}}
```

CLI Example:

```
salt '*' makeconf.append_var 'LINGUAS' 'en'
```

salt.modules.makeconf.**cflags_contains**(*value*)
Verify if CFLAGS variable contains a value in make.conf

Return True if value is set for var

CLI Example:

```
salt '*' makeconf.cflags_contains '-pipe'
```

salt.modules.makeconf.**chost_contains**(*value*)
Verify if CHOST variable contains a value in make.conf

Return True if value is set for var

CLI Example:

```
salt '*' makeconf.chost_contains 'x86_64-pc-linux-gnu'
```

salt.modules.makeconf.**cxxflags_contains**(*value*)
Verify if CXXFLAGS variable contains a value in make.conf

Return True if value is set for var

CLI Example:

```
salt '*' makeconf.cxxflags_contains '-pipe'
```

salt.modules.makeconf.**emerge_default_opts_contains**(*value*)
Verify if EMERGE_DEFAULT_OPTS variable contains a value in make.conf

Return True if value is set for var

CLI Example:

```
salt '*' makeconf.emerge_default_opts_contains '--jobs'
```

salt.modules.makeconf.**features_contains**(*value*)
Verify if FEATURES variable contains a value in make.conf

Return True if value is set for var

CLI Example:

```
salt '*' makeconf.features_contains 'webrsync-gpg'
```

salt.modules.makeconf.**gentoo_mirrors_contains**(*value*)
Verify if GENTOO_MIRRORS variable contains a value in make.conf

Return True if value is set for var

CLI Example:

```
salt '*' makeconf.gentoo_mirrors_contains 'http://distfiles.gentoo.org'
```

salt.modules.makeconf.**get_cflags**()
Get the value of CFLAGS variable in the make.conf

Return the value of the variable or None if the variable is not in the make.conf

CLI Example:

```
salt '*' makeconf.get_cflags
```

salt.modules.makeconf.**get_chost**()
Get the value of CHOST variable in the make.conf

Return the value of the variable or None if the variable is not in the make.conf

CLI Example:

```
salt '*' makeconf.get_chost
```

salt.modules.makeconf.**get_cxxflags**()
Get the value of CXXFLAGS variable in the make.conf

Return the value of the variable or None if the variable is not in the make.conf

CLI Example:

```
salt '*' makeconf.get_cxxflags
```

salt.modules.makeconf.**get_emerge_default_opts**()
Get the value of EMERGE_DEFAULT_OPTS variable in the make.conf

Return the value of the variable or None if the variable is not in the make.conf

CLI Example:

```
salt '*' makeconf.get_emerge_default_opts
```

salt.modules.makeconf.**get_features**()
Get the value of FEATURES variable in the make.conf

Return the value of the variable or None if the variable is not in the make.conf

CLI Example:

```
salt '*' makeconf.get_features
```

salt.modules.makeconf.**get_gentoo_mirrors**()
Get the value of GENTOO_MIRRORS variable in the make.conf

Return the value of the variable or None if the variable is not in the make.conf

CLI Example:

```
salt '*' makeconf.get_gentoo_mirrors
```

salt.modules.makeconf.**get_makeopts**()
Get the value of MAKEOPTS variable in the make.conf

Return the value of the variable or None if the variable is not in the make.conf

CLI Example:

```
salt '*' makeconf.get_makeopts
```

salt.modules.makeconf.**get_sync**()
Get the value of SYNC variable in the make.conf

Return the value of the variable or None if the variable is not in the make.conf

CLI Example:

```
salt '*' makeconf.get_sync
```

salt.modules.makeconf.**get_var**(*var*)
Get the value of a variable in make.conf

Return the value of the variable or None if the variable is not in make.conf

CLI Example:

```
salt '*' makeconf.get_var 'LINGUAS'
```

salt.modules.makeconf.**makeopts_contains**(*value*)
Verify if MAKEOPTS variable contains a value in make.conf

Return True if value is set for var

CLI Example:

```
salt '*' makeconf.makeopts_contains '-j3'
```

salt.modules.makeconf.**remove_var**(*var*)
Remove a variable from the make.conf

Return a dict containing the new value for the variable:

```
{'<variable>': {'old': '<old-value>',
                'new': '<new-value>'}}
```

CLI Example:

```
salt '*' makeconf.remove_var 'LINGUAS'
```

salt.modules.makeconf.**set_cflags**(*value*)
Set the CFLAGS variable

Return a dict containing the new value for variable:

```
{'<variable>': {'old': '<old-value>',
                'new': '<new-value>'}}
```

CLI Example:

```
salt '*' makeconf.set_cflags '-march=native -O2 -pipe'
```

salt.modules.makeconf.**set_chost**(*value*)
Set the CHOST variable

Return a dict containing the new value for variable:

```
{'<variable>': {'old': '<old-value>',
                'new': '<new-value>'}}
```

CLI Example:

```
salt '*' makeconf.set_chost 'x86_64-pc-linux-gnu'
```

salt.modules.makeconf.**set_cxxflags**(*value*)
Set the CXXFLAGS variable

Return a dict containing the new value for variable:

```
{'<variable>': {'old': '<old-value>',
                'new': '<new-value>'}}
```

CLI Example:

```
salt '*' makeconf.set_cxxflags '-march=native -O2 -pipe'
```

salt.modules.makeconf.**set_emerge_default_opts**(*value*)
Set the EMERGE_DEFAULT_OPTS variable

Return a dict containing the new value for variable:

```
{'<variable>': {'old': '<old-value>',
                'new': '<new-value>'}}
```

CLI Example:

```
salt '*' makeconf.set_emerge_default_opts '--jobs'
```

salt.modules.makeconf.**set_gentoo_mirrors**(*value*)
Set the GENTOO_MIRRORS variable

Return a dict containing the new value for variable:

```
{'<variable>': {'old': '<old-value>',
                'new': '<new-value>'}}
```

CLI Example:

```
salt '*' makeconf.set_gentoo_mirrors 'http://distfiles.gentoo.org'
```

salt.modules.makeconf.**set_makeopts**(*value*)
Set the MAKEOPTS variable

Return a dict containing the new value for variable:

```
{'<variable>': {'old': '<old-value>',
                'new': '<new-value>'}}
```

CLI Example:

```
salt '*' makeconf.set_makeopts '-j3'
```

salt.modules.makeconf.**set_sync**(*value*)
Set the SYNC variable

Return a dict containing the new value for variable:

```
{'<variable>': {'old': '<old-value>',
                'new': '<new-value>'}}
```

CLI Example:

```
salt '*' makeconf.set_sync 'rsync://rsync.namerica.gentoo.org/gentoo-portage'
```

salt.modules.makeconf.**set_var**(*var*, *value*)
Set a variable in the make.conf

Return a dict containing the new value for variable:

```
{'<variable>': {'old': '<old-value>',
                'new': '<new-value>'}}
```

CLI Example:

```
salt '*' makeconf.set_var 'LINGUAS' 'en'
```

salt.modules.makeconf.**sync_contains**(*value*)
Verify if SYNC variable contains a value in make.conf

Return True if value is set for var

CLI Example:

```
salt '*' makeconf.sync_contains 'rsync://rsync.namerica.gentoo.org/gentoo-portage'
```

salt.modules.makeconf.**trim_cflags**(*value*)
Remove a value from CFLAGS variable in the make.conf

Return a dict containing the new value for variable:

```
{'<variable>': {'old': '<old-value>',
                'new': '<new-value>'}}
```

CLI Example:

```
salt '*' makeconf.trim_cflags '-pipe'
```

salt.modules.makeconf.**trim_cxxflags**(*value*)
Remove a value from CXXFLAGS variable in the make.conf

Return a dict containing the new value for variable:

```
{'<variable>': {'old': '<old-value>',
                'new': '<new-value>'}}
```

CLI Example:

```
salt '*' makeconf.trim_cxxflags '-pipe'
```

salt.modules.makeconf.**trim_emerge_default_opts**(*value*)
Remove a value from EMERGE_DEFAULT_OPTS variable in the make.conf

Return a dict containing the new value for variable:

```
{'<variable>': {'old': '<old-value>',
                'new': '<new-value>'}}
```

CLI Example:

```
salt '*' makeconf.trim_emerge_default_opts '--jobs'
```

salt.modules.makeconf.**trim_features**(*value*)

Remove a value from FEATURES variable in the make.conf

Return a dict containing the new value for variable:

```
{'<variable>': {'old': '<old-value>',
                'new': '<new-value>'}}
```

CLI Example:

```
salt '*' makeconf.trim_features 'webrsync-gpg'
```

salt.modules.makeconf.**trim_gentoo_mirrors**(*value*)

Remove a value from GENTOO_MIRRORS variable in the make.conf

Return a dict containing the new value for variable:

```
{'<variable>': {'old': '<old-value>',
                'new': '<new-value>'}}
```

CLI Example:

```
salt '*' makeconf.trim_gentoo_mirrors 'http://distfiles.gentoo.org'
```

salt.modules.makeconf.**trim_makeopts**(*value*)

Remove a value from MAKEOPTS variable in the make.conf

Return a dict containing the new value for variable:

```
{'<variable>': {'old': '<old-value>',
                'new': '<new-value>'}}
```

CLI Example:

```
salt '*' makeconf.trim_makeopts '-j3'
```

salt.modules.makeconf.**trim_var**(*var*, *value*)

Remove a value from a variable in the make.conf

Return a dict containing the new value for variable:

```
{'<variable>': {'old': '<old-value>',
                'new': '<new-value>'}}
```

CLI Example:

```
salt '*' makeconf.trim_var 'LINGUAS' 'en'
```

salt.modules.makeconf.**var_contains**(*var*, *value*)

Verify if variable contains a value in make.conf

Return True if value is set for var

CLI Example:

```
salt '*' makeconf.var_contains 'LINGUAS' 'en'
```

22.16.128 salt.modules.match

The match module allows for match routines to be run and determine target specs

salt.modules.match.**compound**(*tgt*, *minion_id=None*)
Return True if the minion ID matches the given compound target

minion_id Specify the minion ID to match against the target expression

New in version 2014.7.0.

CLI Example:

```
salt '*' match.compound 'L@cheese,foo and *'
```

salt.modules.match.**data**(*tgt*)
Return True if the minion matches the given data target

CLI Example:

```
salt '*' match.data 'spam:eggs'
```

salt.modules.match.**filter_by**(*lookup*, *expr_form='compound'*, *minion_id=None*)
Return the first match in a dictionary of target patterns

New in version 2014.7.0.

CLI Example:

```
salt '*' match.filter_by '{foo*: Foo!, bar*: Bar!}' minion_id=bar03
```

Pillar Example:

```
{% set roles = salt['match.filter_by']({
    'web*': ['app', 'caching'],
    'db*': ['db'],
}) %}
```

salt.modules.match.**glob**(*tgt*, *minion_id=None*)
Return True if the minion ID matches the given glob target

minion_id Specify the minion ID to match against the target expression

New in version 2014.7.0.

CLI Example:

```
salt '*' match.glob '*'
```

salt.modules.match.**grain**(*tgt*, *delimiter=':'*, *delim=None*)
Return True if the minion matches the given grain target. The `delimiter` argument can be used to specify a different delimiter.

CLI Example:

```
salt '*' match.grain 'os:Ubuntu'
salt '*' match.grain 'ipv6|2001:db8::ff00:42:8329' delimiter='|'
```

delimiter Specify an alternate delimiter to use when traversing a nested dict

New in version 2014.7.0.

delim Specify an alternate delimiter to use when traversing a nested dict

New in version 0.16.4.

Deprecated since version 2014.7.0.

salt.modules.match.**grain_pcre**(*tgt*, *delimiter=':'*, *delim=None*)

Return True if the minion matches the given grain_pcre target. The `delimiter` argument can be used to specify a different delimiter.

CLI Example:

```
salt '*' match.grain_pcre 'os:Fedo.*'
salt '*' match.grain_pcre 'ipv6|2001:.*' delimiter='|'
```

delimiter Specify an alternate delimiter to use when traversing a nested dict

New in version 2014.7.0.

delim Specify an alternate delimiter to use when traversing a nested dict

New in version 0.16.4.

Deprecated since version 2014.7.0.

salt.modules.match.**ipcidr**(*tgt*)

Return True if the minion matches the given ipcidr target

CLI Example:

```
salt '*' match.ipcidr '192.168.44.0/24'
```

salt.modules.match.**list_**(*tgt*, *minion_id=None*)

Return True if the minion ID matches the given list target

minion_id Specify the minion ID to match against the target expression

New in version 2014.7.0.

CLI Example:

```
salt '*' match.list 'server1,server2'
```

salt.modules.match.**pcre**(*tgt*, *minion_id=None*)

Return True if the minion ID matches the given pcre target

minion_id Specify the minion ID to match against the target expression

New in version 2014.7.0.

CLI Example:

```
salt '*' match.pcre '.*'
```

salt.modules.match.**pillar**(*tgt*, *delimiter=':'*, *delim=None*)

Return True if the minion matches the given pillar target. The `delimiter` argument can be used to specify a different delimiter.

CLI Example:

```
salt '*' match.pillar 'cheese:foo'
salt '*' match.pillar 'clone_url|https://github.com/saltstack/salt.git' delimiter='|'
```

delimiter Specify an alternate delimiter to use when traversing a nested dict

New in version 2014.7.0.

delim Specify an alternate delimiter to use when traversing a nested dict

New in version 0.16.4.

Deprecated since version 2014.7.0.

`salt.modules.match.`**`pillar_pcre`**(*tgt*, *delimiter=':'*, *delim=None*)

Return True if the minion matches the given pillar_pcre target. The `delimiter` argument can be used to specify a different delimiter.

CLI Example:

```
salt '*' match.pillar_pcre 'cheese:(swiss|american)'
salt '*' match.pillar_pcre 'clone_url|https://github\.com/.*\.git' delimiter='|'
```

delimiter Specify an alternate delimiter to use when traversing a nested dict

New in version 2014.7.0.

delim Specify an alternate delimiter to use when traversing a nested dict

New in version 0.16.4.

Deprecated since version 2014.7.0.

22.16.129 salt.modules.mdadm

Salt module to manage RAID arrays with mdadm

`salt.modules.mdadm.`**`assemble`**(*name*, *devices*, *test_mode=False*, ***kwargs*)

Assemble a RAID device.

CLI Examples:

```
salt '*' raid.assemble /dev/md0 ['/dev/xvdd', '/dev/xvde']
```

Note: Adding `test_mode=True` as an argument will print out the mdadm command that would have been run.

---

name The name of the array to assemble.

devices The list of devices comprising the array to assemble.

kwargs Optional arguments to be passed to mdadm.

returns

test_mode=True: Prints out the full command.

test_mode=False (Default): Executes command on the host(s) and prints out the mdadm output.

For more info, read the `mdadm` manpage.

`salt.modules.mdadm.`**`create`**(*name*, *level*, *devices*, *metadata='default'*, *test_mode=False*, ***kwargs*)

Create a RAID device.

Changed in version 2014.7.0.

Warning: Use with CAUTION, as this function can be very destructive if not used properly!

CLI Examples:

```
salt '*' raid.create /dev/md0 level=1 chunk=256 devices="['/dev/xvdd', '/dev/xvde']" test_mode=T
```

Note: Adding `test_mode=True` as an argument will print out the mdadm command that would have been run.

---

name The name of the array to create.

level The RAID level to use when creating the raid.

devices A list of devices used to build the array.

metadata Version of metadata to use when creating the array.

kwargs Optional arguments to be passed to mdadm.

returns

test_mode=True: Prints out the full command.

test_mode=False (Default): Executes command on remote the host(s) and Prints out the mdadm output.

Note: It takes time to create a RAID array. You can check the progress in ``resync_status:'' field of the results from the following command:

```
salt '*' raid.detail /dev/md0
```

For more info, read the `mdadm(8)` manpage

`salt.modules.mdadm.`**`destroy`**(*device*)

Destroy a RAID device.

WARNING This will zero the superblock of all members of the RAID array..

CLI Example:

```
salt '*' raid.destroy /dev/md0
```

`salt.modules.mdadm.`**`detail`**(*device='/dev/md0'*)

Show detail for a specified RAID device

CLI Example:

```
salt '*' raid.detail '/dev/md0'
```

`salt.modules.mdadm.`**`list_`**()

List the RAID devices.

CLI Example:

```
salt '*' raid.list
```

`salt.modules.mdadm.`**`save_config`**()

Save RAID configuration to config file.

Same as: mdadm --detail --scan >> /etc/mdadm/mdadm.conf

Fixes this issue with Ubuntu REF: http://askubuntu.com/questions/209702/why-is-my-raid-dev-md1-showing-up-as-dev-md126-is-mdadm-conf-being-ignored

CLI Example:

```
salt '*' raid.save_config
```

salt.modules.mdadm.**stop**()

Shut down all arrays that can be shut down (i.e. are not currently in use).

CLI Example:

```
salt '*' raid.stop
```

22.16.130 salt.modules.memcached

Module for Management of Memcached Keys

New in version 2014.1.0.

salt.modules.memcached.**add**(*key*, *value*, *host=`127.0.0.1'*, *port=11211*, *time=0*, *min_compress_len=0*)

Add a key to the memcached server, but only if it does not exist. Returns False if the key already exists.

CLI Example:

```
salt '*' memcached.add <key> <value>
```

salt.modules.memcached.**decrement**(*key*, *delta=1*, *host=`127.0.0.1'*, *port=11211*)

Decrement the value of a key

CLI Example:

```
salt '*' memcached.decrement <key>
salt '*' memcached.decrement <key> 2
```

salt.modules.memcached.**delete**(*key*, *host=`127.0.0.1'*, *port=11211*, *time=0*)

Delete a key from memcache server

CLI Example:

```
salt '*' memcached.delete <key>
```

salt.modules.memcached.**get**(*key*, *host=`127.0.0.1'*, *port=11211*)

Retrieve value for a key

CLI Example:

```
salt '*' memcached.get <key>
```

salt.modules.memcached.**increment**(*key*, *delta=1*, *host=`127.0.0.1'*, *port=11211*)

Increment the value of a key

CLI Example:

```
salt '*' memcached.increment <key>
salt '*' memcached.increment <key> 2
```

salt.modules.memcached.**replace**(*key*, *value*, *host=`127.0.0.1'*, *port=11211*, *time=0*, *min_compress_len=0*)

Replace a key on the memcached server. This only succeeds if the key already exists. This is the opposite of `memcached.add`

CLI Example:

```
salt '*' memcached.replace <key> <value>
```

salt.modules.memcached.**set_**(*key*, *value*, *host=`127.0.0.1'*, *port=11211*, *time=0*, *min_compress_len=0*)
Set a key on the memcached server, overwriting the value if it exists.

CLI Example:

```
salt '*' memcached.set <key> <value>
```

salt.modules.memcached.**status**(*host=`127.0.0.1'*, *port=11211*)
Get memcached status

CLI Example:

```
salt '*' memcached.status
```

22.16.131 salt.modules.mine

The function cache system allows for data to be stored on the master so it can be easily read by other minions

salt.modules.mine.**delete**(*fun*)
Remove specific function contents of minion. Returns True on success.

CLI Example:

```
salt '*' mine.delete 'network.interfaces'
```

salt.modules.mine.**flush**()
Remove all mine contents of minion. Returns True on success.

CLI Example:

```
salt '*' mine.flush
```

salt.modules.mine.**get**(*tgt*, *fun*, *expr_form='glob'*)
Get data from the mine based on the target, function and expr_form

Targets can be matched based on any standard matching system that can be matched on the master via these keywords:

```
glob
pcre
grain
grain_pcre
compound
pillar
pillar_pcre
```

Note that all pillar matches, whether using the compound matching system or the pillar matching system, will be exact matches, with globbing disabled.

CLI Example:

```
salt '*' mine.get '*' network.interfaces
salt '*' mine.get 'os:Fedora' network.interfaces grain
salt '*' mine.get 'os:Fedora and S@192.168.5.0/24' network.ipaddrs compound
```

salt.modules.mine.**get_docker**(*interfaces=None, cidrs=None*)
Get all mine data for `docker.get_containers' and run an aggregation routine. The ``interfaces'' parameter allows for specifying which network interfaces to select ip addresses from. The ``cidrs'' parameter allows for specifying a list of cidrs which the ip address must match.

CLI Example:

```
salt '*' mine.get_docker
salt '*' mine.get_docker interfaces='eth0'
salt '*' mine.get_docker interfaces='["eth0", "eth1"]'
salt '*' mine.get_docker cidrs='107.170.147.0/24'
salt '*' mine.get_docker cidrs='["107.170.147.0/24", "172.17.42.0/24"]'
salt '*' mine.get_docker interfaces='["eth0", "eth1"]' cidrs='["107.170.147.0/24", "172.17.42.0/
```

salt.modules.mine.**send**(*func, *args, **kwargs*)
Send a specific function to the mine.

CLI Example:

```
salt '*' mine.send network.ip_addrs eth0
salt '*' mine.send eth0_ip_addrs mine_function=network.ip_addrs eth0
```

salt.modules.mine.**update**(*clear=False*)
Execute the configured functions and send the data back up to the master The functions to be executed are merged from the master config, pillar and minion config under the option ``function_cache'':

```
mine_functions:
  network.ip_addrs:
    - eth0
  disk.usage: []
```

The function cache will be populated with information from executing these functions

CLI Example:

```
salt '*' mine.update
```

22.16.132 salt.modules.mod_random

New in version 2014.7.0.

Provides access to randomness generators.

salt.modules.mod_random.**get_str**(*length=20*)
New in version 2014.7.0.

Returns a random string of the specified length.

length [20] Any valid number of bytes.

CLI Example:

```
salt '*' random.get_str 128
```

salt.modules.mod_random.**hash**(*value, algorithm='sha512'*)
New in version 2014.7.0.

Encodes a value with the specified encoder.

value The value to be hashed.

algorithm [sha512] The algorithm to use. May be any valid algorithm supported by hashlib.

CLI Example:

```
salt '*' random.hash 'I am a string' md5
```

salt.modules.mod_random.**shadow_hash**(*crypt_salt=None*, *password=None*, *algorithm='sha512'*)

Generates a salted hash suitable for /etc/shadow.

crypt_salt [None] Salt to be used in the generation of the hash. If one is not provided, a random salt will be generated.

password [None] Value to be salted and hashed. If one is not provided, a random password will be generated.

algorithm [sha512] Hash algorithm to use.

CLI Example:

```
salt '*' random.shadow_hash 'My5alT' 'MyP@asswd' md5
```

salt.modules.mod_random.**str_encode**(*value*, *encoder='base64'*)

New in version 2014.7.0.

value The value to be encoded.

encoder [base64] The encoder to use on the subsequent string.

CLI Example:

```
salt '*' random.str_encode 'I am a new string' base64
```

22.16.133 salt.modules.modjk

Control Modjk via the Apache Tomcat ``Status'' worker (http://tomcat.apache.org/connectors-doc/reference/status.html)

Below is an example of the configuration needed for this module. This configuration data can be placed either in *grains* or *pillar*.

If using grains, this can be accomplished *statically* or via a *grain module*.

If using pillar, the yaml configuration can be placed directly into a pillar SLS file, making this both the easier and more dynamic method of configuring this module.

```
modjk:
  default:
    url: http://localhost/jkstatus
    user: modjk
    pass: secret
    realm: authentication realm for digest passwords
    timeout: 5
  otherVhost:
    url: http://otherVhost/jkstatus
    user: modjk
    pass: secret2
    realm: authentication realm2 for digest passwords
    timeout: 600
```

salt.modules.modjk.**bulk_activate**(*workers*, *lbn*, *profile='default'*)

Activate all the given workers in the specific load balancer

CLI Examples:

```
    salt '*' modjk.bulk_activate node1,node2,node3 loadbalancer1
    salt '*' modjk.bulk_activate node1,node2,node3 loadbalancer1 other-profile

    salt '*' modjk.bulk_activate ["node1","node2","node3"] loadbalancer1
    salt '*' modjk.bulk_activate ["node1","node2","node3"] loadbalancer1 other-profile
```

salt.modules.modjk.**bulk_disable**(*workers*, *lbn*, *profile='default'*)
Disable all the given workers in the specific load balancer

CLI Examples:

```
    salt '*' modjk.bulk_disable node1,node2,node3 loadbalancer1
    salt '*' modjk.bulk_disable node1,node2,node3 loadbalancer1 other-profile

    salt '*' modjk.bulk_disable ["node1","node2","node3"] loadbalancer1
    salt '*' modjk.bulk_disable ["node1","node2","node3"] loadbalancer1 other-profile
```

salt.modules.modjk.**bulk_recover**(*workers*, *lbn*, *profile='default'*)
Recover all the given workers in the specific load balancer

CLI Examples:

```
    salt '*' modjk.bulk_recover node1,node2,node3 loadbalancer1
    salt '*' modjk.bulk_recover node1,node2,node3 loadbalancer1 other-profile

    salt '*' modjk.bulk_recover ["node1","node2","node3"] loadbalancer1
    salt '*' modjk.bulk_recover ["node1","node2","node3"] loadbalancer1 other-profile
```

salt.modules.modjk.**bulk_stop**(*workers*, *lbn*, *profile='default'*)
Stop all the given workers in the specific load balancer

CLI Examples:

```
    salt '*' modjk.bulk_stop node1,node2,node3 loadbalancer1
    salt '*' modjk.bulk_stop node1,node2,node3 loadbalancer1 other-profile

    salt '*' modjk.bulk_stop ["node1","node2","node3"] loadbalancer1
    salt '*' modjk.bulk_stop ["node1","node2","node3"] loadbalancer1 other-profile
```

salt.modules.modjk.**dump_config**(*profile='default'*)
Dump the original configuration that was loaded from disk

CLI Examples:

```
    salt '*' modjk.dump_config
    salt '*' modjk.dump_config other-profile
```

salt.modules.modjk.**get_running**(*profile='default'*)
Get the current running config (not from disk)

CLI Examples:

```
    salt '*' modjk.get_running
    salt '*' modjk.get_running other-profile
```

salt.modules.modjk.**lb_edit**(*lbn*, *settings*, *profile='default'*)
Edit the loadbalancer settings

Note: http://tomcat.apache.org/connectors-doc/reference/status.html Data Parameters for the standard Update Action

CLI Examples:

```
salt '*' modjk.lb_edit loadbalancer1 "{'vlr': 1, 'vlt': 60}"
salt '*' modjk.lb_edit loadbalancer1 "{'vlr': 1, 'vlt': 60}" other-profile
```

salt.modules.modjk.**list_configured_members**(*lbn*, *profile='default'*)
Return a list of member workers from the configuration files

CLI Examples:

```
salt '*' modjk.list_configured_members loadbalancer1
salt '*' modjk.list_configured_members loadbalancer1 other-profile
```

salt.modules.modjk.**recover_all**(*lbn*, *profile='default'*)
Set the all the workers in lbn to recover and activate them if they are not

CLI Examples:

```
salt '*' modjk.recover_all loadbalancer1
salt '*' modjk.recover_all loadbalancer1 other-profile
```

salt.modules.modjk.**reset_stats**(*lbn*, *profile='default'*)
Reset all runtime statistics for the load balancer

CLI Examples:

```
salt '*' modjk.reset_stats loadbalancer1
salt '*' modjk.reset_stats loadbalancer1 other-profile
```

salt.modules.modjk.**version**(*profile='default'*)
Return the modjk version

CLI Examples:

```
salt '*' modjk.version
salt '*' modjk.version other-profile
```

salt.modules.modjk.**worker_activate**(*worker*, *lbn*, *profile='default'*)
Set the worker to activate state in the lbn load balancer

CLI Examples:

```
salt '*' modjk.worker_activate node1 loadbalancer1
salt '*' modjk.worker_activate node1 loadbalancer1 other-profile
```

salt.modules.modjk.**worker_disable**(*worker*, *lbn*, *profile='default'*)
Set the worker to disable state in the lbn load balancer

CLI Examples:

```
salt '*' modjk.worker_disable node1 loadbalancer1
salt '*' modjk.worker_disable node1 loadbalancer1 other-profile
```

salt.modules.modjk.**worker_edit**(*worker*, *lbn*, *settings*, *profile='default'*)
Edit the worker settings

Note: http://tomcat.apache.org/connectors-doc/reference/status.html Data Parameters for the standard Update Action

CLI Examples:

```
    salt '*' modjk.worker_edit node1 loadbalancer1 "{'vwf': 500, 'vwd': 60}"
    salt '*' modjk.worker_edit node1 loadbalancer1 "{'vwf': 500, 'vwd': 60}" other-profile
```

salt.modules.modjk.**worker_recover**(*worker*, *lbn*, *profile='default'*)
Set the worker to recover this module will fail if it is in OK state

CLI Examples:

```
    salt '*' modjk.worker_recover node1 loadbalancer1
    salt '*' modjk.worker_recover node1 loadbalancer1 other-profile
```

salt.modules.modjk.**worker_status**(*worker*, *profile='default'*)
Return the state of the worker

CLI Examples:

```
    salt '*' modjk.worker_status node1
    salt '*' modjk.worker_status node1 other-profile
```

salt.modules.modjk.**worker_stop**(*worker*, *lbn*, *profile='default'*)
Set the worker to stopped state in the lbn load balancer

CLI Examples:

```
    salt '*' modjk.worker_activate node1 loadbalancer1
    salt '*' modjk.worker_activate node1 loadbalancer1 other-profile
```

salt.modules.modjk.**workers**(*profile='default'*)
Return a list of member workers and their status

CLI Examples:

```
    salt '*' modjk.workers
    salt '*' modjk.workers other-profile
```

22.16.134 salt.modules.mongodb

Module to provide MongoDB functionality to Salt

configuration This module uses PyMongo, and accepts configuration details as parameters as well as configuration settings:

```
        mongodb.host: 'localhost'
        mongodb.port: 27017
        mongodb.user: ''
        mongodb.password: ''
```

This data can also be passed into pillar. Options passed into opts will overwrite options passed into pillar.

salt.modules.mongodb.**db_exists**(*name*, *user=None*, *password=None*, *host=None*, *port=None*)
Checks if a database exists in Mongodb

CLI Example:

```
    salt '*' mongodb.db_exists <name> <user> <password> <host> <port>
```

salt.modules.mongodb.**db_list**(*user=None*, *password=None*, *host=None*, *port=None*)
List all Mongodb databases

CLI Example:

```
salt '*' mongodb.db_list <user> <password> <host> <port>
```

salt.modules.mongodb.**db_remove**(*name*, *user=None*, *password=None*, *host=None*, *port=None*)

Remove a Mongodb database

CLI Example:

```
salt '*' mongodb.db_remove <name> <user> <password> <host> <port>
```

salt.modules.mongodb.**find**(*collection*, *query=None*, *user=None*, *password=None*, *host=None*, *port=None*, *database='admin'*)

salt.modules.mongodb.**insert**(*objects*, *collection*, *user=None*, *password=None*, *host=None*, *port=None*, *database='admin'*)

Insert an object or list of objects into a collection

CLI Example:

```
salt '*' mongodb.insert '[{"foo": "FOO", "bar": "BAR"}, {"foo": "BAZ", "bar": "BAM"}]' mycollect
```

salt.modules.mongodb.**remove**(*collection*, *query=None*, *user=None*, *password=None*, *host=None*, *port=None*, *database='admin'*, *w=1*)

Remove an object or list of objects into a collection

CLI Example:

```
salt '*' mongodb.remove mycollection '[{"foo": "FOO", "bar": "BAR"}, {"foo": "BAZ", "bar": "BAM"
```

salt.modules.mongodb.**user_create**(*name*, *passwd*, *user=None*, *password=None*, *host=None*, *port=None*, *database='admin'*)

Create a Mongodb user

CLI Example:

```
salt '*' mongodb.user_create <name> <user> <password> <host> <port> <database>
```

salt.modules.mongodb.**user_exists**(*name*, *user=None*, *password=None*, *host=None*, *port=None*, *database='admin'*)

Checks if a user exists in Mongodb

CLI Example:

```
salt '*' mongodb.user_exists <name> <user> <password> <host> <port> <database>
```

salt.modules.mongodb.**user_list**(*user=None*, *password=None*, *host=None*, *port=None*, *database='admin'*)

List users of a Mongodb database

CLI Example:

```
salt '*' mongodb.user_list <user> <password> <host> <port> <database>
```

salt.modules.mongodb.**user_remove**(*name*, *user=None*, *password=None*, *host=None*, *port=None*, *database='admin'*)

Remove a Mongodb user

CLI Example:

```
salt '*' mongodb.user_remove <name> <user> <password> <host> <port> <database>
```

22.16.135 salt.modules.monit

Monit service module. This module will create a monit type service watcher.

salt.modules.monit.**monitor**(*name*)
monitor service via monit

CLI Example:

```
salt '*' monit.monitor <service name>
```

salt.modules.monit.**restart**(*name*)
Restart service via monit

CLI Example:

```
salt '*' monit.restart <service name>
```

salt.modules.monit.**start**(*name*)
CLI Example:

```
salt '*' monit.start <service name>
```

salt.modules.monit.**status**(*svc_name=''*)
Display a process status from monit

CLI Example:

```
salt '*' monit.status
salt '*' monit.status <service name>
```

salt.modules.monit.**stop**(*name*)
Stops service via monit

CLI Example:

```
salt '*' monit.stop <service name>
```

salt.modules.monit.**summary**(*svc_name=''*)
Display a summary from monit

CLI Example:

```
salt '*' monit.summary
salt '*' monit.summary <service name>
```

salt.modules.monit.**unmonitor**(*name*)
Unmonitor service via monit

CLI Example:

```
salt '*' monit.unmonitor <service name>
```

22.16.136 salt.modules.moosefs

Module for gathering and managing information about MooseFS

salt.modules.moosefs.**dirinfo**(*path*, *opts=None*)
Return information on a directory located on the Moose

CLI Example:

```
    salt '*' moosefs.dirinfo /path/to/dir/ [-[n][h|H]]
```

salt.modules.moosefs.**fileinfo**(*path*)
Return information on a file located on the Moose

CLI Example:

```
    salt '*' moosefs.fileinfo /path/to/dir/
```

salt.modules.moosefs.**getgoal**(*path*, *opts=None*)
Return goal(s) for a file or directory

CLI Example:

```
    salt '*' moosefs.getgoal /path/to/file [-[n][h|H]]
    salt '*' moosefs.getgoal /path/to/dir/ [-[n][h|H][r]]
```

salt.modules.moosefs.**mounts**()
Return a list of current MooseFS mounts

CLI Example:

```
    salt '*' moosefs.mounts
```

22.16.137 salt.modules.mount

Salt module to manage unix mounts and the fstab file

salt.modules.mount.**active**(*extended=False*)
List the active mounts.

CLI Example:

```
    salt '*' mount.active
```

salt.modules.mount.**automaster**(*config='/etc/auto_salt'*)
List the contents of the fstab

CLI Example:

```
    salt '*' mount.fstab
```

salt.modules.mount.**fstab**(*config='/etc/fstab'*)
List the contents of the fstab

CLI Example:

```
    salt '*' mount.fstab
```

salt.modules.mount.**is_fuse_exec**(*cmd*)
Returns true if the command passed is a fuse mountable application.

CLI Example:

```
    salt '*' mount.is_fuse_exec sshfs
```

salt.modules.mount.**is_mounted**(*name*)
New in version 2014.7.0.

Provide information if the path is mounted

CLI Example:

```
salt '*' mount.is_mounted /mnt/share
```

salt.modules.mount.**mount**(*name*, *device*, *mkmnt=False*, *fstype=''*, *opts='defaults'*, *user=None*)
Mount a device

CLI Example:

```
salt '*' mount.mount /mnt/foo /dev/sdz1 True
```

salt.modules.mount.**remount**(*name*, *device*, *mkmnt=False*, *fstype=''*, *opts='defaults'*, *user=None*)
Attempt to remount a device, if the device is not already mounted, mount is called

CLI Example:

```
salt '*' mount.remount /mnt/foo /dev/sdz1 True
```

salt.modules.mount.**rm_automaster**(*name*, *device*, *config='/etc/auto_salt'*)
Remove the mount point from the auto_master

CLI Example:

```
salt '*' mount.rm_automaster /mnt/foo
```

salt.modules.mount.**rm_fstab**(*name*, *device*, *config='/etc/fstab'*)
Remove the mount point from the fstab

CLI Example:

```
salt '*' mount.rm_fstab /mnt/foo
```

salt.modules.mount.**set_automaster**(*name*, *device*, *fstype*, *opts=''*, *config='/etc/auto_salt'*, *test=False*, ***kwargs*)
Verify that this mount is represented in the auto_salt, change the mount to match the data passed, or add the mount if it is not present.

CLI Example:

```
salt '*' mount.set_automaster /mnt/foo /dev/sdz1 ext4
```

salt.modules.mount.**set_fstab**(*name*, *device*, *fstype*, *opts='defaults'*, *dump=0*, *pass_num=0*, *config='/etc/fstab'*, *test=False*, *match_on='auto'*, ***kwargs*)
Verify that this mount is represented in the fstab, change the mount to match the data passed, or add the mount if it is not present.

CLI Example:

```
salt '*' mount.set_fstab /mnt/foo /dev/sdz1 ext4
```

salt.modules.mount.**swapoff**(*name*)
Deactivate a named swap mount

CLI Example:

```
salt '*' mount.swapoff /root/swapfile
```

salt.modules.mount.**swapon**(*name*, *priority=None*)
Activate a swap disk

CLI Example:

```
    salt '*' mount.swapon /root/swapfile
```

salt.modules.mount.**swaps**()
Return a dict containing information on active swap

CLI Example:

```
    salt '*' mount.swaps
```

salt.modules.mount.**umount**(*name*, *device=None*, *user=None*)
Attempt to unmount a device by specifying the directory it is mounted on

CLI Example:

```
    salt '*' mount.umount /mnt/foo

    .. versionadded:: 2015.5.0

    salt '*' mount.umount /mnt/foo /dev/xvdc1
```

22.16.138 salt.modules.munin

Run munin plugins/checks from salt and format the output as data.

salt.modules.munin.**list_plugins**()
List all the munin plugins

CLI Example:

```
    salt '*' munin.list_plugins
```

salt.modules.munin.**run**(*plugins*)
Run one or more named munin plugins

CLI Example:

```
    salt '*' munin.run uptime
    salt '*' munin.run uptime,cpu,load,memory
```

salt.modules.munin.**run_all**()
Run all the munin plugins

CLI Example:

```
    salt '*' munin.run_all
```

22.16.139 salt.modules.mysql

Module to provide MySQL compatibility to salt.

depends

- MySQLdb Python module

Note: On CentOS 5 (and possibly RHEL 5) both MySQL-python and python26-mysqldb need to be installed.

configuration In order to connect to MySQL, certain configuration is required in /etc/salt/minion on the relevant minions. Some sample configs might look like:

```
        mysql.host: 'localhost'
        mysql.port: 3306
        mysql.user: 'root'
        mysql.pass: ''
        mysql.db: 'mysql'
        mysql.unix_socket: '/tmp/mysql.sock'
        mysql.charset: 'utf8'
```

You can also use a defaults file:

```
        mysql.default_file: '/etc/mysql/debian.cnf'
```

Changed in version 2014.1.0: charset connection argument added. This is a MySQL charset, not a python one

Changed in version 0.16.2: Connection arguments from the minion config file can be overridden on the CLI by using the arguments defined *here*. Additionally, it is now possible to setup a user with no password.

`salt.modules.mysql.`**`db_check`**(*name*, *table=None*, ***connection_args*)

Repairs the full database or just a given table

CLI Example:

```
    salt '*' mysql.db_check dbname
    salt '*' mysql.db_check dbname dbtable
```

`salt.modules.mysql.`**`db_create`**(*name*, *character_set=None*, *collate=None*, ***connection_args*)

Adds a databases to the MySQL server.

name The name of the database to manage

character_set The character set, if left empty the MySQL default will be used

collate The collation, if left empty the MySQL default will be used

CLI Example:

```
    salt '*' mysql.db_create 'dbname'
    salt '*' mysql.db_create 'dbname' 'utf8' 'utf8_general_ci'
```

`salt.modules.mysql.`**`db_exists`**(*name*, ***connection_args*)

Checks if a database exists on the MySQL server.

CLI Example:

```
    salt '*' mysql.db_exists 'dbname'
```

`salt.modules.mysql.`**`db_list`**(***connection_args*)

Return a list of databases of a MySQL server using the output from the `SHOW DATABASES` query.

CLI Example:

```
    salt '*' mysql.db_list
```

`salt.modules.mysql.`**`db_optimize`**(*name*, *table=None*, ***connection_args*)

Optimizes the full database or just a given table

CLI Example:

```
    salt '*' mysql.db_optimize dbname
```

`salt.modules.mysql.`**`db_remove`**(*name*, ***connection_args*)

Removes a databases from the MySQL server.

CLI Example:

```
salt '*' mysql.db_remove 'dbname'
```

salt.modules.mysql.**db_repair**(*name*, *table=None*, ***connection_args*)

Repairs the full database or just a given table

CLI Example:

```
salt '*' mysql.db_repair dbname
```

salt.modules.mysql.**db_tables**(*name*, ***connection_args*)

Shows the tables in the given MySQL database (if exists)

CLI Example:

```
salt '*' mysql.db_tables 'database'
```

salt.modules.mysql.**free_slave**(***connection_args*)

Frees a slave from its master. This is a WIP, do not use.

CLI Example:

```
salt '*' mysql.free_slave
```

salt.modules.mysql.**get_master_status**(***connection_args*)

Retrieves the master status from the minion.

Returns:

{`host.domain.com': {`Binlog_Do_DB': `', `Binlog_Ignore_DB': `', `File': `mysql-bin.000021', `Position': 107}}

CLI Example:

```
salt '*' mysql.get_master_status
```

salt.modules.mysql.**get_slave_status**(***connection_args*)

Retrieves the slave status from the minion.

Returns:

```
{'host.domain.com': {'Connect_Retry': 60,
                     'Exec_Master_Log_Pos': 107,
                     'Last_Errno': 0,
                     'Last_Error': '',
                     'Last_IO_Errno': 0,
                     'Last_IO_Error': '',
                     'Last_SQL_Errno': 0,
                     'Last_SQL_Error': '',
                     'Master_Host': 'comet.scion-eng.com',
                     'Master_Log_File': 'mysql-bin.000021',
                     'Master_Port': 3306,
                     'Master_SSL_Allowed': 'No',
                     'Master_SSL_CA_File': '',
                     'Master_SSL_CA_Path': '',
                     'Master_SSL_Cert': '',
                     'Master_SSL_Cipher': '',
                     'Master_SSL_Key': '',
                     'Master_SSL_Verify_Server_Cert': 'No',
                     'Master_Server_Id': 1,
                     'Master_User': 'replu',
```

```
                    'Read_Master_Log_Pos': 107,
                    'Relay_Log_File': 'klo-relay-bin.000071',
                    'Relay_Log_Pos': 253,
                    'Relay_Log_Space': 553,
                    'Relay_Master_Log_File': 'mysql-bin.000021',
                    'Replicate_Do_DB': '',
                    'Replicate_Do_Table': '',
                    'Replicate_Ignore_DB': '',
                    'Replicate_Ignore_Server_Ids': '',
                    'Replicate_Ignore_Table': '',
                    'Replicate_Wild_Do_Table': '',
                    'Replicate_Wild_Ignore_Table': '',
                    'Seconds_Behind_Master': 0,
                    'Skip_Counter': 0,
                    'Slave_IO_Running': 'Yes',
                    'Slave_IO_State': 'Waiting for master to send event',
                    'Slave_SQL_Running': 'Yes',
                    'Until_Condition': 'None',
                    'Until_Log_File': '',
                    'Until_Log_Pos': 0}}
```

CLI Example:

```
salt '*' mysql.get_slave_status
```

salt.modules.mysql.**grant_add**(*grant*, *database*, *user*, *host='localhost'*, *grant_option=False*, *escape=True*, *ssl_option=False*, ***connection_args*)

Adds a grant to the MySQL server.

For database, make sure you specify database.table or database.*

CLI Example:

```
salt '*' mysql.grant_add 'SELECT,INSERT,UPDATE,...' 'database.*' 'frank' 'localhost'
```

salt.modules.mysql.**grant_exists**(*grant*, *database*, *user*, *host='localhost'*, *grant_option=False*, *escape=True*, ***connection_args*)

Checks to see if a grant exists in the database

CLI Example:

```
salt '*' mysql.grant_exists 'SELECT,INSERT,UPDATE,...' 'database.*' 'frank' 'localh
```

salt.modules.mysql.**grant_revoke**(*grant*, *database*, *user*, *host='localhost'*, *grant_option=False*, *escape=True*, ***connection_args*)

Removes a grant from the MySQL server.

CLI Example:

```
salt '*' mysql.grant_revoke 'SELECT,INSERT,UPDATE' 'database.*' 'frank' 'localhost'
```

salt.modules.mysql.**processlist**(***connection_args*)

Retrieves the processlist from the MySQL server via ``SHOW FULL PROCESSLIST''.

Returns: a list of dicts, with each dict representing a process:

{`Command': `Query', `Host': `localhost', `Id': 39, `Info': `SHOW FULL PROCESSLIST', `Rows_examined': 0, `Rows_read': 1, `Rows_sent': 0, `State': None, `Time': 0, `User': `root', `db': `mysql'}

CLI Example:

```
salt '*' mysql.processlist
```

salt.modules.mysql.**query**(*database*, *query*, ***connection_args*)

Run an arbitrary SQL query and return the results or the number of affected rows.

CLI Example:

```
salt '*' mysql.query mydb "UPDATE mytable set myfield=1 limit 1"
```

Return data:

```
{'query time': {'human': '39.0ms', 'raw': '0.03899'}, 'rows affected': 1L}
```

CLI Example:

```
salt '*' mysql.query mydb "SELECT id,name,cash from users limit 3"
```

Return data:

```
{'columns': ('id', 'name', 'cash'),
    'query time': {'human': '1.0ms', 'raw': '0.001'},
    'results': ((1L, 'User 1', Decimal('110.000000')),
                (2L, 'User 2', Decimal('215.636756')),
                (3L, 'User 3', Decimal('0.040000'))),
    'rows returned': 3L}
```

CLI Example:

```
salt '*' mysql.query mydb 'INSERT into users values (null,"user 4", 5)'
```

Return data:

```
{'query time': {'human': '25.6ms', 'raw': '0.02563'}, 'rows affected': 1L}
```

CLI Example:

```
salt '*' mysql.query mydb 'DELETE from users where id = 4 limit 1'
```

Return data:

```
{'query time': {'human': '39.0ms', 'raw': '0.03899'}, 'rows affected': 1L}
```

Jinja Example: Run a query on `mydb` and use row 0, column 0's data.

```
{{ salt['mysql.query']('mydb', 'SELECT info from mytable limit 1')['results'][0][0] }}
```

salt.modules.mysql.**quote_identifier**(*identifier*, *for_grants=False*)

Return an identifier name (column, table, database, etc) escaped for MySQL

This means surrounded by ```" character and escaping this character inside. It also means doubling the `%' character for MySQLdb internal usage.

Parameters

- **identifier** -- the table, column or database identifier
- **for_grants** -- is False by default, when using database names on grant queries you should set it to True to also escape ``_" and ``%" characters as requested by MySQL. Note that theses characters should only be escaped when requesting grants on the database level (*my_%db.**) but not for table level grants (*my_%db.`foo`*)

CLI Example:

```
salt '*' mysql.quote_identifier 'foo`bar'
```

salt.modules.mysql.**showglobal**(***connection_args*)

Retrieves the show global variables from the minion.

Returns:: show global variables full dict

CLI Example:

```
salt '*' mysql.showglobal
```

salt.modules.mysql.**showvariables**(***connection_args*)

Retrieves the show variables from the minion.

Returns:: show variables full dict

CLI Example:

```
salt '*' mysql.showvariables
```

salt.modules.mysql.**slave_lag**(***connection_args*)

Return the number of seconds that a slave SQL server is lagging behind the master, if the host is not a slave it will return -1. If the server is configured to be a slave for replication but slave IO is not running then -2 will be returned. If there was an error connecting to the database or checking the slave status, -3 will be returned.

CLI Example:

```
salt '*' mysql.slave_lag
```

salt.modules.mysql.**status**(***connection_args*)

Return the status of a MySQL server using the output from the `SHOW STATUS` query.

CLI Example:

```
salt '*' mysql.status
```

salt.modules.mysql.**tokenize_grant**(*grant*)

External wrapper function :param grant: :return: dict

CLI Example:

```
salt '*' mysql.tokenize_grant             "GRANT SELECT, INSERT ON testdb.* TO 'testuser'@'local
```

salt.modules.mysql.**user_chpass**(*user*, *host='localhost'*, *password=None*, *password_hash=None*, *allow_passwordless=False*, *unix_socket=None*, ***connection_args*)

Change password for a MySQL user

host Host for which this user/password combo applies

password The password to set for the new user. Will take precedence over the `password_hash` option if both are specified.

password_hash The password in hashed form. Be sure to quote the password because YAML doesn't like the `*`. A password hash can be obtained from the mysql command-line client like so:

```
mysql> SELECT PASSWORD('mypass');
+-------------------------------------------+
| PASSWORD('mypass')                        |
+-------------------------------------------+
| *6C8989366EAF75BB670AD8EA7A7FC1176A95CEF4 |
```

```
        +-------------------------------------------+
        1 row in set (0.00 sec)
```

allow_passwordless If `True`, then `password` and `password_hash` can be omitted (or set to `None`) to permit a passwordless login.

New in version 0.16.2: The `allow_passwordless` option was added.

CLI Examples:

```
salt '*' mysql.user_chpass frank localhost newpassword
salt '*' mysql.user_chpass frank localhost password_hash='hash'
salt '*' mysql.user_chpass frank localhost allow_passwordless=True
```

`salt.modules.mysql.`**`user_create`**(*user*, *host='localhost'*, *password=None*, *password_hash=None*, *allow_passwordless=False*, *unix_socket=False*, ***connection_args*)

Creates a MySQL user

host Host for which this user/password combo applies

password The password to use for the new user. Will take precedence over the `password_hash` option if both are specified.

password_hash The password in hashed form. Be sure to quote the password because YAML doesn't like the `*`. A password hash can be obtained from the mysql command-line client like so:

```
        mysql> SELECT PASSWORD('mypass');
        +-------------------------------------------+
        | PASSWORD('mypass')                        |
        +-------------------------------------------+
        | *6C8989366EAF75BB670AD8EA7A7FC1176A95CEF4 |
        +-------------------------------------------+
        1 row in set (0.00 sec)
```

allow_passwordless If `True`, then `password` and `password_hash` can be omitted (or set to `None`) to permit a passwordless login.

unix_socket If `True` and allow_passwordless is `True` then will be used unix_socket auth plugin.

New in version 0.16.2: The `allow_passwordless` option was added.

CLI Examples:

```
salt '*' mysql.user_create 'username' 'hostname' 'password'
salt '*' mysql.user_create 'username' 'hostname' password_hash='hash'
salt '*' mysql.user_create 'username' 'hostname' allow_passwordless=True
```

`salt.modules.mysql.`**`user_exists`**(*user*, *host='localhost'*, *password=None*, *password_hash=None*, *passwordless=False*, *unix_socket=False*, ***connection_args*)

Checks if a user exists on the MySQL server. A login can be checked to see if passwordless login is permitted by omitting `password` and `password_hash`, and using `passwordless=True`.

New in version 0.16.2: The `passwordless` option was added.

CLI Example:

```
salt '*' mysql.user_exists 'username' 'hostname' 'password'
salt '*' mysql.user_exists 'username' 'hostname' password_hash='hash'
salt '*' mysql.user_exists 'username' passwordless=True
```

`salt.modules.mysql.`**`user_grants`**(*user*, *host='localhost'*, ***connection_args*)

Shows the grants for the given MySQL user (if it exists)

CLI Example:

```
salt '*' mysql.user_grants 'frank' 'localhost'
```

salt.modules.mysql.**user_info**(*user*, *host='localhost'*, ***connection_args*)
Get full info on a MySQL user

CLI Example:

```
salt '*' mysql.user_info root localhost
```

salt.modules.mysql.**user_list**(***connection_args*)
Return a list of users on a MySQL server

CLI Example:

```
salt '*' mysql.user_list
```

salt.modules.mysql.**user_remove**(*user*, *host='localhost'*, ***connection_args*)
Delete MySQL user

CLI Example:

```
salt '*' mysql.user_remove frank localhost
```

salt.modules.mysql.**version**(***connection_args*)
Return the version of a MySQL server using the output from the `SELECT VERSION()` query.

CLI Example:

```
salt '*' mysql.version
```

22.16.140 salt.modules.nagios

Run nagios plugins/checks from salt and get the return as data.

salt.modules.nagios.**list_plugins**()
List all the nagios plugins

CLI Example:

```
salt '*' nagios.list_plugins
```

salt.modules.nagios.**retcode**(*plugin*, *args=''*, *key_name=None*)
Run one nagios plugin and return retcode of the execution

CLI Example:

```
salt '*' nagios.run check_apt
salt '*' nagios.run check_icmp '8.8.8.8'
```

salt.modules.nagios.**retcode_pillar**(*pillar_name*)
Run one or more nagios plugins from pillar data and get the result of cmd.retcode The pillar have to be in this format:

```
------
webserver:
    Ping_google:
        - check_icmp: 8.8.8.8
        - check_icmp: google.com
```

```
        Load:
            - check_load: -w 0.8 -c 1
        APT:
            - check_apt
    -------
```

webserver is the role to check, the next keys are the group and the items the check with the arguments if needed

You must to group different checks(one o more) and always it will return the highest value of all the checks

CLI Example:

```
    salt '*' nagios.retcode webserver
```

salt.modules.nagios.**run**(*plugin*, *args=''*)
Run nagios plugin and return all the data execution with cmd.run

salt.modules.nagios.**run_all**(*plugin*, *args=''*)
Run nagios plugin and return all the data execution with cmd.run_all

salt.modules.nagios.**run_all_pillar**(*pillar_name*)
Run one or more nagios plugins from pillar data and get the result of cmd.run_all The pillar have to be in this format:

```
    ------
    webserver:
        Ping_google:
            - check_icmp: 8.8.8.8
            - check_icmp: google.com
        Load:
            - check_load: -w 0.8 -c 1
        APT:
            - check_apt
    -------
```

webserver is the role to check, the next keys are the group and the items the check with the arguments if needed

You have to group different checks in a group

CLI Example:

```
    salt '*' nagios.run webserver
```

salt.modules.nagios.**run_pillar**(*pillar_name*)
Run one or more nagios plugins from pillar data and get the result of cmd.run The pillar have to be in this format:

```
    ------
    webserver:
        Ping_google:
            - check_icmp: 8.8.8.8
            - check_icmp: google.com
        Load:
            - check_load: -w 0.8 -c 1
        APT:
            - check_apt
    -------
```

webserver is the role to check, the next keys are the group and the items the check with the arguments if needed

You have to group different checks in a group

CLI Example:

```
salt '*' nagios.run webserver
```

22.16.141 salt.modules.netbsd_sysctl

Module for viewing and modifying sysctl parameters

salt.modules.netbsd_sysctl.**assign**(*name*, *value*)
Assign a single sysctl parameter for this minion

CLI Example:

```
salt '*' sysctl.assign net.inet.icmp.icmplim 50
```

salt.modules.netbsd_sysctl.**get**(*name*)
Return a single sysctl parameter for this minion

CLI Example:

```
salt '*' sysctl.get hw.physmem
```

salt.modules.netbsd_sysctl.**persist**(*name*, *value*, *config='/etc/sysctl.conf'*)
Assign and persist a simple sysctl parameter for this minion

CLI Example:

```
salt '*' sysctl.persist net.inet.icmp.icmplim 50
```

salt.modules.netbsd_sysctl.**show**(*config_file=False*)
Return a list of sysctl parameters for this minion

CLI Example:

```
salt '*' sysctl.show
```

22.16.142 salt.modules.netbsdservice

The service module for NetBSD

salt.modules.netbsdservice.**available**(*name*)
Returns `True` if the specified service is available, otherwise returns `False`.

CLI Example:

```
salt '*' service.available sshd
```

salt.modules.netbsdservice.**disable**(*name*, ***kwargs*)
Disable the named service to start at boot

CLI Example:

```
salt '*' service.disable <service name>
```

salt.modules.netbsdservice.**disabled**(*name*)
Return True if the named service is enabled, false otherwise

CLI Example:

```
salt '*' service.disabled <service name>
```

salt.modules.netbsdservice.**enable**(*name*, ***kwargs*)
Enable the named service to start at boot

CLI Example:

```
salt '*' service.enable <service name>
```

salt.modules.netbsdservice.**enabled**(*name*, ***kwargs*)
Return True if the named service is enabled, false otherwise

CLI Example:

```
salt '*' service.enabled <service name>
```

salt.modules.netbsdservice.**force_reload**(*name*)
Force-reload the named service

CLI Example:

```
salt '*' service.force_reload <service name>
```

salt.modules.netbsdservice.**get_all**()
Return all available boot services

CLI Example:

```
salt '*' service.get_all
```

salt.modules.netbsdservice.**get_disabled**()
Return a set of services that are installed but disabled

CLI Example:

```
salt '*' service.get_disabled
```

salt.modules.netbsdservice.**get_enabled**()
Return a list of service that are enabled on boot

CLI Example:

```
salt '*' service.get_enabled
```

salt.modules.netbsdservice.**missing**(*name*)
The inverse of service.available. Returns `True` if the specified service is not available, otherwise returns `False`.

CLI Example:

```
salt '*' service.missing sshd
```

salt.modules.netbsdservice.**reload_**(*name*)
Reload the named service

CLI Example:

```
salt '*' service.reload <service name>
```

salt.modules.netbsdservice.**restart**(*name*)
Restart the named service

CLI Example:

```
salt '*' service.restart <service name>
```

salt.modules.netbsdservice.**start**(*name*)
Start the specified service

CLI Example:

```
salt '*' service.start <service name>
```

salt.modules.netbsdservice.**status**(*name*, *sig=None*)
Return the status for a service, returns a bool whether the service is running.

CLI Example:

```
salt '*' service.status <service name>
```

salt.modules.netbsdservice.**stop**(*name*)
Stop the specified service

CLI Example:

```
salt '*' service.stop <service name>
```

22.16.143 salt.modules.netscaler

salt.modules.netscaler.**server_add**(*s_name*, *s_ip*, *s_state=None*, ***connection_args*)
Add a server Note: The default server state is ENABLED

CLI Example:

```
salt '*' netscaler.server_add 'serverName' 'serverIpAddress'
salt '*' netscaler.server_add 'serverName' 'serverIpAddress' 'serverState'
```

salt.modules.netscaler.**server_delete**(*s_name*, ***connection_args*)
Delete a server

CLI Example:

```
salt '*' netscaler.server_delete 'serverName'
```

salt.modules.netscaler.**server_disable**(*s_name*, ***connection_args*)
Disable a server globally

CLI Example:

```
salt '*' netscaler.server_disable 'serverName'
```

salt.modules.netscaler.**server_enable**(*s_name*, ***connection_args*)
Enables a server globally

CLI Example:

```
salt '*' netscaler.server_enable 'serverName'
```

salt.modules.netscaler.**server_enabled**(*s_name*, ***connection_args*)
Check if a server is enabled globally

CLI Example:

```
salt '*' netscaler.server_enabled 'serverName'
```

salt.modules.netscaler.**server_exists**(*s_name*, *ip=None*, *s_state=None*, ***connection_args*)
Checks if a server exists

CLI Example:

```
salt '*' netscaler.server_exists 'serverName'
```

salt.modules.netscaler.**server_update**(*s_name*, *s_ip*, ***connection_args*)
Update a server's attributes

CLI Example:

```
salt '*' netscaler.server_update 'serverName' 'serverIP'
```

salt.modules.netscaler.**service_disable**(*s_name*, *s_delay=None*, ***connection_args*)
Disable a service

CLI Example:

```
salt '*' netscaler.service_disable 'serviceName'
salt '*' netscaler.service_disable 'serviceName' 'delayInSeconds'
```

salt.modules.netscaler.**service_enable**(*s_name*, ***connection_args*)
Enable a service

CLI Example:

```
salt '*' netscaler.service_enable 'serviceName'
```

salt.modules.netscaler.**service_exists**(*s_name*, ***connection_args*)
Checks if a service exists

CLI Example:

```
salt '*' netscaler.service_exists 'serviceName'
```

salt.modules.netscaler.**service_up**(*s_name*, ***connection_args*)
Checks if a service is UP

CLI Example:

```
salt '*' netscaler.service_up 'serviceName'
```

salt.modules.netscaler.**servicegroup_add**(*sg_name*, *sg_type='HTTP'*, ***connection_args*)
Add a new service group If no service type is specified, HTTP will be used. Most common service types: HTTP, SSL, and SSL_BRIDGE

CLI Example:

```
salt '*' netscaler.servicegroup_add 'serviceGroupName'
salt '*' netscaler.servicegroup_add 'serviceGroupName' 'serviceGroupType'
```

salt.modules.netscaler.**servicegroup_delete**(*sg_name*, ***connection_args*)
Delete a new service group

CLI Example:

```
salt '*' netscaler.servicegroup_delete 'serviceGroupName'
```

salt.modules.netscaler.**servicegroup_exists**(*sg_name*, *sg_type=None*, ***connection_args*)
Checks if a service group exists

CLI Example:

```
salt '*' netscaler.servicegroup_exists 'serviceGroupName'
```

salt.modules.netscaler.**servicegroup_server_add**(*sg_name*, *s_name*, *s_port*, ***connection_args*)

Add a server:port member to a servicegroup

CLI Example:

```
salt '*' netscaler.servicegroup_server_add 'serviceGroupName' 'serverName' 'serverPort'
```

salt.modules.netscaler.**servicegroup_server_delete**(*sg_name*, *s_name*, *s_port*, ***connection_args*)

Remove a server:port member from a servicegroup

CLI Example:

```
salt '*' netscaler.servicegroup_server_delete 'serviceGroupName' 'serverName' 'serverPort'
```

salt.modules.netscaler.**servicegroup_server_disable**(*sg_name*, *s_name*, *s_port*, ***connection_args*)

Disable a server:port member of a servicegroup

CLI Example:

```
salt '*' netscaler.servicegroup_server_disable 'serviceGroupName' 'serverName' 'serverPort'
```

salt.modules.netscaler.**servicegroup_server_enable**(*sg_name*, *s_name*, *s_port*, ***connection_args*)

Enable a server:port member of a servicegroup

CLI Example:

```
salt '*' netscaler.servicegroup_server_enable 'serviceGroupName' 'serverName' 'serverPort'
```

salt.modules.netscaler.**servicegroup_server_exists**(*sg_name*, *s_name*, *s_port=None*, ***connection_args*)

Check if a server:port combination is a member of a servicegroup

CLI Example:

```
salt '*' netscaler.servicegroup_server_exists 'serviceGroupName' 'serverName' 'serverPort'
```

salt.modules.netscaler.**servicegroup_server_up**(*sg_name*, *s_name*, *s_port*, ***connection_args*)

Check if a server:port combination is in state UP in a servicegroup

CLI Example:

```
salt '*' netscaler.servicegroup_server_up 'serviceGroupName' 'serverName' 'serverPort'
```

salt.modules.netscaler.**vserver_add**(*v_name*, *v_ip*, *v_port*, *v_type*, ***connection_args*)

Add a new lb vserver

CLI Example:

```
salt '*' netscaler.vserver_add 'vserverName' 'vserverIP' 'vserverPort' 'vserverType'
salt '*' netscaler.vserver_add 'alex.patate.chaude.443' '1.2.3.4' '443' 'SSL'
```

salt.modules.netscaler.**vserver_delete**(*v_name*, ***connection_args*)

Delete a lb vserver

CLI Example:

```
salt '*' netscaler.vserver_delete 'vserverName'
```

salt.modules.netscaler.**vserver_exists**(*v_name*, *v_ip=None*, *v_port=None*, *v_type=None*, ***connection_args*)

Checks if a vserver exists

CLI Example:

```
salt '*' netscaler.vserver_exists 'vserverName'
```

salt.modules.netscaler.**vserver_servicegroup_add**(*v_name*, *sg_name*, ***connection_args*)

Bind a servicegroup to a vserver

CLI Example:

```
salt '*' netscaler.vserver_servicegroup_add 'vserverName' 'serviceGroupName'
```

salt.modules.netscaler.**vserver_servicegroup_delete**(*v_name*, *sg_name*, ***connection_args*)

Unbind a servicegroup from a vserver

CLI Example:

```
salt '*' netscaler.vserver_servicegroup_delete 'vserverName' 'serviceGroupName'
```

salt.modules.netscaler.**vserver_servicegroup_exists**(*v_name*, *sg_name*, ***connection_args*)

Checks if a servicegroup is tied to a vserver

CLI Example:

```
salt '*' netscaler.vserver_servicegroup_exists 'vserverName' 'serviceGroupName'
```

salt.modules.netscaler.**vserver_sslcert_add**(*v_name*, *sc_name*, ***connection_args*)

Binds a SSL certificate to a vserver

CLI Example:

```
salt '*' netscaler.vserver_sslcert_add 'vserverName' 'sslCertificateName'
```

salt.modules.netscaler.**vserver_sslcert_delete**(*v_name*, *sc_name*, ***connection_args*)

Unbinds a SSL certificate from a vserver

CLI Example:

```
salt '*' netscaler.vserver_sslcert_delete 'vserverName' 'sslCertificateName'
```

salt.modules.netscaler.**vserver_sslcert_exists**(*v_name*, *sc_name*, ***connection_args*)

Checks if a SSL certificate is tied to a vserver

CLI Example:

```
salt '*' netscaler.vserver_sslcert_exists 'vserverName' 'sslCertificateName'
```

22.16.144 salt.modules.network

Module for gathering and managing network information

salt.modules.network.**active_tcp**()

Return a dict containing information on all of the running TCP connections

CLI Example:

```
salt '*' network.active_tcp
```

salt.modules.network.**arp**()
Return the arp table from the minion

CLI Example:

```
salt '*' network.arp
```

salt.modules.network.**connect**(*host*, *port=None*, ***kwargs*)
Test connectivity to a host using a particular port from the minion.

New in version 2014.7.

CLI Example:

```
salt '*' network.connect archlinux.org 80

salt '*' network.connect archlinux.org 80 timeout=3

salt '*' network.connect archlinux.org 80 timeout=3 family=ipv4

salt '*' network.connect google-public-dns-a.google.com port=53 proto=udp timeout=3
```

salt.modules.network.**default_route**(*family=None*)
Return default route(s) from routing table

CLI Example:

```
salt '*' network.default_route
```

salt.modules.network.**dig**(*host*)
Performs a DNS lookup with dig

CLI Example:

```
salt '*' network.dig archlinux.org
```

salt.modules.network.**get_bufsize**(*iface*)
Return network buffer sizes as a dict

CLI Example:

```
salt '*' network.getbufsize
```

salt.modules.network.**get_hostname**()
Get hostname

CLI Example:

```
salt '*' network.get_hostname
```

salt.modules.network.**hw_addr**(*iface*)
Return the hardware address (a.k.a. MAC address) for a given interface

CLI Example:

```
salt '*' network.hw_addr eth0
```

salt.modules.network.**hwaddr**(*iface*)
Return the hardware address (a.k.a. MAC address) for a given interface

CLI Example:

```
salt '*' network.hw_addr eth0
```

salt.modules.network.**in_subnet**(*cidr*)
Returns True if host is within specified subnet, otherwise False.

CLI Example:

```
salt '*' network.in_subnet 10.0.0.0/16
```

salt.modules.network.**interface**(*iface*)
Return the inet address for a given interface

New in version 2014.7.

CLI Example:

```
salt '*' network.interface eth0
```

salt.modules.network.**interface_ip**(*iface*)
Return the inet address for a given interface

New in version 2014.7.

CLI Example:

```
salt '*' network.interface_ip eth0
```

salt.modules.network.**interfaces**()
Return a dictionary of information about all the interfaces on the minion

CLI Example:

```
salt '*' network.interfaces
```

salt.modules.network.**ip_addrs**(*interface=None*, *include_loopback=False*, *cidr=None*)
Returns a list of IPv4 addresses assigned to the host. 127.0.0.1 is ignored, unless `include_loopback=True' is indicated. If `interface' is provided, then only IP addresses from that interface will be returned. Providing a CIDR via `cidr="10.0.0.0/8"' will return only the addresses which are within that subnet.

CLI Example:

```
salt '*' network.ip_addrs
```

salt.modules.network.**ip_addrs6**(*interface=None*, *include_loopback=False*)
Returns a list of IPv6 addresses assigned to the host. ::1 is ignored, unless `include_loopback=True' is indicated. If `interface' is provided, then only IP addresses from that interface will be returned.

CLI Example:

```
salt '*' network.ip_addrs6
```

salt.modules.network.**ip_in_subnet**(*ip_addr*, *cidr*)
Returns True if given IP is within specified subnet, otherwise False.

CLI Example:

```
salt '*' network.ip_in_subnet 172.17.0.4 172.16.0.0/12
```

salt.modules.network.**ipaddrs**(*interface=None*, *include_loopback=False*, *cidr=None*)
Returns a list of IPv4 addresses assigned to the host. 127.0.0.1 is ignored, unless `include_loopback=True' is indicated. If `interface' is provided, then only IP addresses from that interface will be returned. Providing a CIDR via `cidr="10.0.0.0/8"' will return only the addresses which are within that subnet.

CLI Example:

```
salt '*' network.ip_addrs
```

salt.modules.network.**ipaddrs6**(*interface=None*, *include_loopback=False*)
Returns a list of IPv6 addresses assigned to the host. ::1 is ignored, unless `include_loopback=True' is indicated. If `interface' is provided, then only IP addresses from that interface will be returned.

CLI Example:

```
salt '*' network.ip_addrs6
```

salt.modules.network.**is_loopback**(*ip_addr*)
Check if the given IP address is a loopback address

New in version 2014.7.0.

CLI Example:

```
salt '*' network.is_loopback 127.0.0.1
```

salt.modules.network.**is_private**(*ip_addr*)
Check if the given IP address is a private address

New in version 2014.7.0.

CLI Example:

```
salt '*' network.is_private 10.0.0.3
```

salt.modules.network.**mod_bufsize**(*iface*, **args*, ***kwargs*)
Modify network interface buffers (currently linux only)

CLI Example:

```
salt '*' network.getBuffers
```

salt.modules.network.**mod_hostname**(*hostname*)
Modify hostname

CLI Example:

```
salt '*' network.mod_hostname   master.saltstack.com
```

salt.modules.network.**netstat**()
Return information on open ports and states

Note: On BSD minions, the output contains PID info (where available) for each netstat entry, fetched from sockstat/fstat output.

Changed in version 2014.1.4: Added support for OpenBSD, FreeBSD, and NetBSD

CLI Example:

```
salt '*' network.netstat
```

salt.modules.network.**ping**(*host*, *timeout=False*, *return_boolean=False*)
Performs a ping to a host

CLI Example:

```
salt '*' network.ping archlinux.org
```

New in version 2015.5.0.

Return a True or False instead of ping output.

salt `*' network.ping archlinux.org return_boolean=True

Set the time to wait for a response in seconds.

salt `*' network.ping archlinux.org timeout=3

salt.modules.network.**reverse_ip**(*ip_addr*)
Returns the reversed IP address

CLI Example:

```
salt '*' network.reverse_ip 172.17.0.4
```

salt.modules.network.**routes**(*family=None*)
Return currently configured routes from routing table

CLI Example:

```
salt '*' network.routes
```

salt.modules.network.**subnets**()
Returns a list of subnets to which the host belongs

CLI Example:

```
salt '*' network.subnets
```

salt.modules.network.**traceroute**(*host*)
Performs a traceroute to a 3rd party host

CLI Example:

```
salt '*' network.traceroute archlinux.org
```

salt.modules.network.**wol**(*mac*, *bcast=`255.255.255.255'*, *destport=9*)
Send Wake On Lan packet to a host

CLI Example:

```
salt '*' network.wol 08-00-27-13-69-77
salt '*' network.wol 080027136977 255.255.255.255 7
salt '*' network.wol 08:00:27:13:69:77 255.255.255.255 7
```

22.16.145 salt.modules.neutron

Module for handling OpenStack Neutron calls

depends

- neutronclient Python module

configuration This module is not usable until the user, password, tenant, and auth URL are specified either in a pillar or in the minion's config file. For example:

```
keystone.user: 'admin'
keystone.password: 'password'
keystone.tenant: 'admin'
keystone.auth_url: 'http://127.0.0.1:5000/v2.0/'
keystone.region_name: 'RegionOne'
keystone.service_type: 'network'
```

If configuration for multiple OpenStack accounts is required, they can be set up as different configuration profiles: For example:

```
openstack1:
  keystone.user: 'admin'
  keystone.password: 'password'
  keystone.tenant: 'admin'
  keystone.auth_url: 'http://127.0.0.1:5000/v2.0/'
  keystone.region_name: 'RegionOne'
  keystone.service_type: 'network'

openstack2:
  keystone.user: 'admin'
  keystone.password: 'password'
  keystone.tenant: 'admin'
  keystone.auth_url: 'http://127.0.0.2:5000/v2.0/'
  keystone.region_name: 'RegionOne'
  keystone.service_type: 'network'
```

With this configuration in place, any of the neutron functions can make use of a configuration profile by declaring it explicitly. For example:

```
salt '*' neutron.network_list profile=openstack1
```

`salt.modules.neutron.`**`add_gateway_router`**(*router*, *ext_network*, *profile=None*)

Adds an external network gateway to the specified router

CLI Example:

```
salt '*' neutron.add_gateway_router router-name ext-network-name
```

Parameters

- **`router`** -- ID or name of the router
- **`ext_network`** -- ID or name of the external network the gateway
- **`profile`** -- Profile to build on (Optional)

Returns Added Gateway router information

`salt.modules.neutron.`**`add_interface_router`**(*router*, *subnet*, *profile=None*)

Adds an internal network interface to the specified router

CLI Example:

```
salt '*' neutron.add_interface_router router-name subnet-name
```

Parameters

- **`router`** -- ID or name of the router

- **subnet** -- ID or name of the subnet
- **profile** -- Profile to build on (Optional)

Returns Added interface information

salt.modules.neutron.**create_floatingip**(*floating_network*, *port=None*, *profile=None*)

Creates a new floatingIP

CLI Example:

```
salt '*' neutron.create_floatingip network-name port-name
```

Parameters

- **floating_network** -- Network name or ID to allocate floatingIP from
- **port** -- Of the port to be associated with the floatingIP (Optional)
- **profile** -- Profile to build on (Optional)

Returns Created floatingIP information

salt.modules.neutron.**create_ikepolicy**(*name*, *profile=None*, ***kwargs*)

Creates a new IKEPolicy

CLI Example:

```
salt '*' neutron.create_ikepolicy ikepolicy-name
        phase1_negotiation_mode=main auth_algorithm=sha1
        encryption_algorithm=aes-128 pfs=group5
```

Parameters

- **name** -- Name of the IKE policy
- **phase1_negotiation_mode** -- IKE Phase1 negotiation mode in lowercase, default: main (Optional)
- **auth_algorithm** -- Authentication algorithm in lowercase, default: sha1 (Optional)
- **encryption_algorithm** -- Encryption algorithm in lowercase. default:aes-128 (Optional)
- **pfs** -- Prefect Forward Security in lowercase, default: group5 (Optional)
- **units** -- IKE lifetime attribute. default: seconds (Optional)
- **value** -- IKE lifetime attribute. default: 3600 (Optional)
- **ike_version** -- IKE version in lowercase, default: v1 (Optional)
- **profile** -- Profile to build on (Optional)
- **kwargs** --

Returns Created IKE policy information

salt.modules.neutron.**create_ipsec_site_connection**(*name*, *ipsecpolicy*, *ikepolicy*, *vpnservice*, *peer_cidrs*, *peer_address*, *peer_id*, *psk*, *admin_state_up=True*, *profile=None*, ***kwargs*)

Creates a new IPsecSiteConnection

CLI Example:

```
salt '*' neutron.show_ipsec_site_connection connection-name
        ipsec-policy-name ikepolicy-name vpnservice-name
        192.168.XXX.XXX/24 192.168.XXX.XXX 192.168.XXX.XXX secret
```

Parameters

- **name** -- Set friendly name for the connection
- **ipsecpolicy** -- IPSec policy ID or name associated with this connection
- **ikepolicy** -- IKE policy ID or name associated with this connection
- **vpnservice** -- VPN service instance ID or name associated with this connection
- **peer_cidrs** -- Remote subnet(s) in CIDR format
- **peer_address** -- Peer gateway public IPv4/IPv6 address or FQDN
- **peer_id** -- Peer router identity for authentication Can be IPv4/IPv6 address, e-mail address, key id, or FQDN
- **psk** -- Pre-shared key string
- **initiator** -- Initiator state in lowercase, default:bi-directional
- **admin_state_up** -- Set admin state up to true or false, default: True (Optional)
- **mtu** -- size for the connection, default:1500 (Optional)
- **dpd_action** -- Dead Peer Detection attribute: hold/clear/disabled/ restart/restart-by-peer (Optional)
- **dpd_interval** -- Dead Peer Detection attribute (Optional)
- **dpd_timeout** -- Dead Peer Detection attribute (Optional)
- **profile** -- Profile to build on (Optional)

Returns Created IPSec site connection information

`salt.modules.neutron.`**`create_ipsecpolicy`**(*name*, *profile=None*, ***kwargs*)

Creates a new IPsecPolicy

CLI Example:

```
salt '*' neutron.create_ipsecpolicy ipsecpolicy-name
        transform_protocol=esp auth_algorithm=sha1
        encapsulation_mode=tunnel encryption_algorithm=aes-128
```

Parameters

- **name** -- Name of the IPSec policy
- **transform_protocol** -- Transform protocol in lowercase, default: esp (Optional)
- **auth_algorithm** -- Authentication algorithm in lowercase, default: sha1 (Optional)
- **encapsulation_mode** -- Encapsulation mode in lowercase, default: tunnel (Optional)
- **encryption_algorithm** -- Encryption algorithm in lowercase, default:aes-128 (Optional)
- **pfs** -- Prefect Forward Security in lowercase, default: group5 (Optional)

- **units** -- IPSec lifetime attribute. default: seconds (Optional)
- **value** -- IPSec lifetime attribute. default: 3600 (Optional)
- **profile** -- Profile to build on (Optional)

Returns Created IPSec policy information

salt.modules.neutron.**create_network**(*name*, *router_ext=False*, *profile=None*)

Creates a new network

CLI Example:

```
salt '*' neutron.create_network network-name
salt '*' neutron.create_network network-name profile=openstack1
```

Parameters

- **name** -- Name of network to create
- **router_ext** -- True then if create the external network, default: False (Optional)
- **profile** -- Profile to build on (Optional)

Returns Created network information

salt.modules.neutron.**create_port**(*name*, *network*, *device_id=None*, *admin_state_up=True*, *profile=None*)

Creates a new port

CLI Example:

```
salt '*' neutron.create_port network-name port-name
```

Parameters

- **name** -- Name of port to create
- **network** -- Network name or ID
- **device_id** -- ID of device (Optional)
- **admin_state_up** -- Set admin state up to true or false, default: true (Optional)
- **profile** -- Profile to build on (Optional)

Returns Created port information

salt.modules.neutron.**create_router**(*name*, *ext_network=None*, *admin_state_up=True*, *profile=None*)

Creates a new router

CLI Example:

```
salt '*' neutron.create_router new-router-name
```

Parameters

- **name** -- Name of router to create (must be first)
- **ext_network** -- ID or name of the external for the gateway (Optional)
- **admin_state_up** -- Set admin state up to true or false, default:true (Optional)

- **profile** -- Profile to build on (Optional)

Returns Created router information

salt.modules.neutron.**create_security_group**(*name=None*, *description=None*, *profile=None*)

Creates a new security group

CLI Example:

```
salt '*' neutron.create_security_group security-group-name                    description='Security
```

Parameters

- **name** -- Name of security group (Optional)
- **description** -- Description of security group (Optional)
- **profile** -- Profile to build on (Optional)

Returns Created security group information

salt.modules.neutron.**create_security_group_rule**(*security_group*, *remote_group_id=None*, *direction='ingress'*, *protocol=None*, *port_range_min=None*, *port_range_max=None*, *ethertype='IPv4'*, *profile=None*)

Creates a new security group rule

CLI Example:

```
salt '*' neutron.show_security_group_rule security-group-rule-id
```

Parameters

- **security_group** -- Security group name or ID to add rule
- **remote_group_id** -- Remote security group name or ID to apply rule (Optional)
- **direction** -- Direction of traffic: ingress/egress, default: ingress (Optional)
- **protocol** -- Protocol of packet: null/icmp/tcp/udp, default: null (Optional)
- **port_range_min** -- Starting port range (Optional)
- **port_range_max** -- Ending port range (Optional)
- **ethertype** -- IPv4/IPv6, default: IPv4 (Optional)
- **profile** -- Profile to build on (Optional)

Returns Created security group rule information

salt.modules.neutron.**create_subnet**(*network*, *cidr*, *name=None*, *ip_version=4*, *profile=None*)

Creates a new subnet

CLI Example:

```
salt '*' neutron.create_subnet network-name 192.168.1.0/24
```

Parameters

- **network** -- Network ID or name this subnet belongs to

- **cidr** -- CIDR of subnet to create (Ex. `192.168.1.0/24')
- **name** -- Name of the subnet to create (Optional)
- **ip_version** -- Version to use, default is 4(IPv4) (Optional)
- **profile** -- Profile to build on (Optional)

Returns Created subnet information

salt.modules.neutron.**create_vpnservice**(*subnet*, *router*, *name*, *admin_state_up=True*, *profile=None*)

Creates a new VPN service

CLI Example:

```
salt '*' neutron.create_vpnservice router-name name
```

Parameters

- **subnet** -- Subnet unique identifier for the VPN service deployment
- **router** -- Router unique identifier for the VPN service
- **name** -- Set a name for the VPN service
- **admin_state_up** -- Set admin state up to true or false, default:True (Optional)
- **profile** -- Profile to build on (Optional)

Returns Created VPN service information

salt.modules.neutron.**delete_floatingip**(*floatingip_id*, *profile=None*)

Deletes the specified floating IP

CLI Example:

```
salt '*' neutron.delete_floatingip floatingip-id
```

Parameters

- **floatingip_id** -- ID of floatingIP to delete
- **profile** -- Profile to build on (Optional)

Returns True(Succeed) or False

salt.modules.neutron.**delete_ikepolicy**(*ikepolicy*, *profile=None*)

Deletes the specified IKEPolicy

CLI Example:

```
salt '*' neutron.delete_ikepolicy ikepolicy-name
```

Parameters

- **ikepolicy** -- ID or name of IKE policy to delete
- **profile** -- Profile to build on (Optional)

Returns True(Succeed) or False

salt.modules.neutron.**delete_ipsec_site_connection**(*ipsec_site_connection*, *profile=None*)

Deletes the specified IPsecSiteConnection

CLI Example:

```
salt '*' neutron.delete_ipsec_site_connection connection-name
```

Parameters

- **ipsec_site_connection** -- ID or name of ipsec site connection to delete
- **profile** -- Profile to build on (Optional)

Returns True(Succeed) or False

salt.modules.neutron.**delete_ipsecpolicy**(*ipsecpolicy*, *profile=None*)

Deletes the specified IPsecPolicy

CLI Example:

```
salt '*' neutron.delete_ipsecpolicy ipsecpolicy-name
```

Parameters

- **ipsecpolicy** -- ID or name of IPSec policy to delete
- **profile** -- Profile to build on (Optional)

Returns True(Succeed) or False

salt.modules.neutron.**delete_network**(*network*, *profile=None*)

Deletes the specified network

CLI Example:

```
salt '*' neutron.delete_network network-name
salt '*' neutron.delete_network network-name profile=openstack1
```

Parameters

- **network** -- ID or name of network to delete
- **profile** -- Profile to build on (Optional)

Returns True(Succeed) or False

salt.modules.neutron.**delete_port**(*port*, *profile=None*)

Deletes the specified port

CLI Example:

```
salt '*' neutron.delete_network port-name
salt '*' neutron.delete_network port-name profile=openstack1
```

Parameters

- **port** -- port name or ID
- **profile** -- Profile to build on (Optional)

Returns True(Succeed) or False

salt.modules.neutron.**delete_quota**(*tenant_id*, *profile=None*)
Delete the specified tenant's quota value

CLI Example:

```
salt '*' neutron.update_quota tenant-id
salt '*' neutron.update_quota tenant-id profile=openstack1
```

Parameters

- **tenant_id** -- ID of tenant to quota delete
- **profile** -- Profile to build on (Optional)

Returns True(Delete succeed) or False(Delete failed)

salt.modules.neutron.**delete_router**(*router*, *profile=None*)
Delete the specified router

CLI Example:

```
salt '*' neutron.delete_router router-name
```

Parameters

- **router** -- ID or name of router to delete
- **profile** -- Profile to build on (Optional)

Returns True(Succeed) or False

salt.modules.neutron.**delete_security_group**(*security_group*, *profile=None*)
Deletes the specified security group

CLI Example:

```
salt '*' neutron.delete_security_group security-group-name
```

Parameters

- **security_group** -- ID or name of security group to delete
- **profile** -- Profile to build on (Optional)

Returns True(Succeed) or False

salt.modules.neutron.**delete_security_group_rule**(*security_group_rule_id*, *profile=None*)
Deletes the specified security group rule

CLI Example:

```
salt '*' neutron.delete_security_group_rule security-group-rule-id
```

Parameters

- **security_group_rule_id** -- ID of security group rule to delete
- **profile** -- Profile to build on (Optional)

Returns True(Succeed) or False

salt.modules.neutron.**delete_subnet**(*subnet*, *profile=None*)
Deletes the specified subnet

CLI Example:

```
salt '*' neutron.delete_subnet subnet-name
salt '*' neutron.delete_subnet subnet-name profile=openstack1
```

Parameters

- **subnet** -- ID or name of subnet to delete
- **profile** -- Profile to build on (Optional)

Returns True(Succeed) or False

salt.modules.neutron.**delete_vpnservice**(*vpnservice*, *profile=None*)
Deletes the specified VPN service

CLI Example:

```
salt '*' neutron.delete_vpnservice vpnservice-name
```

Parameters

- **vpnservice** -- ID or name of vpn service to delete
- **profile** -- Profile to build on (Optional)

Returns True(Succeed) or False

salt.modules.neutron.**get_quotas_tenant**(*profile=None*)
Fetches tenant info in server's context for following quota operation

CLI Example:

```
salt '*' neutron.get_quotas_tenant
salt '*' neutron.get_quotas_tenant profile=openstack1
```

Parameters **profile** -- Profile to build on (Optional)

Returns Quotas information

salt.modules.neutron.**list_extensions**(*profile=None*)
Fetches a list of all extensions on server side

CLI Example:

```
salt '*' neutron.list_extensions
salt '*' neutron.list_extensions profile=openstack1
```

Parameters **profile** -- Profile to build on (Optional)

Returns List of extensions

salt.modules.neutron.**list_floatingips**(*profile=None*)
Fetch a list of all floatingIPs for a tenant

CLI Example:

```
    salt '*' neutron.list_floatingips
    salt '*' neutron.list_floatingips profile=openstack1
```

Parameters **profile** -- Profile to build on (Optional)

Returns List of floatingIP

salt.modules.neutron.**list_ikepolicies**(*profile=None*)
Fetches a list of all configured IKEPolicies for a tenant

CLI Example:

```
    salt '*' neutron.list_ikepolicies
    salt '*' neutron.list_ikepolicies profile=openstack1
```

Parameters **profile** -- Profile to build on (Optional)

Returns List of IKE policy

salt.modules.neutron.**list_ipsec_site_connections**(*profile=None*)
Fetches all configured IPsec Site Connections for a tenant

CLI Example:

```
    salt '*' neutron.list_ipsec_site_connections
    salt '*' neutron.list_ipsec_site_connections profile=openstack1
```

Parameters **profile** -- Profile to build on (Optional)

Returns List of IPSec site connection

salt.modules.neutron.**list_ipsecpolicies**(*profile=None*)
Fetches a list of all configured IPsecPolicies for a tenant

CLI Example:

```
    salt '*' neutron.list_ipsecpolicies ipsecpolicy-name
    salt '*' neutron.list_ipsecpolicies ipsecpolicy-name profile=openstack1
```

Parameters **profile** -- Profile to build on (Optional)

Returns List of IPSec policy

salt.modules.neutron.**list_networks**(*profile=None*)
Fetches a list of all networks for a tenant

CLI Example:

```
    salt '*' neutron.list_networks
    salt '*' neutron.list_networks profile=openstack1
```

Parameters **profile** -- Profile to build on (Optional)

Returns List of network

salt.modules.neutron.**list_ports**(*profile=None*)
Fetches a list of all networks for a tenant

CLI Example:

```
salt '*' neutron.list_ports
salt '*' neutron.list_ports profile=openstack1
```

Parameters **profile** -- Profile to build on (Optional)

Returns List of port

salt.modules.neutron.**list_quotas**(*profile=None*)
Fetches all tenants quotas

CLI Example:

```
salt '*' neutron.list_quotas
salt '*' neutron.list_quotas profile=openstack1
```

Parameters **profile** -- Profile to build on (Optional)

Returns List of quotas

salt.modules.neutron.**list_routers**(*profile=None*)
Fetches a list of all routers for a tenant

CLI Example:

```
salt '*' neutron.list_routers
salt '*' neutron.list_routers profile=openstack1
```

Parameters **profile** -- Profile to build on (Optional)

Returns List of router

salt.modules.neutron.**list_security_group_rules**(*profile=None*)
Fetches a list of all security group rules for a tenant

CLI Example:

```
salt '*' neutron.list_security_group_rules
salt '*' neutron.list_security_group_rules profile=openstack1
```

Parameters **profile** -- Profile to build on (Optional)

Returns List of security group rule

salt.modules.neutron.**list_security_groups**(*profile=None*)
Fetches a list of all security groups for a tenant

CLI Example:

```
salt '*' neutron.list_security_groups
salt '*' neutron.list_security_groups profile=openstack1
```

Parameters **profile** -- Profile to build on (Optional)

Returns List of security group

salt.modules.neutron.**list_subnets**(*profile=None*)
Fetches a list of all networks for a tenant

CLI Example:

```
salt '*' neutron.list_subnets
salt '*' neutron.list_subnets profile=openstack1
```

Parameters **profile** -- Profile to build on (Optional)

Returns List of subnet

salt.modules.neutron.**list_vpnservices**(*retrive_all=True*, *profile=None*, ***kwargs*)
Fetches a list of all configured VPN services for a tenant

CLI Example:

```
salt '*' neutron.list_vpnservices
```

Parameters

- **retrive_all** -- True or False, default: True (Optional)
- **profile** -- Profile to build on (Optional)

Returns List of VPN service

salt.modules.neutron.**remove_gateway_router**(*router*, *profile=None*)
Removes an external network gateway from the specified router

CLI Example:

```
salt '*' neutron.remove_gateway_router router-name
```

Parameters

- **router** -- ID or name of router
- **profile** -- Profile to build on (Optional)

Returns True(Succeed) or False

salt.modules.neutron.**remove_interface_router**(*router*, *subnet*, *profile=None*)
Removes an internal network interface from the specified router

CLI Example:

```
salt '*' neutron.remove_interface_router router-name subnet-name
```

Parameters

- **router** -- ID or name of the router
- **subnet** -- ID or name of the subnet
- **profile** -- Profile to build on (Optional)

Returns True(Succeed) or False

salt.modules.neutron.**show_floatingip**(*floatingip_id*, *profile=None*)
Fetches information of a certain floatingIP

CLI Example:

```
salt '*' neutron.show_floatingip floatingip-id
```

Parameters

- **floatingip_id** -- ID of floatingIP to look up
- **profile** -- Profile to build on (Optional)

Returns Floating IP information

salt.modules.neutron.**show_ikepolicy**(*ikepolicy*, *profile=None*)
Fetches information of a specific IKEPolicy

CLI Example:

```
salt '*' neutron.show_ikepolicy ikepolicy-name
```

Parameters

- **ikepolicy** -- ID or name of ikepolicy to look up
- **profile** -- Profile to build on (Optional)

Returns IKE policy information

salt.modules.neutron.**show_ipsec_site_connection**(*ipsec_site_connection*, *profile=None*)
Fetches information of a specific IPsecSiteConnection

CLI Example:

```
salt '*' neutron.show_ipsec_site_connection connection-name
```

Parameters

- **ipsec_site_connection** -- ID or name of ipsec site connection to look up
- **profile** -- Profile to build on (Optional)

Returns IPSec site connection information

salt.modules.neutron.**show_ipsecpolicy**(*ipsecpolicy*, *profile=None*)
Fetches information of a specific IPsecPolicy

CLI Example:

```
salt '*' neutron.show_ipsecpolicy ipsecpolicy-name
```

Parameters

- **ipsecpolicy** -- ID or name of IPSec policy to look up
- **profile** -- Profile to build on (Optional)

Returns IPSec policy information

salt.modules.neutron.**show_network**(*network*, *profile=None*)
Fetches information of a certain network

CLI Example:

```
salt '*' neutron.show_network network-name
salt '*' neutron.show_network network-name profile=openstack1
```

Parameters

- **network** -- ID or name of network to look up
- **profile** -- Profile to build on (Optional)

Returns Network information

salt.modules.neutron.**show_port**(*port*, *profile=None*)
Fetches information of a certain port

CLI Example:

```
salt '*' neutron.show_port port-id
salt '*' neutron.show_port port-id profile=openstack1
```

Parameters

- **port** -- ID or name of port to look up
- **profile** -- Profile to build on (Optional)

Returns Port information

salt.modules.neutron.**show_quota**(*tenant_id*, *profile=None*)
Fetches information of a certain tenant's quotas

CLI Example:

```
salt '*' neutron.show_quota tenant-id
salt '*' neutron.show_quota tenant-id profile=openstack1
```

Parameters

- **tenant_id** -- ID of tenant
- **profile** -- Profile to build on (Optional)

Returns Quota information

salt.modules.neutron.**show_router**(*router*, *profile=None*)
Fetches information of a certain router

CLI Example:

```
salt '*' neutron.show_router router-name
```

Parameters

- **router** -- ID or name of router to look up
- **profile** -- Profile to build on (Optional)

Returns Router information

salt.modules.neutron.**show_security_group**(*security_group*, *profile=None*)
Fetches information of a certain security group

CLI Example:

```
salt '*' neutron.show_security_group security-group-name
```

Parameters

- **security_group** -- ID or name of security group to look up
- **profile** -- Profile to build on (Optional)

Returns Security group information

salt.modules.neutron.**show_security_group_rule**(*security_group_rule_id*, *profile=None*)
Fetches information of a certain security group rule

CLI Example:

```
salt '*' neutron.show_security_group_rule security-group-rule-id
```

Parameters

- **security_group_rule_id** -- ID of security group rule to look up
- **profile** -- Profile to build on (Optional)

Returns Security group rule information

salt.modules.neutron.**show_subnet**(*subnet*, *profile=None*)
Fetches information of a certain subnet

CLI Example:

```
salt '*' neutron.show_subnet subnet-name
```

Parameters

- **subnet** -- ID or name of subnet to look up
- **profile** -- Profile to build on (Optional)

Returns Subnet information

salt.modules.neutron.**show_vpnservice**(*vpnservice*, *profile=None*, ***kwargs*)
Fetches information of a specific VPN service

CLI Example:

```
salt '*' neutron.show_vpnservice vpnservice-name
```

Parameters

- **vpnservice** -- ID or name of vpn service to look up
- **profile** -- Profile to build on (Optional)

Returns VPN service information

salt.modules.neutron.**update_floatingip**(*floatingip_id*, *port*, *profile=None*)
Updates a floatingIP

CLI Example:

```
salt '*' neutron.update_floatingip network-name port-name
```

Parameters

- **floatingip_id** -- ID of floatingIP
- **port** -- ID or name of port
- **profile** -- Profile to build on (Optional)

Returns Value of updated floating IP information

salt.modules.neutron.**update_network**(*network*, *name*, *profile=None*)
Updates a network

CLI Example:

```
salt '*' neutron.update_network network-name new-network-name
```

Parameters

- **network** -- ID or name of network to update
- **name** -- Name of this network
- **profile** -- Profile to build on (Optional)

Returns Value of updated network information

salt.modules.neutron.**update_port**(*port*, *name*, *admin_state_up=True*, *profile=None*)
Updates a port

CLI Example:

```
salt '*' neutron.update_port port-name network-name new-port-name
```

Parameters

- **port** -- Port name or ID
- **name** -- Name of this port
- **admin_state_up** -- Set admin state up to true or false, default: true (Optional)
- **profile** -- Profile to build on (Optional)

Returns Value of updated port information

salt.modules.neutron.**update_quota**(*tenant_id*, *subnet=None*, *router=None*, *network=None*, *floatingip=None*, *port=None*, *security_group=None*, *security_group_rule=None*, *profile=None*)
Update a tenant's quota

CLI Example:

```
salt '*' neutron.update_quota tenant-id subnet=40 router=50
                              network=10 floatingip=30 port=30
```

Parameters

- **tenant_id** -- ID of tenant
- **subnet** -- Value of subnet quota (Optional)
- **router** -- Value of router quota (Optional)
- **network** -- Value of network quota (Optional)
- **floatingip** -- Value of floatingip quota (Optional)
- **port** -- Value of port quota (Optional)
- **security_group** -- Value of security group (Optional)
- **security_group_rule** -- Value of security group rule (Optional)
- **profile** -- Profile to build on (Optional)

Returns Value of updated quota

salt.modules.neutron.**update_router**(*router*, *name=None*, *admin_state_up=None*, *profile=None*, ***kwargs*)

Updates a router

CLI Example:

```
salt '*' neutron.update_router router_id name=new-router-name
        admin_state_up=True
```

Parameters

- **router** -- ID or name of router to update
- **name** -- Name of this router
- **ext_network** -- ID or name of the external for the gateway (Optional)
- **admin_state_up** -- Set admin state up to true or false, default: true (Optional)
- **profile** -- Profile to build on (Optional)
- **kwargs** --

Returns Value of updated router information

salt.modules.neutron.**update_security_group**(*security_group*, *name=None*, *description=None*, *profile=None*)

Updates a security group

CLI Example:

```
salt '*' neutron.update_security_group security-group-name                  new-security-group-na
```

Parameters

- **security_group** -- ID or name of security group to update
- **name** -- Name of this security group (Optional)
- **description** -- Description of security group (Optional)
- **profile** -- Profile to build on (Optional)

Returns Value of updated security group information

salt.modules.neutron.**update_subnet**(*subnet*, *name*, *profile=None*)
Updates a subnet

CLI Example:

```
salt '*' neutron.update_subnet subnet-name new-subnet-name
```

Parameters

- **subnet** -- ID or name of subnet to update
- **name** -- Name of this subnet
- **profile** -- Profile to build on (Optional)

Returns Value of updated subnet information

salt.modules.neutron.**update_vpnservice**(*vpnservice*, *desc*, *profile=None*)
Updates a VPN service

CLI Example:

```
salt '*' neutron.update_vpnservice vpnservice-name desc='VPN Service1'
```

Parameters

- **vpnservice** -- ID or name of vpn service to update
- **desc** -- Set a description for the VPN service
- **profile** -- Profile to build on (Optional)

Returns Value of updated VPN service information

22.16.146 salt.modules.nfs3

Module for managing NFS version 3.

salt.modules.nfs3.**del_export**(*exports='/etc/exports'*, *path=None*)
Remove an export

CLI Example:

```
salt '*' nfs.del_export /media/storage
```

salt.modules.nfs3.**list_exports**(*exports='/etc/exports'*)
List configured exports

CLI Example:

```
salt '*' nfs.list_exports
```

22.16.147 salt.modules.nftables

Support for nftables

salt.modules.nftables.**append**(*table='filter'*, *chain=None*, *rule=None*, *family='ipv4'*)
Append a rule to the specified table & chain.

This function accepts a rule in a standard nftables command format, starting with the chain. Trying to force users to adapt to a new method of creating rules would be irritating at best, and we already have a parser that can handle it.

CLI Example:

```
salt '*' nftables.append filter input \
    rule='input tcp dport 22 log accept'

IPv6:
salt '*' nftables.append filter input \
    rule='input tcp dport 22 log accept' \
    family=ipv6
```

salt.modules.nftables.**build_rule**(*table=None, chain=None, command=None, position='', full=None, family='ipv4', **kwargs*)

Build a well-formatted nftables rule based on kwargs. A *table* and *chain* are not required, unless *full* is True.

If *full* is *True*, then *table*, *chain* and *command* are required. *command* may be specified as either insert, append, or delete. This will return the nftables command, exactly as it would be used from the command line.

If a position is required (as with *insert* or *delete*), it may be specified as *position*. This will only be useful if *full* is True.

If *connstate* is passed in, it will automatically be changed to *state*.

CLI Examples:

```
salt '*' nftables.build_rule match=state \
    connstate=RELATED,ESTABLISHED jump=ACCEPT
salt '*' nftables.build_rule filter input command=insert position=3 \
    full=True match=state state=related,established jump=accept

IPv6:
salt '*' nftables.build_rule match=state \
    connstate=related,established jump=accept \
    family=ipv6
salt '*' nftables.build_rule filter input command=insert position=3 \
    full=True match=state state=related,established jump=accept \
    family=ipv6
```

salt.modules.nftables.**check**(*table='filter', chain=None, rule=None, family='ipv4'*)

Check for the existence of a rule in the table and chain

This function accepts a rule in a standard nftables command format, starting with the chain. Trying to force users to adapt to a new method of creating rules would be irritating at best, and we already have a parser that can handle it.

CLI Example:

```
salt '*' nftables.check filter input \
    rule='input tcp dport 22 log accept'

IPv6:
salt '*' nftables.check filter input \
    rule='input tcp dport 22 log accept' \
    family=ipv6
```

salt.modules.nftables.**check_chain**(*table='filter', chain=None, family='ipv4'*)

New in version 2014.7.0.

Check for the existence of a chain in the table

CLI Example:

```
salt '*' nftables.check_chain filter input

IPv6:
salt '*' nftables.check_chain filter input family=ipv6
```

salt.modules.nftables.**check_table**(*table=None*, *family='ipv4'*)

Check for the existence of a table

CLI Example:

```
salt '*' nftables.check_table nat
```

salt.modules.nftables.**delete**(*table*, *chain=None*, *position=None*, *rule=None*, *family='ipv4'*)

Delete a rule from the specified table & chain, specifying either the rule in its entirety, or the rule's position in the chain.

This function accepts a rule in a standard nftables command format, starting with the chain. Trying to force users to adapt to a new method of creating rules would be irritating at best, and we already have a parser that can handle it.

CLI Examples:

```
salt '*' nftables.delete filter input position=3

salt '*' nftables.delete filter input \
    rule='input tcp dport 22 log accept'

IPv6:
salt '*' nftables.delete filter input position=3 family=ipv6

salt '*' nftables.delete filter input \
    rule='input tcp dport 22 log accept' \
    family=ipv6
```

salt.modules.nftables.**delete_chain**(*table='filter'*, *chain=None*, *family='ipv4'*)

New in version 2014.7.0.

Delete the chain from the specified table.

CLI Example:

```
salt '*' nftables.delete_chain filter input

salt '*' nftables.delete_chain filter foo

IPv6:
salt '*' nftables.delete_chain filter input family=ipv6

salt '*' nftables.delete_chain filter foo family=ipv6
```

salt.modules.nftables.**delete_table**(*table*, *family='ipv4'*)

New in version 2014.7.0.

Create new custom table.

CLI Example:

```
salt '*' nftables.delete_table filter
```

```
    IPv6:
    salt '*' nftables.delete_table filter family=ipv6
```

salt.modules.nftables.**flush**(*table='filter'*, *chain=''*, *family='ipv4'*)

Flush the chain in the specified table, flush all chains in the specified table if chain is not specified.

CLI Example:

```
    salt '*' nftables.flush filter

    salt '*' nftables.flush filter input

    IPv6:
    salt '*' nftables.flush filter input family=ipv6
```

salt.modules.nftables.**get_rule_handle**(*table='filter'*, *chain=None*, *rule=None*, *family='ipv4'*)

Get the handle for a particular rule

This function accepts a rule in a standard nftables command format, starting with the chain. Trying to force users to adapt to a new method of creating rules would be irritating at best, and we already have a parser that can handle it.

CLI Example:

```
    salt '*' nftables.get_rule_handle filter input \
        rule='input tcp dport 22 log accept'

    IPv6:
    salt '*' nftables.get_rule_handle filter input \
        rule='input tcp dport 22 log accept' \
        family=ipv6
```

salt.modules.nftables.**get_rules**(*family='ipv4'*)

Return a data structure of the current, in-memory rules

CLI Example:

```
    salt '*' nftables.get_rules

    salt '*' nftables.get_rules family=ipv6
```

salt.modules.nftables.**get_saved_rules**(*conf_file=None*, *family='ipv4'*)

Return a data structure of the rules in the conf file

CLI Example:

```
    salt '*' nftables.get_saved_rules
```

salt.modules.nftables.**insert**(*table='filter'*, *chain=None*, *position=None*, *rule=None*, *family='ipv4'*)

Insert a rule into the specified table & chain, at the specified position.

If position is not specified, rule will be inserted in first position.

This function accepts a rule in a standard nftables command format, starting with the chain. Trying to force users to adapt to a new method of creating rules would be irritating at best, and we already have a parser that can handle it.

CLI Examples:

```
    salt '*' nftables.insert filter input \
        rule='input tcp dport 22 log accept'
```

```
    salt '*' nftables.insert filter input position=3 \
        rule='input tcp dport 22 log accept'

    IPv6:
    salt '*' nftables.insert filter input \
        rule='input tcp dport 22 log accept' \
        family=ipv6

    salt '*' nftables.insert filter input position=3 \
        rule='input tcp dport 22 log accept' \
        family=ipv6
```

salt.modules.nftables.**new_chain**(*table='filter'*, *chain=None*, *table_type=None*, *hook=None*, *priority=None*, *family='ipv4'*)

New in version 2014.7.0.

Create new chain to the specified table.

CLI Example:

```
    salt '*' nftables.new_chain filter input

    salt '*' nftables.new_chain filter input \
            table_type=filter hook=input priority=0

    salt '*' nftables.new_chain filter foo

    IPv6:
    salt '*' nftables.new_chain filter input family=ipv6

    salt '*' nftables.new_chain filter input \
            table_type=filter hook=input priority=0 family=ipv6

    salt '*' nftables.new_chain filter foo family=ipv6
```

salt.modules.nftables.**new_table**(*table*, *family='ipv4'*)

New in version 2014.7.0.

Create new custom table.

CLI Example:

```
    salt '*' nftables.new_table filter

    IPv6:
    salt '*' nftables.new_table filter family=ipv6
```

salt.modules.nftables.**save**(*filename=None*, *family='ipv4'*)

Save the current in-memory rules to disk

CLI Example:

```
    salt '*' nftables.save /etc/nftables
```

salt.modules.nftables.**version**()

Return version from nftables --version

CLI Example:

```
salt '*' nftables.version
```

22.16.148 salt.modules.nginx

Support for nginx

salt.modules.nginx.**configtest**()
test configuration and exit

CLI Example:

```
salt '*' nginx.configtest
```

salt.modules.nginx.**signal**(*signal=None*)
Signals nginx to start, reload, reopen or stop.

CLI Example:

```
salt '*' nginx.signal reload
```

salt.modules.nginx.**status**(*url='http://127.0.0.1/status'*)
Return the data from an Nginx status page as a dictionary. http://wiki.nginx.org/HttpStubStatusModule

url The URL of the status page. Defaults to `http://127.0.0.1/status`

CLI Example:

```
salt '*' nginx.status
```

salt.modules.nginx.**version**()
Return server version from nginx -v

CLI Example:

```
salt '*' nginx.version
```

22.16.149 salt.modules.nova

Module for handling OpenStack Nova calls

depends

- novaclient Python module

configuration This module is not usable until the user, password, tenant, and auth URL are specified either in a pillar or in the minion's config file. For example:

```
keystone.user: admin
keystone.password: verybadpass
keystone.tenant: admin
keystone.auth_url: 'http://127.0.0.1:5000/v2.0/'
# Optional
keystone.region_name: 'regionOne'
```

If configuration for multiple OpenStack accounts is required, they can be set up as different configuration profiles: For example:

```
        openstack1:
          keystone.user: admin
          keystone.password: verybadpass
          keystone.tenant: admin
          keystone.auth_url: 'http://127.0.0.1:5000/v2.0/'

        openstack2:
          keystone.user: admin
          keystone.password: verybadpass
          keystone.tenant: admin
          keystone.auth_url: 'http://127.0.0.2:5000/v2.0/'
```

With this configuration in place, any of the nova functions can make use of a configuration profile by declaring it explicitly. For example:

```
salt '*' nova.flavor_list profile=openstack1
```

salt.modules.nova.**boot**(*name*, *flavor_id=0*, *image_id=0*, *profile=None*, *timeout=300*)

Boot (create) a new instance

name Name of the new instance (must be first)

flavor_id Unique integer ID for the flavor

image_id Unique integer ID for the image

timeout How long to wait, after creating the instance, for the provider to return information about it (default 300 seconds).

New in version 2014.1.0.

CLI Example:

```
salt '*' nova.boot myinstance flavor_id=4596 image_id=2
```

The flavor_id and image_id are obtained from nova.flavor_list and nova.image_list

```
salt '*' nova.flavor_list
salt '*' nova.image_list
```

salt.modules.nova.**delete**(*instance_id*, *profile=None*)

Delete an instance

instance_id ID of the instance to be deleted

CLI Example:

```
salt '*' nova.delete 1138
```

salt.modules.nova.**flavor_create**(*name*, *flavor_id=0*, *ram=0*, *disk=0*, *vcpus=1*, *profile=None*)

Add a flavor to nova (nova flavor-create). The following parameters are required:

name Name of the new flavor (must be first)

flavor_id Unique integer ID for the new flavor

ram Memory size in MB

disk Disk size in GB

vcpus Number of vcpus

CLI Example:

```
    salt '*' nova.flavor_create myflavor flavor_id=6                    ram=4096 disk=10 vcpus=1
```

salt.modules.nova.**flavor_delete**(*flavor_id*, *profile=None*)
Delete a flavor from nova by id (nova flavor-delete)

CLI Example:

```
    salt '*' nova.flavor_delete 7
```

salt.modules.nova.**flavor_list**(*profile=None*)
Return a list of available flavors (nova flavor-list)

CLI Example:

```
    salt '*' nova.flavor_list
```

salt.modules.nova.**image_list**(*name=None*, *profile=None*)
Return a list of available images (nova images-list + nova image-show) If a name is provided, only that image will be displayed.

CLI Examples:

```
    salt '*' nova.image_list
    salt '*' nova.image_list myimage
```

salt.modules.nova.**image_meta_delete**(*image_id=None*, *name=None*, *keys=None*, *profile=None*)
Delete a key=value pair from the metadata for an image (nova image-meta set)

CLI Examples:

```
    salt '*' nova.image_meta_delete                 6f52b2ff-0b31-4d84-8fd1-af45b84824f6 keys=cheese
    salt '*' nova.image_meta_delete name=myimage keys=salad,beans
```

salt.modules.nova.**image_meta_set**(*image_id=None*, *name=None*, *profile=None*, ***kwargs*)
Sets a key=value pair in the metadata for an image (nova image-meta set)

CLI Examples:

```
    salt '*' nova.image_meta_set 6f52b2ff-0b31-4d84-8fd1-af45b84824f6                cheese=gruyere
    salt '*' nova.image_meta_set name=myimage salad=pasta beans=baked
```

salt.modules.nova.**keypair_add**(*name*, *pubfile=None*, *pubkey=None*, *profile=None*)
Add a keypair to nova (nova keypair-add)

CLI Examples:

```
    salt '*' nova.keypair_add mykey pubfile='/home/myuser/.ssh/id_rsa.pub'
    salt '*' nova.keypair_add mykey pubkey='ssh-rsa <key> myuser@mybox'
```

salt.modules.nova.**keypair_delete**(*name*, *profile=None*)
Add a keypair to nova (nova keypair-delete)

CLI Example:

```
    salt '*' nova.keypair_delete mykey'
```

salt.modules.nova.**keypair_list**(*profile=None*)
Return a list of available keypairs (nova keypair-list)

CLI Example:

```
salt '*' nova.keypair_list
```

salt.modules.nova.**list_**(*profile=None*)
To maintain the feel of the nova command line, this function simply calls the server_list function.

salt.modules.nova.**lock**(*instance_id*, *profile=None*)
Lock an instance

instance_id ID of the instance to be locked

CLI Example:

```
salt '*' nova.lock 1138
```

salt.modules.nova.**resume**(*instance_id*, *profile=None*)
Resume an instance

instance_id ID of the instance to be resumed

CLI Example:

```
salt '*' nova.resume 1138
```

salt.modules.nova.**secgroup_create**(*name*, *description*, *profile=None*)
Add a secgroup to nova (nova secgroup-create)

CLI Example:

```
salt '*' nova.secgroup_create mygroup 'This is my security group'
```

salt.modules.nova.**secgroup_delete**(*name*, *profile=None*)
Delete a secgroup to nova (nova secgroup-delete)

CLI Example:

```
salt '*' nova.secgroup_delete mygroup
```

salt.modules.nova.**secgroup_list**(*profile=None*)
Return a list of available security groups (nova items-list)

CLI Example:

```
salt '*' nova.secgroup_list
```

salt.modules.nova.**server_by_name**(*name*, *profile=None*)
Return information about a server

name Server Name

CLI Example:

```
salt '*' nova.server_by_name myserver profile=openstack
```

salt.modules.nova.**server_list**(*profile=None*)
Return list of active servers

CLI Example:

```
salt '*' nova.show
```

salt.modules.nova.**server_list_detailed**(*profile=None*)
Return detailed list of active servers

CLI Example:

```
salt '*' nova.server_list_detailed
```

salt.modules.nova.**server_show**(*server_id*, *profile=None*)
Return detailed information for an active server

CLI Example:

```
salt '*' nova.server_show <server_id>
```

salt.modules.nova.**show**(*server_id*, *profile=None*)
To maintain the feel of the nova command line, this function simply calls the server_show function.

CLI Example:

```
salt '*' nova.show
```

salt.modules.nova.**suspend**(*instance_id*, *profile=None*)
Suspend an instance

instance_id ID of the instance to be suspended

CLI Example:

```
salt '*' nova.suspend 1138
```

salt.modules.nova.**volume_attach**(*name*, *server_name*, *device='/dev/xvdb'*, *profile=None*, *timeout=300*)
Attach a block storage volume

name Name of the new volume to attach

server_name Name of the server to attach to

device Name of the device on the server

profile Profile to build on

CLI Example:

```
salt '*' nova.volume_attach myblock slice.example.com profile=openstack
salt '*' nova.volume_attach myblock server.example.com                 device='/dev/xvdb' profil
```

salt.modules.nova.**volume_create**(*name*, *size=100*, *snapshot=None*, *voltype=None*, *profile=None*)
Create a block storage volume

name Name of the new volume (must be first)

size Volume size

snapshot Block storage snapshot id

voltype Type of storage

profile Profile to build on

CLI Example:

```
salt '*' nova.volume_create myblock size=300 profile=openstack
```

salt.modules.nova.**volume_delete**(*name*, *profile=None*)
Destroy the volume

name Name of the volume

profile Profile to build on

CLI Example:

```
salt '*' nova.volume_delete myblock profile=openstack
```

salt.modules.nova.**volume_detach**(*name*, *profile=None*, *timeout=300*)
Attach a block storage volume

name Name of the new volume to attach

server_name Name of the server to detach from

profile Profile to build on

CLI Example:

```
salt '*' nova.volume_detach myblock profile=openstack
```

salt.modules.nova.**volume_list**(*search_opts=None*, *profile=None*)
List storage volumes

search_opts Dictionary of search options

profile Profile to use

CLI Example:

```
salt '*' nova.volume_list                search_opts='{"display_name": "myblock"}'
```

salt.modules.nova.**volume_show**(*name*, *profile=None*)
Create a block storage volume

name Name of the volume

profile Profile to use

CLI Example:

```
salt '*' nova.volume_show myblock profile=openstack
```

22.16.150 salt.modules.npm

Manage and query NPM packages.

salt.modules.npm.**install**(*pkg=None*, *pkgs=None*, *dir=None*, *runas=None*, *registry=None*, *env=None*)
Install an NPM package.

If no directory is specified, the package will be installed globally. If no package is specified, the dependencies (from package.json) of the package in the given directory will be installed.

pkg A package name in any format accepted by NPM, including a version identifier

pkgs A list of package names in the same format as the `name` parameter

New in version 2014.7.0.

dir The target directory in which to install the package, or None for global installation

runas The user to run NPM with

registry The NPM registry to install the package from.

New in version 2014.7.0.

env Environment variables to set when invoking npm. Uses the same `env` format as the `cmd.run` execution function.

New in version 2014.7.0.

CLI Example:

```
salt '*' npm.install coffee-script

salt '*' npm.install coffee-script@1.0.1
```

`salt.modules.npm.`**`list_`**(*pkg=None*, *dir=None*, *runas=None*, *env=None*)
List installed NPM packages.

If no directory is specified, this will return the list of globally- installed packages.

pkg Limit package listing by name

dir The directory whose packages will be listed, or None for global installation

runas The user to run NPM with

New in version 2014.7.0.

env Environment variables to set when invoking npm. Uses the same `env` format as the `cmd.run` execution function.

New in version 2014.7.0.

CLI Example:

```
salt '*' npm.list
```

`salt.modules.npm.`**`uninstall`**(*pkg*, *dir=None*, *runas=None*)
Uninstall an NPM package.

If no directory is specified, the package will be uninstalled globally.

pkg A package name in any format accepted by NPM

dir The target directory from which to uninstall the package, or None for global installation

runas The user to run NPM with

CLI Example:

```
salt '*' npm.uninstall coffee-script
```

22.16.151 salt.modules.omapi

This module interacts with an ISC DHCP Server via OMAPI. server_ip and server_port params may be set in the minion config or pillar:

```
omapi.server_ip: 127.0.0.1
omapi.server_port: 7991
```

depends pypureomapi Python module

`salt.modules.omapi.`**`add_host`**(*mac*, *name=None*, *ip=None*, *ddns=False*, *group=None*, *supersede_host=False*)
Add a host object for the given mac.

CLI Example:

```
salt dhcp-server omapi.add_host ab:ab:ab:ab:ab:ab name=host1
```

Add ddns-hostname and a fixed-ip statements:

```
salt dhcp-server omapi.add_host ab:ab:ab:ab:ab:ab name=host1 ip=10.1.1.1 ddns=true
```

salt.modules.omapi.**delete_host**(*mac=None*, *name=None*)
Delete the host with the given mac or name.

CLI Examples:

```
salt dhcp-server omapi.delete_host name=host1
salt dhcp-server omapi.delete_host mac=ab:ab:ab:ab:ab:ab
```

22.16.152 salt.modules.openbsd_sysctl

Module for viewing and modifying OpenBSD sysctl parameters

salt.modules.openbsd_sysctl.**assign**(*name*, *value*)
Assign a single sysctl parameter for this minion

CLI Example:

```
salt '*' sysctl.assign net.inet.ip.forwarding 1
```

salt.modules.openbsd_sysctl.**get**(*name*)
Return a single sysctl parameter for this minion

CLI Example:

```
salt '*' sysctl.get hw.physmem
```

salt.modules.openbsd_sysctl.**persist**(*name*, *value*, *config='/etc/sysctl.conf'*)
Assign and persist a simple sysctl parameter for this minion

CLI Example:

```
salt '*' sysctl.persist net.inet.ip.forwarding 1
```

salt.modules.openbsd_sysctl.**show**(*config_file=False*)
Return a list of sysctl parameters for this minion

CLI Example:

```
salt '*' sysctl.show
```

22.16.153 salt.modules.openbsdpkg

Package support for OpenBSD

salt.modules.openbsdpkg.**install**(*name=None*, *pkgs=None*, *sources=None*, ***kwargs*)
Install the passed package

Return a dict containing the new package names and versions:

```
{'<package>': {'old': '<old-version>',
               'new': '<new-version>'}}
```

CLI Example, Install one package:

```
salt '*' pkg.install <package name>
```

CLI Example, Install more than one package:

```
salt '*' pkg.install pkgs='["<package name>", "<package name>"]'
```

CLI Example, Install more than one package from a alternate source (e.g. salt file-server, HTTP, FTP, local filesystem):

```
salt '*' pkg.install sources='[{"<pkg name>": "salt://pkgs/<pkg filename>"}]'
```

salt.modules.openbsdpkg.**latest_version**(**names*, ***kwargs*)

The available version of the package in the repository

CLI Example:

```
salt '*' pkg.latest_version <package name>
```

salt.modules.openbsdpkg.**list_pkgs**(*versions_as_list=False*, ***kwargs*)

List the packages currently installed as a dict:

```
{'<package_name>': '<version>'}
```

CLI Example:

```
salt '*' pkg.list_pkgs
```

salt.modules.openbsdpkg.**purge**(*name=None*, *pkgs=None*, ***kwargs*)

Package purges are not supported, this function is identical to `remove()`.

name The name of the package to be deleted.

Multiple Package Options:

pkgs A list of packages to delete. Must be passed as a python list. The `name` parameter will be ignored if this option is passed.

New in version 0.16.0.

Returns a dict containing the changes.

CLI Example:

```
salt '*' pkg.purge <package name>
salt '*' pkg.purge <package1>,<package2>,<package3>
salt '*' pkg.purge pkgs='["foo", "bar"]'
```

salt.modules.openbsdpkg.**remove**(*name=None*, *pkgs=None*, ***kwargs*)

Remove a single package with pkg_delete

Multiple Package Options:

pkgs A list of packages to delete. Must be passed as a python list. The `name` parameter will be ignored if this option is passed.

New in version 0.16.0.

Returns a dict containing the changes.

CLI Example:

```
    salt '*' pkg.remove <package name>
    salt '*' pkg.remove <package1>,<package2>,<package3>
    salt '*' pkg.remove pkgs='["foo", "bar"]'
```

salt.modules.openbsdpkg.**version**(**names*, ***kwargs*)

Returns a string representing the package version or an empty string if not installed. If more than one package name is specified, a dict of name/version pairs is returned.

CLI Example:

```
    salt '*' pkg.version <package name>
    salt '*' pkg.version <package1> <package2> <package3> ...
```

22.16.154 salt.modules.openbsdrcctl

The rcctl service module for OpenBSD

salt.modules.openbsdrcctl.**available**(*name*)

Return True if the named service is available.

CLI Example:

```
    salt '*' service.available sshd
```

salt.modules.openbsdrcctl.**disable**(*name*, ***kwargs*)

Disable the named service to not start at boot.

CLI Example:

```
    salt '*' service.disable <service name>
```

salt.modules.openbsdrcctl.**disabled**(*name*)

Return True if the named service is disabled at boot, False otherwise.

CLI Example:

```
    salt '*' service.disabled <service name>
```

salt.modules.openbsdrcctl.**enable**(*name*, ***kwargs*)

Enable the named service to start at boot.

flags [None] Set optional flags to run the service with.

service.flags can be used to change the default flags.

CLI Example:

```
    salt '*' service.enable <service name>
    salt '*' service.enable <service name> flags=<flags>
```

salt.modules.openbsdrcctl.**enabled**(*name*, ***kwargs*)

Return True if the named service is enabled at boot and the provided flags match the configured ones (if any). Return False otherwise.

name Service name

CLI Example:

```
    salt '*' service.enabled <service name>
    salt '*' service.enabled <service name> flags=<flags>
```

salt.modules.openbsdrcctl.**get_all**()
Return all installed services.

CLI Example:

```
salt '*' service.get_all
```

salt.modules.openbsdrcctl.**get_disabled**()
Return what services are available but not enabled to start at boot.

CLI Example:

```
salt '*' service.get_disabled
```

salt.modules.openbsdrcctl.**get_enabled**()
Return what services are set to run on boot.

CLI Example:

```
salt '*' service.get_enabled
```

salt.modules.openbsdrcctl.**missing**(*name*)
The inverse of service.available. Return True if the named service is not available.

CLI Example:

```
salt '*' service.missing sshd
```

salt.modules.openbsdrcctl.**reload_**(*name*)
Reload the named service.

CLI Example:

```
salt '*' service.reload <service name>
```

salt.modules.openbsdrcctl.**restart**(*name*)
Restart the named service.

CLI Example:

```
salt '*' service.restart <service name>
```

salt.modules.openbsdrcctl.**start**(*name*)
Start the named service.

CLI Example:

```
salt '*' service.start <service name>
```

salt.modules.openbsdrcctl.**status**(*name*, *sig=None*)
Return the status for a service, returns a bool whether the service is running.

CLI Example:

```
salt '*' service.status <service name>
```

salt.modules.openbsdrcctl.**stop**(*name*)
Stop the named service.

CLI Example:

```
salt '*' service.stop <service name>
```

22.16.155 salt.modules.openbsdservice

The service module for OpenBSD

salt.modules.openbsdservice.**available**(*name*)

New in version 2014.7.0.

Returns `True` if the specified service is available, otherwise returns `False`.

CLI Example:

```
salt '*' service.available sshd
```

salt.modules.openbsdservice.**disabled**(*name*)

New in version 2014.7.0.

Return True if the named service is disabled, false otherwise

CLI Example:

```
salt '*' service.disabled <service name>
```

salt.modules.openbsdservice.**enabled**(*name*, ***kwargs*)

New in version 2014.7.0.

Return True if the named service is enabled, false otherwise

CLI Example:

```
salt '*' service.enabled <service name>
```

salt.modules.openbsdservice.**get_all**()

New in version 2014.7.0.

Return all available boot services

CLI Example:

```
salt '*' service.get_all
```

salt.modules.openbsdservice.**get_disabled**()

New in version 2014.7.0.

Return a set of services that are installed but disabled

CLI Example:

```
salt '*' service.get_disabled
```

salt.modules.openbsdservice.**get_enabled**()

New in version 2014.7.0.

Return a list of service that are enabled on boot

CLI Example:

```
salt '*' service.get_enabled
```

salt.modules.openbsdservice.**missing**(*name*)

New in version 2014.7.0.

The inverse of service.available. Returns `True` if the specified service is not available, otherwise returns `False`.

CLI Example:

```
salt '*' service.missing sshd
```

salt.modules.openbsdservice.**reload_**(*name*)
New in version 2014.7.0.

Reload the named service

CLI Example:

```
salt '*' service.reload <service name>
```

salt.modules.openbsdservice.**restart**(*name*)
Restart the named service

CLI Example:

```
salt '*' service.restart <service name>
```

salt.modules.openbsdservice.**start**(*name*)
Start the specified service

CLI Example:

```
salt '*' service.start <service name>
```

salt.modules.openbsdservice.**status**(*name*, *sig=None*)
Return the status for a service, returns a bool whether the service is running.

CLI Example:

```
salt '*' service.status <service name>
```

salt.modules.openbsdservice.**stop**(*name*)
Stop the specified service

CLI Example:

```
salt '*' service.stop <service name>
```

22.16.156 salt.modules.openstack_config

Modify, retrieve, or delete values from OpenStack configuration files.

maintainer Jeffrey C. Ollie <jeff@ocjtech.us>

maturity new

depends

platform linux

salt.modules.openstack_config.**delete**(*filename*, *section*, *parameter*)
Delete a value from an OpenStack configuration file.

filename The full path to the configuration file

section The section from which to delete the parameter

parameter The parameter to delete

CLI Example:

```
salt-call openstack_config.delete /etc/keystone/keystone.conf sql connection
```

salt.modules.openstack_config.**get**(*filename*, *section*, *parameter*)
Get a value from an OpenStack configuration file.

filename The full path to the configuration file

section The section from which to search for the parameter

parameter The parameter to return

CLI Example:

```
salt-call openstack_config.get /etc/keystone/keystone.conf sql connection
```

salt.modules.openstack_config.**set_**(*filename*, *section*, *parameter*, *value*)
Set a value in an OpenStack configuration file.

filename The full path to the configuration file

section The section in which the parameter will be set

parameter The parameter to change

value The value to set

CLI Example:

```
salt-call openstack_config.set /etc/keystone/keystone.conf sql connection foo
```

22.16.157 salt.modules.oracle

Oracle DataBase connection module

maintainer Vladimir Bormotov <bormotov@gmail.com>

maturity new

depends cx_Oracle

platform all

configuration module provide connections for multiple Oracle DB instances.

OS Environment

```
ORACLE_HOME: path to oracle product
PATH: path to Oracle Client libs need to be in PATH
```

pillar

```
oracle.dbs: list of known based
oracle.dbs.<db>.uri: connection credentials in format:
    user/password@host[:port]/sid[ as {sysdba|sysoper}]
```

salt.modules.oracle.**client_version**()
Oracle Client Version

CLI Example:

```
salt '*' oracle.client_version
```

salt.modules.oracle.**run_query**(*db*, *query*)
Run SQL query and return result

CLI Example:

```
salt '*' oracle.run_query my_db "select * from my_table"
```

salt.modules.oracle.**show_dbs**(**dbs*)
Show databases configuration from pillar. Filter by **args*

CLI Example:

```
salt '*' oracle.show_dbs
salt '*' oracle.show_dbs my_db
```

salt.modules.oracle.**show_env**()
Show Environment used by Oracle Client

CLI Example:

```
salt '*' oracle.show_env
```

Note: at first _connect() NLS_LANG will forced to `.AL32UTF8'

salt.modules.oracle.**show_pillar**(*item=None*)
Show Pillar segment oracle.* and subitem with notation ``item:subitem''

CLI Example:

```
salt '*' oracle.show_pillar
salt '*' oracle.show_pillar dbs:my_db
```

salt.modules.oracle.**version**(**dbs*)
Server Version (select banner from v$version)

CLI Example:

```
salt '*' oracle.version
salt '*' oracle.version my_db
```

22.16.158 salt.modules.osxdesktop

Mac OS X implementations of various commands in the ``desktop'' interface

salt.modules.osxdesktop.**get_output_volume**()
Get the output volume (range 0 to 100)

CLI Example:

```
salt '*' desktop.get_output_volume
```

salt.modules.osxdesktop.**lock**()
Lock the desktop session

CLI Example:

```
salt '*' desktop.lock
```

salt.modules.osxdesktop.**say**(**words*)
Say some words.

CLI Example:

```
salt '*' desktop.say <word0> <word1> ... <wordN>
```

salt.modules.osxdesktop.**screensaver**()
Launch the screensaver

CLI Example:

```
salt '*' desktop.screensaver
```

salt.modules.osxdesktop.**set_output_volume**(*volume*)
Set the volume of sound (range 0 to 100)

CLI Example:

```
salt '*' desktop.set_output_volume <volume>
```

22.16.159 salt.modules.pacman

A module to wrap pacman calls, since Arch is the best (https://wiki.archlinux.org/index.php/Arch_is_the_best)

salt.modules.pacman.**file_dict**(**packages*)
List the files that belong to a package, grouped by package. Not specifying any packages will return a list of _every_ file on the system's package database (not generally recommended).

CLI Examples:

```
salt '*' pkg.file_list httpd
salt '*' pkg.file_list httpd postfix
salt '*' pkg.file_list
```

salt.modules.pacman.**file_list**(**packages*)
List the files that belong to a package. Not specifying any packages will return a list of _every_ file on the system's package database (not generally recommended).

CLI Examples:

```
salt '*' pkg.file_list httpd
salt '*' pkg.file_list httpd postfix
salt '*' pkg.file_list
```

salt.modules.pacman.**install**(*name=None, refresh=False, sysupgrade=False, pkgs=None, sources=None, **kwargs*)
Install (pacman -S) the passed package, add refresh=True to install with -y, add sysupgrade=True to install with -u.

name The name of the package to be installed. Note that this parameter is ignored if either ``pkgs'' or ``sources'' is passed. Additionally, please note that this option can only be used to install packages from a software repository. To install a package file manually, use the ``sources'' option.

CLI Example:

```
salt '*' pkg.install <package name>
```

refresh Whether or not to refresh the package database before installing.

sysupgrade Whether or not to upgrade the system packages before installing.

Multiple Package Installation Options:

pkgs A list of packages to install from a software repository. Must be passed as a python list. A specific version number can be specified by using a single-element dict representing the package and its version. As with the `version` parameter above, comparison operators can be used to target a specific version of a package.

CLI Examples:

```
salt '*' pkg.install pkgs='["foo", "bar"]'
salt '*' pkg.install pkgs='["foo", {"bar": "1.2.3-4"}]'
salt '*' pkg.install pkgs='["foo", {"bar": "<1.2.3-4"}]'
```

sources A list of packages to install. Must be passed as a list of dicts, with the keys being package names, and the values being the source URI or local path to the package.

CLI Example:

```
salt '*' pkg.install                    sources='[{"foo": "salt://foo.pkg.tar.xz"},
```

Returns a dict containing the new package names and versions:

```
{'<package>': {'old': '<old-version>',
               'new': '<new-version>'}}
```

`salt.modules.pacman.`**`latest_version`**(**names*, ***kwargs*)

Return the latest version of the named package available for upgrade or installation. If more than one package name is specified, a dict of name/version pairs is returned.

If the latest version of a given package is already installed, an empty string will be returned for that package.

CLI Example:

```
salt '*' pkg.latest_version <package name>
salt '*' pkg.latest_version <package1> <package2> <package3> ...
```

`salt.modules.pacman.`**`list_pkgs`**(*versions_as_list=False*, ***kwargs*)

List the packages currently installed as a dict:

```
{'<package_name>': '<version>'}
```

CLI Example:

```
salt '*' pkg.list_pkgs
```

`salt.modules.pacman.`**`list_upgrades`**(*refresh=False*)

List all available package upgrades on this system

CLI Example:

```
salt '*' pkg.list_upgrades
```

`salt.modules.pacman.`**`owner`**(**paths*)

New in version 2014.7.0.

Return the name of the package that owns the file. Multiple file paths can be passed. Like `pkg.version` `<salt.modules.yumpkg.version`, if a single path is passed, a string will be returned, and if multiple paths are passed, a dictionary of file/package name pairs will be returned.

If the file is not owned by a package, or is not present on the minion, then an empty string will be returned for that path.

CLI Example:

salt `*' pkg.owner /usr/bin/apachectl salt `*' pkg.owner /usr/bin/apachectl /usr/bin/zsh

salt.modules.pacman.**purge**(*name=None, pkgs=None, **kwargs*)

Recursively remove a package and all dependencies which were installed with it, this will call a `pacman -Rsc`

name The name of the package to be deleted.

Multiple Package Options:

pkgs A list of packages to delete. Must be passed as a python list. The `name` parameter will be ignored if this option is passed.

New in version 0.16.0.

Returns a dict containing the changes.

CLI Example:

```
salt '*' pkg.purge <package name>
salt '*' pkg.purge <package1>,<package2>,<package3>
salt '*' pkg.purge pkgs='["foo", "bar"]'
```

salt.modules.pacman.**refresh_db**()

Just run a `pacman -Sy`, return a dict:

```
{'<database name>': Bool}
```

CLI Example:

```
salt '*' pkg.refresh_db
```

salt.modules.pacman.**remove**(*name=None, pkgs=None, **kwargs*)

Remove packages with `pacman -R`.

name The name of the package to be deleted.

Multiple Package Options:

pkgs A list of packages to delete. Must be passed as a python list. The `name` parameter will be ignored if this option is passed.

New in version 0.16.0.

Returns a dict containing the changes.

CLI Example:

```
salt '*' pkg.remove <package name>
salt '*' pkg.remove <package1>,<package2>,<package3>
salt '*' pkg.remove pkgs='["foo", "bar"]'
```

salt.modules.pacman.**upgrade**(*refresh=False*)

Run a full system upgrade, a pacman -Syu

refresh Whether or not to refresh the package database before installing.

Return a dict containing the new package names and versions:

```
{'<package>': {'old': '<old-version>',
               'new': '<new-version>'}}
```

CLI Example:

```
salt '*' pkg.upgrade
```

salt.modules.pacman.**upgrade_available**(*name*)
Check whether or not an upgrade is available for a given package

CLI Example:

```
salt '*' pkg.upgrade_available <package name>
```

salt.modules.pacman.**version**(**names*, ***kwargs*)
Returns a string representing the package version or an empty string if not installed. If more than one package name is specified, a dict of name/version pairs is returned.

CLI Example:

```
salt '*' pkg.version <package name>
salt '*' pkg.version <package1> <package2> <package3> ...
```

22.16.160 salt.modules.pagerduty

Module for Firing Events via PagerDuty

New in version 2014.1.0.

configuration This module can be used by specifying the name of a configuration profile in the minion config, minion pillar, or master config.

For example:

```
my-pagerduty-account:
    pagerduty.api_key: F3Rbyjbve43rfFWf2214
    pagerduty.subdomain: mysubdomain
```

salt.modules.pagerduty.**create_event**(*service_key=None*, *description=None*, *details=None*, *incident_key=None*, *profile=None*)
Create an event in PagerDuty. Designed for use in states.

CLI Example:

```
salt myminion pagerduty.create_event <service_key> <description> <details> profile=my-pa
```

The following parameters are required:

service_key This key can be found by using pagerduty.list_services.

description This is a short description of the event.

details This can be a more detailed description of the event.

profile This refers to the configuration profile to use to connect to the PagerDuty service.

salt.modules.pagerduty.**list_escalation_policies**(*profile=None*, *api_key=None*)
List escalation policies belonging to this account

CLI Example:

salt myminion pagerduty.list_policies my-pagerduty-account salt myminion pagerduty.list_escalation_policies my-pagerduty-account

salt.modules.pagerduty.**list_incidents**(*profile=None*, *api_key=None*)
List incidents belonging to this account

CLI Example:

salt myminion pagerduty.list_incidents my-pagerduty-account

salt.modules.pagerduty.**list_maintenance_windows**(*profile=None*, *api_key=None*)
List maintenance windows belonging to this account

CLI Example:

salt myminion pagerduty.list_windows my-pagerduty-account salt myminion pagerduty.list_maintenance_windows my-pagerduty-account

salt.modules.pagerduty.**list_policies**(*profile=None*, *api_key=None*)
List escalation policies belonging to this account

CLI Example:

salt myminion pagerduty.list_policies my-pagerduty-account salt myminion pagerduty.list_escalation_policies my-pagerduty-account

salt.modules.pagerduty.**list_schedules**(*profile=None*, *api_key=None*)
List schedules belonging to this account

CLI Example:

salt myminion pagerduty.list_schedules my-pagerduty-account

salt.modules.pagerduty.**list_services**(*profile=None*, *api_key=None*)
List services belonging to this account

CLI Example:

salt myminion pagerduty.list_services my-pagerduty-account

salt.modules.pagerduty.**list_users**(*profile=None*, *api_key=None*)
List users belonging to this account

CLI Example:

salt myminion pagerduty.list_users my-pagerduty-account

salt.modules.pagerduty.**list_windows**(*profile=None*, *api_key=None*)
List maintenance windows belonging to this account

CLI Example:

salt myminion pagerduty.list_windows my-pagerduty-account salt myminion pagerduty.list_maintenance_windows my-pagerduty-account

22.16.161 salt.modules.pam

Support for pam

salt.modules.pam.**read_file**(*file_name*)
This is just a test function, to make sure parsing works

CLI Example:

```
salt '*' pam.read_file /etc/pam.d/login
```

22.16.162 salt.modules.parted

Module for managing partitions on POSIX-like systems.

depends

- parted, partprobe, lsblk (usually parted and util-linux packages)

Some functions may not be available, depending on your version of parted.

Check the manpage for `parted(8)` for more information, or the online docs at:

http://www.gnu.org/software/parted/manual/html_chapter/parted_2.html

In light of parted not directly supporting partition IDs, some of this module has been written to utilize sfdisk instead. For further information, please reference the man page for `sfdisk(8)`.

`salt.modules.parted.`**`align_check`**(*device*, *part_type*, *partition*)
partition.align_check device part_type partition

Check if partition satisfies the alignment constraint of part_type. Type must be ``minimal'' or ``optimal''.

CLI Example:

```
salt '*' partition.align_check /dev/sda minimal 1
```

`salt.modules.parted.`**`check`**(*device*, *minor*)
partition.check device minor

Checks if the file system on partition <minor> has any errors.

CLI Example:

```
salt '*' partition.check 1
```

`salt.modules.parted.`**`cp`**(*device*, *from_minor*, *to_minor*)
partition.check device from_minor to_minor

Copies the file system on the partition <from-minor> to partition <to-minor>, deleting the original contents of the destination partition.

CLI Example:

```
salt '*' partition.cp /dev/sda 2 3
```

`salt.modules.parted.`**`exists`**(*device=''*)
partition.exists device

Check to see if the partition exists

CLI Example:

```
salt '*' partition.exists /dev/sdb1
```

`salt.modules.parted.`**`get_block_device`**()
Retrieve a list of disk devices

New in version 2014.7.0.

CLI Example:

```
salt '*' partition.get_block_device
```

`salt.modules.parted.`**`get_id`**(*device*, *minor*)
Prints the system ID for the partition. Some typical values are:

```
 b: FAT32 (vfat)
 7: HPFS/NTFS
82: Linux Swap
83: Linux
8e: Linux LVM
fd: Linux RAID Auto
```

CLI Example:

```
salt '*' partition.get_id /dev/sda 1
```

salt.modules.parted.**list_**(*device*, *unit=None*)
partition.list device unit

Prints partition information of given <device>

CLI Examples:

```
salt '*' partition.list /dev/sda
salt '*' partition.list /dev/sda unit=s
salt '*' partition.list /dev/sda unit=kB
```

salt.modules.parted.**mkfs**(*device*, *fs_type*)
partition.mkfs device fs_type

Makes a file system <fs_type> on partition <device>, destroying all data that resides on that partition. <fs_type> must be one of ``ext2'', ``fat32'', ``fat16'', ``linux-swap'' or ``reiserfs'' (if libreiserfs is installed)

CLI Example:

```
salt '*' partition.mkfs /dev/sda2 fat32
```

salt.modules.parted.**mklabel**(*device*, *label_type*)
partition.mklabel device label_type

Create a new disklabel (partition table) of label_type. Type should be one of ``aix'', ``amiga'', ``bsd'', ``dvh'', ``gpt'', ``loop'', ``mac'', ``msdos'', ``pc98'', or ``sun''.

CLI Example:

```
salt '*' partition.mklabel /dev/sda msdos
```

salt.modules.parted.**mkpart**(*device*, *part_type*, *fs_type=None*, *start=None*, *end=None*)
partition.mkpart device part_type fs_type start end

Make a part_type partition for filesystem fs_type, beginning at start and ending at end (by default in megabytes). part_type should be one of ``primary'', ``logical'', or ``extended''.

CLI Examples:

```
salt '*' partition.mkpart /dev/sda primary fs_type=fat32 start=0 end=639
salt '*' partition.mkpart /dev/sda primary start=0 end=639
```

salt.modules.parted.**mkpartfs**(*device*, *part_type*, *fs_type*, *start*, *end*)
partition.mkpartfs device part_type fs_type start end

Make a <part_type> partition with a new filesystem of <fs_type>, beginning at <start> and ending at <end> (by default in megabytes). <part_type> should be one of ``primary'', ``logical'', or ``extended''. <fs_type> must be one of ``ext2'', ``fat32'', ``fat16'', ``linux-swap'' or ``reiserfs'' (if libreiserfs is installed)

CLI Example:

```
salt '*' partition.mkpartfs /dev/sda logical ext2 440 670
```

salt.modules.parted.**name**(*device*, *partition*, *name*)
partition.name device partition name

Set the name of partition to name. This option works only on Mac, PC98, and GPT disklabels. The name can be placed in quotes, if necessary.

CLI Example:

```
salt '*' partition.name /dev/sda 1 'My Documents'
```

salt.modules.parted.**part_list**(*device*, *unit=None*)

Deprecated. Calls partition.list.

CLI Examples:

```
salt '*' partition.part_list /dev/sda
salt '*' partition.part_list /dev/sda unit=s
salt '*' partition.part_list /dev/sda unit=kB
```

salt.modules.parted.**probe**(**devices*, ***kwargs*)

Ask the kernel to update its local partition data. When no args are specified all block devices are tried.

Caution: Generally only works on devices with no mounted partitions and may take a long time to return if specified devices are in use.

CLI Examples:

```
salt '*' partition.probe
salt '*' partition.probe /dev/sda
salt '*' partition.probe /dev/sda /dev/sdb
```

salt.modules.parted.**rescue**(*device*, *start*, *end*)

partition.rescue device start end

Rescue a lost partition that was located somewhere between start and end. If a partition is found, parted will ask if you want to create an entry for it in the partition table.

CLI Example:

```
salt '*' partition.rescue /dev/sda 0 8056
```

salt.modules.parted.**resize**(*device*, *minor*, *start*, *end*)

partition.resize device minor, start, end

Resizes the partition with number <minor>. The partition will start <start> from the beginning of the disk, and end <end> from the beginning of the disk. resize never changes the minor number. Extended partitions can be resized, so long as the new extended partition completely contains all logical partitions.

CLI Example:

```
salt '*' partition.resize /dev/sda 3 200 850
```

salt.modules.parted.**rm**(*device*, *minor*)

partition.rm device minor

Removes the partition with number <minor>.

CLI Example:

```
salt '*' partition.rm /dev/sda 5
```

salt.modules.parted.**set_**(*device*, *minor*, *flag*, *state*)

partition.set device minor flag state

Changes a flag on the partition with number <minor>. A flag can be either ``on'' or ``off''. Some or all of these flags will be available, depending on what disk label you are using.

CLI Example:

```
salt '*' partition.set /dev/sda 1 boot on
```

salt.modules.parted.**set_id**(*device*, *minor*, *system_id*)
Sets the system ID for the partition. Some typical values are:

```
 b: FAT32 (vfat)
 7: HPFS/NTFS
82: Linux Swap
83: Linux
8e: Linux LVM
fd: Linux RAID Auto
```

CLI Example:

```
salt '*' partition.set_id /dev/sda 1 83
```

salt.modules.parted.**system_types**()
List the system types that are supported by the installed version of sfdisk

CLI Example:

```
salt '*' partition.system_types
```

salt.modules.parted.**toggle**(*device*, *partition*, *flag*)
partition.toggle device partition flag

Toggle the state of <flag> on <partition>

CLI Example:

```
salt '*' partition.name /dev/sda 1 boot
```

22.16.163 salt.modules.pecl

Manage PHP pecl extensions.

salt.modules.pecl.**install**(*pecls*, *defaults=False*, *force=False*, *preferred_state='stable'*)
New in version 0.17.0.

Installs one or several pecl extensions.

pecls The pecl extensions to install.

defaults Use default answers for extensions such as pecl_http which ask questions before installation. Without this option, the pecl.installed state will hang indefinitely when trying to install these extensions.

force Whether to force the installed version or not

CLI Example:

```
salt '*' pecl.install fuse
```

salt.modules.pecl.**list_**(*channel=None*)
List installed pecl extensions.

CLI Example:

```
salt '*' pecl.list
```

salt.modules.pecl.**uninstall**(*pecls*)
Uninstall one or several pecl extensions.

pecls The pecl extensions to uninstall.

CLI Example:

```
salt '*' pecl.uninstall fuse
```

salt.modules.pecl.**update**(*pecls*)
Update one or several pecl extensions.

pecls The pecl extensions to update.

CLI Example:

```
salt '*' pecl.update fuse
```

22.16.164 salt.modules.pillar

Extract the pillar data for this minion

salt.modules.pillar.**ext**(*external*, *pillar=None*)
Generate the pillar and apply an explicit external pillar

CLI Example:

pillar [None] If specified, allows for a dictionary of pillar data to be made available to pillar and ext_pillar rendering. These pillar variables will also override any variables of the same name in pillar or ext_pillar.

New in version 2015.5.0.

```
salt '*' pillar.ext '{libvirt: _}'
```

salt.modules.pillar.**get**(*key*, *default=<type `exceptions.KeyError'>*, *merge=False*, *delimiter=':'*)
New in version 0.14.

Attempt to retrieve the named value from pillar, if the named value is not available return the passed default. The default return is an empty string except __opts__['PILLAR_RAISE_ON_MISSING'] is set to True, in which case a KeyError will be raised.

If the merge parameter is set to `True`, the default will be recursively merged into the returned pillar data.

The value can also represent a value in a nested dict using a ":" delimiter for the dict. This means that if a dict in pillar looks like this:

```
{'pkg': {'apache': 'httpd'}}
```

To retrieve the value associated with the apache key in the pkg dict this key can be passed:

```
pkg:apache
```

merge Specify whether or not the retrieved values should be recursively merged into the passed default.

New in version 2014.7.0.

delimiter Specify an alternate delimiter to use when traversing a nested dict

New in version 2014.7.0.

CLI Example:

```
salt '*' pillar.get pkg:apache
```

salt.modules.pillar.**item**(**args*, ***kwargs*)

New in version 0.16.2.

Return one or more pillar entries

pillar [none] if specified, allows for a dictionary of pillar data to be made available to pillar and ext_pillar rendering. these pillar variables will also override any variables of the same name in pillar or ext_pillar.

New in version 2015.5.0.

CLI Examples:

```
salt '*' pillar.item foo
salt '*' pillar.item foo bar baz
```

salt.modules.pillar.**items**(**args*, ***kwargs*)

Calls the master for a fresh pillar and generates the pillar data on the fly

Contrast with `raw()` which returns the pillar data that is currently loaded into the minion.

pillar [none] if specified, allows for a dictionary of pillar data to be made available to pillar and ext_pillar rendering. these pillar variables will also override any variables of the same name in pillar or ext_pillar.

New in version 2015.5.0.

CLI Example:

```
salt '*' pillar.items
```

salt.modules.pillar.**raw**(*key=None*)

Return the raw pillar data that is currently loaded into the minion.

Contrast with `items()` which calls the master to fetch the most up-to-date Pillar.

CLI Example:

```
salt '*' pillar.raw
```

With the optional key argument, you can select a subtree of the pillar raw data.:

```
salt '*' pillar.raw key='roles'
```

22.16.165 salt.modules.pip

Install Python packages with pip to either the system or a virtualenv

Windows Support

New in version 2014.7.4.

Salt now uses a portable python. As a result the entire pip module is now functional on the salt installation itself. You can pip install dependencies for your custom modules. You can even upgrade salt itself using pip. For this to work properly, you must specify the Current Working Directory (`cwd`) and the Pip Binary (`bin_env`) salt should use.

For example, the following command will list all software installed using pip to your current salt environment:

```
salt <minion> pip.list cwd='C:\salt\bin\Scripts' bin_env='C:\salt\bin\Scripts\pip.exe'
```

Specifying the `cwd` and `bin_env` options ensures you're modifying the salt environment. If these are omitted, it will default to the local installation of python. If python is not installed locally it will fail saying it couldn't find pip.

State File Support

This functionality works in states as well. If you need to pip install colorama with a state, for example, the following will work:

```
install_colorama:
  pip.installed:
    - name: colorama
    - cwd: 'C:\salt\bin\scripts'
    - bin_env: 'C:\salt\bin\scripts\pip.exe'
    - upgrade: True
```

Upgrading Salt using Pip

You can now update salt using pip to any version from the 2014.7 branch forward. Previous version require recompiling some of the dependencies which is painful in windows.

To do this you just use pip with git to update to the version you want and then restart the service. Here is a sample state file that upgrades salt to the head of the 2015.5 branch:

```
install_salt:
  pip.installed:
    - cwd: 'C:\salt\bin\scripts'
    - bin_env: 'C:\salt\bin\scripts\pip.exe'
    - editable: git+https://github.com/saltstack/salt@2015.5#egg=salt
    - upgrade: True

restart_service:
  service.running:
    - name: salt-minion
    - enable: True
    - watch:
      - pip: install_salt
```

Note: If you're having problems, you might try doubling the back slashes. For example, cwd: `C:\salt\bin\scripts'. Sometimes python thinks the single back slash is an escape character.

`salt.modules.pip.`**`freeze`**(*bin_env=None*, *user=None*, *cwd=None*, *use_vt=False*)

Return a list of installed packages either globally or in the specified virtualenv

bin_env path to pip bin or path to virtualenv. If doing an uninstall from the system python and want to use a specific pip bin (pip-2.7, pip-2.6, etc..) just specify the pip bin you want. If uninstalling from a virtualenv, just use the path to the virtualenv (/home/code/path/to/virtualenv/)

user The user under which to run pip

cwd Current working directory to run pip from

CLI Example:

```
salt '*' pip.freeze /home/code/path/to/virtualenv/
```

salt.modules.pip.**install**(*pkgs=None*, *requirements=None*, *env=None*, *bin_env=None*, *use_wheel=False*, *no_use_wheel=False*, *log=None*, *proxy=None*, *timeout=None*, *editable=None*, *find_links=None*, *index_url=None*, *extra_index_url=None*, *no_index=False*, *mirrors=None*, *build=None*, *target=None*, *download=None*, *download_cache=None*, *source=None*, *upgrade=False*, *force_reinstall=False*, *ignore_installed=False*, *exists_action=None*, *no_deps=False*, *no_install=False*, *no_download=False*, *global_options=None*, *install_options=None*, *user=None*, *no_chown=False*, *cwd=None*, *activate=False*, *pre_releases=False*, *cert=None*, *allow_all_external=False*, *allow_external=None*, *allow_unverified=None*, *process_dependency_links=False*, *__env__=None*, *saltenv='base'*, *env_vars=None*, *use_vt=False*)

Install packages with pip

Install packages individually or from a pip requirements file. Install packages globally or to a virtualenv.

pkgs Comma separated list of packages to install

requirements Path to requirements

bin_env Path to pip bin or path to virtualenv. If doing a system install, and want to use a specific pip bin (pip-2.7, pip-2.6, etc..) just specify the pip bin you want. If installing into a virtualenv, just use the path to the virtualenv (/home/code/path/to/virtualenv/)

env Deprecated, use bin_env now

use_wheel Prefer wheel archives (requires pip>=1.4)

no_use_wheel Force to not use wheel archives (requires pip>=1.4)

log Log file where a complete (maximum verbosity) record will be kept

proxy Specify a proxy in the form user:passwd@proxy.server:port. Note that the user:password@ is optional and required only if you are behind an authenticated proxy. If you provide user@proxy.server:port then you will be prompted for a password.

timeout Set the socket timeout (default 15 seconds)

editable install something editable (i.e. git+https://github.com/worldcompany/djangoembed.git#egg=djangoembed)

find_links URL to look for packages at

index_url Base URL of Python Package Index

extra_index_url Extra URLs of package indexes to use in addition to `index_url`

no_index Ignore package index

mirrors Specific mirror URL(s) to query (automatically adds --use-mirrors)

build Unpack packages into `build` dir

target Install packages into `target` dir

download Download packages into `download` instead of installing them

download_cache Cache downloaded packages in `download_cache` dir

source Check out `editable` packages into `source` dir

upgrade Upgrade all packages to the newest available version

force_reinstall When upgrading, reinstall all packages even if they are already up-to-date.

ignore_installed Ignore the installed packages (reinstalling instead)

exists_action Default action when a path already exists: (s)witch, (i)gnore, (w)ipe, (b)ackup

no_deps Ignore package dependencies

no_install Download and unpack all packages, but don't actually install them

no_download Don't download any packages, just install the ones already downloaded (completes an install run with --no-install)

install_options Extra arguments to be supplied to the setup.py install command (use like --install-option="--install- scripts=/usr/local/bin"). Use multiple --install- option options to pass multiple options to setup.py install. If you are using an option with a directory path, be sure to use absolute path.

global_options Extra global options to be supplied to the setup.py call before the install command.

user The user under which to run pip

no_chown When user is given, do not attempt to copy and chown a requirements file

cwd Current working directory to run pip from

activate Activates the virtual environment, if given via bin_env, before running install.

Deprecated since version 2014.7.2: If *bin_env* is given, pip will already be sourced from that virualenv, making *activate* effectively a noop.

pre_releases Include pre-releases in the available versions

cert Provide a path to an alternate CA bundle

allow_all_external Allow the installation of all externally hosted files

allow_external Allow the installation of externally hosted files (comma separated list)

allow_unverified Allow the installation of insecure and unverifiable files (comma separated list)

process_dependency_links Enable the processing of dependency links

use_vt Use VT terminal emulation (see ouptut while installing)

env_vars Set environment variables that some builds will depend on. For example, a Python C-module may have a Makefile that needs INCLUDE_PATH set to pick up a header file while compiling.

CLI Example:

```
salt '*' pip.install <package name>,<package2 name>
salt '*' pip.install requirements=/path/to/requirements.txt
salt '*' pip.install <package name> bin_env=/path/to/virtualenv
salt '*' pip.install <package name> bin_env=/path/to/pip_bin
```

Complicated CLI example:

```
salt '*' pip.install markdown,django                    editable=git+https://github.com/worldcompan
```

`salt.modules.pip.`**`list_`**(*prefix=None*, *bin_env=None*, *user=None*, *cwd=None*)

Filter list of installed apps from `freeze` and check to see if `prefix` exists in the list of packages installed.

CLI Example:

```
salt '*' pip.list salt
```

`salt.modules.pip.`**`list_upgrades`**(*bin_env=None*, *user=None*, *cwd=None*)

Check whether or not an upgrade is available for all packages

CLI Example:

```
salt '*' pip.list_upgrades
```

salt.modules.pip.**uninstall**(*pkgs=None*, *requirements=None*, *bin_env=None*, *log=None*, *proxy=None*, *timeout=None*, *user=None*, *no_chown=False*, *cwd=None*, *__env__=None*, *saltenv='base'*, *use_vt=False*)

Uninstall packages with pip

Uninstall packages individually or from a pip requirements file. Uninstall packages globally or from a virtualenv.

pkgs comma separated list of packages to install

requirements path to requirements.

bin_env path to pip bin or path to virtualenv. If doing an uninstall from the system python and want to use a specific pip bin (pip-2.7, pip-2.6, etc..) just specify the pip bin you want. If uninstalling from a virtualenv, just use the path to the virtualenv (/home/code/path/to/virtualenv/)

log Log file where a complete (maximum verbosity) record will be kept

proxy Specify a proxy in the form user:passwd@proxy.server:port. Note that the user:password@ is optional and required only if you are behind an authenticated proxy. If you provide user@proxy.server:port then you will be prompted for a password.

timeout Set the socket timeout (default 15 seconds)

user The user under which to run pip

no_chown When user is given, do not attempt to copy and chown a requirements file

cwd Current working directory to run pip from

use_vt Use VT terminal emulation (see ouptut while installing)

CLI Example:

```
salt '*' pip.uninstall <package name>,<package2 name>
salt '*' pip.uninstall requirements=/path/to/requirements.txt
salt '*' pip.uninstall <package name> bin_env=/path/to/virtualenv
salt '*' pip.uninstall <package name> bin_env=/path/to/pip_bin
```

salt.modules.pip.**upgrade**(*bin_env=None*, *user=None*, *cwd=None*, *use_vt=False*)

New in version 2015.5.0.

Upgrades outdated pip packages

Returns a dict containing the changes.

{`<package>': {`old': `<old-version>', `new': `<new-version>'}}

CLI Example:

```
salt '*' pip.upgrade
```

salt.modules.pip.**upgrade_available**(*pkg*, *bin_env=None*, *user=None*, *cwd=None*)

New in version 2015.5.0.

Check whether or not an upgrade is available for a given package

CLI Example:

```
salt '*' pip.upgrade_available <package name>
```

salt.modules.pip.**version**(*bin_env=None*)

New in version 0.17.0.

Returns the version of pip. Use `bin_env` to specify the path to a virtualenv and get the version of pip in that virtualenv.

If unable to detect the pip version, returns `None`.

CLI Example:

```
salt '*' pip.version
```

22.16.166 salt.modules.pkg_resource

Resources needed by pkg providers

`salt.modules.pkg_resource.`**`add_pkg`**(*pkgs*, *name*, *version*)

Add a package to a dict of installed packages.

CLI Example:

```
salt '*' pkg_resource.add_pkg '{}' bind 9
```

`salt.modules.pkg_resource.`**`check_extra_requirements`**(*pkgname*, *pkgver*)

Check if the installed package already has the given requirements. This function will simply try to call ``pkg.check_extra_requirements''.

CLI Example:

```
salt '*' pkg_resource.check_extra_requirements <pkgname> <extra_requirements>
```

`salt.modules.pkg_resource.`**`pack_sources`**(*sources*)

Accepts list of dicts (or a string representing a list of dicts) and packs the key/value pairs into a single dict.

`'[{"foo": "salt://foo.rpm"}, {"bar": "salt://bar.rpm"}]'` would become `{"foo": "salt://foo.rpm", "bar": "salt://bar.rpm"}`

CLI Example:

```
salt '*' pkg_resource.pack_sources '[{"foo": "salt://foo.rpm"}, {"bar": "salt://bar.rpm"}]'
```

`salt.modules.pkg_resource.`**`parse_targets`**(*name=None*, *pkgs=None*, *sources=None*, *saltenv='base'*, *normalize=True*, ***kwargs*)

Parses the input to pkg.install and returns back the package(s) to be installed. Returns a list of packages, as well as a string noting whether the packages are to come from a repository or a binary package.

CLI Example:

```
salt '*' pkg_resource.parse_targets
```

`salt.modules.pkg_resource.`**`sort_pkglist`**(*pkgs*)

Accepts a dict obtained from pkg.list_pkgs() and sorts in place the list of versions for any packages that have multiple versions installed, so that two package lists can be compared to one another.

CLI Example:

```
salt '*' pkg_resource.sort_pkglist '["3.45", "2.13"]'
```

`salt.modules.pkg_resource.`**`stringify`**(*pkgs*)

Takes a dict of package name/version information and joins each list of installed versions into a string.

CLI Example:

```
    salt '*' pkg_resource.stringify 'vim: 7.127'
```

salt.modules.pkg_resource.**version**(**names*, ***kwargs*)

Common interface for obtaining the version of installed packages.

CLI Example:

```
    salt '*' pkg_resource.version vim
    salt '*' pkg_resource.version foo bar baz
    salt '*' pkg_resource.version 'python*'
```

salt.modules.pkg_resource.**version_clean**(*version*)

Clean the version string removing extra data. This function will simply try to call `pkg.version_clean`.

CLI Example:

```
    salt '*' pkg_resource.version_clean <version_string>
```

22.16.167 salt.modules.pkgin

Package support for pkgin based systems, inspired from freebsdpkg module

salt.modules.pkgin.**available_version**(**names*, ***kwargs*)

Return the latest version of the named package available for upgrade or installation.

If the latest version of a given package is already installed, an empty string will be returned for that package.

CLI Example:

```
    salt '*' pkg.latest_version <package name>
    salt '*' pkg.latest_version <package1> <package2> ...
```

salt.modules.pkgin.**file_dict**(*package*)

List the files that belong to a package.

CLI Examples:

```
    salt '*' pkg.file_list nginx
```

salt.modules.pkgin.**file_list**(*package*)

List the files that belong to a package.

CLI Examples:

```
    salt '*' pkg.file_list nginx
```

salt.modules.pkgin.**install**(*name=None*, *refresh=False*, *fromrepo=None*, *pkgs=None*, *sources=None*, ***kwargs*)

Install the passed package

name The name of the package to be installed.

refresh Whether or not to refresh the package database before installing.

fromrepo Specify a package repository to install from.

Multiple Package Installation Options:

pkgs A list of packages to install from a software repository. Must be passed as a python list.

CLI Example:

```
salt '*' pkg.install pkgs='["foo","bar"]'
```

sources A list of packages to install. Must be passed as a list of dicts, with the keys being package names, and the values being the source URI or local path to the package.

CLI Example:

```
salt '*' pkg.install sources='[{"foo": "salt://foo.deb"},{"bar": "salt://bar.deb"}]'
```

Return a dict containing the new package names and versions:

```
{'<package>': {'old': '<old-version>',
               'new': '<new-version>'}}
```

CLI Example:

```
salt '*' pkg.install <package name>
```

salt.modules.pkgin.**latest_version**(**names*, ***kwargs*)

Return the latest version of the named package available for upgrade or installation.

If the latest version of a given package is already installed, an empty string will be returned for that package.

CLI Example:

```
salt '*' pkg.latest_version <package name>
salt '*' pkg.latest_version <package1> <package2> ...
```

salt.modules.pkgin.**list_pkgs**(*versions_as_list=False*, ***kwargs*)

List the packages currently installed as a dict:

```
{'<package_name>': '<version>'}
```

CLI Example:

```
salt '*' pkg.list_pkgs
```

salt.modules.pkgin.**purge**(*name=None*, *pkgs=None*, ***kwargs*)

Package purges are not supported, this function is identical to `remove()`.

name The name of the package to be deleted.

Multiple Package Options:

pkgs A list of packages to delete. Must be passed as a python list. The `name` parameter will be ignored if this option is passed.

New in version 0.16.0.

Returns a dict containing the changes.

CLI Example:

```
salt '*' pkg.purge <package name>
salt '*' pkg.purge <package1>,<package2>,<package3>
salt '*' pkg.purge pkgs='["foo", "bar"]'
```

salt.modules.pkgin.**refresh_db**()

Use pkg update to get latest pkg_summary

CLI Example:

```
salt '*' pkg.refresh_db
```

salt.modules.pkgin.**remove**(*name=None, pkgs=None, **kwargs*)

name The name of the package to be deleted.

Multiple Package Options:

pkgs A list of packages to delete. Must be passed as a python list. The `name` parameter will be ignored if this option is passed.

New in version 0.16.0.

Returns a list containing the removed packages.

CLI Example:

```
salt '*' pkg.remove <package name>
salt '*' pkg.remove <package1>,<package2>,<package3>
salt '*' pkg.remove pkgs='["foo", "bar"]'
```

salt.modules.pkgin.**search**(*pkg_name*)

Searches for an exact match using pkgin ^package$

CLI Example:

```
salt '*' pkg.search 'mysql-server'
```

salt.modules.pkgin.**upgrade**()

Run pkg upgrade, if pkgin used. Otherwise do nothing

Return a dict containing the new package names and versions:

```
{'<package>': {'old': '<old-version>',
               'new': '<new-version>'}}
```

CLI Example:

```
salt '*' pkg.upgrade
```

salt.modules.pkgin.**version**(**names, **kwargs*)

Returns a string representing the package version or an empty string if not installed. If more than one package name is specified, a dict of name/version pairs is returned.

CLI Example:

```
salt '*' pkg.version <package name>
salt '*' pkg.version <package1> <package2> <package3> ...
```

22.16.168 salt.modules.pkgng

Support for `pkgng`, the new package manager for FreeBSD

Warning: This module has been completely rewritten. Up to and including version 0.17.x, it was available as the `pkgng` module, (`pkgng.install`, `pkgng.delete`, etc.), but moving forward this module will no longer be available as `pkgng`, as it will behave like a normal Salt `pkg` provider. The documentation below should not be considered to apply to this module in versions <= 0.17.x. If your minion is running a 0.17.x release or older, then the documentation for this module can be viewed using the `sys.doc` function:

```
salt bsdminion sys.doc pkgng
```

This module provides an interface to `pkg(8)`. It acts as the default package provider for FreeBSD 10 and newer. For FreeBSD hosts which have been upgraded to use pkgng, you will need to override the `pkg` provider by setting the `providers` parameter in your Minion config file, in order to use this module to manage packages, like so:

```
providers:
  pkg: pkgng
```

`salt.modules.pkgng.`**`audit`**(*jail=None*, *chroot=None*)

Audits installed packages against known vulnerabilities

CLI Example:

```
salt '*' pkg.audit
```

jail Audit packages within the specified jail

CLI Example:

```
salt '*' pkg.audit jail=<jail name or id>
```

chroot Audit packages within the specified chroot (ignored if `jail` is specified)

CLI Example:

```
salt '*' pkg.audit chroot=/path/to/chroot
```

`salt.modules.pkgng.`**`autoremove`**(*jail=None*, *chroot=None*, *dryrun=False*)

Delete packages which were automatically installed as dependencies and are not required anymore.

dryrun Dry-run mode. The list of changes to packages is always printed, but no changes are actually made.

CLI Example:

```
salt '*' pkg.autoremove
salt '*' pkg.autoremove jail=<jail name or id>
salt '*' pkg.autoremove dryrun=True
salt '*' pkg.autoremove jail=<jail name or id> dryrun=True
```

`salt.modules.pkgng.`**`backup`**(*file_name*, *jail=None*, *chroot=None*)

Export installed packages into yaml+mtree file

CLI Example:

```
salt '*' pkg.backup /tmp/pkg
```

jail Backup packages from the specified jail. Note that this will run the command within the jail, and so the path to the backup file will be relative to the root of the jail

CLI Example:

```
salt '*' pkg.backup /tmp/pkg jail=<jail name or id>
```

chroot Backup packages from the specified chroot (ignored if `jail` is specified). Note that this will run the command within the chroot, and so the path to the backup file will be relative to the root of the chroot.

CLI Example:

```
salt '*' pkg.backup /tmp/pkg chroot=/path/to/chroot
```

salt.modules.pkgng.**check**(*jail=None*, *chroot=None*, *depends=False*, *recompute=False*, *checksum=False*)

Sanity checks installed packages

jail Perform the sanity check in the specified jail

CLI Example:

```
salt '*' pkg.check jail=<jail name or id>
```

chroot Perform the sanity check in the specified chroot (ignored if `jail` is specified)

CLI Example:

```
salt '*' pkg.check chroot=/path/to/chroot
```

Of the below, at least one must be set to `True`.

depends Check for and install missing dependencies.

CLI Example:

```
salt '*' pkg.check recompute=True
```

recompute Recompute sizes and checksums of installed packages.

CLI Example:

```
salt '*' pkg.check depends=True
```

checksum Find invalid checksums for installed packages.

CLI Example:

```
salt '*' pkg.check checksum=True
```

salt.modules.pkgng.**clean**(*jail=None*, *chroot=None*)

Cleans the local cache of fetched remote packages

CLI Example:

```
salt '*' pkg.clean
salt '*' pkg.clean jail=<jail name or id>
salt '*' pkg.clean chroot=/path/to/chroot
```

salt.modules.pkgng.**fetch**(*name*, *jail=None*, *chroot=None*, *fetch_all=False*, *quiet=False*, *fromrepo=None*, *glob=True*, *regex=False*, *pcre=False*, *local=False*, *depends=False*)

Fetches remote packages

CLI Example:

```
salt '*' pkg.fetch <package name>
```

jail Fetch package in the specified jail

CLI Example:

```
salt '*' pkg.fetch <package name> jail=<jail name or id>
```

chroot Fetch package in the specified chroot (ignored if `jail` is specified)

CLI Example:

```
salt '*' pkg.fetch <package name> chroot=/path/to/chroot
```

fetch_all Fetch all packages.

CLI Example:

```
salt '*' pkg.fetch <package name> fetch_all=True
```

quiet Quiet mode. Show less output.

CLI Example:

```
salt '*' pkg.fetch <package name> quiet=True
```

fromrepo Fetches packages from the given repo if multiple repo support is enabled. See `pkg.conf(5)`.

CLI Example:

```
salt '*' pkg.fetch <package name> fromrepo=repo
```

glob Treat pkg_name as a shell glob pattern.

CLI Example:

```
salt '*' pkg.fetch <package name> glob=True
```

regex Treat pkg_name as a regular expression.

CLI Example:

```
salt '*' pkg.fetch <regular expression> regex=True
```

pcre Treat pkg_name is an extended regular expression.

CLI Example:

```
salt '*' pkg.fetch <extended regular expression> pcre=True
```

local Skip updating the repository catalogs with pkg-update(8). Use the local cache only.

CLI Example:

```
salt '*' pkg.fetch <package name> local=True
```

depends Fetch the package and its dependencies as well.

CLI Example:

```
salt '*' pkg.fetch <package name> depends=True
```

salt.modules.pkgng.**install**(*name=None*, *fromrepo=None*, *pkgs=None*, *sources=None*, *jail=None*, *chroot=None*, *orphan=False*, *force=False*, *glob=False*, *local=False*, *dryrun=False*, *quiet=False*, *reinstall_requires=False*, *regex=False*, *pcre=False*, ***kwargs*)

Install package(s) from a repository

name The name of the package to install

CLI Example:

```
salt '*' pkg.install <package name>
```

jail Install the package into the specified jail

chroot Install the package into the specified chroot (ignored if `jail` is specified)

orphan Mark the installed package as orphan. Will be automatically removed if no other packages depend on them. For more information please refer to `pkg-autoremove(8)`.

CLI Example:

```
salt '*' pkg.install <package name> orphan=True
```

force Force the reinstallation of the package if already installed.

CLI Example:

```
salt '*' pkg.install <package name> force=True
```

glob Treat the package names as shell glob patterns.

CLI Example:

```
salt '*' pkg.install <package name> glob=True
```

local Do not update the repository catalogs with `pkg-update(8)`. A value of `True` here is equivalent to using the `-U` flag with `pkg install`.

CLI Example:

```
salt '*' pkg.install <package name> local=True
```

dryrun Dru-run mode. The list of changes to packages is always printed, but no changes are actually made.

CLI Example:

```
salt '*' pkg.install <package name> dryrun=True
```

quiet Force quiet output, except when dryrun is used, where pkg install will always show packages to be installed, upgraded or deleted.

CLI Example:

```
salt '*' pkg.install <package name> quiet=True
```

reinstall_requires When used with force, reinstalls any packages that require the given package.

CLI Example:

```
salt '*' pkg.install <package name> reinstall_requires=True force=True
```

Changed in version 2014.7.0: `require` kwarg renamed to `reinstall_requires`

fromrepo In multi-repo mode, override the pkg.conf ordering and only attempt to download packages from the named repository.

CLI Example:

```
salt '*' pkg.install <package name> fromrepo=repo
```

regex Treat the package names as a regular expression

CLI Example:

```
salt '*' pkg.install <regular expression> regex=True
```

pcre Treat the package names as extended regular expressions.

CLI Example:

```
salt '*' pkg.install <extended regular expression> pcre=True
```

salt.modules.pkgng.**latest_version**(**names*, ***kwargs*)

Return the latest version of the named package available for upgrade or installation. If more than one package name is specified, a dict of name/version pairs is returned.

If the latest version of a given package is already installed, an empty string will be returned for that package.

CLI Example:

```
salt '*' pkg.latest_version <package name>
salt '*' pkg.latest_version <package name> jail=<jail name or id>
salt '*' pkg.latest_version <package name> chroot=/path/to/chroot
```

salt.modules.pkgng.**list_pkgs**(*versions_as_list=False*, *jail=None*, *chroot=None*, *with_origin=False*, ***kwargs*)

List the packages currently installed as a dict:

```
{'<package_name>': '<version>'}
```

jail List the packages in the specified jail

chroot List the packages in the specified chroot (ignored if `jail` is specified)

with_origin [False] Return a nested dictionary containing both the origin name and version for each installed package.

New in version 2014.1.0.

CLI Example:

```
salt '*' pkg.list_pkgs
salt '*' pkg.list_pkgs jail=<jail name or id>
salt '*' pkg.list_pkgs chroot=/path/to/chroot
```

salt.modules.pkgng.**parse_config**(*file_name='/usr/local/etc/pkg.conf'*)

Return dict of uncommented global variables.

CLI Example:

```
salt '*' pkg.parse_config
```

`NOTE:` not working properly right now

salt.modules.pkgng.**refresh_db**(*jail=None*, *chroot=None*, *force=False*)
Refresh PACKAGESITE contents

Note: This function can accessed using `pkg.update` in addition to `pkg.refresh_db`, to more closely match the CLI usage of `pkg(8)`.

CLI Example:

```
salt '*' pkg.refresh_db
```

jail Refresh the pkg database within the specified jail

chroot Refresh the pkg database within the specified chroot (ignored if `jail` is specified)

force Force a full download of the repository catalog without regard to the respective ages of the local and remote copies of the catalog.

CLI Example:

```
salt '*' pkg.refresh_db force=True
```

salt.modules.pkgng.**remove**(*name=None*, *pkgs=None*, *jail=None*, *chroot=None*, *all_installed=False*, *force=False*, *glob=False*, *dryrun=False*, *recurse=False*, *regex=False*, *pcre=False*, ***kwargs*)
Remove a package from the database and system

Note: This function can accessed using `pkg.delete` in addition to `pkg.remove`, to more closely match the CLI usage of `pkg(8)`.

---

name The package to remove

CLI Example:

```
salt '*' pkg.remove <package name>
```

jail Delete the package from the specified jail

chroot Delete the package from the specified chroot (ignored if `jail` is specified)

all_installed Deletes all installed packages from the system and empties the database. USE WITH CAUTION!

CLI Example:

```
salt '*' pkg.remove all all_installed=True force=True
```

force Forces packages to be removed despite leaving unresolved dependencies.

CLI Example:

```
salt '*' pkg.remove <package name> force=True
```

glob Treat the package names as shell glob patterns.

CLI Example:

```
salt '*' pkg.remove <package name> glob=True
```

dryrun Dry run mode. The list of packages to delete is always printed, but no packages are actually deleted.

CLI Example:

```
salt '*' pkg.remove <package name> dryrun=True
```

recurse Delete all packages that require the listed package as well.

CLI Example:

```
salt '*' pkg.remove <package name> recurse=True
```

regex Treat the package names as regular expressions.

CLI Example:

```
salt '*' pkg.remove <regular expression> regex=True
```

pcre Treat the package names as extended regular expressions.

CLI Example:

```
salt '*' pkg.remove <extended regular expression> pcre=True
```

`salt.modules.pkgng.`**`restore`**(*file_name*, *jail=None*, *chroot=None*)

Reads archive created by pkg backup -d and recreates the database.

CLI Example:

```
salt '*' pkg.restore /tmp/pkg
```

jail Restore database to the specified jail. Note that this will run the command within the jail, and so the path to the file from which the pkg database will be restored is relative to the root of the jail.

CLI Example:

```
salt '*' pkg.restore /tmp/pkg jail=<jail name or id>
```

chroot Restore database to the specified chroot (ignored if `jail` is specified). Note that this will run the command within the chroot, and so the path to the file from which the pkg database will be restored is relative to the root of the chroot.

CLI Example:

```
salt '*' pkg.restore /tmp/pkg chroot=/path/to/chroot
```

`salt.modules.pkgng.`**`search`**(*name*, *jail=None*, *chroot=None*, *exact=False*, *glob=False*, *regex=False*, *pcre=False*, *comment=False*, *desc=False*, *full=False*, *depends=False*, *size=False*, *quiet=False*, *origin=False*, *prefix=False*)

Searches in remote package repositories

CLI Example:

```
salt '*' pkg.search pattern
```

jail Perform the search using the `pkg.conf(5)` from the specified jail

CLI Example:

```
salt '*' pkg.search pattern jail=<jail name or id>
```

chroot Perform the search using the `pkg.conf(5)` from the specified chroot (ignored if `jail` is specified)

CLI Example:

```
salt '*' pkg.search pattern chroot=/path/to/chroot
```

exact Treat pattern as exact pattern.

CLI Example:

```
salt '*' pkg.search pattern exact=True
```

glob Treat pattern as a shell glob pattern.

CLI Example:

```
salt '*' pkg.search pattern glob=True
```

regex Treat pattern as a regular expression.

CLI Example:

```
salt '*' pkg.search pattern regex=True
```

pcre Treat pattern as an extended regular expression.

CLI Example:

```
salt '*' pkg.search pattern pcre=True
```

comment Search for pattern in the package comment one-line description.

CLI Example:

```
salt '*' pkg.search pattern comment=True
```

desc Search for pattern in the package description.

CLI Example:

```
salt '*' pkg.search pattern desc=True
```

full Displays full information about the matching packages.

CLI Example:

```
salt '*' pkg.search pattern full=True
```

depends Displays the dependencies of pattern.

CLI Example:

```
salt '*' pkg.search pattern depends=True
```

size Displays the size of the package

CLI Example:

```
salt '*' pkg.search pattern size=True
```

quiet Be quiet. Prints only the requested information without displaying many hints.

CLI Example:

```
salt '*' pkg.search pattern quiet=True
```

origin Displays pattern origin.

CLI Example:

```
salt '*' pkg.search pattern origin=True
```

prefix Displays the installation prefix for each package matching pattern.

CLI Example:

```
salt '*' pkg.search pattern prefix=True
```

salt.modules.pkgng.**stats**(*local=False*, *remote=False*, *jail=None*, *chroot=None*)

Return pkgng stats.

CLI Example:

```
salt '*' pkg.stats
```

local Display stats only for the local package database.

CLI Example:

```
salt '*' pkg.stats local=True
```

remote Display stats only for the remote package database(s).

CLI Example:

```
salt '*' pkg.stats remote=True
```

jail Retrieve stats from the specified jail.

CLI Example:

```
salt '*' pkg.stats jail=<jail name or id>
salt '*' pkg.stats jail=<jail name or id> local=True
salt '*' pkg.stats jail=<jail name or id> remote=True
```

chroot Retrieve stats from the specified chroot (ignored if `jail` is specified).

CLI Example:

```
salt '*' pkg.stats chroot=/path/to/chroot
salt '*' pkg.stats chroot=/path/to/chroot local=True
salt '*' pkg.stats chroot=/path/to/chroot remote=True
```

salt.modules.pkgng.**update_package_site**(*new_url*)

Updates remote package repo URL, PACKAGESITE var to be exact.

Must use `http://`, `ftp://`, or `https://` protocol

CLI Example:

```
salt '*' pkg.update_package_site http://127.0.0.1/
```

salt.modules.pkgng.**updating**(*name*, *jail=None*, *chroot=None*, *filedate=None*, *filename=None*)

` Displays UPDATING entries of software packages

CLI Example:

```
salt '*' pkg.updating foo
```

jail Perform the action in the specified jail

CLI Example:

```
salt '*' pkg.updating foo jail=<jail name or id>
```

chroot Perform the action in the specified chroot (ignored if `jail` is specified)

CLI Example:

```
salt '*' pkg.updating foo chroot=/path/to/chroot
```

filedate Only entries newer than date are shown. Use a YYYYMMDD date format.

CLI Example:

```
salt '*' pkg.updating foo filedate=20130101
```

filename Defines an alternative location of the UPDATING file.

CLI Example:

```
salt '*' pkg.updating foo filename=/tmp/UPDATING
```

`salt.modules.pkgng.`**`upgrade`**`(`**names, **kwargs*`)`

Upgrade named or all packages (run a `pkg upgrade`). If <package name> is ommitted, the operation is executed on all packages.

CLI Example:

```
salt '*' pkg.upgrade <package name>
```

jail Audit packages within the specified jail

CLI Example:

```
salt '*' pkg.upgrade <package name> jail=<jail name or id>
```

chroot Audit packages within the specified chroot (ignored if `jail` is specified)

CLI Example:

```
salt '*' pkg.upgrade <package name> chroot=/path/to/chroot
```

Any of the below options can also be used with `jail` or `chroot`.

force Force reinstalling/upgrading the whole set of packages.

CLI Example:

```
salt '*' pkg.upgrade <package name> force=True
```

local Do not update the repository catalogs with `pkg-update(8)`. A value of `True` here is equivalent to using the `-U` flag with `pkg upgrade`.

CLI Example:

```
salt '*' pkg.upgrade <package name> local=True
```

dryrun Dry-run mode: show what packages have updates available, but do not perform any upgrades. Repository catalogs will be updated as usual unless the local option is also given.

CLI Example:

```
salt '*' pkg.upgrade <package name> dryrun=True
```

salt.modules.pkgng.**version**(**names*, ***kwargs*)

Returns a string representing the package version or an empty string if not installed. If more than one package name is specified, a dict of name/version pairs is returned.

Note: This function can accessed using `pkg.info` in addition to `pkg.version`, to more closely match the CLI usage of `pkg(8)`.

---

jail Get package version information for the specified jail

chroot Get package version information for the specified chroot (ignored if `jail` is specified)

with_origin [False] Return a nested dictionary containing both the origin name and version for each specified package.

New in version 2014.1.0.

CLI Example:

```
salt '*' pkg.version <package name>
salt '*' pkg.version <package name> jail=<jail name or id>
salt '*' pkg.version <package1> <package2> <package3> ...
```

salt.modules.pkgng.**which**(*path*, *jail=None*, *chroot=None*, *origin=False*, *quiet=False*)

Displays which package installed a specific file

CLI Example:

```
salt '*' pkg.which <file name>
```

jail Perform the check in the specified jail

CLI Example:

```
salt '*' pkg.which <file name> jail=<jail name or id>
```

chroot Perform the check in the specified chroot (ignored if `jail` is specified)

CLI Example:

```
salt '*' pkg.which <file name> chroot=/path/to/chroot
```

origin Shows the origin of the package instead of name-version.

CLI Example:

```
salt '*' pkg.which <file name> origin=True
```

quiet Quiet output.

CLI Example:

```
salt '*' pkg.which <file name> quiet=True
```

22.16.169 salt.modules.pkgutil

Pkgutil support for Solaris

salt.modules.pkgutil.**install**(*name=None, refresh=False, version=None, pkgs=None, **kwargs*)

Install packages using the pkgutil tool.

CLI Example:

```
salt '*' pkg.install <package_name>
salt '*' pkg.install SMClgcc346
```

Multiple Package Installation Options:

pkgs A list of packages to install from OpenCSW. Must be passed as a python list.

CLI Example:

```
salt '*' pkg.install pkgs='["foo", "bar"]'
salt '*' pkg.install pkgs='["foo", {"bar": "1.2.3"}]'
```

Returns a dict containing the new package names and versions:

```
{'<package>': {'old': '<old-version>',
               'new': '<new-version>'}}
```

salt.modules.pkgutil.**latest_version**(**names, **kwargs*)

Return the latest version of the named package available for upgrade or installation. If more than one package name is specified, a dict of name/version pairs is returned.

If the latest version of a given package is already installed, an empty string will be returned for that package.

CLI Example:

```
salt '*' pkgutil.latest_version CSWpython
salt '*' pkgutil.latest_version <package1> <package2> <package3> ...
```

salt.modules.pkgutil.**list_pkgs**(*versions_as_list=False, **kwargs*)

List the packages currently installed as a dict:

```
{'<package_name>': '<version>'}
```

CLI Example:

```
salt '*' pkg.list_pkgs
salt '*' pkg.list_pkgs versions_as_list=True
```

salt.modules.pkgutil.**list_upgrades**(*refresh=True*)

List all available package upgrades on this system

CLI Example:

```
salt '*' pkgutil.list_upgrades
```

salt.modules.pkgutil.**purge**(*name=None, pkgs=None, **kwargs*)

Package purges are not supported, this function is identical to `remove()`.

name The name of the package to be deleted.

Multiple Package Options:

pkgs A list of packages to delete. Must be passed as a python list. The `name` parameter will be ignored if this option is passed.

New in version 0.16.0.

Returns a dict containing the changes.

CLI Example:

```
salt '*' pkg.purge <package name>
salt '*' pkg.purge <package1>,<package2>,<package3>
salt '*' pkg.purge pkgs='["foo", "bar"]'
```

salt.modules.pkgutil.**refresh_db**()

Updates the pkgutil repo database (pkgutil -U)

CLI Example:

```
salt '*' pkgutil.refresh_db
```

salt.modules.pkgutil.**remove**(*name=None*, *pkgs=None*, ***kwargs*)

Remove a package and all its dependencies which are not in use by other packages.

name The name of the package to be deleted.

Multiple Package Options:

pkgs A list of packages to delete. Must be passed as a python list. The `name` parameter will be ignored if this option is passed.

New in version 0.16.0.

Returns a dict containing the changes.

CLI Example:

```
salt '*' pkg.remove <package name>
salt '*' pkg.remove <package1>,<package2>,<package3>
salt '*' pkg.remove pkgs='["foo", "bar"]'
```

salt.modules.pkgutil.**upgrade**(*refresh=True*)

Upgrade all of the packages to the latest available version.

Returns a dict containing the changes:

```
{'<package>': {'old': '<old-version>',
               'new': '<new-version>'}}
```

CLI Example:

```
salt '*' pkgutil.upgrade
```

salt.modules.pkgutil.**upgrade_available**(*name*)

Check if there is an upgrade available for a certain package

CLI Example:

```
salt '*' pkgutil.upgrade_available CSWpython
```

salt.modules.pkgutil.**version**(**names*, ***kwargs*)

Returns a version if the package is installed, else returns an empty string

CLI Example:

```
salt '*' pkgutil.version CSWpython
```

22.16.170 salt.modules.portage_config

Configure `portage(5)`

salt.modules.portage_config.**append_to_package_conf**(*conf*, *atom=''*, *flags=None*, *string=''*, *overwrite=False*)

Append a string or a list of flags for a given package or DEPEND atom to a given configuration file.

CLI Example:

```
salt '*' portage_config.append_to_package_conf use string="app-admin/salt ldap -libvirt"
salt '*' portage_config.append_to_package_conf use atom="> = app-admin/salt-0.14.1" flags="['lda
```

salt.modules.portage_config.**append_use_flags**(*atom*, *uses=None*, *overwrite=False*)

Append a list of use flags for a given package or DEPEND atom

CLI Example:

```
salt '*' portage_config.append_use_flags "app-admin/salt[ldap, -libvirt]"
salt '*' portage_config.append_use_flags ">=app-admin/salt-0.14.1" "['ldap', '-libvirt']"
```

salt.modules.portage_config.**enforce_nice_config**()

Enforce a nice tree structure for /etc/portage/package.* configuration files.

See also:

salt.modules.ebuild.ex_mod_init() for information on automatically running this when pkg is used.

CLI Example:

```
salt '*' portage_config.enforce_nice_config
```

salt.modules.portage_config.**get_flags_from_package_conf**(*conf*, *atom*)

Get flags for a given package or DEPEND atom. Warning: This only works if the configuration files tree is in the correct format (the one enforced by enforce_nice_config)

CLI Example:

```
salt '*' portage_config.get_flags_from_package_conf license salt
```

salt.modules.portage_config.**get_missing_flags**(*conf*, *atom*, *flags*)

Find out which of the given flags are currently not set. CLI Example:

```
salt '*' portage_config.get_missing_flags use salt "['ldap', '-libvirt', 'openssl']"
```

salt.modules.portage_config.**has_flag**(*conf*, *atom*, *flag*)

Verify if the given package or DEPEND atom has the given flag. Warning: This only works if the configuration files tree is in the correct format (the one enforced by enforce_nice_config)

CLI Example:

```
salt '*' portage_config.has_flag license salt Apache-2.0
```

salt.modules.portage_config.**has_use**(*atom*, *use*)

Verify if the given package or DEPEND atom has the given use flag. Warning: This only works if the configuration files tree is in the correct format (the one enforced by enforce_nice_config)

CLI Example:

```
salt '*' portage_config.has_use salt libvirt
```

salt.modules.portage_config.**is_present**(*conf*, *atom*)
Tell if a given package or DEPEND atom is present in the configuration files tree. Warning: This only works if the configuration files tree is in the correct format (the one enforced by enforce_nice_config)

CLI Example:

```
salt '*' portage_config.is_present unmask salt
```

22.16.171 salt.modules.postfix

Support for Postfix

This module is currently little more than a config file viewer and editor. It is able to read the master.cf file (which is one style) and files in the style of main.cf (which is a different style, that is used in multiple postfix configuration files).

The design of this module is such that when files are edited, a minimum of changes are made to them. Each file should look as if it has been edited by hand; order, comments and whitespace are all preserved.

salt.modules.postfix.**delete**(*queue_id*)
Delete message(s) from the mail queue

CLI Example:

```
salt '*' postfix.delete 5C33CA0DEA

salt '*' postfix.delete ALL
```

salt.modules.postfix.**hold**(*queue_id*)
Put message(s) on hold from the mail queue

CLI Example:

```
salt '*' postfix.hold 5C33CA0DEA

salt '*' postfix.hold ALL
```

salt.modules.postfix.**requeue**(*queue_id*)
Requeue message(s) in the mail queue

CLI Example:

```
salt '*' postfix.requeue 5C33CA0DEA

salt '*' postfix.requeue ALL
```

salt.modules.postfix.**set_main**(*key*, *value*, *path='/etc/postfix/main.cf'*)
Set a single config value in the main.cf file. If the value does not already exist, it will be appended to the end.

CLI Example:

salt <minion> postfix.set_main mailq_path /usr/bin/mailq

salt.modules.postfix.**set_master**(*service*, *conn_type*, *private='y'*, *unpriv='y'*, *chroot='y'*, *wakeup='n'*, *maxproc='100'*, *command=''*, *write_conf=True*, *path='/etc/postfix/master.cf'*)
Set a single config value in the master.cf file. If the value does not already exist, it will be appended to the end.

Because of shell parsing issues, `-` cannot be set as a value, as is normal in the master.cf file; either `y', `n' or a number should be used when calling this function from the command line. If the value used matches the default, it will internally be converted to a `-`. Calling this function from the Python API is not affected by this limitation

The settings and their default values, in order, are: service (required), conn_type (required), private (y), unpriv (y), chroot (y), wakeup (n), maxproc (100), command (required).

By default, this function will write out the changes to the master.cf file, and then returns the full contents of the file. By setting the `write_conf` option to `False`, it will skip writing the file.

CLI Example:

salt <minion> postfix.set_master smtp inet n y n n 100 smtpd

`salt.modules.postfix.`**`show_main`**(*path='/etc/postfix/main.cf'*)

Return a dict of active config values. This does not include comments, spacing or order. Bear in mind that order is functionally important in the main.cf file, since keys can be referred to as variables. This means that the data returned from this function should not be used for direct modification of the main.cf file; other functions are available for that.

CLI Examples:

salt <minion> postfix.show_main salt <minion> postfix.show_main path=/path/to/main.cf

`salt.modules.postfix.`**`show_master`**(*path='/etc/postfix/master.cf'*)

Return a dict of active config values. This does not include comments, spacing or order.

The data returned from this function should not be used for direct modification of the main.cf file; other functions are available for that.

CLI Examples:

salt <minion> postfix.show_master salt <minion> postfix.show_master path=/path/to/master.cf

`salt.modules.postfix.`**`show_queue`**()

Show contents of the mail queue

CLI Example:

```
salt '*' postfix.show_queue
```

`salt.modules.postfix.`**`unhold`**(*queue_id*)

Set held message(s) in the mail queue to unheld

CLI Example:

```
salt '*' postfix.unhold 5C33CA0DEA

salt '*' postfix.unhold ALL
```

22.16.172 salt.modules.postgres

Module to provide Postgres compatibility to salt.

configuration In order to connect to Postgres, certain configuration is required in /etc/salt/minion on the relevant minions. Some sample configs might look like:

```
postgres.host: 'localhost'
postgres.port: '5432'
postgres.user: 'postgres' -> db user
```

```
        postgres.pass: ''
        postgres.maintenance_db: 'postgres'
```

The default for the maintenance_db is `postgres' and in most cases it can be left at the default setting. This data can also be passed into pillar. Options passed into opts will overwrite options passed into pillar

note This module uses MD5 hashing which may not be compliant with certain security audits.

salt.modules.postgres.**available_extensions**(*user=None*, *host=None*, *port=None*, *maintenance_db=None*, *password=None*, *runas=None*)

List available postgresql extensions

CLI Example:

```
salt '*' postgres.available_extensions
```

salt.modules.postgres.**create_extension**(*name*, *if_not_exists=None*, *schema=None*, *ext_version=None*, *from_version=None*, *user=None*, *host=None*, *port=None*, *maintenance_db=None*, *password=None*, *runas=None*)

Install a postgresql extension

CLI Example:

```
salt '*' postgres.create_extension 'adminpack'
```

salt.modules.postgres.**create_metadata**(*name*, *ext_version=None*, *schema=None*, *user=None*, *host=None*, *port=None*, *maintenance_db=None*, *password=None*, *runas=None*)

Get lifecycle information about an extension

CLI Example:

```
salt '*' postgres.create_metadata adminpack
```

salt.modules.postgres.**db_alter**(*name*, *user=None*, *host=None*, *port=None*, *maintenance_db=None*, *password=None*, *tablespace=None*, *owner=None*, *runas=None*)

Change tablespace or/and owner of database.

CLI Example:

```
salt '*' postgres.db_alter dbname owner=otheruser
```

salt.modules.postgres.**db_create**(*name*, *user=None*, *host=None*, *port=None*, *maintenance_db=None*, *password=None*, *tablespace=None*, *encoding=None*, *lc_collate=None*, *lc_ctype=None*, *owner=None*, *template=None*, *runas=None*)

Adds a databases to the Postgres server.

CLI Example:

```
salt '*' postgres.db_create 'dbname'

salt '*' postgres.db_create 'dbname' template=template_postgis
```

salt.modules.postgres.**db_exists**(*name*, *user=None*, *host=None*, *port=None*, *maintenance_db=None*, *password=None*, *runas=None*)

Checks if a database exists on the Postgres server.

CLI Example:

```
salt '*' postgres.db_exists 'dbname'
```

salt.modules.postgres.**db_list**(*user=None*, *host=None*, *port=None*, *maintenance_db=None*, *password=None*, *runas=None*)

Return dictionary with information about databases of a Postgres server.

CLI Example:

```
salt '*' postgres.db_list
```

salt.modules.postgres.**db_remove**(*name*, *user=None*, *host=None*, *port=None*, *maintenance_db=None*, *password=None*, *runas=None*)

Removes a databases from the Postgres server.

CLI Example:

```
salt '*' postgres.db_remove 'dbname'
```

salt.modules.postgres.**drop_extension**(*name*, *if_exists=None*, *restrict=None*, *cascade=None*, *user=None*, *host=None*, *port=None*, *maintenance_db=None*, *password=None*, *runas=None*)

Drop an installed postgresql extension

CLI Example:

```
salt '*' postgres.drop_extension 'adminpack'
```

salt.modules.postgres.**get_available_extension**(*name*, *user=None*, *host=None*, *port=None*, *maintenance_db=None*, *password=None*, *runas=None*)

Get info about an available postgresql extension

CLI Example:

```
salt '*' postgres.get_available_extension plpgsql
```

salt.modules.postgres.**get_installed_extension**(*name*, *user=None*, *host=None*, *port=None*, *maintenance_db=None*, *password=None*, *runas=None*)

Get info about an installed postgresql extension

CLI Example:

```
salt '*' postgres.get_installed_extension plpgsql
```

salt.modules.postgres.**group_create**(*groupname*, *user=None*, *host=None*, *port=None*, *maintenance_db=None*, *password=None*, *createdb=None*, *createuser=None*, *createroles=None*, *encrypted=None*, *login=None*, *inherit=None*, *superuser=None*, *replication=None*, *rolepassword=None*, *groups=None*, *runas=None*)

Creates a Postgres group. A group is postgres is similar to a user, but cannot login.

CLI Example:

```
salt '*' postgres.group_create 'groupname' user='user' \
        host='hostname' port='port' password='password' \
        rolepassword='rolepassword'
```

salt.modules.postgres.**group_remove**(*groupname*, *user=None*, *host=None*, *port=None*, *maintenance_db=None*, *password=None*, *runas=None*)

Removes a group from the Postgres server.

CLI Example:

```
salt '*' postgres.group_remove 'groupname'
```

salt.modules.postgres.**group_update**(*groupname*, *user=None*, *host=None*, *port=None*, *maintenance_db=None*, *password=None*, *createdb=None*, *createroles=None*, *createuser=None*, *encrypted=None*, *inherit=None*, *login=None*, *superuser=None*, *replication=None*, *rolepassword=None*, *groups=None*, *runas=None*)

Updates a postgres group

CLI Examples:

```
salt '*' postgres.group_update 'username' user='user' \
        host='hostname' port='port' password='password' \
        rolepassword='rolepassword'
```

salt.modules.postgres.**installed_extensions**(*user=None*, *host=None*, *port=None*, *maintenance_db=None*, *password=None*, *runas=None*)

List installed postgresql extensions

CLI Example:

```
salt '*' postgres.installed_extensions
```

salt.modules.postgres.**is_available_extension**(*name*, *user=None*, *host=None*, *port=None*, *maintenance_db=None*, *password=None*, *runas=None*)

Test if a specific extension is available

CLI Example:

```
salt '*' postgres.is_available_extension
```

salt.modules.postgres.**is_installed_extension**(*name*, *user=None*, *host=None*, *port=None*, *maintenance_db=None*, *password=None*, *runas=None*)

Test if a specific extension is installed

CLI Example:

```
salt '*' postgres.is_installed_extension
```

salt.modules.postgres.**owner_to**(*dbname*, *ownername*, *user=None*, *host=None*, *port=None*, *password=None*, *runas=None*)

Set the owner of all schemas, functions, tables, views and sequences to the given username.

CLI Example:

```
salt '*' postgres.owner_to 'dbname' 'username'
```

salt.modules.postgres.**psql_query**(*query*, *user=None*, *host=None*, *port=None*, *maintenance_db=None*, *password=None*, *runas=None*)

Run an SQL-Query and return the results as a list. This command only supports SELECT statements. This limitation can be worked around with a query like this:

WITH updated AS (UPDATE pg_authid SET rolconnlimit = 2000 WHERE rolname = `rolename' RETURNING rolconnlimit) SELECT * FROM updated;

CLI Example:

```
salt '*' postgres.psql_query 'select * from pg_stat_activity'
```

salt.modules.postgres.**role_get**(*name*, *user=None*, *host=None*, *port=None*, *maintenance_db=None*, *password=None*, *runas=None*, *return_password=False*)

Return a dict with information about users of a Postgres server.

Set return_password to True to get password hash in the result.

CLI Example:

```
salt '*' postgres.role_get postgres
```

salt.modules.postgres.**schema_create**(*dbname*, *name*, *owner=None*, *user=None*, *db_user=None*, *db_password=None*, *db_host=None*, *db_port=None*)

Creates a Postgres schema.

CLI Example:

```
salt '*' postgres.schema_create dbname name owner='owner' \
        user='user' \
        db_user='user' db_password='password'
        db_host='hostname' db_port='port'
```

salt.modules.postgres.**schema_exists**(*dbname*, *name*, *db_user=None*, *db_password=None*, *db_host=None*, *db_port=None*)

Checks if a schema exists on the Postgres server.

CLI Example:

```
salt '*' postgres.schema_exists dbname schemaname
```

dbname Database name we query on

name Schema name we look for

db_user database username if different from config or default

db_password user password if any password for a specified user

db_host Database host if different from config or default

db_port Database port if different from config or default

salt.modules.postgres.**schema_get**(*dbname*, *name*, *db_user=None*, *db_password=None*, *db_host=None*, *db_port=None*)

Return a dict with information about schemas in a database.

CLI Example:

```
salt '*' postgres.schema_get dbname name
```

dbname Database name we query on

name Schema name we look for

db_user database username if different from config or default

db_password user password if any password for a specified user

db_host Database host if different from config or default

db_port Database port if different from config or default

salt.modules.postgres.**schema_list**(*dbname*, *db_user=None*, *db_password=None*, *db_host=None*, *db_port=None*)

Return a dict with information about schemas in a Postgres database.

CLI Example:

```
salt '*' postgres.schema_list dbname
```

dbname Database name we query on

db_user database username if different from config or default

db_password user password if any password for a specified user

db_host Database host if different from config or default

db_port Database port if different from config or default

salt.modules.postgres.**schema_remove**(*dbname*, *name*, *user=None*, *db_user=None*, *db_password=None*, *db_host=None*, *db_port=None*)

Removes a schema from the Postgres server.

CLI Example:

```
salt '*' postgres.schema_remove dbname schemaname
```

dbname Database name we work on

schemaname The schema's name we'll remove

user System user all operations should be performed on behalf of

db_user database username if different from config or default

db_password user password if any password for a specified user

db_host Database host if different from config or default

db_port Database port if different from config or default

salt.modules.postgres.**user_create**(*username*, *user=None*, *host=None*, *port=None*, *maintenance_db=None*, *password=None*, *createdb=None*, *createuser=None*, *createroles=None*, *inherit=None*, *login=None*, *connlimit=None*, *encrypted=None*, *superuser=None*, *replication=None*, *rolepassword=None*, *groups=None*, *runas=None*)

Creates a Postgres user.

CLI Examples:

```
salt '*' postgres.user_create 'username' user='user' \
        host='hostname' port='port' password='password' \
        rolepassword='rolepassword'
```

salt.modules.postgres.**user_exists**(*name*, *user=None*, *host=None*, *port=None*, *maintenance_db=None*, *password=None*, *runas=None*)

Checks if a user exists on the Postgres server.

CLI Example:

```
salt '*' postgres.user_exists 'username'
```

salt.modules.postgres.**user_list**(*user=None, host=None, port=None, maintenance_db=None, password=None, runas=None, return_password=False*)

Return a dict with information about users of a Postgres server.

Set return_password to True to get password hash in the result.

CLI Example:

```
salt '*' postgres.user_list
```

salt.modules.postgres.**user_remove**(*username, user=None, host=None, port=None, maintenance_db=None, password=None, runas=None*)

Removes a user from the Postgres server.

CLI Example:

```
salt '*' postgres.user_remove 'username'
```

salt.modules.postgres.**user_update**(*username, user=None, host=None, port=None, maintenance_db=None, password=None, createdb=None, createuser=None, createroles=None, encrypted=None, superuser=None, inherit=None, login=None, connlimit=None, replication=None, rolepassword=None, groups=None, runas=None*)

Updates a Postgres user.

CLI Examples:

```
salt '*' postgres.user_update 'username' user='user' \
        host='hostname' port='port' password='password' \
        rolepassword='rolepassword'
```

salt.modules.postgres.**version**(*user=None, host=None, port=None, maintenance_db=None, password=None, runas=None*)

Return the version of a Postgres server.

CLI Example:

```
salt '*' postgres.version
```

22.16.173 salt.modules.poudriere

Support for poudriere

salt.modules.poudriere.**bulk_build**(*jail, pkg_file, keep=False*)

Run bulk build on poudriere server.

Return number of pkg builds, failures, and errors, on error dump to CLI

CLI Example:

```
salt -N buildbox_group poudriere.bulk_build 90amd64 /root/pkg_list
```

salt.modules.poudriere.**create_jail**(*name, arch, version='9.0-RELEASE'*)

Creates a new poudriere jail if one does not exist

NOTE creating a new jail will take some time the master is not hanging

CLI Example:

```
salt '*' poudriere.create_jail 90amd64 amd64
```

salt.modules.poudriere.**create_ports_tree**()
Not working need to run portfetch non interactive

salt.modules.poudriere.**delete_jail**(*name*)
Deletes poudriere jail with *name*

CLI Example:

```
salt '*' poudriere.delete_jail 90amd64
```

salt.modules.poudriere.**is_jail**(*name*)
Return True if jail exists False if not

CLI Example:

```
salt '*' poudriere.is_jail <jail name>
```

salt.modules.poudriere.**list_jails**()
Return a list of current jails managed by poudriere

CLI Example:

```
salt '*' poudriere.list_jails
```

salt.modules.poudriere.**list_ports**()
Return a list of current port trees managed by poudriere

CLI Example:

```
salt '*' poudriere.list_ports
```

salt.modules.poudriere.**make_pkgng_aware**(*jname*)
Make jail `jname` pkgng aware

CLI Example:

```
salt '*' poudriere.make_pkgng_aware <jail name>
```

salt.modules.poudriere.**parse_config**(*config_file=None*)
Returns a dict of poudriere main configuration definitions

CLI Example:

```
salt '*' poudriere.parse_config
```

salt.modules.poudriere.**update_jail**(*name*)
Run freebsd-update on *name* poudriere jail

CLI Example:

```
salt '*' poudriere.update_jail freebsd:10:x86:64
```

salt.modules.poudriere.**update_ports_tree**(*ports_tree*)
Updates the ports tree, either the default or the *ports_tree* specified

CLI Example:

```
salt '*' poudriere.update_ports_tree staging
```

salt.modules.poudriere.**version**()
Return poudriere version

CLI Example:

```
salt '*' poudriere.version
```

22.16.174 salt.modules.powerpath

powerpath support.

Assumes RedHat

salt.modules.powerpath.**add_license**(*key*)
Add a license

salt.modules.powerpath.**has_powerpath**()

salt.modules.powerpath.**list_licenses**()
returns a list of applied powerpath license keys

salt.modules.powerpath.**remove_license**(*key*)
Remove a license

22.16.175 salt.modules.ps

A salt interface to psutil, a system and process library. See http://code.google.com/p/psutil.

depends

- psutil Python module, version 0.3.0 or later
- python-utmp package (optional)

salt.modules.ps.**boot_time**(*time_format=None*)
Return the boot time in number of seconds since the epoch began.

CLI Example:

time_format Optionally specify a strftime format string. Use `time_format='%c'` to get a nicely-formatted locale specific date and time (i.e. `Fri May 2 19:08:32 2014`).

New in version 2014.1.4.

```
salt '*' ps.boot_time
```

salt.modules.ps.**cpu_percent**(*interval=0.1*, *per_cpu=False*)
Return the percent of time the CPU is busy.

interval the number of seconds to sample CPU usage over

per_cpu if True return an array of CPU percent busy for each CPU, otherwise aggregate all percents into one number

CLI Example:

```
salt '*' ps.cpu_percent
```

salt.modules.ps.**cpu_times**(*per_cpu=False*)
Return the percent of time the CPU spends in each state, e.g. user, system, idle, nice, iowait, irq, softirq.

per_cpu if True return an array of percents for each CPU, otherwise aggregate all percents into one number

CLI Example:

```
salt '*' ps.cpu_times
```

salt.modules.ps.**disk_io_counters**(*device=None*)
Return disk I/O statistics.

CLI Example:

```
salt '*' ps.disk_io_counters

salt '*' ps.disk_io_counters device=sda1
```

salt.modules.ps.**disk_partition_usage**(*all=False*)
Return a list of disk partitions plus the mount point, filesystem and usage statistics.

CLI Example:

```
salt '*' ps.disk_partition_usage
```

salt.modules.ps.**disk_partitions**(*all=False*)
Return a list of disk partitions and their device, mount point, and filesystem type.

all if set to False, only return local, physical partitions (hard disk, USB, CD/DVD partitions). If True, return all filesystems.

CLI Example:

```
salt '*' ps.disk_partitions
```

salt.modules.ps.**disk_usage**(*path*)
Given a path, return a dict listing the total available space as well as the free space, and used space.

CLI Example:

```
salt '*' ps.disk_usage /home
```

salt.modules.ps.**get_pid_list**()
Return a list of process ids (PIDs) for all running processes.

CLI Example:

```
salt '*' ps.get_pid_list
```

salt.modules.ps.**get_users**()
Return logged-in users.

CLI Example:

```
salt '*' ps.get_users
```

salt.modules.ps.**kill_pid**(*pid*, *signal=15*)
Kill a process by PID.

```
salt 'minion' ps.kill_pid pid [signal=signal_number]
```

pid PID of process to kill.

signal Signal to send to the process. See manpage entry for kill for possible values. Default: 15 (SIGTERM).

Example:

Send SIGKILL to process with PID 2000:

```
salt 'minion' ps.kill_pid 2000 signal=9
```

salt.modules.ps.**network_io_counters**(*interface=None*)

Return network I/O statistics.

CLI Example:

```
salt '*' ps.network_io_counters

salt '*' ps.network_io_counters interface=eth0
```

salt.modules.ps.**num_cpus**()

Return the number of CPUs.

CLI Example:

```
salt '*' ps.num_cpus
```

salt.modules.ps.**pgrep**(*pattern*, *user=None*, *full=False*)

Return the pids for processes matching a pattern.

If full is true, the full command line is searched for a match, otherwise only the name of the command is searched.

```
salt '*' ps.pgrep pattern [user=username] [full=(true|false)]
```

pattern Pattern to search for in the process list.

user Limit matches to the given username. Default: All users.

full A boolean value indicating whether only the name of the command or the full command line should be matched against the pattern.

Examples:

Find all httpd processes on all `www' minions:

```
salt 'www.*' ps.pgrep httpd
```

Find all bash processes owned by user `tom':

```
salt '*' ps.pgrep bash user=tom
```

salt.modules.ps.**pkill**(*pattern*, *user=None*, *signal=15*, *full=False*)

Kill processes matching a pattern.

```
salt '*' ps.pkill pattern [user=username] [signal=signal_number] \
        [full=(true|false)]
```

pattern Pattern to search for in the process list.

user Limit matches to the given username. Default: All users.

signal Signal to send to the process(es). See manpage entry for kill for possible values. Default: 15 (SIGTERM).

full A boolean value indicating whether only the name of the command or the full command line should be matched against the pattern.

Examples:

Send SIGHUP to all httpd processes on all `www' minions:

```
salt 'www.*' ps.pkill httpd signal=1
```

Send SIGKILL to all bash processes owned by user `tom':

```
salt '*' ps.pkill bash signal=9 user=tom
```

salt.modules.ps.**proc_info**(*pid*, *attrs=None*)

Return a dictionary of information for a process id (PID).

CLI Example:

```
salt '*' ps.proc_info 2322
salt '*' ps.proc_info 2322 attrs='["pid", "name"]'
```

pid PID of process to query.

attrs Optional list of desired process attributes. The list of possible attributes can be found here: http://pythonhosted.org/psutil/#psutil.Process

salt.modules.ps.**swap_memory**()

New in version 2014.7.0.

Return a dict that describes swap memory statistics.

Note: This function is only available in psutil version 0.6.0 and above.

CLI Example:

```
salt '*' ps.swap_memory
```

salt.modules.ps.**top**(*num_processes=5*, *interval=3*)

Return a list of top CPU consuming processes during the interval. num_processes = return the top N CPU consuming processes interval = the number of seconds to sample CPU usage over

CLI Examples:

```
salt '*' ps.top

salt '*' ps.top 5 10
```

salt.modules.ps.**total_physical_memory**()

Return the total number of bytes of physical memory.

CLI Example:

```
salt '*' ps.total_physical_memory
```

salt.modules.ps.**virtual_memory**()

New in version 2014.7.0.

Return a dict that describes statistics about system memory usage.

Note: This function is only available in psutil version 0.6.0 and above.

CLI Example:

```
salt '*' ps.virtual_memory
```

22.16.176 salt.modules.publish

Publish a command from a minion to a target

salt.modules.publish.**full_data**(*tgt*, *fun*, *arg=None*, *expr_form='glob'*, *returner='`*, *timeout=5*)
Return the full data about the publication, this is invoked in the same way as the publish function

CLI Example:

```
salt system.example.com publish.full_data '*' cmd.run 'ls -la /tmp'
```

Attention

If you need to pass a value to a function argument and that value contains an equal sign, you **must** include the argument name. For example:

```
salt '*' publish.full_data test.kwarg arg='cheese=spam'
```

---

salt.modules.publish.**publish**(*tgt*, *fun*, *arg=None*, *expr_form='glob'*, *returner='`*, *timeout=5*)
Publish a command from the minion out to other minions.

Publications need to be enabled on the Salt master and the minion needs to have permission to publish the command. The Salt master will also prevent a recursive publication loop, this means that a minion cannot command another minion to command another minion as that would create an infinite command loop.

The expr_form argument is used to pass a target other than a glob into the execution, the available options are:

- glob
- pcre
- grain
- grain_pcre
- pillar
- pillar_pcre
- ipcidr
- range
- compound

Note that for pillar matches must be exact, both in the pillar matcher and the compound matcher. No globbing is supported.

The arguments sent to the minion publish function are separated with commas. This means that for a minion executing a command with multiple args it will look like this:

```
salt system.example.com publish.publish '*' user.add 'foo,1020,1020'
salt system.example.com publish.publish 'os:Fedora' network.interfaces '' grain
```

CLI Example:

```
salt system.example.com publish.publish '*' cmd.run 'ls -la /tmp'
```

Attention

If you need to pass a value to a function argument and that value contains an equal sign, you **must** include the argument name. For example:

```
salt '*' publish.publish test.kwarg arg='cheese=spam'
```

Multiple keyword arguments should be passed as a list.

```
salt '*' publish.publish test.kwarg arg="['cheese=spam','spam=cheese']"
```

salt.modules.publish.**runner**(*fun*, *arg=None*, *timeout=5*)

Execute a runner on the master and return the data from the runner function

CLI Example:

```
salt publish.runner manage.down
```

22.16.177 salt.modules.puppet

Execute puppet routines

salt.modules.puppet.**disable**()

New in version 2014.7.0.

Disable the puppet agent

CLI Example:

```
salt '*' puppet.disable
```

salt.modules.puppet.**enable**()

New in version 2014.7.0.

Enable the puppet agent

CLI Example:

```
salt '*' puppet.enable
```

salt.modules.puppet.**fact**(*name*, *puppet=False*)

Run facter for a specific fact

CLI Example:

```
salt '*' puppet.fact kernel
```

salt.modules.puppet.**facts**(*puppet=False*)

Run facter and return the results

CLI Example:

```
salt '*' puppet.facts
```

salt.modules.puppet.**noop**(**args*, ***kwargs*)

Execute a puppet noop run and return a dict with the stderr, stdout, return code, etc. Usage is the same as for puppet.run.

CLI Example:

```
    salt '*' puppet.noop
    salt '*' puppet.noop tags=basefiles::edit,apache::server
    salt '*' puppet.noop debug
    salt '*' puppet.noop apply /a/b/manifest.pp modulepath=/a/b/modules tags=basefiles::edit,apache:
```

salt.modules.puppet.**run**(**args*, ***kwargs*)

Execute a puppet run and return a dict with the stderr, stdout, return code, etc. The first positional argument given is checked as a subcommand. Following positional arguments should be ordered with arguments required by the subcommand first, followed by non-keyword arguments. Tags are specified by a tag keyword and comma separated list of values. -- http://docs.puppetlabs.com/puppet/latest/reference/lang_tags.html

CLI Examples:

```
    salt '*' puppet.run
    salt '*' puppet.run tags=basefiles::edit,apache::server
    salt '*' puppet.run agent onetime no-daemonize no-usecacheonfailure no-splay ignorecache
    salt '*' puppet.run debug
    salt '*' puppet.run apply /a/b/manifest.pp modulepath=/a/b/modules tags=basefiles::edit,apache::
```

salt.modules.puppet.**status**()

New in version 2014.7.0.

Display puppet agent status

CLI Example:

```
    salt '*' puppet.status
```

salt.modules.puppet.**summary**()

New in version 2014.7.0.

Show a summary of the last puppet agent run

CLI Example:

```
    salt '*' puppet.summary
```

22.16.178 salt.modules.pw_group

Manage groups on FreeBSD

salt.modules.pw_group.**add**(*name*, *gid=None*, ***kwargs*)

Add the specified group

CLI Example:

```
    salt '*' group.add foo 3456
```

salt.modules.pw_group.**chgid**(*name*, *gid*)

Change the gid for a named group

CLI Example:

```
    salt '*' group.chgid foo 4376
```

salt.modules.pw_group.**delete**(*name*)

Remove the named group

CLI Example:

```
salt '*' group.delete foo
```

salt.modules.pw_group.**getent**(*refresh=False*)
Return info on all groups

CLI Example:

```
salt '*' group.getent
```

salt.modules.pw_group.**info**(*name*)
Return information about a group

CLI Example:

```
salt '*' group.info foo
```

22.16.179 salt.modules.pw_user

Manage users with the useradd command

salt.modules.pw_user.**add**(*name*, *uid=None*, *gid=None*, *groups=None*, *home=None*, *shell=None*, *unique=True*, *fullname=''*, *roomnumber=''*, *workphone=''*, *homephone=''*, *createhome=True*, ***kwargs*)
Add a user to the minion

CLI Example:

```
salt '*' user.add name <uid> <gid> <groups> <home> <shell>
```

salt.modules.pw_user.**chfullname**(*name*, *fullname*)
Change the user's Full Name

CLI Example:

```
salt '*' user.chfullname foo "Foo Bar"
```

salt.modules.pw_user.**chgid**(*name*, *gid*)
Change the default group of the user

CLI Example:

```
salt '*' user.chgid foo 4376
```

salt.modules.pw_user.**chgroups**(*name*, *groups*, *append=False*)
Change the groups this user belongs to, add append to append the specified groups

CLI Example:

```
salt '*' user.chgroups foo wheel,root True
```

salt.modules.pw_user.**chhome**(*name*, *home*, *persist=False*)
Change the home directory of the user, pass true for persist to copy files to the new home dir

CLI Example:

```
salt '*' user.chhome foo /home/users/foo True
```

salt.modules.pw_user.**chhomephone**(*name*, *homephone*)
Change the user's Home Phone

CLI Example:

```
salt '*' user.chhomephone foo "7735551234"
```

salt.modules.pw_user.**chroomnumber**(*name*, *roomnumber*)
Change the user's Room Number

CLI Example:

```
salt '*' user.chroomnumber foo 123
```

salt.modules.pw_user.**chshell**(*name*, *shell*)
Change the default shell of the user

CLI Example:

```
salt '*' user.chshell foo /bin/zsh
```

salt.modules.pw_user.**chuid**(*name*, *uid*)
Change the uid for a named user

CLI Example:

```
salt '*' user.chuid foo 4376
```

salt.modules.pw_user.**chworkphone**(*name*, *workphone*)
Change the user's Work Phone

CLI Example:

```
salt '*' user.chworkphone foo "7735550123"
```

salt.modules.pw_user.**delete**(*name*, *remove=False*, *force=False*)
Remove a user from the minion

CLI Example:

```
salt '*' user.delete name remove=True force=True
```

salt.modules.pw_user.**getent**(*refresh=False*)
Return the list of all info for all users

CLI Example:

```
salt '*' user.getent
```

salt.modules.pw_user.**info**(*name*)
Return user information

CLI Example:

```
salt '*' user.info root
```

salt.modules.pw_user.**list_groups**(*name*)
Return a list of groups the named user belongs to

CLI Example:

```
salt '*' user.list_groups foo
```

salt.modules.pw_user.**list_users**()
Return a list of all users

CLI Example:

```
    salt '*' user.list_users
```

salt.modules.pw_user.**rename**(*name*, *new_name*)
Change the username for a named user

CLI Example:

```
    salt '*' user.rename name new_name
```

22.16.180 salt.modules.pyenv

Manage python installations with pyenv.

New in version v2014.04.

salt.modules.pyenv.**default**(*python=None*, *runas=None*)
Returns or sets the currently defined default python.

python=None The version to set as the default. Should match one of the versions listed by `pyenv.versions`. Leave blank to return the current default.

CLI Example:

```
    salt '*' pyenv.default
    salt '*' pyenv.default 2.0.0-p0
```

salt.modules.pyenv.**do**(*cmdline=None*, *runas=None*)
Execute a python command with pyenv's shims from the user or the system.

CLI Example:

```
    salt '*' pyenv.do 'gem list bundler'
    salt '*' pyenv.do 'gem list bundler' deploy
```

salt.modules.pyenv.**do_with_python**(*python*, *cmdline*, *runas=None*)
Execute a python command with pyenv's shims using a specific python version.

CLI Example:

```
    salt '*' pyenv.do_with_python 2.0.0-p0 'gem list bundler'
    salt '*' pyenv.do_with_python 2.0.0-p0 'gem list bundler' deploy
```

salt.modules.pyenv.**install**(*runas=None*, *path=None*)
Install pyenv systemwide

CLI Example:

```
    salt '*' pyenv.install
```

salt.modules.pyenv.**install_python**(*python*, *runas=None*)
Install a python implementation.

python The version of python to install, should match one of the versions listed by pyenv.list

CLI Example:

```
    salt '*' pyenv.install_python 2.0.0-p0
```

salt.modules.pyenv.**is_installed**(*runas=None*)
Check if pyenv is installed.

CLI Example:

```
salt '*' pyenv.is_installed
```

salt.modules.pyenv.**list_**(*runas=None*)
List the installable versions of python.

CLI Example:

```
salt '*' pyenv.list
```

salt.modules.pyenv.**rehash**(*runas=None*)
Run pyenv rehash to update the installed shims.

CLI Example:

```
salt '*' pyenv.rehash
```

salt.modules.pyenv.**uninstall_python**(*python*, *runas=None*)
Uninstall a python implementation.

python The version of python to uninstall. Should match one of the versions listed by `pyenv.versions`

CLI Example:

```
salt '*' pyenv.uninstall_python 2.0.0-p0
```

salt.modules.pyenv.**update**(*runas=None*, *path=None*)
Updates the current versions of pyenv and python-Build

CLI Example:

```
salt '*' pyenv.update
```

salt.modules.pyenv.**versions**(*runas=None*)
List the installed versions of python.

CLI Example:

```
salt '*' pyenv.versions
```

22.16.181 salt.modules.qemu_img

Qemu-img Command Wrapper

The qemu img command is wrapped for specific functions

depends qemu-img

salt.modules.qemu_img.**make_image**(*location*, *size*, *fmt*)
Create a blank virtual machine image file of the specified size in megabytes. The image can be created in any format supported by qemu

CLI Example:

```
salt '*' qemu_img.make_image /tmp/image.qcow 2048 qcow2
salt '*' qemu_img.make_image /tmp/image.raw 10240 raw
```

22.16.182 salt.modules.qemu_nbd

Qemu Command Wrapper

The qemu system comes with powerful tools, such as qemu-img and qemu-nbd which are used here to build up kvm images.

`salt.modules.qemu_nbd.`**`clear`**(*mnt*)
Pass in the mnt dict returned from nbd_mount to unmount and disconnect the image from nbd. If all of the partitions are unmounted return an empty dict, otherwise return a dict containing the still mounted partitions

CLI Example:

```
salt '*' qemu_nbd.clear '{"/mnt/foo": "/dev/nbd0p1"}'
```

`salt.modules.qemu_nbd.`**`connect`**(*image*)
Activate nbd for an image file.

CLI Example:

```
salt '*' qemu_nbd.connect /tmp/image.raw
```

`salt.modules.qemu_nbd.`**`init`**(*image*)
Mount the named image via qemu-nbd and return the mounted roots

CLI Example:

```
salt '*' qemu_nbd.init /srv/image.qcow2
```

`salt.modules.qemu_nbd.`**`mount`**(*nbd*)
Pass in the nbd connection device location, mount all partitions and return a dict of mount points

CLI Example:

```
salt '*' qemu_nbd.mount /dev/nbd0
```

22.16.183 salt.modules.quota

Module for managing quotas on POSIX-like systems.

`salt.modules.quota.`**`get_mode`**(*device*)
Report whether the quota system for this device is on or off

CLI Example:

```
salt '*' quota.get_mode
```

`salt.modules.quota.`**`off`**(*device*)
Turns off the quota system

CLI Example:

```
salt '*' quota.off
```

`salt.modules.quota.`**`on`**(*device*)
Turns on the quota system

CLI Example:

```
salt '*' quota.on
```

salt.modules.quota.**report**(*mount*)
Report on quotas for a specific volume

CLI Example:

```
salt '*' quota.report /media/data
```

salt.modules.quota.**set_**(*device*, ***kwargs*)
Calls out to setquota, for a specific user or group

CLI Example:

```
salt '*' quota.set /media/data user=larry block-soft-limit=1048576
salt '*' quota.set /media/data group=painters file-hard-limit=1000
```

salt.modules.quota.**stats**()
Runs the quotastats command, and returns the parsed output

CLI Example:

```
salt '*' quota.stats
```

salt.modules.quota.**warn**()
Runs the warnquota command, to send warning emails to users who are over their quota limit.

CLI Example:

```
salt '*' quota.warn
```

22.16.184 salt.modules.rabbitmq

Module to provide RabbitMQ compatibility to Salt. Todo: A lot, need to add cluster support, logging, and minion configuration data.

salt.modules.rabbitmq.**add_user**(*name*, *password=None*, *runas=None*)
Add a rabbitMQ user via rabbitmqctl user_add <user> <password>

CLI Example:

```
salt '*' rabbitmq.add_user rabbit_user password
```

salt.modules.rabbitmq.**add_vhost**(*vhost*, *runas=None*)
Adds a vhost via rabbitmqctl add_vhost.

CLI Example:

```
salt '*' rabbitmq add_vhost '<vhost_name>'
```

salt.modules.rabbitmq.**change_password**(*name*, *password*, *runas=None*)
Changes a user's password.

CLI Example:

```
salt '*' rabbitmq.change_password rabbit_user password
```

salt.modules.rabbitmq.**clear_password**(*name*, *runas=None*)
Removes a user's password.

CLI Example:

```
salt '*' rabbitmq.clear_password rabbit_user
```

salt.modules.rabbitmq.**cluster_status**(*runas=None*)
return rabbitmq cluster_status

CLI Example:

```
salt '*' rabbitmq.cluster_status
```

salt.modules.rabbitmq.**delete_policy**(*vhost*, *name*, *runas=None*)
Delete a policy based on rabbitmqctl clear_policy.

Reference: http://www.rabbitmq.com/ha.html

CLI Example:

```
salt '*' rabbitmq.delete_policy / HA'
```

salt.modules.rabbitmq.**delete_user**(*name*, *runas=None*)
Deletes a user via rabbitmqctl delete_user.

CLI Example:

```
salt '*' rabbitmq.delete_user rabbit_user
```

salt.modules.rabbitmq.**delete_vhost**(*vhost*, *runas=None*)
Deletes a vhost rabbitmqctl delete_vhost.

CLI Example:

```
salt '*' rabbitmq.delete_vhost '<vhost_name>'
```

salt.modules.rabbitmq.**disable_plugin**(*name*, *runas=None*)
Disable a RabbitMQ plugin via the rabbitmq-plugins command.

CLI Example:

```
salt '*' rabbitmq.disable_plugin foo
```

salt.modules.rabbitmq.**enable_plugin**(*name*, *runas=None*)
Enable a RabbitMQ plugin via the rabbitmq-plugins command.

CLI Example:

```
salt '*' rabbitmq.enable_plugin foo
```

salt.modules.rabbitmq.**force_reset**(*runas=None*)
Forcefully Return a RabbitMQ node to its virgin state

CLI Example:

```
salt '*' rabbitmq.force_reset
```

salt.modules.rabbitmq.**join_cluster**(*host*, *user='rabbit'*, *ram_node=None*, *runas=None*)
Join a rabbit cluster

CLI Example:

```
salt '*' rabbitmq.join_cluster 'rabbit.example.com' 'rabbit'
```

salt.modules.rabbitmq.**list_permissions**(*vhost*, *runas=None*)
Lists permissions for vhost via rabbitmqctl list_permissions

CLI Example:

```
salt '*' rabbitmq.list_permissions '/myvhost'
```

salt.modules.rabbitmq.**list_policies**(*runas=None*)

Return a dictionary of policies nested by vhost and name based on the data returned from rabbitmqctl list_policies.

Reference: http://www.rabbitmq.com/ha.html

CLI Example:

```
salt '*' rabbitmq.list_policies'
```

salt.modules.rabbitmq.**list_queues**(*runas=None*, **kwargs*)

Returns queue details of the / virtual host

CLI Example:

```
salt '*' rabbitmq.list_queues messages consumers
```

salt.modules.rabbitmq.**list_queues_vhost**(*vhost*, *runas=None*, **kwargs*)

Returns queue details of specified virtual host. This command will consider first parameter as the vhost name and rest will be treated as queueinfoitem. For getting details on vhost `/`, use `list_queues` instead).

CLI Example:

```
salt '*' rabbitmq.list_queues messages consumers
```

salt.modules.rabbitmq.**list_user_permissions**(*name*, *runas=None*)

List permissions for a user via rabbitmqctl list_user_permissions

CLI Example:

```
salt '*' rabbitmq.list_user_permissions 'user'.
```

salt.modules.rabbitmq.**list_users**(*runas=None*)

Return a list of users based off of rabbitmqctl user_list.

CLI Example:

```
salt '*' rabbitmq.list_users
```

salt.modules.rabbitmq.**list_vhosts**(*runas=None*)

Return a list of vhost based on rabbitmqctl list_vhosts.

CLI Example:

```
salt '*' rabbitmq.list_vhosts
```

salt.modules.rabbitmq.**plugin_is_enabled**(*name*, *runas=None*)

Return whether the plugin is enabled.

CLI Example:

```
salt '*' rabbitmq.plugin_is_enabled foo
```

salt.modules.rabbitmq.**policy_exists**(*vhost*, *name*, *runas=None*)

Return whether the policy exists based on rabbitmqctl list_policies.

Reference: http://www.rabbitmq.com/ha.html

CLI Example:

```
salt '*' rabbitmq.policy_exists / HA
```

salt.modules.rabbitmq.**reset**(*runas=None*)
Return a RabbitMQ node to its virgin state

CLI Example:

```
salt '*' rabbitmq.reset
```

salt.modules.rabbitmq.**set_permissions**(*vhost*, *user*, *conf='.*'*, *write='.*'*, *read='.*'*, *runas=None*)
Sets permissions for vhost via rabbitmqctl set_permissions

CLI Example:

```
salt '*' rabbitmq.set_permissions 'myvhost' 'myuser'
```

salt.modules.rabbitmq.**set_policy**(*vhost*, *name*, *pattern*, *definition*, *priority=None*, *runas=None*)
Set a policy based on rabbitmqctl set_policy.

Reference: http://www.rabbitmq.com/ha.html

CLI Example:

```
salt '*' rabbitmq.set_policy / HA '.*' '{"ha-mode": "all"}'
```

salt.modules.rabbitmq.**set_user_tags**(*name*, *tags*, *runas=None*)
Add user tags via rabbitmqctl set_user_tags

CLI Example:

```
salt '*' rabbitmq.set_user_tags 'myadmin' 'administrator'
```

salt.modules.rabbitmq.**start_app**(*runas=None*)
Start the RabbitMQ application.

CLI Example:

```
salt '*' rabbitmq.start_app
```

salt.modules.rabbitmq.**status**(*runas=None*)
return rabbitmq status

CLI Example:

```
salt '*' rabbitmq.status
```

salt.modules.rabbitmq.**stop_app**(*runas=None*)
Stops the RabbitMQ application, leaving the Erlang node running.

CLI Example:

```
salt '*' rabbitmq.stop_app
```

salt.modules.rabbitmq.**user_exists**(*name*, *runas=None*)
Return whether the user exists based on rabbitmqctl list_users.

CLI Example:

```
salt '*' rabbitmq.user_exists rabbit_user
```

salt.modules.rabbitmq.**vhost_exists**(*name*, *runas=None*)
Return whether the vhost exists based on rabbitmqctl list_vhosts.

CLI Example:

```
salt '*' rabbitmq.vhost_exists rabbit_host
```

22.16.185 salt.modules.raet_publish

Publish a command from a minion to a target

salt.modules.raet_publish.**full_data**(*tgt*, *fun*, *arg=None*, *expr_form='glob'*, *returner=''*, *timeout=5*)

Return the full data about the publication, this is invoked in the same way as the publish function

CLI Example:

```
salt system.example.com publish.full_data '*' cmd.run 'ls -la /tmp'
```

Attention

If you need to pass a value to a function argument and that value contains an equal sign, you **must** include the argument name. For example:

```
salt '*' publish.full_data test.kwarg arg='cheese=spam'
```

---

salt.modules.raet_publish.**publish**(*tgt*, *fun*, *arg=None*, *expr_form='glob'*, *returner=''*, *timeout=5*)

Publish a command from the minion out to other minions.

Publications need to be enabled on the Salt master and the minion needs to have permission to publish the command. The Salt master will also prevent a recursive publication loop, this means that a minion cannot command another minion to command another minion as that would create an infinite command loop.

The expr_form argument is used to pass a target other than a glob into the execution, the available options are:

- glob
- pcre
- grain
- grain_pcre
- pillar
- pillar_pcre
- ipcidr
- range
- compound

The arguments sent to the minion publish function are separated with commas. This means that for a minion executing a command with multiple args it will look like this:

```
salt system.example.com publish.publish '*' user.add 'foo,1020,1020'
salt system.example.com publish.publish 'os:Fedora' network.interfaces '' grain
```

CLI Example:

```
salt system.example.com publish.publish '*' cmd.run 'ls -la /tmp'
```

Attention

If you need to pass a value to a function argument and that value contains an equal sign, you **must** include the argument name. For example:

```
salt '*' publish.publish test.kwarg arg='cheese=spam'
```

salt.modules.raet_publish.**runner**(*fun*, *arg=None*, *timeout=5*)
Execute a runner on the master and return the data from the runner function

CLI Example:

```
salt publish.runner manage.down
```

22.16.186 salt.modules.rbenv

Manage ruby installations with rbenv.

New in version 0.16.0.

salt.modules.rbenv.**default**(*ruby=None*, *runas=None*)
Returns or sets the currently defined default ruby.

ruby=None The version to set as the default. Should match one of the versions listed by `rbenv.versions`. Leave blank to return the current default.

CLI Example:

```
salt '*' rbenv.default
salt '*' rbenv.default 2.0.0-p0
```

salt.modules.rbenv.**do**(*cmdline=None*, *runas=None*)
Execute a ruby command with rbenv's shims from the user or the system.

CLI Example:

```
salt '*' rbenv.do 'gem list bundler'
salt '*' rbenv.do 'gem list bundler' deploy
```

salt.modules.rbenv.**do_with_ruby**(*ruby*, *cmdline*, *runas=None*)
Execute a ruby command with rbenv's shims using a specific ruby version.

CLI Example:

```
salt '*' rbenv.do_with_ruby 2.0.0-p0 'gem list bundler'
salt '*' rbenv.do_with_ruby 2.0.0-p0 'gem list bundler' deploy
```

salt.modules.rbenv.**install**(*runas=None*, *path=None*)
Install Rbenv systemwide

CLI Example:

```
salt '*' rbenv.install
```

salt.modules.rbenv.**install_ruby**(*ruby*, *runas=None*)
Install a ruby implementation.

ruby The version of Ruby to install, should match one of the versions listed by rbenv.list

Additional environment variables can be configured in pillar / grains / master:

```
rbenv:
  build_env: 'CONFIGURE_OPTS="--no-tcmalloc" CFLAGS="-fno-tree-dce"'
```

CLI Example:

```
salt '*' rbenv.install_ruby 2.0.0-p0
```

salt.modules.rbenv.**is_installed**(*runas=None*)
Check if Rbenv is installed.

CLI Example:

```
salt '*' rbenv.is_installed
```

salt.modules.rbenv.**list_**(*runas=None*)
List the installable versions of ruby.

CLI Example:

```
salt '*' rbenv.list
```

salt.modules.rbenv.**rehash**(*runas=None*)
Run rbenv rehash to update the installed shims.

CLI Example:

```
salt '*' rbenv.rehash
```

salt.modules.rbenv.**uninstall_ruby**(*ruby*, *runas=None*)
Uninstall a ruby implementation.

ruby The version of ruby to uninstall. Should match one of the versions listed by `rbenv.versions`

CLI Example:

```
salt '*' rbenv.uninstall_ruby 2.0.0-p0
```

salt.modules.rbenv.**update**(*runas=None*, *path=None*)
Updates the current versions of Rbenv and Ruby-Build

CLI Example:

```
salt '*' rbenv.update
```

salt.modules.rbenv.**versions**(*runas=None*)
List the installed versions of ruby.

CLI Example:

```
salt '*' rbenv.versions
```

22.16.187 salt.modules.rdp

Manage RDP Service on Windows servers

salt.modules.rdp.**disable**()
Disable RDP the service on the server

CLI Example:

```
salt '*' rdp.disable
```

salt.modules.rdp.**enable**()
Enable RDP the service on the server

CLI Example:

```
salt '*' rdp.enable
```

salt.modules.rdp.**status**()
Show if rdp is enabled on the server

CLI Example:

```
salt '*' rdp.status
```

22.16.188 salt.modules.redis

Module to provide redis functionality to Salt

New in version 2014.7.0.

configuration This module requires the redis python module and uses the following defaults which may be overridden in the minion configuration:

```
redis.host: 'localhost'
redis.port: 6379
redis.db: 0
redis.password: None
```

salt.modules.redismod.**bgrewriteaof**(*host=None*, *port=None*, *db=None*, *password=None*)
Asynchronously rewrite the append-only file

CLI Example:

```
salt '*' redis.bgrewriteaof
```

salt.modules.redismod.**bgsave**(*host=None*, *port=None*, *db=None*, *password=None*)
Asynchronously save the dataset to disk

CLI Example:

```
salt '*' redis.bgsave
```

salt.modules.redismod.**config_get**(*pattern='*'*, *host=None*, *port=None*, *db=None*, *password=None*)
Get redis server configuration values

CLI Example:

```
salt '*' redis.config_get
salt '*' redis.config_get port
```

salt.modules.redismod.**config_set**(*name*, *value*, *host=None*, *port=None*, *db=None*, *password=None*)
Set redis server configuration values

CLI Example:

```
salt '*' redis.config_set masterauth luv_kittens
```

salt.modules.redismod.**dbsize**(*host=None*, *port=None*, *db=None*, *password=None*)
Return the number of keys in the selected database

CLI Example:

```
salt '*' redis.dbsize
```

salt.modules.redismod.**delete**(**keys*, ***connection_args*)
Deletes the keys from redis, returns number of keys deleted

CLI Example:

```
salt '*' redis.delete foo
```

salt.modules.redismod.**exists**(*key*, *host=None*, *port=None*, *db=None*, *password=None*)
Return true if the key exists in redis

CLI Example:

```
salt '*' redis.exists foo
```

salt.modules.redismod.**expire**(*key*, *seconds*, *host=None*, *port=None*, *db=None*, *password=None*)
Set a keys time to live in seconds

CLI Example:

```
salt '*' redis.expire foo 300
```

salt.modules.redismod.**expireat**(*key*, *timestamp*, *host=None*, *port=None*, *db=None*, *password=None*)
Set a keys expire at given UNIX time

CLI Example:

```
salt '*' redis.expireat foo 1400000000
```

salt.modules.redismod.**flushall**(*host=None*, *port=None*, *db=None*, *password=None*)
Remove all keys from all databases

CLI Example:

```
salt '*' redis.flushall
```

salt.modules.redismod.**flushdb**(*host=None*, *port=None*, *db=None*, *password=None*)
Remove all keys from the selected database

CLI Example:

```
salt '*' redis.flushdb
```

salt.modules.redismod.**get_key**(*key*, *host=None*, *port=None*, *db=None*, *password=None*)
Get redis key value

CLI Example:

```
salt '*' redis.get_key foo
```

salt.modules.redismod.**hget**(*key*, *field*, *host=None*, *port=None*, *db=None*, *password=None*)
Get specific field value from a redis hash, returns dict

CLI Example:

```
    salt '*' redis.hget foo_hash bar_field
```

salt.modules.redismod.**hgetall**(*key*, *host=None*, *port=None*, *db=None*, *password=None*)
Get all fields and values from a redis hash, returns dict

CLI Example:

```
    salt '*' redis.hgetall foo_hash
```

salt.modules.redismod.**info**(*host=None*, *port=None*, *db=None*, *password=None*)
Get information and statistics about the server

CLI Example:

```
    salt '*' redis.info
```

salt.modules.redismod.**key_type**(*key*, *host=None*, *port=None*, *db=None*, *password=None*)
Get redis key type

CLI Example:

```
    salt '*' redis.type foo
```

salt.modules.redismod.**keys**(*pattern='*'*, *host=None*, *port=None*, *db=None*, *password=None*)
Get redis keys, supports glob style patterns

CLI Example:

```
    salt '*' redis.keys
    salt '*' redis.keys test*
```

salt.modules.redismod.**lastsave**(*host=None*, *port=None*, *db=None*, *password=None*)
Get the UNIX time in seconds of the last successful save to disk

CLI Example:

```
    salt '*' redis.lastsave
```

salt.modules.redismod.**llen**(*key*, *host=None*, *port=None*, *db=None*, *password=None*)
Get the length of a list in Redis

CLI Example:

```
    salt '*' redis.llen foo_list
```

salt.modules.redismod.**lrange**(*key*, *start*, *stop*, *host=None*, *port=None*, *db=None*, *password=None*)
Get a range of values from a list in Redis

CLI Example:

```
    salt '*' redis.lrange foo_list 0 10
```

salt.modules.redismod.**ping**(*host=None*, *port=None*, *db=None*, *password=None*)
Ping the server, returns False on connection errors

CLI Example:

```
    salt '*' redis.ping
```

salt.modules.redismod.**save**(*host=None*, *port=None*, *db=None*, *password=None*)
Synchronously save the dataset to disk

CLI Example:

```
salt '*' redis.save
```

salt.modules.redismod.**set_key**(*key*, *value*, *host=None*, *port=None*, *db=None*, *password=None*)
Set redis key value

CLI Example:

```
salt '*' redis.set_key foo bar
```

salt.modules.redismod.**shutdown**(*host=None*, *port=None*, *db=None*, *password=None*)
Synchronously save the dataset to disk and then shut down the server

CLI Example:

```
salt '*' redis.shutdown
```

salt.modules.redismod.**slaveof**(*master_host=None*, *master_port=None*, *host=None*, *port=None*, *db=None*, *password=None*)
Make the server a slave of another instance, or promote it as master

CLI Example:

```
# Become slave of redis-n01.example.com:6379
salt '*' redis.slaveof redis-n01.example.com 6379
salt '*' redis.slaveof redis-n01.example.com
# Become master
salt '*' redis.slaveof
```

salt.modules.redismod.**smembers**(*key*, *host=None*, *port=None*, *db=None*, *password=None*)
Get members in a Redis set

CLI Example:

```
salt '*' redis.smembers foo_set
```

salt.modules.redismod.**time**(*host=None*, *port=None*, *db=None*, *password=None*)
Return the current server UNIX time in seconds

CLI Example:

```
salt '*' redis.time
```

salt.modules.redismod.**zcard**(*key*, *host=None*, *port=None*, *db=None*, *password=None*)
Get the length of a sorted set in Redis

CLI Example:

```
salt '*' redis.zcard foo_sorted
```

salt.modules.redismod.**zrange**(*key*, *start*, *stop*, *host=None*, *port=None*, *db=None*, *password=None*)
Get a range of values from a sorted set in Redis by index

CLI Example:

```
salt '*' redis.zrange foo_sorted 0 10
```

22.16.189 salt.modules.reg

Manage the registry on Windows

depends

- winreg Python module

class salt.modules.reg.**Registry**
Delay `_winreg' usage until this module is used

salt.modules.reg.**create_key**(*hkey*, *path*, *key*, *value=None*, *reflection=True*)
Create a registry key

CLI Example:

```
salt '*' reg.create_key HKEY_CURRENT_USER 'SOFTWARE\Salt' 'version' '0.97'
```

salt.modules.reg.**delete_key**(*hkey*, *path*, *key*, *reflection=True*)
Delete a registry key

Note: This cannot delete a key with subkeys

CLI Example:

```
salt '*' reg.delete_key HKEY_CURRENT_USER 'SOFTWARE\Salt' 'version'
```

salt.modules.reg.**read_key**(*hkey*, *path*, *key*, *reflection=True*)
Read registry key value

CLI Example:

```
salt '*' reg.read_key HKEY_LOCAL_MACHINE 'SOFTWARE\Salt' 'version'
```

salt.modules.reg.**set_key**(*hkey*, *path*, *key*, *value*, *vtype='REG_DWORD'*, *reflection=True*)
Set a registry key vtype: http://docs.python.org/2/library/_winreg.html#value-types

CLI Example:

```
salt '*' reg.set_key HKEY_CURRENT_USER 'SOFTWARE\Salt' 'version' '0.97' REG_DWORD
```

22.16.190 salt.modules.rest_package

Service support for the REST example

salt.modules.rest_package.**install**(*name=None*, *refresh=False*, *fromrepo=None*, *pkgs=None*, *sources=None*, ***kwargs*)

salt.modules.rest_package.**installed**(*name*, *version=None*, *refresh=False*, *fromrepo=None*, *skip_verify=False*, *pkgs=None*, *sources=None*, ***kwargs*)

salt.modules.rest_package.**list_pkgs**(*versions_as_list=False*, ***kwargs*)

salt.modules.rest_package.**remove**(*name=None*, *pkgs=None*, ***kwargs*)

salt.modules.rest_package.**version**(**names*, ***kwargs*)
Returns a string representing the package version or an empty string if not installed. If more than one package name is specified, a dict of name/version pairs is returned.

CLI Example:

```
salt '*' pkg.version <package name>
salt '*' pkg.version <package1> <package2> <package3> ...
```

22.16.191 salt.modules.rest_sample

Module for interfacing to the REST example

pre-pre-ALPHA QUALITY code.

salt.modules.rest_sample.**grains_refresh**()
Refresh the cache.

salt.modules.rest_sample.**ping**()

22.16.192 salt.modules.rest_service

Service support for the REST example

salt.modules.rest_service.**list_**()
List services.

CLI Example:

```
salt '*' rest_service.list <service name>
```

salt.modules.rest_service.**restart**(*name*)
Restart the named service

CLI Example:

```
salt '*' rest_service.restart <service name>
```

salt.modules.rest_service.**start**(*name*)
Start the specified service

CLI Example:

```
salt '*' rest_service.start <service name>
```

salt.modules.rest_service.**status**(*name*)
Return the status for a service, returns a bool whether the service is running.

CLI Example:

```
salt '*' rest_service.status <service name>
```

salt.modules.rest_service.**stop**(*name*)
Stop the specified service

CLI Example:

```
salt '*' rest_service.stop <service name>
```

22.16.193 salt.modules.ret

Module to integrate with the returner system and retrieve data sent to a salt returner

salt.modules.ret.**get_fun**(*returner*, *fun*)
Return info about last time fun was called on each minion

CLI Example:

```
salt '*' ret.get_fun mysql network.interfaces
```

salt.modules.ret.**get_jid**(*returner*, *jid*)
Return the information for a specified job id

CLI Example:

```
salt '*' ret.get_jid redis 20421104181954700505
```

salt.modules.ret.**get_jids**(*returner*)
Return a list of all job ids

CLI Example:

```
salt '*' ret.get_jids mysql
```

salt.modules.ret.**get_minions**(*returner*)
Return a list of all minions

CLI Example:

```
salt '*' ret.get_minions mysql
```

22.16.194 salt.modules.rh_ip

The networking module for RHEL/Fedora based distros

salt.modules.rh_ip.**apply_network_settings**(***settings*)
Apply global network configuration.

CLI Example:

```
salt '*' ip.apply_network_settings
```

salt.modules.rh_ip.**build_bond**(*iface*, ***settings*)
Create a bond script in /etc/modprobe.d with the passed settings and load the bonding kernel module.

CLI Example:

```
salt '*' ip.build_bond bond0 mode=balance-alb
```

salt.modules.rh_ip.**build_interface**(*iface*, *iface_type*, *enabled*, ***settings*)
Build an interface script for a network interface.

CLI Example:

```
salt '*' ip.build_interface eth0 eth <settings>
```

salt.modules.rh_ip.**build_network_settings**(***settings*)
Build the global network script.

CLI Example:

```
salt '*' ip.build_network_settings <settings>
```

salt.modules.rh_ip.**build_routes**(*iface*, ***settings*)
Build a route script for a network interface.

CLI Example:

```
salt '*' ip.build_routes eth0 <settings>
```

salt.modules.rh_ip.**down**(*iface*, *iface_type*)
Shutdown a network interface

CLI Example:

```
salt '*' ip.down eth0
```

salt.modules.rh_ip.**get_bond**(*iface*)
Return the content of a bond script

CLI Example:

```
salt '*' ip.get_bond bond0
```

salt.modules.rh_ip.**get_interface**(*iface*)
Return the contents of an interface script

CLI Example:

```
salt '*' ip.get_interface eth0
```

salt.modules.rh_ip.**get_network_settings**()
Return the contents of the global network script.

CLI Example:

```
salt '*' ip.get_network_settings
```

salt.modules.rh_ip.**get_routes**(*iface*)
Return the contents of the interface routes script.

CLI Example:

```
salt '*' ip.get_routes eth0
```

salt.modules.rh_ip.**up**(*iface*, *iface_type*)
Start up a network interface

CLI Example:

```
salt '*' ip.up eth0
```

22.16.195 salt.modules.rh_service

Service support for RHEL-based systems, including support for both upstart and sysvinit

salt.modules.rh_service.**available**(*name*, *limit=''*)
Return True if the named service is available. Use the `limit` param to restrict results to services of that type.

CLI Examples:

```
salt '*' service.available sshd
salt '*' service.available sshd limit=upstart
salt '*' service.available sshd limit=sysvinit
```

salt.modules.rh_service.**disable**(*name*, ***kwargs*)
Disable the named service to start at boot

CLI Example:

```
salt '*' service.disable <service name>
```

salt.modules.rh_service.**disabled**(*name*)
Check to see if the named service is disabled to start on boot

CLI Example:

```
salt '*' service.disabled <service name>
```

salt.modules.rh_service.**enable**(*name*, ***kwargs*)
Enable the named service to start at boot

CLI Example:

```
salt '*' service.enable <service name>
```

salt.modules.rh_service.**enabled**(*name*, ***kwargs*)
Check to see if the named service is enabled to start on boot

CLI Example:

```
salt '*' service.enabled <service name>
```

salt.modules.rh_service.**get_all**(*limit=''*)
Return all installed services. Use the `limit` param to restrict results to services of that type.

CLI Example:

```
salt '*' service.get_all
salt '*' service.get_all limit=upstart
salt '*' service.get_all limit=sysvinit
```

salt.modules.rh_service.**get_disabled**(*limit=''*)
Return the disabled services. Use the `limit` param to restrict results to services of that type.

CLI Example:

```
salt '*' service.get_disabled
salt '*' service.get_disabled limit=upstart
salt '*' service.get_disabled limit=sysvinit
```

salt.modules.rh_service.**get_enabled**(*limit=''*)
Return the enabled services. Use the `limit` param to restrict results to services of that type.

CLI Examples:

```
salt '*' service.get_enabled
salt '*' service.get_enabled limit=upstart
salt '*' service.get_enabled limit=sysvinit
```

salt.modules.rh_service.**missing**(*name*, *limit=''*)
The inverse of service.available. Return True if the named service is not available. Use the `limit` param to restrict results to services of that type.

CLI Examples:

```
salt '*' service.missing sshd
salt '*' service.missing sshd limit=upstart
salt '*' service.missing sshd limit=sysvinit
```

salt.modules.rh_service.**reload_**(*name*)
Reload the named service

CLI Example:

```
salt '*' service.reload <service name>
```

salt.modules.rh_service.**restart**(*name*)
Restart the named service

CLI Example:

```
salt '*' service.restart <service name>
```

salt.modules.rh_service.**start**(*name*)
Start the specified service

CLI Example:

```
salt '*' service.start <service name>
```

salt.modules.rh_service.**status**(*name*, *sig=None*)
Return the status for a service, returns a bool whether the service is running.

CLI Example:

```
salt '*' service.status <service name>
```

salt.modules.rh_service.**stop**(*name*)
Stop the specified service

CLI Example:

```
salt '*' service.stop <service name>
```

22.16.196 salt.modules.riak

Riak Salt Module

Author: David Boucha <boucha@gmail.com>

salt.modules.riak.**cluster_commit**()
Commit Cluster Changes

CLI Example:

```
salt '*' riak.cluster_commit
```

salt.modules.riak.**cluster_join**(*riak_user=None*, *riak_host=None*)
Join a Riak cluster

CLI Example:

```
salt '*' riak.cluster_join <user> <host>
```

salt.modules.riak.**cluster_plan**()
Review Cluster Plan

CLI Example:

```
salt '*' riak.cluster_plan
```

salt.modules.riak.**member_status**()
Get cluster member status

CLI Example:

```
salt '*' riak.member_status
```

salt.modules.riak.**start**()
Start Riak

CLI Example:

```
salt '*' riak.start
```

salt.modules.riak.**stop**()
Stop Riak

CLI Example:

```
salt '*' riak.stop
```

22.16.197 salt.modules.rpm

Support for rpm

salt.modules.rpm.**diff**(*package*, *path*)
Return a formatted diff between current file and original in a package. NOTE: this function includes all files (configuration and not), but does not work on binary content.

Parameters

- **package** -- The name of the package
- **path** -- Full path to the installed file

Returns Difference or empty string. For binary files only a notification.

CLI example:

```
salt '*' lowpkg.diff apache2 /etc/apache2/httpd.conf
```

salt.modules.rpm.**file_dict**(**packages*)
List the files that belong to a package, sorted by group. Not specifying any packages will return a list of _every_ file on the system's rpm database (not generally recommended).

CLI Examples:

```
salt '*' lowpkg.file_dict httpd
salt '*' lowpkg.file_dict httpd postfix
salt '*' lowpkg.file_dict
```

salt.modules.rpm.**file_list**(**packages*)
List the files that belong to a package. Not specifying any packages will return a list of _every_ file on the system's rpm database (not generally recommended).

CLI Examples:

```
salt '*' lowpkg.file_list httpd
salt '*' lowpkg.file_list httpd postfix
salt '*' lowpkg.file_list
```

salt.modules.rpm.**list_pkgs**(**packages*)
List the packages currently installed in a dict:

```
{'<package_name>': '<version>'}
```

CLI Example:

```
salt '*' lowpkg.list_pkgs
```

salt.modules.rpm.**modified**(**packages*, ***flags*)
List the modified files that belong to a package. Not specifying any packages will return a list of _all_ modified files on the system's RPM database.

New in version 2015.5.0.

CLI examples:

```
salt '*' lowpkg.modified httpd
salt '*' lowpkg.modified httpd postfix
salt '*' lowpkg.modified
```

salt.modules.rpm.**owner**(**paths*)
Return the name of the package that owns the file. Multiple file paths can be passed. If a single path is passed, a string will be returned, and if multiple paths are passed, a dictionary of file/package name pairs will be returned.

If the file is not owned by a package, or is not present on the minion, then an empty string will be returned for that path.

CLI Examples:

```
salt '*' lowpkg.owner /usr/bin/apachectl
salt '*' lowpkg.owner /usr/bin/apachectl /etc/httpd/conf/httpd.conf
```

salt.modules.rpm.**verify**(**package*, ***kwargs*)
Runs an rpm -Va on a system, and returns the results in a dict

Files with an attribute of config, doc, ghost, license or readme in the package header can be ignored using the `ignore_types` keyword argument

CLI Example:

```
salt '*' lowpkg.verify
salt '*' lowpkg.verify httpd
salt '*' lowpkg.verify 'httpd postfix'
salt '*' lowpkg.verify 'httpd postfix' ignore_types=['config','doc']
```

22.16.198 salt.modules.rsync

Wrapper for rsync

New in version 2014.1.0.

This data can also be passed into *pillar*. Options passed into opts will overwrite options passed into pillar.

salt.modules.rsync.**config**(*confile='/etc/rsyncd.conf'*)
Return rsync config

CLI Example:

```
    salt '*' rsync.config
```

salt.modules.rsync.**rsync**(*src*, *dst*, *delete=False*, *force=False*, *update=False*, *passwordfile=None*, *exclude=None*, *excludefrom=None*)

Rsync files from src to dst

CLI Example:

```
    salt '*' rsync.rsync {src} {dst} {delete=True} {update=True} {passwordfile=/etc/pass.crt} {exclu
    salt '*' rsync.rsync {src} {dst} {delete=True} {excludefrom=/xx.ini}
```

salt.modules.rsync.**version**()

Return rsync version

CLI Example:

```
    salt '*' rsync.version
```

22.16.199 salt.modules.runit

runit service module

This module is compatible with the `service` states, so it can be used to maintain services using the `provider` argument:

```
myservice:
  service:
    - running
    - provider: runit
```

Note that the `enabled` argument is not available with this provider.

salt.modules.runit.**available**(*name*)

Returns `True` if the specified service is available, otherwise returns `False`.

CLI Example:

```
    salt '*' runit.available foo
```

salt.modules.runit.**full_restart**(*name*)

Calls runit.restart() function

CLI Example:

```
    salt '*' runit.full_restart <service name>
```

salt.modules.runit.**get_all**()

Return a list of all available services

CLI Example:

```
    salt '*' runit.get_all
```

salt.modules.runit.**missing**(*name*)

The inverse of runit.available. Returns `True` if the specified service is not available, otherwise returns `False`.

CLI Example:

```
    salt '*' runit.missing foo
```

salt.modules.runit.**reload_**(*name*)
Send a HUP to service via runit

CLI Example:

```
salt '*' runit.reload <service name>
```

salt.modules.runit.**restart**(*name*)
Restart service via runit. This will stop/start service

CLI Example:

```
salt '*' runit.restart <service name>
```

salt.modules.runit.**start**(*name*)
Starts service via runit

CLI Example:

```
salt '*' runit.start <service name>
```

salt.modules.runit.**status**(*name*, *sig=None*)
Return the status for a service via runit, return pid if running

CLI Example:

```
salt '*' runit.status <service name>
```

salt.modules.runit.**stop**(*name*)
Stops service via runit

CLI Example:

```
salt '*' runit.stop <service name>
```

salt.modules.runit.**term**(*name*)
Send a TERM to service via runit

CLI Example:

```
salt '*' runit.term <service name>
```

22.16.200 salt.modules.rvm

Manage ruby installations and gemsets with RVM, the Ruby Version Manager.

salt.modules.rvm.**do**(*ruby*, *command*, *runas=None*, *cwd=None*)
Execute a command in an RVM controlled environment.

ruby: The ruby to use.

command: The command to execute.

runas [None] The user to run rvm as.

cwd [None] The current working directory.

CLI Example:

```
salt '*' rvm.do 2.0.0 <command>
```

salt.modules.rvm.**gemset_copy**(*source*, *destination*, *runas=None*)
Copy all gems from one gemset to another.

source The name of the gemset to copy, complete with ruby version.

destination The destination gemset.

runas [None] The user to run rvm as.

CLI Example:

```
salt '*' rvm.gemset_copy foobar bazquo
```

salt.modules.rvm.**gemset_create**(*ruby*, *gemset*, *runas=None*)
Creates a gemset.

ruby The ruby version to create the gemset for.

gemset The name of the gemset to create.

runas [None] The user to run rvm as.

CLI Example:

```
salt '*' rvm.gemset_create 2.0.0 foobar
```

salt.modules.rvm.**gemset_delete**(*ruby*, *gemset*, *runas=None*)
Deletes a gemset.

ruby The ruby version the gemset belongs to.

gemset The gemset to delete.

runas [None] The user to run rvm as.

CLI Example:

```
salt '*' rvm.gemset_delete 2.0.0 foobar
```

salt.modules.rvm.**gemset_empty**(*ruby*, *gemset*, *runas=None*)
Remove all gems from a gemset.

ruby The ruby version the gemset belongs to.

gemset The gemset to empty.

runas [None] The user to run rvm as.

CLI Example:

```
salt '*' rvm.gemset_empty 2.0.0 foobar
```

salt.modules.rvm.**gemset_list**(*ruby='default'*, *runas=None*)
List all gemsets for the given ruby.

ruby [default] The ruby version to list the gemsets for

runas [None] The user to run rvm as.

CLI Example:

```
salt '*' rvm.gemset_list
```

salt.modules.rvm.**gemset_list_all**(*runas=None*)
List all gemsets for all installed rubies.

Note that you must have set a default ruby before this can work.

runas [None] The user to run rvm as.

CLI Example:

```
salt '*' rvm.gemset_list_all
```

salt.modules.rvm.**get**(*version='stable'*, *runas=None*)
Update RVM.

version [stable] Which version of RVM to install, e.g. stable or head.

ruby The version of ruby to reinstall.

CLI Example:

```
salt '*' rvm.get
```

salt.modules.rvm.**install**(*runas=None*)
Install RVM system wide.

CLI Example:

```
salt '*' rvm.install
```

salt.modules.rvm.**install_ruby**(*ruby*, *runas=None*)
Install a ruby implementation.

ruby The version of ruby to install.

runas [None] The user to run rvm as.

CLI Example:

```
salt '*' rvm.install_ruby 1.9.3-p385
```

salt.modules.rvm.**is_installed**(*runas=None*)
Check if RVM is installed.

CLI Example:

```
salt '*' rvm.is_installed
```

salt.modules.rvm.**list_**(*runas=None*)
List all rvm installed rubies.

runas [None] The user to run rvm as.

CLI Example:

```
salt '*' rvm.list
```

salt.modules.rvm.**reinstall_ruby**(*ruby*, *runas=None*)
Reinstall a ruby implementation.

ruby The version of ruby to reinstall.

runas [None] The user to run rvm as.

CLI Example:

```
salt '*' rvm.reinstall_ruby 1.9.3-p385
```

salt.modules.rvm.**rubygems**(*ruby*, *version*, *runas=None*)
Installs a specific rubygems version in the given ruby.

ruby The ruby to install rubygems for.

version The version of rubygems to install or `remove' to use the version that ships with 1.9

runas [None] The user to run rvm as.

CLI Example:

```
salt '*' rvm.rubygems 2.0.0 1.8.24
```

salt.modules.rvm.**set_default**(*ruby*, *runas=None*)
Set the default ruby.

ruby The version of ruby to make the default.

runas [None] The user to run rvm as.

CLI Example:

```
salt '*' rvm.set_default 2.0.0
```

salt.modules.rvm.**wrapper**(*ruby_string*, *wrapper_prefix*, *runas=None*, **binaries*)
Install RVM wrapper scripts.

ruby_string Ruby/gemset to install wrappers for.

wrapper_prefix What to prepend to the name of the generated wrapper binaries.

runas [None] The user to run rvm as.

binaries [None] The names of the binaries to create wrappers for. When nothing is given, wrappers for ruby, gem, rake, irb, rdoc, ri and testrb are generated.

CLI Example:

```
salt '*' rvm.wrapper <ruby_string> <wrapper_prefix>
```

22.16.201 salt.modules.s3

Connection module for Amazon S3

configuration This module accepts explicit s3 credentials but can also utilize IAM roles assigned to the instance trough Instance Profiles. Dynamic credentials are then automatically obtained from AWS API and no further configuration is necessary. More Information available at:

```
http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/iam-roles-for-amazon-ec2.html
```

If IAM roles are not used you need to specify them either in a pillar or in the minion's config file:

```
s3.keyid: GKTADJGHEIQSXMKKRBJ08H
s3.key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs
```

A service_url may also be specified in the configuration:

```
s3.service_url: s3.amazonaws.com
```

If a service_url is not specified, the default is s3.amazonaws.com. This may appear in various documentation as an ``endpoint''. A comprehensive list for Amazon S3 may be found at:

```
http://docs.aws.amazon.com/general/latest/gr/rande.html#s3_region
```

The service_url will form the basis for the final endpoint that is used to query the service.

SSL verification may also be turned off in the configuration:

s3.verify_ssl: False

This is required if using S3 bucket names that contain a period, as these will not match Amazon's S3 wildcard certificates. Certificate verification is enabled by default.

This module should be usable to query other S3-like services, such as Eucalyptus.

depends requests

`salt.modules.s3.`**`delete`**(*bucket*, *path=None*, *action=None*, *key=None*, *keyid=None*, *service_url=None*, *verify_ssl=None*)

Delete a bucket, or delete an object from a bucket.

CLI Example to delete a bucket:

```
salt myminion s3.delete mybucket
```

CLI Example to delete an object from a bucket:

```
salt myminion s3.delete mybucket remoteobject
```

`salt.modules.s3.`**`get`**(*bucket=None*, *path=None*, *return_bin=False*, *action=None*, *local_file=None*, *key=None*, *keyid=None*, *service_url=None*, *verify_ssl=None*)

List the contents of a bucket, or return an object from a bucket. Set return_bin to True in order to retrieve an object wholesale. Otherwise, Salt will attempt to parse an XML response.

CLI Example to list buckets:

```
salt myminion s3.get
```

CLI Example to list the contents of a bucket:

```
salt myminion s3.get mybucket
```

CLI Example to return the binary contents of an object:

```
salt myminion s3.get mybucket myfile.png return_bin=True
```

CLI Example to save the binary contents of an object to a local file:

```
salt myminion s3.get mybucket myfile.png local_file=/tmp/myfile.png
```

It is also possible to perform an action on a bucket. Currently, S3 supports the following actions:

```
acl
cors
lifecycle
policy
location
logging
notification
tagging
versions
requestPayment
versioning
website
```

To perform an action on a bucket:

```
salt myminion s3.get mybucket myfile.png action=acl
```

salt.modules.s3.**head**(*bucket*, *path=None*, *key=None*, *keyid=None*, *service_url=None*, *verify_ssl=None*)
Return the metadata for a bucket, or an object in a bucket.

CLI Examples:

```
salt myminion s3.head mybucket
salt myminion s3.head mybucket myfile.png
```

salt.modules.s3.**put**(*bucket*, *path=None*, *return_bin=False*, *action=None*, *local_file=None*, *key=None*, *keyid=None*, *service_url=None*, *verify_ssl=None*)
Create a new bucket, or upload an object to a bucket.

CLI Example to create a bucket:

```
salt myminion s3.put mybucket
```

CLI Example to upload an object to a bucket:

```
salt myminion s3.put mybucket remotepath local_file=/path/to/file
```

22.16.202 salt.modules.saltcloudmod

Control a salt cloud system

salt.modules.saltcloudmod.**create**(*name*, *profile*)
Create the named vm

CLI Example:

```
salt <minion-id> saltcloud.create webserver rackspace_centos_512
```

22.16.203 salt.modules.saltutil

The Saltutil module is used to manage the state of the salt minion itself. It is used to manage minion modules as well as automate updates to the salt minion.

depends

- esky Python module for update functionality

salt.modules.saltutil.**clear_cache**()
Forcibly removes all caches on a minion.

New in version 2014.7.0.

WARNING: The safest way to clear a minion cache is by first stopping the minion and then deleting the cache files before restarting it.

CLI Example:

```
salt '*' saltutil.clear_cache
```

salt.modules.saltutil.**cmd**(*tgt*, *fun*, *arg=()*, *timeout=None*, *expr_form='glob'*, *ret=''*, *kwarg=None*, *ssh=False*, ***kwargs*)
Assuming this minion is a master, execute a salt command

CLI Example:

```
salt '*' saltutil.cmd
```

salt.modules.saltutil.**cmd_iter**(*tgt*, *fun*, *arg=()*, *timeout=None*, *expr_form='glob'*, *ret='`*, *kwarg=None*, *ssh=False*, ***kwargs*)

Assuming this minion is a master, execute a salt command

CLI Example:

```
salt '*' saltutil.cmd_iter
```

salt.modules.saltutil.**find_cached_job**(*jid*)

Return the data for a specific cached job id

CLI Example:

```
salt '*' saltutil.find_cached_job <job id>
```

salt.modules.saltutil.**find_job**(*jid*)

Return the data for a specific job id

CLI Example:

```
salt '*' saltutil.find_job <job id>
```

salt.modules.saltutil.**is_running**(*fun*)

If the named function is running return the data associated with it/them. The argument can be a glob

CLI Example:

```
salt '*' saltutil.is_running state.highstate
```

salt.modules.saltutil.**kill_job**(*jid*)

Sends a kill signal (SIGKILL 9) to the named salt job's process

CLI Example:

```
salt '*' saltutil.kill_job <job id>
```

salt.modules.saltutil.**mmodule**(*saltenv*, *fun*, **args*, ***kwargs*)

Loads minion modules from an environment so that they can be used in pillars for that environment

CLI Example:

```
salt '*' saltutil.mmodule base test.ping
```

salt.modules.saltutil.**pillar_refresh**()

Signal the minion to refresh the pillar data.

CLI Example:

```
salt '*' saltutil.refresh_pillar
```

salt.modules.saltutil.**refresh_modules**()

Signal the minion to refresh the module and grain data

CLI Example:

```
salt '*' saltutil.refresh_modules
```

salt.modules.saltutil.**refresh_pillar**()

Signal the minion to refresh the pillar data.

CLI Example:

```
salt '*' saltutil.refresh_pillar
```

salt.modules.saltutil.**regen_keys**()
Used to regenerate the minion keys.

CLI Example:

```
salt '*' saltutil.regen_keys
```

salt.modules.saltutil.**revoke_auth**(*preserve_minion_cache=False*)
The minion sends a request to the master to revoke its own key. Note that the minion session will be revoked and the minion may not be able to return the result of this command back to the master.

If the `preserve_minion_cache' flag is set to True, the master cache for this minion will not be removed.

CLI Example:

```
salt '*' saltutil.revoke_auth
```

salt.modules.saltutil.**runner**(*fun*, ***kwargs*)
Execute a runner module (this function must be run on the master)

New in version 2014.7.

name The name of the function to run

kwargs Any keyword arguments to pass to the runner function

CLI Example:

```
salt '*' saltutil.runner jobs.list_jobs
```

salt.modules.saltutil.**running**()
Return the data on all running salt processes on the minion

CLI Example:

```
salt '*' saltutil.running
```

salt.modules.saltutil.**signal_job**(*jid*, *sig*)
Sends a signal to the named salt job's process

CLI Example:

```
salt '*' saltutil.signal_job <job id> 15
```

salt.modules.saltutil.**sync_all**(*saltenv=None*, *refresh=True*)
Sync down all of the dynamic modules from the file server for a specific environment

refresh [True] Also refresh the execution modules available to the minion.

CLI Example:

```
salt '*' saltutil.sync_all
```

salt.modules.saltutil.**sync_grains**(*saltenv=None*, *refresh=True*)
Sync the grains from the _grains directory on the salt master file server. This function is environment aware, pass the desired environment to grab the contents of the _grains directory, base is the default environment.

CLI Example:

```
salt '*' saltutil.sync_grains
```

salt.modules.saltutil.**sync_modules**(*saltenv=None*, *refresh=True*)
Sync the modules from the _modules directory on the salt master file server. This function is environment aware, pass the desired environment to grab the contents of the _modules directory, base is the default environment.

CLI Example:

```
salt '*' saltutil.sync_modules
```

salt.modules.saltutil.**sync_outputters**(*saltenv=None*, *refresh=True*)
Sync the outputters from the _outputters directory on the salt master file server. This function is environment aware, pass the desired environment to grab the contents of the _outputters directory, base is the default environment.

CLI Example:

```
salt '*' saltutil.sync_outputters
```

salt.modules.saltutil.**sync_renderers**(*saltenv=None*, *refresh=True*)
Sync the renderers from the _renderers directory on the salt master file server. This function is environment aware, pass the desired environment to grab the contents of the _renderers directory, base is the default environment.

CLI Example:

```
salt '*' saltutil.sync_renderers
```

salt.modules.saltutil.**sync_returners**(*saltenv=None*, *refresh=True*)
Sync the returners from the _returners directory on the salt master file server. This function is environment aware, pass the desired environment to grab the contents of the _returners directory, base is the default environment.

CLI Example:

```
salt '*' saltutil.sync_returners
```

salt.modules.saltutil.**sync_states**(*saltenv=None*, *refresh=True*)
Sync the states from the _states directory on the salt master file server. This function is environment aware, pass the desired environment to grab the contents of the _states directory, base is the default environment.

CLI Example:

```
salt '*' saltutil.sync_states
```

salt.modules.saltutil.**sync_utils**(*saltenv=None*, *refresh=True*)
Sync utility source files from the _utils directory on the salt master file server. This function is environment aware, pass the desired environment to grab the contents of the _utils directory, base is the default environment.

CLI Example:

```
salt '*' saltutil.sync_utils
```

salt.modules.saltutil.**term_job**(*jid*)
Sends a termination signal (SIGTERM 15) to the named salt job's process

CLI Example:

```
salt '*' saltutil.term_job <job id>
```

salt.modules.saltutil.**update**(*version=None*)

Update the salt minion from the URL defined in opts['update_url'] SaltStack, Inc provides the latest builds here: update_url: http://docs.saltstack.com/downloads/

Be aware that as of 2014-8-11 there's a bug in esky such that only the latest version available in the update_url can be downloaded and installed.

This feature requires the minion to be running a bdist_esky build.

The version number is optional and will default to the most recent version available at opts['update_url'].

Returns details about the transaction upon completion.

CLI Example:

```
salt '*' saltutil.update
salt '*' saltutil.update 0.10.3
```

salt.modules.saltutil.**wheel**(*fun*, ***kwargs*)

Execute a wheel module (this function must be run on the master)

New in version 2014.7.

name The name of the function to run

kwargs Any keyword arguments to pass to the wheel function

CLI Example:

```
salt '*' saltutil.wheel key.accept match=jerry
```

22.16.204 salt.modules.schedule

Module for managing the Salt schedule on a minion

New in version 2014.7.0.

salt.modules.schedule.**add**(*name*, ***kwargs*)

Add a job to the schedule

CLI Example:

```
salt '*' schedule.add job1 function='test.ping' seconds=3600
# If function have some arguments, use job_args
salt '*' schedule.add job2 function='cmd.run' job_args=['date >> /tmp/date.log'] seconds=60
```

salt.modules.schedule.**build_schedule_item**(*name*, ***kwargs*)

Build a schedule job

CLI Example:

```
salt '*' schedule.build_schedule_item job1 function='test.ping' seconds=3600
```

salt.modules.schedule.**copy**(*name*, *target*, ***kwargs*)

Copy scheduled job to another minion or minions.

CLI Example:

```
salt '*' schedule.copy jobname target
```

salt.modules.schedule.**delete**(*name*, ***kwargs*)
Delete a job from the minion's schedule

CLI Example:

```
salt '*' schedule.delete job1
```

salt.modules.schedule.**disable**(***kwargs*)
Disable all scheduled jobs on the minion

CLI Example:

```
salt '*' schedule.disable
```

salt.modules.schedule.**disable_job**(*name*, ***kwargs*)
Disable a job in the minion's schedule

CLI Example:

```
salt '*' schedule.disable_job job1
```

salt.modules.schedule.**enable**(***kwargs*)
Enable all scheduled jobs on the minion

CLI Example:

```
salt '*' schedule.enable
```

salt.modules.schedule.**enable_job**(*name*, ***kwargs*)
Enable a job in the minion's schedule

CLI Example:

```
salt '*' schedule.enable_job job1
```

salt.modules.schedule.**list_**(*show_all=False*, *return_yaml=True*)
List the jobs currently scheduled on the minion

CLI Example:

```
salt '*' schedule.list

salt '*' schedule.list show_all=True
```

salt.modules.schedule.**modify**(*name*, ***kwargs*)
Modify an existing job in the schedule

CLI Example:

```
salt '*' schedule.modify job1 function='test.ping' seconds=3600
```

salt.modules.schedule.**move**(*name*, *target*, ***kwargs*)
Move scheduled job to another minion or minions.

CLI Example:

```
salt '*' schedule.move jobname target
```

salt.modules.schedule.**purge**(***kwargs*)
Purge all the jobs currently scheduled on the minion

CLI Example:

```
    salt '*' schedule.purge
```

salt.modules.schedule.**reload_**()
Reload saved scheduled jobs on the minion

CLI Example:

```
    salt '*' schedule.reload
```

salt.modules.schedule.**run_job**(*name*, *force=False*)
Run a scheduled job on the minion immediately

CLI Example:

```
    salt '*' schedule.run_job job1

    salt '*' schedule.run_job job1 force=True
    Force the job to run even if it is disabled.
```

salt.modules.schedule.**save**()
Save all scheduled jobs on the minion

CLI Example:

```
    salt '*' schedule.save
```

22.16.205 salt.modules.scsi

SCSI administration module

salt.modules.scsi.**ls_**()
List SCSI devices, with details

CLI Example:

```
    salt '*' scsi.ls
```

salt.modules.scsi.**rescan_all**(*host*)
List scsi devices

CLI Example:

```
    salt '*' scsi.rescan_all(0)
```

22.16.206 salt.modules.sdb

Module for Manipulating Data via the Salt DB API

salt.modules.sdb.**get**(*uri*)
Get a value from a db, using a uri in the form of sdb://<profile>/<key>. If the uri provided does not start with sdb://, then it will be returned as-is.

CLI Example:

```
    salt '*' sdb.get sdb://mymemcached/foo
```

salt.modules.sdb.**set_**(*uri*, *value*)

Set a value in a db, using a uri in the form of `sdb://<profile>/<key>`. If the uri provided does not start with `sdb://` or the value is not successfully set, return `False`.

CLI Example:

```
salt '*' sdb.set sdb://mymemcached/foo bar
```

22.16.207 salt.modules.seed

Virtual machine image management tools

salt.modules.seed.**apply_**(*path*, *id_=None*, *config=None*, *approve_key=True*, *install=True*, *prep_install=False*)

Seed a location (disk image, directory, or block device) with the minion config, approve the minion's key, and/or install salt-minion.

CLI Example:

```
salt 'minion' seed.apply path id [config=config_data] \
        [gen_key=(true|false)] [approve_key=(true|false)] \
        [install=(true|false)]
```

path Full path to the directory, device, or disk image on the target minion's file system.

id Minion id with which to seed the path.

config Minion configuration options. By default, the `master' option is set to the target host's `master'.

approve_key Request a pre-approval of the generated minion key. Requires that the salt-master be configured to either auto-accept all keys or expect a signing request from the target host. Default: true.

install Install salt-minion, if absent. Default: true.

prep_install Prepare the bootstrap script, but don't run it. Default: false

salt.modules.seed.**mkconfig**(*config=None*, *tmp=None*, *id_=None*, *approve_key=True*, *pub_key=None*, *priv_key=None*)

Generate keys and config and put them in a tmp directory.

pub_key absolute path or file content of an optional preseeded salt key

priv_key absolute path or file content of an optional preseeded salt key

CLI Example:

```
salt 'minion' seed.mkconfig [config=config_data] [tmp=tmp_dir] \
        [id_=minion_id] [approve_key=(true|false)]
```

salt.modules.seed.**prep_bootstrap**(*mpt*)

Update and get the random script to a random place

CLI Example:

```
salt '*' seed.prep_bootstrap /tmp
```

22.16.208 salt.modules.selinux

Execute calls on selinux

Note: This module requires the `semanage` and `setsebool` commands to be available on the minion. On RHEL-based distros, this means that the `policycoreutils` and `policycoreutils-python` packages must be installed. If not on a RHEL-based distribution, consult the selinux documentation for your distro to ensure that the proper packages are installed.

`salt.modules.selinux.`**`getenforce`**`()`
Return the mode selinux is running in

CLI Example:

```
salt '*' selinux.getenforce
```

`salt.modules.selinux.`**`getsebool`**`(`*boolean*`)`
Return the information on a specific selinux boolean

CLI Example:

```
salt '*' selinux.getsebool virt_use_usb
```

`salt.modules.selinux.`**`list_sebool`**`()`
Return a structure listing all of the selinux booleans on the system and what state they are in

CLI Example:

```
salt '*' selinux.list_sebool
```

`salt.modules.selinux.`**`selinux_fs_path`**`(`**args*`)`
Return the location of the SELinux VFS directory

CLI Example:

```
salt '*' selinux.selinux_fs_path
```

`salt.modules.selinux.`**`setenforce`**`(`*mode*`)`
Set the SELinux enforcing mode

CLI Example:

```
salt '*' selinux.setenforce enforcing
```

`salt.modules.selinux.`**`setsebool`**`(`*boolean, value, persist=False*`)`
Set the value for a boolean

CLI Example:

```
salt '*' selinux.setsebool virt_use_usb off
```

`salt.modules.selinux.`**`setsebools`**`(`*pairs, persist=False*`)`
Set the value of multiple booleans

CLI Example:

```
salt '*' selinux.setsebools '{virt_use_usb: on, squid_use_tproxy: off}'
```

22.16.209 salt.modules.sensors

Read lm-sensors

New in version 2014.1.3.

salt.modules.sensors.**sense**(*chip*, *fahrenheit=False*)
Gather lm-sensors data from a given chip

To determine the chip to query, use the `sensors' command and see the leading line in the block.

Example:

/usr/bin/sensors

coretemp-isa-0000 Adapter: ISA adapter Physical id 0: +56.0°C (high = +87.0°C, crit = +105.0°C) Core 0: +52.0°C (high = +87.0°C, crit = +105.0°C) Core 1: +50.0°C (high = +87.0°C, crit = +105.0°C) Core 2: +56.0°C (high = +87.0°C, crit = +105.0°C) Core 3: +53.0°C (high = +87.0°C, crit = +105.0°C)

Given the above, the chip is `coretemp-isa-0000'.

22.16.210 salt.modules.serverdensity_device

Wrapper around Server Density API

New in version 2014.7.0.

salt.modules.serverdensity_device.**create**(*name*, ***params*)
Function to create device in Server Density. For more info, see the API docs.

CLI Example:

```
salt '*' serverdensity_device.create lama
salt '*' serverdensity_device.create rich_lama group=lama_band installedRAM=32768
```

salt.modules.serverdensity_device.**delete**(*device_id*)
Delete a device from Server Density. For more information, see the API docs.

CLI Example:

```
salt '*' serverdensity_device.delete 51f7eafcdba4bb235e000ae4
```

salt.modules.serverdensity_device.**get_sd_auth**(*val*, *sd_auth_pillar_name='serverdensity'*)
Returns requested Server Density authentication value from pillar.

CLI Example:

```
salt '*' serverdensity_device.get_sd_auth <val>
```

salt.modules.serverdensity_device.**install_agent**(*agent_key*)
Function downloads Server Density installation agent, and installs sd-agent with agent_key.

CLI Example:

```
salt '*' serverdensity_device.install_agent c2bbdd6689ff46282bdaa07555641498
```

salt.modules.serverdensity_device.**ls**(***params*)
List devices in Server Density

Results will be filtered by any params passed to this function. For more information, see the API docs on listing and searching.

CLI Example:

```
salt '*' serverdensity_device.ls
salt '*' serverdensity_device.ls name=lama
salt '*' serverdensity_device.ls name=lama group=lama_band installedRAM=32768
```

salt.modules.serverdensity_device.**update**(*device_id*, ***params*)

Updates device information in Server Density. For more information see the API docs.

CLI Example:

```
salt '*' serverdensity_device.update 51f7eafcdba4bb235e000ae4 name=lama group=lama_band
salt '*' serverdensity_device.update 51f7eafcdba4bb235e000ae4 name=better_lama group=rock_lamas
```

22.16.211 salt.modules.service

The default service module, if not otherwise specified salt will fall back to this basic module

salt.modules.service.**available**(*name*)

Returns `True` if the specified service is available, otherwise returns `False`.

CLI Example:

```
salt '*' service.available sshd
```

salt.modules.service.**get_all**()

Return a list of all available services

CLI Example:

```
salt '*' service.get_all
```

salt.modules.service.**missing**(*name*)

The inverse of service.available. Returns `True` if the specified service is not available, otherwise returns `False`.

CLI Example:

```
salt '*' service.missing sshd
```

salt.modules.service.**reload_**(*name*)

Refreshes config files by calling service reload. Does not perform a full restart.

CLI Example:

```
salt '*' service.reload <service name>
```

salt.modules.service.**restart**(*name*)

Restart the specified service

CLI Example:

```
salt '*' service.restart <service name>
```

salt.modules.service.**start**(*name*)

Start the specified service

CLI Example:

```
salt '*' service.start <service name>
```

salt.modules.service.**status**(*name*, *sig=None*)
Return the status for a service, returns the PID or an empty string if the service is running or not, pass a signature to use to find the service via ps

CLI Example:

```
salt '*' service.status <service name> [service signature]
```

salt.modules.service.**stop**(*name*)
Stop the specified service

CLI Example:

```
salt '*' service.stop <service name>
```

22.16.212 salt.modules.shadow

Manage the shadow file

salt.modules.shadow.**default_hash**()
Returns the default hash used for unset passwords

CLI Example:

```
salt '*' shadow.default_hash
```

salt.modules.shadow.**del_password**(*name*)
New in version 2014.7.0.

Delete the password from name user

CLI Example:

```
salt '*' shadow.del_password username
```

salt.modules.shadow.**gen_password**(*password*, *crypt_salt=None*, *algorithm='sha512'*)
New in version 2014.7.0.

Generate hashed password

password Plaintext password to be hashed.

crypt_salt Crpytographic salt. If not given, a random 8-character salt will be generated.

algorithm The following hash algorithms are supported:

- md5
- blowfish (not in mainline glibc, only available in distros that add it)
- sha256
- sha512 (default)

CLI Example:

```
salt '*' shadow.gen_password 'I_am_password'
salt '*' shadow.gen_password 'I_am_password' crypt_salt'I_am_salt' algorithm=sha256
```

salt.modules.shadow.**info**(*name*)
Return information for the specified user

CLI Example:

```
salt '*' shadow.info root
```

salt.modules.shadow.**set_date**(*name*, *date*)
Sets the value for the date the password was last changed to days since the epoch (January 1, 1970). See man chage.

CLI Example:

```
salt '*' shadow.set_date username 0
```

salt.modules.shadow.**set_expire**(*name*, *expire*)
Changed in version 2014.7.0.

Sets the value for the date the account expires as days since the epoch (January 1, 1970). Using a value of -1 will clear expiration. See man chage.

CLI Example:

```
salt '*' shadow.set_expire username -1
```

salt.modules.shadow.**set_inactdays**(*name*, *inactdays*)
Set the number of days of inactivity after a password has expired before the account is locked. See man chage.

CLI Example:

```
salt '*' shadow.set_inactdays username 7
```

salt.modules.shadow.**set_maxdays**(*name*, *maxdays*)
Set the maximum number of days during which a password is valid. See man chage.

CLI Example:

```
salt '*' shadow.set_maxdays username 90
```

salt.modules.shadow.**set_mindays**(*name*, *mindays*)
Set the minimum number of days between password changes. See man chage.

CLI Example:

```
salt '*' shadow.set_mindays username 7
```

salt.modules.shadow.**set_password**(*name*, *password*, *use_usermod=False*)
Set the password for a named user. The password must be a properly defined hash. The password hash can be generated with this command:

```
python -c "import crypt; print crypt.crypt('password', '\$6\$SALTsalt')"
```

`SALTsalt` is the 8-character crpytographic salt. Valid characters in the salt are `.`, `/`, and any alphanumeric character.

Keep in mind that the $6 represents a sha512 hash, if your OS is using a different hashing algorithm this needs to be changed accordingly

CLI Example:

```
salt '*' shadow.set_password root '$1$UYCIxa628.9qXjpQCjM4a..'
```

salt.modules.shadow.**set_warndays**(*name*, *warndays*)
Set the number of days of warning before a password change is required. See man chage.

CLI Example:

```
    salt '*' shadow.set_warndays username 7
```

22.16.213 salt.modules.slack_notify

Module for sending messages to Slack

New in version 2015.5.0.

configuration This module can be used by either passing an api key and version directly or by specifying both in a configuration profile in the salt master/minion config.

For example:

```
slack:
  api_key: peWcBiMOS9HrZG15peWcBiMOS9HrZG15
```

salt.modules.slack_notify.**find_room**(*name*, *api_key=None*)

Find a room by name and return it. :param name: The room name. :param api_key: The Slack admin api key. :return: The room object.

CLI Example:

```
salt '*' slack.find_room name="random"

salt '*' slack.find_room name="random" api_key=peWcBiMOS9HrZG15peWcBiMOS9HrZG15
```

salt.modules.slack_notify.**find_user**(*name*, *api_key=None*)

Find a user by name and return it. :param name: The user name. :param api_key: The Slack admin api key. :return: The user object.

CLI Example:

```
salt '*' slack.find_user name="ThomasHatch"

salt '*' slack.find_user name="ThomasHatch" api_key=peWcBiMOS9HrZG15peWcBiMOS9HrZG15
```

salt.modules.slack_notify.**list_rooms**(*api_key=None*)

List all Slack rooms.

Parameters **api_key** -- The Slack admin api key.

Returns The room list.

CLI Example:

```
salt '*' slack.list_rooms

salt '*' slack.list_rooms api_key=peWcBiMOS9HrZG15peWcBiMOS9HrZG15
```

salt.modules.slack_notify.**list_users**(*api_key=None*)

List all Slack users. :param api_key: The Slack admin api key. :return: The user list.

CLI Example:

```
salt '*' slack.list_users

salt '*' slack.list_users api_key=peWcBiMOS9HrZG15peWcBiMOS9HrZG15
```

salt.modules.slack_notify.**post_message**(*channel*, *message*, *from_name*, *api_key=None*)

Send a message to a Slack channel. :param channel: The channel name, either will work. :param message: The

message to send to the HipChat room. :param from_name: Specify who the message is from. :param api_key: The Slack api key, if not specified in the configuration. :return: Boolean if message was sent successfully.

CLI Example:

```
salt '*' slack.send_message channel="Development Room" message="Build is done" from_name="Build
```

22.16.214 salt.modules.smartos_imgadm

Module for running imgadm command on SmartOS

salt.modules.smartos_imgadm.**avail**(*search=None*)
Return a list of available images

CLI Example:

```
salt '*' imgadm.avail [percona]
```

salt.modules.smartos_imgadm.**delete**(*uuid=None*)
Remove an installed image

CLI Example:

```
salt '*' imgadm.delete e42f8c84-bbea-11e2-b920-078fab2aab1f
```

salt.modules.smartos_imgadm.**get**(*uuid=None*)
Return info on an installed image

CLI Example:

```
salt '*' imgadm.get e42f8c84-bbea-11e2-b920-078fab2aab1f
```

salt.modules.smartos_imgadm.**import_image**(*uuid=None*)
Import an image from the repository

CLI Example:

```
salt '*' imgadm.import_image e42f8c84-bbea-11e2-b920-078fab2aab1f
```

salt.modules.smartos_imgadm.**list_installed**()
Return a list of installed images

CLI Example:

```
salt '*' imgadm.list_installed
```

salt.modules.smartos_imgadm.**show**(*uuid=None*)
Show manifest of a given image

CLI Example:

```
salt '*' imgadm.show e42f8c84-bbea-11e2-b920-078fab2aab1f
```

salt.modules.smartos_imgadm.**update_installed**()
Gather info on unknown images (locally installed)

CLI Example:

```
salt '*' imgadm.update_installed()
```

salt.modules.smartos_imgadm.**version**()
Return imgadm version

CLI Example:

```
salt '*' imgadm.version
```

22.16.215 salt.modules.smartos_vmadm

Module for managing VMs on SmartOS

salt.modules.smartos_vmadm.**destroy**(*uuid=None*)
Hard power down the virtual machine, this is equivalent to pulling the power

CLI Example:

```
salt '*' virt.destroy <uuid>
```

salt.modules.smartos_vmadm.**get_macs**(*uuid=None*)
Return a list off MAC addresses from the named VM

CLI Example:

```
salt '*' virt.get_macs <uuid>
```

salt.modules.smartos_vmadm.**init**(***kwargs*)
Initialize a new VM

CLI Example:

```
salt '*' virt.init image_uuid='...' alias='...' [...]
```

salt.modules.smartos_vmadm.**list_active_vms**()
Return a list of uuids for active virtual machine on the minion

CLI Example:

```
salt '*' virt.list_active_vms
```

salt.modules.smartos_vmadm.**list_inactive_vms**()
Return a list of uuids for inactive virtual machine on the minion

CLI Example:

```
salt '*' virt.list_inactive_vms
```

salt.modules.smartos_vmadm.**list_vms**()
Return a list of virtual machine names on the minion

CLI Example:

```
salt '*' virt.list_vms
```

salt.modules.smartos_vmadm.**reboot**(*uuid=None*)
Reboot a domain via ACPI request

CLI Example:

```
salt '*' virt.reboot <uuid>
```

salt.modules.smartos_vmadm.**setmem**(*uuid*, *memory*)
Change the amount of memory allocated to VM. <memory> is to be specified in MB.

Note for KVM : this would require a restart of the VM.

CLI Example:

```
salt '*' virt.setmem <uuid> 512
```

salt.modules.smartos_vmadm.**shutdown**(*uuid=None*)
Send a soft shutdown signal to the named vm

CLI Example:

```
salt '*' virt.shutdown <uuid>
```

salt.modules.smartos_vmadm.**start**(*uuid=None*)
Start a defined domain

CLI Example:

```
salt '*' virt.start <uuid>
```

salt.modules.smartos_vmadm.**vm_info**(*uuid=None*)
Return a dict with information about the specified VM on this CN

CLI Example:

```
salt '*' virt.vm_info <uuid>
```

salt.modules.smartos_vmadm.**vm_virt_type**(*uuid=None*)
Return VM virtualization type : OS or KVM

CLI Example:

```
salt '*' virt.vm_virt_type <uuid>
```

22.16.216 salt.modules.smf

Service support for Solaris 10 and 11, should work with other systems that use SMF also. (e.g. SmartOS)

salt.modules.smf.**available**(*name*)
Returns `True` if the specified service is available, otherwise returns `False`.

We look up the name with the svcs command to get back the FMRI This allows users to use simpler service names

CLI Example:

```
salt '*' service.available net-snmp
```

salt.modules.smf.**disable**(*name*, ***kwargs*)
Disable the named service to start at boot

CLI Example:

```
salt '*' service.disable <service name>
```

salt.modules.smf.**disabled**(*name*)
Check to see if the named service is disabled to start on boot

CLI Example:

```
salt '*' service.disabled <service name>
```

salt.modules.smf.**enable**(*name*, ***kwargs*)
Enable the named service to start at boot

CLI Example:

```
salt '*' service.enable <service name>
```

salt.modules.smf.**enabled**(*name*, ***kwargs*)
Check to see if the named service is enabled to start on boot

CLI Example:

```
salt '*' service.enabled <service name>
```

salt.modules.smf.**get_all**()
Return all installed services

CLI Example:

```
salt '*' service.get_all
```

salt.modules.smf.**get_disabled**()
Return the disabled services

CLI Example:

```
salt '*' service.get_disabled
```

salt.modules.smf.**get_enabled**()
Return the enabled services

CLI Example:

```
salt '*' service.get_enabled
```

salt.modules.smf.**get_running**()
Return the running services

CLI Example:

```
salt '*' service.get_running
```

salt.modules.smf.**get_stopped**()
Return the stopped services

CLI Example:

```
salt '*' service.get_stopped
```

salt.modules.smf.**missing**(*name*)
The inverse of service.available. Returns `True` if the specified service is not available, otherwise returns `False`.

CLI Example:

```
salt '*' service.missing net-snmp
```

salt.modules.smf.**reload_**(*name*)
Reload the named service

CLI Example:

```
salt '*' service.reload <service name>
```

salt.modules.smf.**restart**(*name*)
Restart the named service

CLI Example:

```
salt '*' service.restart <service name>
```

salt.modules.smf.**start**(*name*)
Start the specified service

CLI Example:

```
salt '*' service.start <service name>
```

salt.modules.smf.**status**(*name*, *sig=None*)
Return the status for a service, returns a bool whether the service is running.

CLI Example:

```
salt '*' service.status <service name>
```

salt.modules.smf.**stop**(*name*)
Stop the specified service

CLI Example:

```
salt '*' service.stop <service name>
```

22.16.217 salt.modules.smtp

Module for Sending Messages via SMTP

New in version 2014.7.0.

depends

- smtplib python module

configuration This module can be used by either passing a jid and password directly to send_message, or by specifying the name of a configuration profile in the minion config, minion pillar, or master config.

For example:

```
my-smtp-login:
    smtp.server: smtp.domain.com
    smtp.tls: True
    smtp.sender: admin@domain.com
    smtp.username: myuser
    smtp.password: verybadpass
```

The resourcename refers to the resource that is using this account. It is user-definable, and optional. The following configurations are both valid:

```
my-smtp-login:
    smtp.server: smtp.domain.com
    smtp.tls: True
    smtp.sender: admin@domain.com
    smtp.username: myuser
```

```
            smtp.password: verybadpass

        another-smtp-login:
            smtp.server: smtp.domain.com
            smtp.tls: True
            smtp.sender: admin@domain.com
            smtp.username: myuser
            smtp.password: verybadpass
```

salt.modules.smtp.**send_msg**(*recipient*, *message*, *subject='Message from Salt'*, *sender=None*, *server=None*, *use_ssl='True'*, *username=None*, *password=None*, *profile=None*)

Send a message to an SMTP recipient. Designed for use in states.

CLI Examples:

```
    smtp.send_msg 'admin@example.com' 'This is a salt module test'              profile='my-smtp-acco
    smtp.send_msg 'admin@example.com' 'This is a salt module test'              username='myuser' pas
```

22.16.218 salt.modules.softwareupdate

Support for the softwareupdate command on MacOS.

salt.modules.softwareupdate.**download**(**updates*)

Download a named update so that it can be installed later with the install or upgrade function. It returns a list of all updates that are now downloaded.

CLI Example:

```
    salt '*' softwareupdate.download <update name>
    salt '*' softwareupdate.download "<update with whitespace>"
    salt '*' softwareupdate.download <update1> <update2> <update3>
```

salt.modules.softwareupdate.**download_all**(*rec=False*, *restart=True*)

Download all available updates so that they can be installed later with the install or upgrade function. It returns a list of updates that are now downloaded.

CLI Example:

```
    salt '*' softwareupdate.download_all
```

salt.modules.softwareupdate.**ignore**(**updates*)

Ignore a specific program update. When an update is ignored the `-` and version number at the end will be omitted, so ``SecUpd2014-001-1.0'' becomes ``SecUpd2014-001''. It will be removed automatically if present. An update is successfully ignored when it no longer shows up after list_upgrades.

CLI Example:

```
    salt '*' softwareupdate.ignore <update-name>
    salt '*' softwareupdate.ignore "<update with whitespace>"
    salt '*' softwareupdate.ignore <update1> <update2> <update3>
```

salt.modules.softwareupdate.**install**(**updates*)

Install a named upgrade. Returns a dictionary containing the name of the update and the status of its installation.

Return values: - `True`: The update was installed. - `False`: The update was not installed. - `None`: There is no update available with that name.

CLI Example:

```
salt '*' softwareupdate.install <update-name>
salt '*' softwareupdate.install "<update with whitespace>"
salt '*' softwareupdate.install <update1> <update2> <update3>
```

salt.modules.softwareupdate.**list_downloads**()
Return a list of all updates that have been downloaded locally.

CLI Example:

```
salt '*' softwareupdate.list_downloads
```

salt.modules.softwareupdate.**list_ignored**()
List all upgrades that has been ignored. Ignored updates are shown without the `-` and version number at the end, this is how the softwareupdate command works.

CLI Example:

```
salt '*' softwareupdate.list_ignored
```

salt.modules.softwareupdate.**list_upgrades**(*rec=False*, *restart=False*)
List all available updates.

rec Return only the recommended updates.

restart Return only the updates that require a restart.

CLI Example:

```
salt '*' softwareupdate.list_upgrades
```

salt.modules.softwareupdate.**reset_ignored**()
Make sure the ignored updates are not ignored anymore, returns a list of the updates that are no longer ignored.

CLI Example:

```
salt '*' softwareupdate.reset_ignored
```

salt.modules.softwareupdate.**schedule**(**status*)
Decide if automatic checking for upgrades should be on or off. If no arguments are given it will return the current status. Append on or off to change the status.

Return values: - `True`: Automatic checking is now on, - `False`: Automatic checking is now off, - `None`: Invalid argument.

CLI Example:

```
salt '*' softwareupdate.schedule
salt '*' softwareupdate.schedule on|off
```

salt.modules.softwareupdate.**upgrade**(*rec=False*, *restart=True*)
Install all available upgrades. Returns a dictionary containing the name of the update and the status of its installation.

Return values: - `True`: The update was installed. - `False`: The update was not installed.

rec If set to True, only install all the recommended updates.

restart Set this to False if you do not want to install updates that require a restart.

CLI Example:

```
    salt '*' softwareupdate.upgrade
```

salt.modules.softwareupdate.**upgrade_available**(*update*)
Check whether or not an upgrade is available with a given name.

CLI Example:

```
    salt '*' softwareupdate.upgrade_available <update-name>
    salt '*' softwareupdate.upgrade_available "<update with whitespace>"
```

22.16.219 salt.modules.solaris_group

Manage groups on Solaris

salt.modules.solaris_group.**add**(*name*, *gid=None*, ***kwargs*)
Add the specified group

CLI Example:

```
    salt '*' group.add foo 3456
```

salt.modules.solaris_group.**chgid**(*name*, *gid*)
Change the gid for a named group

CLI Example:

```
    salt '*' group.chgid foo 4376
```

salt.modules.solaris_group.**delete**(*name*)
Remove the named group

CLI Example:

```
    salt '*' group.delete foo
```

salt.modules.solaris_group.**getent**(*refresh=False*)
Return info on all groups

CLI Example:

```
    salt '*' group.getent
```

salt.modules.solaris_group.**info**(*name*)
Return information about a group

CLI Example:

```
    salt '*' group.info foo
```

22.16.220 salt.modules.solaris_shadow

Manage the password database on Solaris systems

salt.modules.solaris_shadow.**default_hash**()
Returns the default hash used for unset passwords

CLI Example:

```
salt '*' shadow.default_hash
```

salt.modules.solaris_shadow.**info**(*name*)
Return information for the specified user

CLI Example:

```
salt '*' shadow.info root
```

salt.modules.solaris_shadow.**set_maxdays**(*name*, *maxdays*)
Set the maximum number of days during which a password is valid. See man passwd.

CLI Example:

```
salt '*' shadow.set_maxdays username 90
```

salt.modules.solaris_shadow.**set_mindays**(*name*, *mindays*)
Set the minimum number of days between password changes. See man passwd.

CLI Example:

```
salt '*' shadow.set_mindays username 7
```

salt.modules.solaris_shadow.**set_password**(*name*, *password*)
Set the password for a named user. The password must be a properly defined hash, the password hash can be generated with this command: `openssl passwd -1 <plaintext password>`

CLI Example:

```
salt '*' shadow.set_password root $1$UYCIxa628.9qXjpQCjM4a..
```

salt.modules.solaris_shadow.**set_warndays**(*name*, *warndays*)
Set the number of days of warning before a password change is required. See man passwd.

CLI Example:

```
salt '*' shadow.set_warndays username 7
```

22.16.221 salt.modules.solaris_user

Manage users with the useradd command

salt.modules.solaris_user.**add**(*name*, *uid=None*, *gid=None*, *groups=None*, *home=None*, *shell=None*, *unique=True*, *fullname=''*, *roomnumber=''*, *workphone=''*, *homephone=''*, *createhome=True*, ***kwargs*)
Add a user to the minion

CLI Example:

```
salt '*' user.add name <uid> <gid> <groups> <home> <shell>
```

salt.modules.solaris_user.**chfullname**(*name*, *fullname*)
Change the user's Full Name

CLI Example:

```
salt '*' user.chfullname foo "Foo Bar"
```

salt.modules.solaris_user.**chgid**(*name*, *gid*)
Change the default group of the user

CLI Example:

```
salt '*' user.chgid foo 4376
```

salt.modules.solaris_user.**chgroups**(*name*, *groups*, *append=False*)
Change the groups this user belongs to, add append to append the specified groups

CLI Example:

```
salt '*' user.chgroups foo wheel,root True
```

salt.modules.solaris_user.**chhome**(*name*, *home*, *persist=False*)
Change the home directory of the user, pass true for persist to copy files to the new home dir

CLI Example:

```
salt '*' user.chhome foo /home/users/foo True
```

salt.modules.solaris_user.**chhomephone**(*name*, *homephone*)
Change the user's Home Phone

CLI Example:

```
salt '*' user.chhomephone foo "7735551234"
```

salt.modules.solaris_user.**chroomnumber**(*name*, *roomnumber*)
Change the user's Room Number

CLI Example:

```
salt '*' user.chroomnumber foo 123
```

salt.modules.solaris_user.**chshell**(*name*, *shell*)
Change the default shell of the user

CLI Example:

```
salt '*' user.chshell foo /bin/zsh
```

salt.modules.solaris_user.**chuid**(*name*, *uid*)
Change the uid for a named user

CLI Example:

```
salt '*' user.chuid foo 4376
```

salt.modules.solaris_user.**chworkphone**(*name*, *workphone*)
Change the user's Work Phone

CLI Example:

```
salt '*' user.chworkphone foo "7735550123"
```

salt.modules.solaris_user.**delete**(*name*, *remove=False*, *force=False*)
Remove a user from the minion

CLI Example:

```
salt '*' user.delete name remove=True force=True
```

salt.modules.solaris_user.**getent**(*refresh=False*)
Return the list of all info for all users

CLI Example:

```
salt '*' user.getent
```

salt.modules.solaris_user.**info**(*name*)
Return user information

CLI Example:

```
salt '*' user.info root
```

salt.modules.solaris_user.**list_groups**(*name*)
Return a list of groups the named user belongs to

CLI Example:

```
salt '*' user.list_groups foo
```

salt.modules.solaris_user.**rename**(*name*, *new_name*)
Change the username for a named user

CLI Example:

```
salt '*' user.rename name new_name
```

22.16.222 salt.modules.solarisips

IPS pkg support for Solaris

This module provides support for Solaris 11 new package management - IPS (Image Packaging System). In order to manage the IPS packages using salt, you need to override the `pkg` provider by setting the `providers` parameter in your Minion config file like this:

```
providers:
  pkg: solarisips
```

Or you can set the provider in sls for each pkg: .. code-block:: yaml

mypackage:

pkg.installed:

- provider: solarisips

salt.modules.solarisips.**available_version**(*name*, ***kwargs*)
The available version of the package in the repository. In case of multiple match, it returns list of all matched packages. Accepts full or partial FMRI.

CLI Example:

```
salt '*' pkg.latest_version pkg://solaris/entire
```

salt.modules.solarisips.**get_fmri**(*name*, ***kwargs*)
Returns FMRI from partial name. Returns `' if not found. In case of multiple match, the function returns list of all matched packages.

CLI Example:

```
salt '*' pkg.get_fmri bash
```

salt.modules.solarisips.**install**(*name=None*, *refresh=False*, *pkgs=None*, *version=None*, *test=False*, ***kwargs*)

Install the named package using the IPS pkg command. Accepts full or partial FMRI.

Returns a dict containing the new package names and versions:

```
{'<package>': {'old': '<old-version>',
               'new': '<new-version>'}}
```

Multiple Package Installation Options:

pkgs A list of packages to install. Must be passed as a python list.

CLI Example:

```
salt '*' pkg.install vim
salt '*' pkg.install pkg://solaris/editor/vim
salt '*' pkg.install pkg://solaris/editor/vim refresh=True
```

salt.modules.solarisips.**is_installed**(*name*, ***kwargs*)

Returns True if the package is installed. Otherwise returns False. Name can be full or partial FMRI. In case of multiple match from partial FMRI name, it returns True.

CLI Example:

```
salt '*' pkg.is_installed bash
```

salt.modules.solarisips.**latest_version**(*name*, ***kwargs*)

The available version of the package in the repository. In case of multiple match, it returns list of all matched packages. Accepts full or partial FMRI.

CLI Example:

```
salt '*' pkg.latest_version pkg://solaris/entire
```

salt.modules.solarisips.**list_pkgs**(*versions_as_list=False*, ***kwargs*)

List the currently installed packages as a dict:

```
{'<package_name>': '<version>'}
```

CLI Example:

```
salt '*' pkg.list_pkgs
```

salt.modules.solarisips.**list_upgrades**(*refresh=False*)

Lists all packages available for update. When run in global zone, it reports only upgradable packages for the global zone. When run in non-global zone, it can report more upgradable packages than ``pkg update -vn'' because ``pkg update'' hides packages that require newer version of pkg://solaris/entire (which means that they can be upgraded only from global zone). Simply said: if you see pkg://solaris/entire in the list of upgrades, you should upgrade the global zone to get all possible updates. You can force full pkg DB refresh before listing.

CLI Example:

```
salt '*' pkg.list_upgrades
salt '*' pkg.list_upgrades refresh=True
```

salt.modules.solarisips.**normalize_name**(*name*, ***kwargs*)

Normalizes pkg name to full FMRI before running pkg.install. In case of multiple match or no match, it returns the name without modifications and lets the ``pkg install'' to decide what to do.

CLI Example:

```
salt '*' pkg.normalize_name vim
```

salt.modules.solarisips.**purge**(*name*, ***kwargs*)

Remove specified package. Accepts full or partial FMRI.

Returns a list containing the removed packages.

CLI Example:

```
salt '*' pkg.purge <package name>
```

salt.modules.solarisips.**refresh_db**(*full=False*)

Updates the remote repos database. You can force the full pkg DB refresh from all publishers regardless the last refresh time.

CLI Example:

```
salt '*' pkg.refresh_db
salt '*' pkg.refresh_db full=True
```

salt.modules.solarisips.**remove**(*name=None*, *pkgs=None*, ***kwargs*)

Remove specified package. Accepts full or partial FMRI. In case of multiple match, the command fails and won't modify the OS.

name The name of the package to be deleted.

Multiple Package Options:

pkgs A list of packages to delete. Must be passed as a python list. The `name` parameter will be ignored if this option is passed.

Returns a list containing the removed packages.

CLI Example:

```
salt '*' pkg.remove <package name>
salt '*' pkg.remove tcsh
salt '*' pkg.remove pkg://solaris/shell/tcsh
salt '*' pkg.remove pkgs='["foo", "bar"]'
```

salt.modules.solarisips.**search**(*name*, *versions_as_list=False*, ***kwargs*)

Searches the repository for given pkg name. The name can be full or partial FMRI. All matches are printed. Globs are also supported.

CLI Example:

```
salt '*' pkg.is_installed bash
```

salt.modules.solarisips.**upgrade**(*refresh=False*, ***kwargs*)

Upgrade all packages to the latest possible version. When run in global zone, it updates also all non-global zones. In non-global zones upgrade is limited by dependency constrains linked to the version of pkg://solaris/entire.

Returns also a raw output of ``pkg update'' command (because if update creates a new boot environment, no immediate changes are visible in ``pkg list'').

CLI Example:

```
salt '*' pkg.upgrade
```

salt.modules.solarisips.**upgrade_available**(*name*)

Check if there is an upgrade available for a certain package Accepts full or partial FMRI. Returns all matches found.

CLI Example:

```
salt '*' pkg.upgrade_available apache-22
```

salt.modules.solarisips.**version**(**names*, ***kwargs*)

Common interface for obtaining the version of installed packages. Accepts full or partial FMRI. If called using pkg_resource, full FMRI is required.

CLI Example:

```
salt '*' pkg.version vim
salt '*' pkg.version foo bar baz
salt '*' pkg_resource.version pkg://solaris/entire
```

22.16.223 salt.modules.solarispkg

Package support for Solaris

salt.modules.solarispkg.**install**(*name=None*, *sources=None*, *saltenv='base'*, ***kwargs*)

Install the passed package. Can install packages from the following sources:

```
* Locally (package already exists on the minion
* HTTP/HTTPS server
* FTP server
* Salt master
```

Returns a dict containing the new package names and versions:

```
{'<package>': {'old': '<old-version>',
               'new': '<new-version>'}}
```

CLI Example, installing a data stream pkg that already exists on the minion:

```
salt '*' pkg.install sources='[{"<pkg name>": "/dir/on/minion/<pkg filename>"}]'
salt '*' pkg.install sources='[{"SMClgcc346": "/var/spool/pkg/gcc-3.4.6-sol10-sparc-local.pkg"}]
```

CLI Example, installing a data stream pkg that exists on the salt master:

```
salt '*' pkg.install sources='[{"<pkg name>": "salt://pkgs/<pkg filename>"}]'
salt '*' pkg.install sources='[{"SMClgcc346": "salt://pkgs/gcc-3.4.6-sol10-sparc-local.pkg"}]'
```

CLI Example, installing a data stream pkg that exists on a HTTP server:

```
salt '*' pkg.install sources='[{"<pkg name>": "http://packages.server.com/<pkg filename>"}]'
salt '*' pkg.install sources='[{"SMClgcc346": "http://packages.server.com/gcc-3.4.6-sol10-sparc-
```

If working with solaris zones and you want to install a package only in the global zone you can pass `current_zone_only=True' to salt to have the package only installed in the global zone. (Behind the scenes this is passing `-G' to the pkgadd command.) Solaris default when installing a package in the global zone is to install it in all zones. This overrides that and installs the package only in the global.

CLI Example, installing a data stream package only in the global zone:

```
salt 'global_zone' pkg.install sources='[{"SMClgcc346": "/var/spool/pkg/gcc-3.4.6-sol10-sparc-lo
```

By default salt automatically provides an adminfile, to automate package installation, with these options set:

```
email=
instance=quit
partial=nocheck
runlevel=nocheck
idepend=nocheck
rdepend=nocheck
space=nocheck
setuid=nocheck
conflict=nocheck
action=nocheck
basedir=default
```

You can override any of these options in two ways. First you can optionally pass any of the options as a kwarg to the module/state to override the default value or you can optionally pass the `admin_source' option providing your own adminfile to the minions.

Note: You can find all of the possible options to provide to the adminfile by reading the admin man page:

```
man -s 4 admin
```

CLI Example - Overriding the `instance' adminfile option when calling the module directly:

```
salt '*' pkg.install sources='[{"<pkg name>": "salt://pkgs/<pkg filename>"}]' instance="overwrit
```

CLI Example - Overriding the `instance' adminfile option when used in a state:

```
SMClgcc346:
  pkg.installed:
    - sources:
      - SMClgcc346: salt://srv/salt/pkgs/gcc-3.4.6-sol10-sparc-local.pkg
    - instance: overwrite
```

Note: the ID declaration is ignored, as the package name is read from the ``sources'' parameter.

CLI Example - Providing your own adminfile when calling the module directly:

```
salt '*' pkg.install sources='[{"<pkg name>": "salt://pkgs/<pkg filename>"}]' admin_source='salt
```

CLI Example - Providing your own adminfile when using states:

```
<pkg name>:
  pkg.installed:
    - sources:
      - <pkg name>: salt://pkgs/<pkg filename>
    - admin_source: salt://pkgs/<adminfile filename>
```

Note: the ID declaration is ignored, as the package name is read from the ``sources'' parameter.

salt.modules.solarispkg.**latest_version**(**names, **kwargs*)

Return the latest version of the named package available for upgrade or installation. If more than one package name is specified, a dict of name/version pairs is returned.

If the latest version of a given package is already installed, an empty string will be returned for that package.

CLI Example:

```
salt '*' pkg.latest_version <package name>
salt '*' pkg.latest_version <package1> <package2> <package3> ...
```

NOTE: As package repositories are not presently supported for Solaris pkgadd, this function will always return an empty string for a given package.

salt.modules.solarispkg.**list_pkgs**(*versions_as_list=False*, ***kwargs*)

List the packages currently installed as a dict:

```
{'<package_name>': '<version>'}
```

CLI Example:

```
salt '*' pkg.list_pkgs
```

salt.modules.solarispkg.**purge**(*name=None*, *pkgs=None*, ***kwargs*)

Package purges are not supported, this function is identical to `remove()`.

name The name of the package to be deleted

Multiple Package Options:

pkgs A list of packages to delete. Must be passed as a python list. The `name` parameter will be ignored if this option is passed.

New in version 0.16.0.

Returns a dict containing the changes.

CLI Example:

```
salt '*' pkg.purge <package name>
salt '*' pkg.purge <package1>,<package2>,<package3>
salt '*' pkg.purge pkgs='["foo", "bar"]'
```

salt.modules.solarispkg.**remove**(*name=None*, *pkgs=None*, *saltenv='base'*, ***kwargs*)

Remove packages with pkgrm

name The name of the package to be deleted

By default salt automatically provides an adminfile, to automate package removal, with these options set:

```
email=
instance=quit
partial=nocheck
runlevel=nocheck
idepend=nocheck
rdepend=nocheck
space=nocheck
setuid=nocheck
conflict=nocheck
action=nocheck
basedir=default
```

You can override any of these options in two ways. First you can optionally pass any of the options as a kwarg to the module/state to override the default value or you can optionally pass the `admin_source' option providing your own adminfile to the minions.

Note: You can find all of the possible options to provide to the adminfile by reading the admin man page:

```
man -s 4 admin
```

Multiple Package Options:

pkgs A list of packages to delete. Must be passed as a python list. The `name` parameter will be ignored if this option is passed.

New in version 0.16.0.

Returns a dict containing the changes.

CLI Example:

```
salt '*' pkg.remove <package name>
salt '*' pkg.remove SUNWgit
salt '*' pkg.remove <package1>,<package2>,<package3>
salt '*' pkg.remove pkgs='["foo", "bar"]'
```

salt.modules.solarispkg.**upgrade_available**(*name*)
Check whether or not an upgrade is available for a given package

CLI Example:

```
salt '*' pkg.upgrade_available <package name>
```

salt.modules.solarispkg.**version**(**names*, ***kwargs*)
Returns a string representing the package version or an empty string if not installed. If more than one package name is specified, a dict of name/version pairs is returned.

CLI Example:

```
salt '*' pkg.version <package name>
salt '*' pkg.version <package1> <package2> <package3> ...
```

22.16.224 salt.modules.solr

Apache Solr Salt Module

Author: Jed Glazner Version: 0.2.1 Modified: 12/09/2011

This module uses HTTP requests to talk to the apache solr request handlers to gather information and report errors. Because of this the minion doesn't necessarily need to reside on the actual slave. However if you want to use the signal function the minion must reside on the physical solr host.

This module supports multi-core and standard setups. Certain methods are master/slave specific. Make sure you set the solr.type. If you have questions or want a feature request please ask.

Coming Features in 0.3

1. Add command for checking for replication failures on slaves
2. Improve match_index_versions since it's pointless on busy solr masters
3. Add additional local fs checks for backups to make sure they succeeded

Override these in the minion config

solr.cores A list of core names e.g. ['core1',core2']. An empty list indicates non-multicore setup.

solr.baseurl The root level URL to access solr via HTTP

solr.request_timeout The number of seconds before timing out an HTTP/HTTPS/FTP request. If nothing is specified then the python global timeout setting is used.

solr.type Possible values are `master' or `slave'

solr.backup_path The path to store your backups. If you are using cores and you can specify to append the core name to the path in the backup method.

solr.num_backups For versions of solr >= 3.5. Indicates the number of backups to keep. This option is ignored if your version is less.

solr.init_script The full path to your init script with start/stop options

solr.dih.options A list of options to pass to the DIH.

Required Options for DIH

clean [False] Clear the index before importing

commit [True] Commit the documents to the index upon completion

optimize [True] Optimize the index after commit is complete

verbose [True] Get verbose output

`salt.modules.solr.`**`abort_import`**`(`*handler, host=None, core_name=None, verbose=False*`)`

MASTER ONLY Aborts an existing import command to the specified handler. This command can only be run if the minion is configured with solr.type=master

handler [str] The name of the data import handler.

host [str (None)] The solr host to query. __opts__['host'] is default.

core [str (None)] The core the handler belongs to.

verbose [boolean (False)] Run the command with verbose output.

Return : dict<str,obj>:

```
{'success':boolean, 'data':dict, 'errors':list, 'warnings':list}
```

CLI Example:

```
salt '*' solr.abort_import dataimport None music {'clean':True}
```

`salt.modules.solr.`**`backup`**`(`*host=None, core_name=None, append_core_to_path=False*`)`

Tell solr make a backup. This method can be mis-leading since it uses the backup API. If an error happens during the backup you are not notified. The status: `OK' in the response simply means that solr received the request successfully.

host [str (None)] The solr host to query. __opts__['host'] is default.

core_name [str (None)] The name of the solr core if using cores. Leave this blank if you are not using cores or if you want to check all cores.

append_core_to_path [boolean (False)] If True add the name of the core to the backup path. Assumes that minion backup path is not None.

Return : dict<str,obj>:

```
{'success':boolean, 'data':dict, 'errors':list, 'warnings':list}
```

CLI Example:

```
salt '*' solr.backup music
```

`salt.modules.solr.`**`core_status`**`(`*host=None, core_name=None*`)`

MULTI-CORE HOSTS ONLY Get the status for a given core or all cores if no core is specified

host [str (None)] The solr host to query. __opts__['host'] is default.

core_name [str] The name of the core to reload

Return : dict<str,obj>:

```
{'success':boolean, 'data':dict, 'errors':list, 'warnings':list}
```

CLI Example:

```
salt '*' solr.core_status None music
```

salt.modules.solr.**delta_import**(*handler*, *host=None*, *core_name=None*, *options=None*, *extra=None*)

Submits an import command to the specified handler using specified options. This command can only be run if the minion is configured with solr.type=master

handler [str] The name of the data import handler.

host [str (None)] The solr host to query. __opts__['host'] is default.

core [str (None)] The core the handler belongs to.

options [dict (__opts__)] A list of options such as clean, optimize commit, verbose, and pause_replication. leave blank to use __opts__ defaults. options will be merged with __opts__

extra [dict ([])] Extra name value pairs to pass to the handler. e.g. ["name=value"]

Return : dict<str,obj>:

```
{'success':boolean, 'data':dict, 'errors':list, 'warnings':list}
```

CLI Example:

```
salt '*' solr.delta_import dataimport None music {'clean':True}
```

salt.modules.solr.**full_import**(*handler*, *host=None*, *core_name=None*, *options=None*, *extra=None*)

MASTER ONLY Submits an import command to the specified handler using specified options. This command can only be run if the minion is configured with solr.type=master

handler [str] The name of the data import handler.

host [str (None)] The solr host to query. __opts__['host'] is default.

core [str (None)] The core the handler belongs to.

options [dict (__opts__)] A list of options such as clean, optimize commit, verbose, and pause_replication. leave blank to use __opts__ defaults. options will be merged with __opts__

extra [dict ([])] Extra name value pairs to pass to the handler. e.g. ["name=value"]

Return : dict<str,obj>:

```
{'success':boolean, 'data':dict, 'errors':list, 'warnings':list}
```

CLI Example:

```
salt '*' solr.full_import dataimport None music {'clean':True}
```

salt.modules.solr.**import_status**(*handler*, *host=None*, *core_name=None*, *verbose=False*)

Submits an import command to the specified handler using specified options. This command can only be run if the minion is configured with solr.type: `master'

handler [str] The name of the data import handler.

host [str (None)] The solr host to query. __opts__['host'] is default.

core [str (None)] The core the handler belongs to.

verbose [boolean (False)] Specifies verbose output

Return : dict<str,obj>:

```
{'success':boolean, 'data':dict, 'errors':list, 'warnings':list}
```

CLI Example:

```
salt '*' solr.import_status dataimport None music False
```

salt.modules.solr.**is_replication_enabled**(*host=None*, *core_name=None*)

SLAVE CALL Check for errors, and determine if a slave is replicating or not.

host [str (None)] The solr host to query. __opts__['host'] is default.

core_name [str (None)] The name of the solr core if using cores. Leave this blank if you are not using cores or if you want to check all cores.

Return : dict<str,obj>:

```
{'success':boolean, 'data':dict, 'errors':list, 'warnings':list}
```

CLI Example:

```
salt '*' solr.is_replication_enabled music
```

salt.modules.solr.**lucene_version**(*core_name=None*)

Gets the lucene version that solr is using. If you are running a multi-core setup you should specify a core name since all the cores run under the same servlet container, they will all have the same version.

core_name [str (None)] The name of the solr core if using cores. Leave this blank if you are not using cores or if you want to check all cores.

Return: dict<str,obj>:

```
{'success':boolean, 'data':dict, 'errors':list, 'warnings':list}
```

CLI Example:

```
salt '*' solr.lucene_version
```

salt.modules.solr.**match_index_versions**(*host=None*, *core_name=None*)

SLAVE CALL Verifies that the master and the slave versions are in sync by comparing the index version. If you are constantly pushing updates the index the master and slave versions will seldom match. A solution to this is pause indexing every so often to allow the slave to replicate and then call this method before allowing indexing to resume.

host [str (None)] The solr host to query. __opts__['host'] is default.

core_name [str (None)] The name of the solr core if using cores. Leave this blank if you are not using cores or if you want to check all cores.

Return : dict<str,obj>:

```
{'success':boolean, 'data':dict, 'errors':list, 'warnings':list}
```

CLI Example:

```
salt '*' solr.match_index_versions music
```

salt.modules.solr.**optimize**(*host=None*, *core_name=None*)

Search queries fast, but it is a very expensive operation. The ideal process is to run this with a master/slave configuration. Then you can optimize the master, and push the optimized index to the slaves. If you are running a single solr instance, or if you are going to run this on a slave be aware than search performance will be horrible while this command is being run. Additionally it can take a LONG time to run and your HTTP request may timeout. If that happens adjust your timeout settings.

host [str (None)] The solr host to query. __opts__['host'] is default.

core_name [str (None)] The name of the solr core if using cores. Leave this blank if you are not using cores or if you want to check all cores.

Return : dict<str,obj>:

```
{'success':boolean, 'data':dict, 'errors':list, 'warnings':list}
```

CLI Example:

```
salt '*' solr.optimize music
```

salt.modules.solr.**ping**(*host=None*, *core_name=None*)

Does a health check on solr, makes sure solr can talk to the indexes.

host [str (None)] The solr host to query. __opts__['host'] is default.

core_name [str (None)] The name of the solr core if using cores. Leave this blank if you are not using cores or if you want to check all cores.

Return : dict<str,obj>:

```
{'success':boolean, 'data':dict, 'errors':list, 'warnings':list}
```

CLI Example:

```
salt '*' solr.ping music
```

salt.modules.solr.**reload_core**(*host=None*, *core_name=None*)

MULTI-CORE HOSTS ONLY Load a new core from the same configuration as an existing registered core. While the ``new'' core is initializing, the ``old'' one will continue to accept requests. Once it has finished, all new request will go to the ``new'' core, and the ``old'' core will be unloaded.

host [str (None)] The solr host to query. __opts__['host'] is default.

core_name [str] The name of the core to reload

Return : dict<str,obj>:

```
{'success':boolean, 'data':dict, 'errors':list, 'warnings':list}
```

CLI Example:

```
salt '*' solr.reload_core None music
```

Return data is in the following format:

```
{'success':bool, 'data':dict, 'errors':list, 'warnings':list}
```

salt.modules.solr.**reload_import_config**(*handler*, *host=None*, *core_name=None*, *verbose=False*)

MASTER ONLY re-loads the handler config XML file. This command can only be run if the minion is a `master' type

handler [str] The name of the data import handler.

host [str (None)] The solr host to query. __opts__['host'] is default.

core [str (None)] The core the handler belongs to.

verbose [boolean (False)] Run the command with verbose output.

Return : dict<str,obj>:

```
{'success':boolean, 'data':dict, 'errors':list, 'warnings':list}
```

CLI Example:

```
salt '*' solr.reload_import_config dataimport None music {'clean':True}
```

salt.modules.solr.**replication_details**(*host=None*, *core_name=None*)

Get the full replication details.

host [str (None)] The solr host to query. __opts__['host'] is default.

core_name [str (None)] The name of the solr core if using cores. Leave this blank if you are not using cores or if you want to check all cores.

Return : dict<str,obj>:

```
{'success':boolean, 'data':dict, 'errors':list, 'warnings':list}
```

CLI Example:

```
salt '*' solr.replication_details music
```

salt.modules.solr.**set_is_polling**(*polling*, *host=None*, *core_name=None*)

SLAVE CALL Prevent the slaves from polling the master for updates.

polling [boolean] True will enable polling. False will disable it.

host [str (None)] The solr host to query. __opts__['host'] is default.

core_name [str (None)] The name of the solr core if using cores. Leave this blank if you are not using cores or if you want to check all cores.

Return : dict<str,obj>:

```
{'success':boolean, 'data':dict, 'errors':list, 'warnings':list}
```

CLI Example:

```
salt '*' solr.set_is_polling False
```

salt.modules.solr.**set_replication_enabled**(*status*, *host=None*, *core_name=None*)

MASTER ONLY Sets the master to ignore poll requests from the slaves. Useful when you don't want the slaves replicating during indexing or when clearing the index.

status [boolean] Sets the replication status to the specified state.

host [str (None)] The solr host to query. __opts__['host'] is default.

core_name [str (None)] The name of the solr core if using cores. Leave this blank if you are not using cores or if you want to set the status on all cores.

Return : dict<str,obj>:

```
{'success':boolean, 'data':dict, 'errors':list, 'warnings':list}
```

CLI Example:

```
salt '*' solr.set_replication_enabled false, None, music
```

salt.modules.solr.**signal**(*signal=None*)

Signals Apache Solr to start, stop, or restart. Obviously this is only going to work if the minion resides on the solr host. Additionally Solr doesn't ship with an init script so one must be created.

signal [str (None)] The command to pass to the apache solr init valid values are `start', `stop', and `restart'

CLI Example:

```
salt '*' solr.signal restart
```

salt.modules.solr.**version**(*core_name=None*)

Gets the solr version for the core specified. You should specify a core here as all the cores will run under the same servlet container and so will all have the same version.

core_name [str (None)] The name of the solr core if using cores. Leave this blank if you are not using cores or if you want to check all cores.

Return : dict<str,obj>:

```
{'success':boolean, 'data':dict, 'errors':list, 'warnings':list}
```

CLI Example:

```
salt '*' solr.version
```

22.16.225 salt.modules.splunk_search

Module for interop with the Splunk API

New in version 2015.5.0.

depends

- splunk-sdk python module

configuration Configure this module by specifying the name of a configuration profile in the minion config, minion pillar, or master config. The module will use the `splunk' key by default, if defined.

For example:

```
splunk:
    username: alice
    password: abc123
    host: example.splunkcloud.com
    port: 8080
```

salt.modules.splunk_search.**create**(*name*, *profile='splunk'*, ***kwargs*)

Create a splunk search

CLI Example:

splunk_search.create `my search name' search='error msg'

salt.modules.splunk_search.**delete**(*name*, *profile='splunk'*)
Delete a splunk search

CLI Example:

splunk_search.delete `my search name'

salt.modules.splunk_search.**get**(*name*, *profile='splunk'*)
Get a splunk search

CLI Example:

splunk_search.get `my search name'

salt.modules.splunk_search.**list**(*profile='splunk'*)
List splunk searches (names only)

CLI Example: splunk_search.list

salt.modules.splunk_search.**list_all**(*prefix=None*, *app=None*, *owner=None*, *description_contains=None*, *name_not_contains=None*, *profile='splunk'*)
Get all splunk search details. Produces results that can be used to create an sls file.

if app or owner are specified, results will be limited to matching saved searches.

if description_contains is specified, results will be limited to those where ``description_contains in description'' is true if name_not_contains is specified, results will be limited to those where ``name_not_contains not in name'' is true.

If prefix parameter is given, alarm names in the output will be prepended with the prefix; alarms that have the prefix will be skipped. This can be used to convert existing alarms to be managed by salt, as follows:

CLI example:

1. **Make a ``backup'' of all existing searches** $ salt-call splunk_search.list_all --out=txt | sed ``s/local: //'' > legacy_searches.sls
2. **Get all searches with new prefixed names** $ salt-call splunk_search.list_all ``prefix=**MANAGED BY SALT** '' --out=txt | sed ``s/local: //'' > managed_searches.sls
3. **Insert the managed searches into splunk** $ salt-call state.sls managed_searches.sls
4. Manually verify that the new searches look right
5. Delete the original searches $ sed s/present/absent/ legacy_searches.sls > remove_legacy_searches.sls $ salt-call state.sls remove_legacy_searches.sls
6. Get all searches again, verify no changes $ salt-call splunk_search.list_all --out=txt | sed ``s/local: //'' > final_searches.sls $ diff final_searches.sls managed_searches.sls

salt.modules.splunk_search.**update**(*name*, *profile='splunk'*, ***kwargs*)
Update a splunk search

CLI Example:

splunk_search.update `my search name' sharing=app

22.16.226 salt.modules.sqlite3

Support for SQLite3

salt.modules.sqlite3.**fetch**(*db=None*, *sql=None*)
Retrieve data from an sqlite3 db (returns all rows, be careful!)

CLI Example:

```
salt '*' sqlite3.fetch /root/test.db 'SELECT * FROM test;'
```

salt.modules.sqlite3.**indexes**(*db=None*)
Show all indices in the database, for people with poor spelling skills

CLI Example:

```
salt '*' sqlite3.indexes /root/test.db
```

salt.modules.sqlite3.**indices**(*db=None*)
Show all indices in the database

CLI Example:

```
salt '*' sqlite3.indices /root/test.db
```

salt.modules.sqlite3.**modify**(*db=None*, *sql=None*)
Issue an SQL query to sqlite3 (with no return data), usually used to modify the database in some way (insert, delete, create, etc)

CLI Example:

```
salt '*' sqlite3.modify /root/test.db 'CREATE TABLE test(id INT, testdata TEXT);'
```

salt.modules.sqlite3.**sqlite_version**()
Return version of sqlite

CLI Example:

```
salt '*' sqlite3.sqlite_version
```

salt.modules.sqlite3.**tables**(*db=None*)
Show all tables in the database

CLI Example:

```
salt '*' sqlite3.tables /root/test.db
```

salt.modules.sqlite3.**version**()
Return version of pysqlite

CLI Example:

```
salt '*' sqlite3.version
```

22.16.227 salt.modules.ssh

Manage client ssh components

Note: This module requires the use of MD5 hashing. Certain security audits may not permit the use of MD5. For those cases, this module should be disabled or removed.

---

salt.modules.ssh.**auth_keys**(*user=None*, *config='.ssh/authorized_keys'*)
Return the authorized keys for users

CLI Example:

```
salt '*' ssh.auth_keys
salt '*' ssh.auth_keys root
salt '*' ssh.auth_keys user=root
salt '*' ssh.auth_keys user="[user1, user2]"
```

salt.modules.ssh.**check_key**(*user*, *key*, *enc*, *comment*, *options*, *config='.ssh/authorized_keys'*, *cache_keys=None*)

Check to see if a key needs updating, returns ``update'', ``add'' or ``exists''

CLI Example:

```
salt '*' ssh.check_key <user> <key> <enc> <comment> <options>
```

salt.modules.ssh.**check_key_file**(*user*, *source*, *config='.ssh/authorized_keys'*, *saltenv='base'*, *env=None*)

Check a keyfile from a source destination against the local keys and return the keys to change

CLI Example:

```
salt '*' root salt://ssh/keyfile
```

salt.modules.ssh.**check_known_host**(*user=None*, *hostname=None*, *key=None*, *fingerprint=None*, *config=None*)

Check the record in known_hosts file, either by its value or by fingerprint (it's enough to set up either key or fingerprint, you don't need to set up both).

If provided key or fingerprint doesn't match with stored value, return ``update'', if no value is found for a given host, return ``add'', otherwise return ``exists''.

If neither key, nor fingerprint is defined, then additional validation is not performed.

CLI Example:

```
salt '*' ssh.check_known_host <user> <hostname> key='AAAA...FAaQ=='
```

salt.modules.ssh.**get_known_host**(*user*, *hostname*, *config=None*)

Return information about known host from the configfile, if any. If there is no such key, return None.

CLI Example:

```
salt '*' ssh.get_known_host <user> <hostname>
```

salt.modules.ssh.**hash_known_hosts**(*user=None*, *config=None*)

Hash all the hostnames in the known hosts file.

New in version 2014.7.0.

CLI Example:

```
salt '*' ssh.hash_known_hosts
```

salt.modules.ssh.**host_keys**(*keydir=None*)

Return the minion's host keys

CLI Example:

```
salt '*' ssh.host_keys
```

salt.modules.ssh.**recv_known_host**(*hostname*, *enc=None*, *port=None*, *hash_hostname=False*)

Retrieve information about host public key from remote server

CLI Example:

```
salt '*' ssh.recv_known_host <hostname> enc=<enc> port=<port>
```

salt.modules.ssh.**rm_auth_key**(*user*, *key*, *config='.ssh/authorized_keys'*)

Remove an authorized key from the specified user's authorized key file

CLI Example:

```
salt '*' ssh.rm_auth_key <user> <key>
```

salt.modules.ssh.**rm_known_host**(*user=None*, *hostname=None*, *config=None*)

Remove all keys belonging to hostname from a known_hosts file.

CLI Example:

```
salt '*' ssh.rm_known_host <user> <hostname>
```

salt.modules.ssh.**set_auth_key**(*user*, *key*, *enc='ssh-rsa'*, *comment=''*, *options=None*, *config='.ssh/authorized_keys'*, *cache_keys=None*)

Add a key to the authorized_keys file. The ``key'' parameter must only be the string of text that is the encoded key. If the key begins with ``ssh-rsa'' or ends with user@host, remove those from the key before passing it to this function.

CLI Example:

```
salt '*' ssh.set_auth_key <user> '<key>' enc='dsa'
```

salt.modules.ssh.**set_auth_key_from_file**(*user*, *source*, *config='.ssh/authorized_keys'*, *saltenv='base'*, *env=None*)

Add a key to the authorized_keys file, using a file as the source.

CLI Example:

```
salt '*' ssh.set_auth_key_from_file <user> salt://ssh_keys/<user>.id_rsa.pub
```

salt.modules.ssh.**set_known_host**(*user=None*, *hostname=None*, *fingerprint=None*, *key=None*, *port=None*, *enc=None*, *hash_hostname=True*, *config=None*)

Download SSH public key from remote host ``hostname'', optionally validate its fingerprint against ``fingerprint'' variable and save the record in the known_hosts file.

If such a record does already exists in there, do nothing.

CLI Example:

```
salt '*' ssh.set_known_host <user> fingerprint='xx:xx:..:xx' enc='ssh-rsa' config='.ssh/known_ho
```

salt.modules.ssh.**user_keys**(*user=None*, *pubfile=None*, *prvfile=None*)

Return the user's ssh keys on the minion

New in version 2014.7.0.

CLI Example:

```
salt '*' ssh.user_keys
salt '*' ssh.user_keys user=user1
salt '*' ssh.user_keys user=user1 pubfile=/home/user1/.ssh/id_rsa.pub prvfile=/home/user1/.ssh/i
salt '*' ssh.user_keys user=user1 prvfile=False
salt '*' ssh.user_keys user="['user1','user2'] pubfile=id_rsa.pub prvfile=id_rsa
```

As you can see you can tell Salt not to read from the user's private (or public) key file by setting the file path to `False`. This can be useful to prevent Salt from publishing private data via Salt Mine or others.

22.16.228 salt.modules.state

Control the state system on the minion

salt.modules.state.**apply_**(*mods=None*, ***kwargs*)

New in version 2015.5.0.

Apply states! This function will call highstate or state.sls based on the arguments passed in, state.apply is intended to be the main gateway for all state executions.

CLI Example:

```
salt '*' state.apply
salt '*' state.apply test
salt '*' state.apply test,pkgs
```

salt.modules.state.**check_request**(*name=None*)

New in version 2015.5.0.

Return the state request information, if any

CLI Example:

```
salt '*' state.check_request
```

salt.modules.state.**clear_cache**()

Clear out cached state files, forcing even cache runs to refresh the cache on the next state execution.

Remember that the state cache is completely disabled by default, this execution only applies if cache=True is used in states

CLI Example:

```
salt '*' state.clear_cache
```

salt.modules.state.**clear_request**(*name=None*)

New in version 2015.5.0.

Clear out the state execution request without executing it

CLI Example:

```
salt '*' state.clear_request
```

salt.modules.state.**disable**(*states*)

Disable state runs.

CLI Example:

```
salt '*' state.disable highstate

salt '*' state.disable highstate,test.succeed_without_changes
```

Note: To disable a state file from running provide the same name that would be passed in a state.sls call.

salt `*' state.disable bind.config

---

salt.modules.state.**enable**(*states*)

Enable state function or sls run

CLI Example:

```
salt '*' state.enable highstate

salt '*' state.enable test.succeed_without_changes
```

Note: To enable a state file from running provide the same name that would be passed in a state.sls call.

salt `*' state.disable bind.config

---

salt.modules.state.**high**(*data*, *test=False*, *queue=False*, ***kwargs*)

Execute the compound calls stored in a single set of high data This function is mostly intended for testing the state system

CLI Example:

```
salt '*' state.high '{"vim": {"pkg": ["installed"]}}'
```

salt.modules.state.**highstate**(*test=None*, *queue=False*, ***kwargs*)

Retrieve the state data from the salt master for this minion and execute it

test Notify states to execute in test-only (dry-run) mode.

Sets the `test` variable in the minion `opts` for the duration of the state run.

pillar Custom Pillar data can be passed with the `pillar` kwarg. Values passed here will override hard-coded Pillar values.

queue [`False`] Instead of failing immediately when another state run is in progress, queue the new state run to begin running once the other has finished.

This option starts a new thread for each queued state run so use this option sparingly.

localconfig: Instead of using running minion opts, load `localconfig` and merge that with the running minion opts. This functionality is intended for using ``roots'' of salt directories (with their own minion config, pillars, file_roots) to run highstate out of.

CLI Example:

```
salt '*' state.highstate

salt '*' state.highstate whitelist=sls1_to_run,sls2_to_run
salt '*' state.highstate exclude=sls_to_exclude
salt '*' state.highstate exclude="[{'id': 'id_to_exclude'}, {'sls': 'sls_to_exclude'}]"

salt '*' state.highstate pillar="{foo: 'Foo!', bar: 'Bar!'}"
```

salt.modules.state.**list_disabled**()

List the states which are currently disabled

CLI Example:

```
salt '*' state.list_disabled
```

salt.modules.state.**low**(*data*, *queue=False*, ***kwargs*)

Execute a single low data call This function is mostly intended for testing the state system

CLI Example:

```
salt '*' state.low '{"state": "pkg", "fun": "installed", "name": "vi"}'
```

salt.modules.state.**pkg**(*pkg_path*, *pkg_sum*, *hash_type*, *test=False*, ***kwargs*)

Execute a packaged state run, the packaged state run will exist in a tarball available locally. This packaged state can be generated using salt-ssh.

CLI Example:

```
salt '*' state.pkg /tmp/state_pkg.tgz
```

salt.modules.state.**request**(*mods=None*, ***kwargs*)

New in version 2015.5.0.

Request that the local admin execute a state run via *salt-call state.run_request* All arguments match state.apply

CLI Example:

```
salt '*' state.request
salt '*' state.request test
salt '*' state.request test,pkgs
```

salt.modules.state.**run_request**(*name='default'*, ***kwargs*)

New in version 2015.5.0.

Execute the pending state request

CLI Example:

```
salt '*' state.run_request
```

salt.modules.state.**running**(*concurrent=False*)

Return a dict of state return data if a state function is already running. This function is used to prevent multiple state calls from being run at the same time.

CLI Example:

```
salt '*' state.running
```

salt.modules.state.**show_highstate**(*queue=False*, ***kwargs*)

Retrieve the highstate data from the salt master and display it

Custom Pillar data can be passed with the `pillar` kwarg.

CLI Example:

```
salt '*' state.show_highstate
```

salt.modules.state.**show_low_sls**(*mods*, *saltenv='base'*, *test=None*, *queue=False*, *env=None*, ***kwargs*)

Display the low data from a specific sls. The default environment is `base`, use `saltenv` (`env` in Salt 0.17.x and older) to specify a different environment.

CLI Example:

```
salt '*' state.show_low_sls foo
```

salt.modules.state.**show_lowstate**(*queue=False*, ***kwargs*)

List out the low data that will be applied to this minion

CLI Example:

```
salt '*' state.show_lowstate
```

salt.modules.state.**show_sls**(*mods*, *saltenv='base'*, *test=None*, *queue=False*, *env=None*, ***kwargs*)
Display the state data from a specific sls or list of sls files on the master. The default environment is `base`, use `saltenv` (`env` in Salt 0.17.x and older) to specify a different environment.

This function does not support topfiles. For `top.sls` please use `show_top` instead.

Custom Pillar data can be passed with the `pillar` kwarg.

CLI Example:

```
salt '*' state.show_sls core,edit.vim dev
```

salt.modules.state.**show_top**(*queue=False*, ***kwargs*)
Return the top data that the minion will use for a highstate

CLI Example:

```
salt '*' state.show_top
```

salt.modules.state.**single**(*fun*, *name*, *test=None*, *queue=False*, ***kwargs*)
Execute a single state function with the named kwargs, returns False if insufficient data is sent to the command

By default, the values of the kwargs will be parsed as YAML. So, you can specify lists values, or lists of single entry key-value maps, as you would in a YAML salt file. Alternatively, JSON format of keyword values is also supported.

CLI Example:

```
salt '*' state.single pkg.installed name=vim
```

salt.modules.state.**sls**(*mods*, *saltenv=None*, *test=None*, *exclude=None*, *queue=False*, *env=None*, ***kwargs*)
Execute a set list of state files from an environment.

test Notify states to execute in test-only (dry-run) mode.

Sets the `test` variable in the minion `opts` for the duration of the state run.

pillar Custom Pillar data can be passed with the `pillar` kwarg. Values passed here will override hard-coded Pillar values.

queue [`False`] Instead of failing immediately when another state run is in progress, queue the new state run to begin running once the other has finished.

This option starts a new thread for each queued state run so use this option sparingly.

saltenv [None] Specify a `file_roots` environment.

Changed in version 0.17.0: Argument name changed from `env` to `saltenv`.

Changed in version 2014.7: Defaults to None. If no saltenv is specified, the minion config will be checked for a saltenv and if found, it will be used. If none is found, base will be used.

concurrent: WARNING: This flag is potentially dangerous. It is designed for use when multiple state runs can safely be run at the same Do not use this flag for performance optimization.

localconfig: Instead of using running minion opts, load `localconfig` and merge that with the running minion opts. This functionality is intended for using ``roots'' of salt directories (with their own minion config, pillars, file_roots) to run highstate out of.

CLI Example:

```
    salt '*' state.sls core,edit.vim dev
    salt '*' state.sls core exclude="[{'id': 'id_to_exclude'}, {'sls': 'sls_to_exclude'}]"

    salt '*' state.sls myslsfile pillar="{foo: 'Foo!', bar: 'Bar!'}"
```

salt.modules.state.**sls_id**(*id_*, *mods*, *saltenv='base'*, *test=None*, *queue=False*, ***kwargs*)
Call a single ID from the named module(s) and handle all requisites

New in version 2014.7.0.

CLI Example:

```
    salt '*' state.sls_id apache http
```

salt.modules.state.**template**(*tem*, *queue=False*, ***kwargs*)
Execute the information stored in a template file on the minion.

This function does not ask a master for a SLS file to render but instead directly processes the file at the provided path on the minion.

CLI Example:

```
    salt '*' state.template '<Path to template on the minion>'
```

salt.modules.state.**template_str**(*tem*, *queue=False*, ***kwargs*)
Execute the information stored in a string from an sls template

CLI Example:

```
    salt '*' state.template_str '<Template String>'
```

salt.modules.state.**top**(*topfn*, *test=None*, *queue=False*, ***kwargs*)
Execute a specific top file instead of the default

CLI Example:

```
    salt '*' state.top reverse_top.sls
    salt '*' state.top reverse_top.sls exclude=sls_to_exclude
    salt '*' state.top reverse_top.sls exclude="[{'id': 'id_to_exclude'}, {'sls': 'sls_to_exclude'}]
```

22.16.229 salt.modules.status

Module for returning various status data about a minion. These data can be useful for compiling into stats later.

salt.modules.status.**all_status**()
Return a composite of all status data and info for this minion. Warning: There is a LOT here!

CLI Example:

```
    salt '*' status.all_status
```

salt.modules.status.**cpuinfo**()
Return the CPU info for this minion

CLI Example:

```
    salt '*' status.cpuinfo
```

salt.modules.status.**cpustats**()
Return the CPU stats for this minion

CLI Example:

```
salt '*' status.cpustats
```

salt.modules.status.**custom**()

Return a custom composite of status data and info for this minion, based on the minion config file. An example config like might be:

```
status.cpustats.custom: [ 'cpu', 'ctxt', 'btime', 'processes' ]
```

Where status refers to status.py, cpustats is the function where we get our data, and custom is this function It is followed by a list of keys that we want returned.

This function is meant to replace all_status(), which returns anything and everything, which we probably don't want.

By default, nothing is returned. Warning: Depending on what you include, there can be a LOT here!

CLI Example:

```
salt '*' status.custom
```

salt.modules.status.**diskstats**()

Return the disk stats for this minion

CLI Example:

```
salt '*' status.diskstats
```

salt.modules.status.**diskusage**(**args*)

Return the disk usage for this minion

Usage:

```
salt '*' status.diskusage [paths and/or filesystem types]
```

CLI Example:

```
salt '*' status.diskusage         # usage for all filesystems
salt '*' status.diskusage / /tmp  # usage for / and /tmp
salt '*' status.diskusage ext?    # usage for ext[234] filesystems
salt '*' status.diskusage / ext?  # usage for / and all ext filesystems
```

salt.modules.status.**loadavg**()

Return the load averages for this minion

CLI Example:

```
salt '*' status.loadavg
```

salt.modules.status.**master**(*master=None*, *connected=True*)

New in version 2014.7.0.

Fire an event if the minion gets disconnected from its master. This function is meant to be run via a scheduled job from the minion. If master_ip is an FQDN/Hostname, is must be resolvable to a valid IPv4 address.

CLI Example:

```
salt '*' status.master
```

salt.modules.status.**meminfo**()

Return the memory info for this minion

CLI Example:

```
salt '*' status.meminfo
```

salt.modules.status.**netdev**()

Return the network device stats for this minion

CLI Example:

```
salt '*' status.netdev
```

salt.modules.status.**netstats**()

Return the network stats for this minion

CLI Example:

```
salt '*' status.netstats
```

salt.modules.status.**nproc**()

Return the number of processing units available on this system

CLI Example:

```
salt '*' status.nproc
```

salt.modules.status.**pid**(*sig*)

Return the PID or an empty string if the process is running or not. Pass a signature to use to find the process via ps. Note you can pass a Python-compatible regular expression to return all pids of processes matching the regexp.

CLI Example:

```
salt '*' status.pid <sig>
```

salt.modules.status.**procs**()

Return the process data

CLI Example:

```
salt '*' status.procs
```

salt.modules.status.**uptime**()

Return the uptime for this minion

CLI Example:

```
salt '*' status.uptime
```

salt.modules.status.**version**()

Return the system version for this minion

CLI Example:

```
salt '*' status.version
```

salt.modules.status.**vmstats**()

Return the virtual memory stats for this minion

CLI Example:

```
salt '*' status.vmstats
```

salt.modules.status.**w**()
Return a list of logged in users for this minion, using the w command

CLI Example:

```
salt '*' status.w
```

22.16.230 salt.modules.sudo

Allow for the calling of execution modules via sudo

salt.modules.sudo.**salt_call**(*runas, fun, *args, **kwargs*)
Wrap a shell execution out to salt call with sudo

CLI Example:

```
salt '*' sudo.salt_call root test.ping
```

22.16.231 salt.modules.supervisord

Provide the service module for system supervisord or supervisord in a virtualenv

salt.modules.supervisord.**add**(*name, user=None, conf_file=None, bin_env=None*)
Activates any updates in config for process/group.

user user to run supervisorctl as

conf_file path to supervisord config file

bin_env path to supervisorctl bin or path to virtualenv with supervisor installed

CLI Example:

```
salt '*' supervisord.add <name>
```

salt.modules.supervisord.**custom**(*command, user=None, conf_file=None, bin_env=None*)
Run any custom supervisord command

user user to run supervisorctl as

conf_file path to supervisord config file

bin_env path to supervisorctl bin or path to virtualenv with supervisor installed

CLI Example:

```
salt '*' supervisord.custom "mstop '*gunicorn*'"
```

salt.modules.supervisord.**options**(*name, conf_file=None*)
New in version 2014.1.0.

Read the config file and return the config options for a given process

name Name of the configured process

conf_file path to supervisord config file

CLI Example:

```
salt '*' supervisord.options foo
```

salt.modules.supervisord.**remove**(*name*, *user=None*, *conf_file=None*, *bin_env=None*)
Removes process/group from active config

user user to run supervisorctl as

conf_file path to supervisord config file

bin_env path to supervisorctl bin or path to virtualenv with supervisor installed

CLI Example:

```
salt '*' supervisord.remove <name>
```

salt.modules.supervisord.**reread**(*user=None*, *conf_file=None*, *bin_env=None*)
Reload the daemon's configuration files

user user to run supervisorctl as

conf_file path to supervisord config file

bin_env path to supervisorctl bin or path to virtualenv with supervisor installed

CLI Example:

```
salt '*' supervisord.reread
```

salt.modules.supervisord.**restart**(*name='all'*, *user=None*, *conf_file=None*, *bin_env=None*)
Restart the named service. Process group names should not include a trailing asterisk.

user user to run supervisorctl as

conf_file path to supervisord config file

bin_env path to supervisorctl bin or path to virtualenv with supervisor installed

CLI Example:

```
salt '*' supervisord.restart <service>
salt '*' supervisord.restart <group>:
```

salt.modules.supervisord.**start**(*name='all'*, *user=None*, *conf_file=None*, *bin_env=None*)
Start the named service. Process group names should not include a trailing asterisk.

user user to run supervisorctl as

conf_file path to supervisord config file

bin_env path to supervisorctl bin or path to virtualenv with supervisor installed

CLI Example:

```
salt '*' supervisord.start <service>
salt '*' supervisord.start <group>:
```

salt.modules.supervisord.**status**(*name=None*, *user=None*, *conf_file=None*, *bin_env=None*)
List programs and its state

user user to run supervisorctl as

conf_file path to supervisord config file

bin_env path to supervisorctl bin or path to virtualenv with supervisor installed

CLI Example:

```
    salt '*' supervisord.status
```

salt.modules.supervisord.**status_raw**(*name=None*, *user=None*, *conf_file=None*, *bin_env=None*)
Display the raw output of status

user user to run supervisorctl as

conf_file path to supervisord config file

bin_env path to supervisorctl bin or path to virtualenv with supervisor installed

CLI Example:

```
    salt '*' supervisord.status_raw
```

salt.modules.supervisord.**stop**(*name='all'*, *user=None*, *conf_file=None*, *bin_env=None*)
Stop the named service. Process group names should not include a trailing asterisk.

user user to run supervisorctl as

conf_file path to supervisord config file

bin_env path to supervisorctl bin or path to virtualenv with supervisor installed

CLI Example:

```
    salt '*' supervisord.stop <service>
    salt '*' supervisord.stop <group>:
```

salt.modules.supervisord.**update**(*user=None*, *conf_file=None*, *bin_env=None*)
Reload config and add/remove as necessary

user user to run supervisorctl as

conf_file path to supervisord config file

bin_env path to supervisorctl bin or path to virtualenv with supervisor installed

CLI Example:

```
    salt '*' supervisord.update
```

22.16.232 salt.modules.svn

Subversion SCM

salt.modules.svn.**add**(*cwd*, *targets*, *user=None*, *username=None*, *password=None*, **opts*)
Add files to be tracked by the Subversion working-copy checkout

cwd The path to the Subversion repository

targets [None] files and directories to pass to the command as arguments

user [None] Run svn as a user other than what the minion runs as

username [None] Connect to the Subversion server as another user

password [None] Connect to the Subversion server with this password

New in version 0.17.0.

CLI Example:

```
salt '*' svn.add /path/to/repo /path/to/new/file
```

salt.modules.svn.**checkout**(*cwd*, *remote*, *target=None*, *user=None*, *username=None*, *password=None*, **opts*)

Download a working copy of the remote Subversion repository directory or file

cwd The path to the Subversion repository

remote [None] URL to checkout

target [None] The name to give the file or directory working copy Default: svn uses the remote basename

user [None] Run svn as a user other than what the minion runs as

username [None] Connect to the Subversion server as another user

password [None] Connect to the Subversion server with this password

New in version 0.17.0.

CLI Example:

```
salt '*' svn.checkout /path/to/repo svn://remote/repo
```

salt.modules.svn.**commit**(*cwd*, *targets=None*, *msg=None*, *user=None*, *username=None*, *password=None*, **opts*)

Commit the current directory, files, or directories to the remote Subversion repository

cwd The path to the Subversion repository

targets [None] files and directories to pass to the command as arguments Default: svn uses `.'

msg [None] Message to attach to the commit log

user [None] Run svn as a user other than what the minion runs as

username [None] Connect to the Subversion server as another user

password [None] Connect to the Subversion server with this password

New in version 0.17.0.

CLI Example:

```
salt '*' svn.commit /path/to/repo
```

salt.modules.svn.**diff**(*cwd*, *targets=None*, *user=None*, *username=None*, *password=None*, **opts*)

Return the diff of the current directory, files, or directories from the remote Subversion repository

cwd The path to the Subversion repository

targets [None] files and directories to pass to the command as arguments Default: svn uses `.'

user [None] Run svn as a user other than what the minion runs as

username [None] Connect to the Subversion server as another user

password [None] Connect to the Subversion server with this password

New in version 0.17.0.

CLI Example:

```
salt '*' svn.diff /path/to/repo
```

salt.modules.svn.**export**(*cwd*, *remote*, *target=None*, *user=None*, *username=None*, *password=None*, *revision='HEAD'*, **opts*)

Create an unversioned copy of a tree.

cwd The path to the Subversion repository

remote [None] URL and path to file or directory checkout

target [None] The name to give the file or directory working copy Default: svn uses the remote basename

user [None] Run svn as a user other than what the minion runs as

username [None] Connect to the Subversion server as another user

password [None] Connect to the Subversion server with this password

New in version 0.17.0.

CLI Example:

```
salt '*' svn.export /path/to/repo svn://remote/repo
```

salt.modules.svn.**info**(*cwd*, *targets=None*, *user=None*, *username=None*, *password=None*, *fmt='str'*)

Display the Subversion information from the checkout.

cwd The path to the Subversion repository

targets [None] files, directories, and URLs to pass to the command as arguments svn uses `.' by default

user [None] Run svn as a user other than what the minion runs as

username [None] Connect to the Subversion server as another user

password [None] Connect to the Subversion server with this password

New in version 0.17.0.

fmt [str] How to fmt the output from info. (str, xml, list, dict)

CLI Example:

```
salt '*' svn.info /path/to/svn/repo
```

salt.modules.svn.**remove**(*cwd*, *targets*, *msg=None*, *user=None*, *username=None*, *password=None*, **opts*)

Remove files and directories from the Subversion repository

cwd The path to the Subversion repository

targets [None] files, directories, and URLs to pass to the command as arguments

msg [None] Message to attach to the commit log

user [None] Run svn as a user other than what the minion runs as

username [None] Connect to the Subversion server as another user

password [None] Connect to the Subversion server with this password

New in version 0.17.0.

CLI Example:

```
salt '*' svn.remove /path/to/repo /path/to/repo/remove
```

salt.modules.svn.**status**(*cwd*, *targets=None*, *user=None*, *username=None*, *password=None*, **opts*)

Display the status of the current directory, files, or directories in the Subversion repository

cwd The path to the Subversion repository

targets [None] files, directories, and URLs to pass to the command as arguments Default: svn uses `.'

user [None] Run svn as a user other than what the minion runs as

username [None] Connect to the Subversion server as another user

password [None] Connect to the Subversion server with this password

New in version 0.17.0.

CLI Example:

```
salt '*' svn.status /path/to/repo
```

salt.modules.svn.**switch**(*cwd*, *remote*, *target=None*, *user=None*, *username=None*, *password=None*, **opts*)

New in version 2014.1.0.

Switch a working copy of a remote Subversion repository directory

cwd The path to the Subversion repository

remote [None] URL to switch

target [None] The name to give the file or directory working copy Default: svn uses the remote basename

user [None] Run svn as a user other than what the minion runs as

username [None] Connect to the Subversion server as another user

password [None] Connect to the Subversion server with this password

CLI Example:

```
salt '*' svn.switch /path/to/repo svn://remote/repo
```

salt.modules.svn.**update**(*cwd*, *targets=None*, *user=None*, *username=None*, *password=None*, **opts*)

Update the current directory, files, or directories from the remote Subversion repository

cwd The path to the Subversion repository

targets [None] files and directories to pass to the command as arguments Default: svn uses `.'

user [None] Run svn as a user other than what the minion runs as

password [None] Connect to the Subversion server with this password

New in version 0.17.0.

username [None] Connect to the Subversion server as another user

CLI Example:

```
salt '*' svn.update /path/to/repo
```

22.16.233 salt.modules.swift

Module for handling OpenStack Swift calls Author: Anthony Stanton <anthony.stanton@gmail.com>

Inspired by the S3 and Nova modules

depends

- swiftclient Python module

configuration This module is not usable until the user, password, tenant, and auth URL are specified either in a pillar or in the minion's config file. For example:

```
keystone.user: admin
keystone.password: verybadpass
keystone.tenant: admin
keystone.auth_url: 'http://127.0.0.1:5000/v2.0/'
```

If configuration for multiple OpenStack accounts is required, they can be set up as different configuration profiles: For example:

```
openstack1:
  keystone.user: admin
  keystone.password: verybadpass
  keystone.tenant: admin
  keystone.auth_url: 'http://127.0.0.1:5000/v2.0/'

openstack2:
  keystone.user: admin
  keystone.password: verybadpass
  keystone.tenant: admin
  keystone.auth_url: 'http://127.0.0.2:5000/v2.0/'
```

With this configuration in place, any of the swift functions can make use of a configuration profile by declaring it explicitly. For example:

```
salt '*' swift.get mycontainer myfile /tmp/file profile=openstack1
```

salt.modules.swift.**delete**(*cont*, *path=None*, *profile=None*)

Delete a container, or delete an object from a container.

CLI Example to delete a container:

```
salt myminion swift.delete mycontainer
```

CLI Example to delete an object from a container:

```
salt myminion swift.delete mycontainer remoteobject
```

salt.modules.swift.**get**(*cont=None*, *path=None*, *local_file=None*, *return_bin=False*, *profile=None*)

List the contents of a container, or return an object from a container. Set return_bin to True in order to retrieve an object wholesale. Otherwise, Salt will attempt to parse an XML response.

CLI Example to list containers:

```
salt myminion swift.get
```

CLI Example to list the contents of a container:

```
salt myminion swift.get mycontainer
```

CLI Example to return the binary contents of an object:

```
salt myminion swift.get mycontainer myfile.png return_bin=True
```

CLI Example to save the binary contents of an object to a local file:

```
salt myminion swift.get mycontainer myfile.png local_file=/tmp/myfile.png
```

salt.modules.swift.**head**()

salt.modules.swift.**put**(*cont*, *path=None*, *local_file=None*, *profile=None*)
Create a new container, or upload an object to a container.

CLI Example to create a container:

```
salt myminion swift.put mycontainer
```

CLI Example to upload an object to a container:

```
salt myminion swift.put mycontainer remotepath local_file=/path/to/file
```

22.16.234 salt.modules.sysbench

The `sysbench' module is used to analyze the performance of the minions, right from the master! It measures various system parameters such as CPU, Memory, File I/O, Threads and Mutex.

salt.modules.sysbench.**cpu**()
Tests for the CPU performance of minions.

CLI Examples:

```
salt '*' sysbench.cpu
```

salt.modules.sysbench.**fileio**()
This tests for the file read and write operations Various modes of operations are

- sequential write
- sequential rewrite
- sequential read
- random read
- random write
- random read and write

The test works with 32 files with each file being 1Gb in size The test consumes a lot of time. Be patient!

CLI Examples:

```
salt '*' sysbench.fileio
```

salt.modules.sysbench.**memory**()
This tests the memory for read and write operations.

CLI Examples:

```
salt '*' sysbench.memory
```

salt.modules.sysbench.**mutex**()
Tests the implementation of mutex

CLI Examples:

```
salt '*' sysbench.mutex
```

salt.modules.sysbench.**ping**()

salt.modules.sysbench.**threads**()
This tests the performance of the processor's scheduler

CLI Example:

```
salt '*' sysbench.threads
```

22.16.235 salt.modules.syslog_ng

Module for getting information about syslog-ng

maintainer Tibor Benke <btibi@sch.bme.hu>

maturity new

depends cmd

platform all

This module is capable of managing syslog-ng instances which were installed via a package manager or from source. Users can use a directory as a parameter in the case of most functions, which contains the syslog-ng and syslog-ng-ctl binaries.

Syslog-ng can be installed via a package manager or from source. In the latter case, the syslog-ng and syslog-ng-ctl binaries are not available from the PATH, so users should set location of the sbin directory with `syslog_ng.set_binary_path`.

Similarly, users can specify the location of the configuration file with `syslog_ng.set_config_file`, then the module will use it. If it is not set, syslog-ng uses the default configuration file.

class salt.modules.syslog_ng.**Argument**(*value=''*)
A TypedParameterValue has one or more Arguments. For example this can be the value of key_file.

Does not need examples.

build()

class salt.modules.syslog_ng.**Buildable**(*iterable*, *join_body_on=''*, *append_extra_newline=True*)
Base class of most classes, which have a build method.

It contains a common build function.

Does not need examples.

build()
Builds the textual representation of the whole configuration object with it's children.

build_body()
Builds the body of a syslog-ng configuration object.

build_header()
Builds the header of a syslog-ng configuration object.

build_tail()
Builds the tail of a syslog-ng configuration object.

class salt.modules.syslog_ng.**GivenStatement**(*value*, *add_newline=True*)

This statement returns a string without modification. It can be used to use existing configuration snippets.

Does not need examples.

build()

class salt.modules.syslog_ng.**NamedStatement**(*type*, *id=''*, *options=None*)
It represents a configuration statement, which has a name, e.g. a source.

Does not need examples.

class salt.modules.syslog_ng.**Option**(*type=''*, *params=None*)
A Statement class contains Option instances.

An instance of Option can represent a file(), tcp(), udp(), etc. option.

Does not need examples.

add_parameter(*param*)

build()

class salt.modules.syslog_ng.**Parameter**(*iterable=None*, *join_body_on=''*)
An Option has one or more Parameter instances.

Does not need examples.

class salt.modules.syslog_ng.**ParameterValue**(*iterable=None*, *join_body_on=''*)
A TypedParameter can have one or more values.

Does not need examples.

class salt.modules.syslog_ng.**SimpleParameter**(*value=''*)
A Parameter is a SimpleParameter, if it's just a simple type, like a string.

For example:

```
destination d_file {
    file(
        "/var/log/messages"
    );
};
```

`/var/log/messages` is a SimpleParameter.

Does not need examples.

build()

class salt.modules.syslog_ng.**SimpleParameterValue**(*value=''*)
A ParameterValuem which holds a simple type, like a string or a number.

For example in ip(127.0.0.1) 127.0.0.1 is a SimpleParameterValue.

Does not need examples.

build()

class salt.modules.syslog_ng.**Statement**(*type*, *id=''*, *options=None*, *has_name=True*)
It represents a syslog-ng configuration statement, e.g. source, destination, filter.

Does not need examples.

add_child(*option*)

build_header()

build_tail()

exception salt.modules.syslog_ng.**SyslogNgError**

class salt.modules.syslog_ng.**TypedParameter**(*type=''*, *values=None*)
A Parameter, which has a type:

```
    destination d_tcp {
        tcp(
            ip(127.0.0.1)
        );
    };
```

`ip(127.0.0.1)` is a TypedParameter.

Does not need examples.

add_value(*value*)

build()

class salt.modules.syslog_ng.**TypedParameterValue**(*type='`*, *arguments=None*)
We have to go deeper...

A TypedParameter can have a `parameter', which also have a type. For example key_file and cert_file:

```
    source demo_tls_source {
        tcp(
            ip(0.0.0.0)
            port(1999)
            tls(
                key_file("/opt/syslog-ng/etc/syslog-ng/key.d/syslog-ng.key")
                cert_file("/opt/syslog-ng/etc/syslog-ng/cert.d/syslog-ng.cert")
            )
        );
    };
```

Does not need examples.

add_argument(*arg*)

build()

class salt.modules.syslog_ng.**UnnamedStatement**(*type*, *options=None*)
It represents a configuration statement, which doesn't have a name, e.g. a log path.

Does not need examples.

salt.modules.syslog_ng.**config**(*name*, *config*, *write=True*)
Builds syslog-ng configuration. This function is intended to be used from the state module, users should not use it directly!

name : the id of the Salt document or it is the format of <statement name>.id config : the parsed YAML code write : if True, it writes the config into the configuration file, otherwise just returns it

CLI Example:

```
    salt '*' syslog_ng.config name="s_local" config="[{'tcp':[{'ip':'127.0.0.1'},{'port':1233}]}]"
```

salt.modules.syslog_ng.**config_test**(*syslog_ng_sbin_dir=None*, *cfgfile=None*)
Runs syntax check against cfgfile. If syslog_ng_sbin_dir is specified, it is added to the PATH during the test.

CLI Example:

```
    salt '*' syslog_ng.config_test
    salt '*' syslog_ng.config_test /home/user/install/syslog-ng/sbin
    salt '*' syslog_ng.config_test /home/user/install/syslog-ng/sbin /etc/syslog-ng/syslog-ng.conf
```

salt.modules.syslog_ng.**get_config_file**()
Returns the configuration directory, which contains syslog-ng.conf.

CLI Example:

```
salt '*' syslog_ng.get_config_file
```

salt.modules.syslog_ng.**modules**(*syslog_ng_sbin_dir=None*)

Returns the available modules. If syslog_ng_sbin_dir is specified, it is added to the PATH during the execution of the command syslog-ng.

CLI Example:

```
salt '*' syslog_ng.modules
salt '*' syslog_ng.modules /home/user/install/syslog-ng/sbin
```

salt.modules.syslog_ng.**reload_**(*name*)

Reloads syslog-ng. This function is intended to be used from states.

If `syslog_ng.set_config_file`, is called before, this function will use the set binary path.

CLI Example:

```
salt '*' syslog_ng.reload
```

salt.modules.syslog_ng.**set_binary_path**(*name*)

Sets the path, where the syslog-ng binary can be found. This function is intended to be used from states.

If syslog-ng is installed via a package manager, users don't need to use this function.

CLI Example:

```
salt '*' syslog_ng.set_binary_path name="/usr/sbin"
```

salt.modules.syslog_ng.**set_config_file**(*name*)

Sets the configuration's name. This function is intended to be used from states.

CLI Example:

```
salt '*' syslog_ng.set_config_file name="/etc/syslog-ng"
```

salt.modules.syslog_ng.**set_parameters**(*version=None*, *binary_path=None*, *config_file=None*, **args*, ***kwargs*)

Sets variables.

CLI Example:

```
salt '*' syslog_ng.set_parameters version="3.6"
salt '*' syslog_ng.set_parameters  binary_path="/home/user/install/syslog-ng/sbin" config_file="
```

salt.modules.syslog_ng.**start**(*name=None*, *user=None*, *group=None*, *chroot=None*, *caps=None*, *no_caps=False*, *pidfile=None*, *enable_core=False*, *fd_limit=None*, *verbose=False*, *debug=False*, *trace=False*, *yydebug=False*, *persist_file=None*, *control=None*, *worker_threads=None*)

Ensures, that syslog-ng is started via the given parameters. This function is intended to be used from the state module.

Users shouldn't use this function, if the service module is available on their system. If `syslog_ng.set_config_file`, is called before, this function will use the set binary path.

CLI Example:

```
salt '*' syslog_ng.start
```

salt.modules.syslog_ng.**stats**(*syslog_ng_sbin_dir=None*)

Returns statistics from the running syslog-ng instance. If syslog_ng_sbin_dir is specified, it is added to the PATH during the execution of the command syslog-ng-ctl.

CLI Example:

```
salt '*' syslog_ng.stats
salt '*' syslog_ng.stats /home/user/install/syslog-ng/sbin
```

salt.modules.syslog_ng.**stop**(*name=None*)

Kills syslog-ng. This function is intended to be used from the state module.

Users shouldn't use this function, if the service module is available on their system. If `syslog_ng.set_config_file` is called before, this function will use the set binary path.

CLI Example:

```
salt '*' syslog_ng.stop
```

salt.modules.syslog_ng.**version**(*syslog_ng_sbin_dir=None*)

Returns the version of the installed syslog-ng. If syslog_ng_sbin_dir is specified, it is added to the PATH during the execution of the command syslog-ng.

CLI Example:

```
salt '*' syslog_ng.version
salt '*' syslog_ng.version /home/user/install/syslog-ng/sbin
```

salt.modules.syslog_ng.**write_config**(*config*, *newlines=2*)

Writes the given parameter config into the config file. This function is intended to be used from states.

If `syslog_ng.set_config_file`, is called before, this function will use the set config file.

CLI Example:

```
salt '*' syslog_ng.write_config config="# comment"
```

salt.modules.syslog_ng.**write_version**(*name*)

Removes the previous configuration file, then creates a new one and writes the name line. This function is intended to be used from states.

If `syslog_ng.set_config_file`, is called before, this function will use the set config file.

CLI Example:

```
salt '*' syslog_ng.write_version name="3.6"
```

22.16.236 salt.modules.sysmod

The sys module provides information about the available functions on the minion

salt.modules.sysmod.**argspec**(*module=''*)

Return the argument specification of functions in Salt execution modules.

CLI Example:

```
salt '*' sys.argspec pkg.install
salt '*' sys.argspec sys
salt '*' sys.argspec
```

Module names can be specified as globs.

New in version 2015.5.0.

```
salt '*' sys.argspec 'pkg.*'
```

salt.modules.sysmod.**doc**(**args*)

Return the docstrings for all modules. Optionally, specify a module or a function to narrow the selection.

The strings are aggregated into a single document on the master for easy reading.

Multiple modules/functions can be specified.

CLI Example:

```
salt '*' sys.doc
salt '*' sys.doc sys
salt '*' sys.doc sys.doc
salt '*' sys.doc network.traceroute user.info
```

Modules can be specified as globs.

New in version 2015.5.0.

```
salt '*' sys.doc 'sys.*'
salt '*' sys.doc 'sys.list_*'
```

salt.modules.sysmod.**list_functions**(**args*, ***kwargs*)

List the functions for all modules. Optionally, specify a module or modules from which to list.

CLI Example:

```
salt '*' sys.list_functions
salt '*' sys.list_functions sys
salt '*' sys.list_functions sys user
```

Function names can be specified as globs.

New in version 2015.5.0.

```
salt '*' sys.list_functions 'sys.list_*'
```

salt.modules.sysmod.**list_modules**(**args*)

List the modules loaded on the minion

New in version 2015.5.0.

CLI Example:

```
salt '*' sys.list_modules
```

Module names can be specified as globs.

```
salt '*' sys.list_modules 's*'
```

salt.modules.sysmod.**list_renderers**(**args*)

List the renderers loaded on the minion

New in version 2015.5.0.

CLI Example:

```
salt '*' sys.list_renderers
```

Render names can be specified as globs.

```
salt '*' sys.list_renderers 'yaml*'
```

salt.modules.sysmod.**list_returner_functions**(**args*, ***kwargs*)

List the functions for all returner modules. Optionally, specify a returner module or modules from which to list.

New in version 2014.7.0.

CLI Example:

```
salt '*' sys.list_returner_functions
salt '*' sys.list_returner_functions mysql
salt '*' sys.list_returner_functions mysql etcd
```

Returner names can be specified as globs.

New in version 2015.5.0.

```
salt '*' sys.list_returner_functions 'sqlite3.get_*'
```

salt.modules.sysmod.**list_returners**(**args*)

List the runners loaded on the minion

New in version 2014.7.0.

CLI Example:

```
salt '*' sys.list_returners
```

Returner names can be specified as globs.

New in version 2015.5.0.

```
salt '*' sys.list_returners 's*'
```

salt.modules.sysmod.**list_runner_functions**(**args*, ***kwargs*)

List the functions for all runner modules. Optionally, specify a runner module or modules from which to list.

New in version 2014.7.0.

CLI Example:

```
salt '*' sys.list_runner_functions
salt '*' sys.list_runner_functions state
salt '*' sys.list_runner_functions state virt
```

Runner function names can be specified as globs.

New in version 2015.5.0.

```
salt '*' sys.list_runner_functions 'state.*' 'virt.*'
```

salt.modules.sysmod.**list_runners**(**args*)

List the runners loaded on the minion

New in version 2014.7.0.

CLI Example:

```
salt '*' sys.list_runners
```

Runner names can be specified as globs.

New in version 2015.5.0.

```
salt '*' sys.list_runners 'm*'
```

salt.modules.sysmod.**list_state_functions**(**args*, ***kwargs*)

List the functions for all state modules. Optionally, specify a state module or modules from which to list.

New in version 2014.7.0.

CLI Example:

```
salt '*' sys.list_state_functions
salt '*' sys.list_state_functions file
salt '*' sys.list_state_functions pkg user
```

State function names can be specified as globs.

New in version 2015.5.0.

```
salt '*' sys.list_state_functions 'file.*'
salt '*' sys.list_state_functions 'file.s*'
```

salt.modules.sysmod.**list_state_modules**(**args*)

List the modules loaded on the minion

New in version 2014.7.0.

CLI Example:

```
salt '*' sys.list_state_modules
```

State module names can be specified as globs.

New in version 2015.5.0.

```
salt '*' sys.list_state_modules 'mysql_*'
```

salt.modules.sysmod.**reload_modules**()

Tell the minion to reload the execution modules

CLI Example:

```
salt '*' sys.reload_modules
```

salt.modules.sysmod.**renderer_doc**(**args*)

Return the docstrings for all renderers. Optionally, specify a renderer or a function to narrow the selection.

The strings are aggregated into a single document on the master for easy reading.

Multiple renderers can be specified.

New in version 2015.5.0.

CLI Example:

```
salt '*' sys.renderer_doc
salt '*' sys.renderer_doc cheetah
salt '*' sys.renderer_doc jinja json
```

Renderer names can be specified as globs.

```
    salt '*' sys.renderer_doc 'c*' 'j*'
```

salt.modules.sysmod.**returner_argspec**(*module=''*)

Return the argument specification of functions in Salt returner modules.

New in version 2015.5.0.

CLI Example:

```
    salt '*' sys.returner_argspec xmpp
    salt '*' sys.returner_argspec xmpp smtp
    salt '*' sys.returner_argspec
```

Returner names can be specified as globs.

```
    salt '*' sys.returner_argspec 'sqlite3.*'
```

salt.modules.sysmod.**returner_doc**(**args*)

Return the docstrings for all returners. Optionally, specify a returner or a function to narrow the selection.

The strings are aggregated into a single document on the master for easy reading.

Multiple returners/functions can be specified.

New in version 2014.7.0.

CLI Example:

```
    salt '*' sys.returner_doc
    salt '*' sys.returner_doc sqlite3
    salt '*' sys.returner_doc sqlite3.get_fun
    salt '*' sys.returner_doc sqlite3.get_fun etcd.get_fun
```

Returner names can be specified as globs.

New in version 2015.5.0.

```
    salt '*' sys.returner_doc 'sqlite3.get_*'
```

salt.modules.sysmod.**runner_argspec**(*module=''*)

Return the argument specification of functions in Salt runner modules.

New in version 2015.5.0.

CLI Example:

```
    salt '*' sys.runner_argspec state
    salt '*' sys.runner_argspec http
    salt '*' sys.runner_argspec
```

Runner names can be specified as globs.

```
    salt '*' sys.runner_argspec 'winrepo.*'
```

salt.modules.sysmod.**runner_doc**(**args*)

Return the docstrings for all runners. Optionally, specify a runner or a function to narrow the selection.

The strings are aggregated into a single document on the master for easy reading.

Multiple runners/functions can be specified.

New in version 2014.7.0.

CLI Example:

```
salt '*' sys.runner_doc
salt '*' sys.runner_doc cache
salt '*' sys.runner_doc cache.grains
salt '*' sys.runner_doc cache.grains mine.get
```

Runner names can be specified as globs.

New in version 2015.5.0.

```
salt '*' sys.runner_doc 'cache.clear_*'
```

salt.modules.sysmod.**state_argspec**(*module=''*)

Return the argument specification of functions in Salt state modules.

New in version 2015.5.0.

CLI Example:

```
salt '*' sys.state_argspec pkg.installed
salt '*' sys.state_argspec file
salt '*' sys.state_argspec
```

State names can be specified as globs.

```
salt '*' sys.state_argspec 'pkg.*'
```

salt.modules.sysmod.**state_doc**(**args*)

Return the docstrings for all states. Optionally, specify a state or a function to narrow the selection.

The strings are aggregated into a single document on the master for easy reading.

Multiple states/functions can be specified.

New in version 2014.7.0.

CLI Example:

```
salt '*' sys.state_doc
salt '*' sys.state_doc service
salt '*' sys.state_doc service.running
salt '*' sys.state_doc service.running ipables.append
```

State names can be specified as globs.

New in version 2015.5.0.

```
salt '*' sys.state_doc 'service.*' 'iptables.*'
```

22.16.237 salt.modules.sysrc

sysrc module for FreeBSD

salt.modules.sysrc.**get**(***kwargs*)

Return system rc configuration variables

CLI Example:

```
    salt '*' sysrc.get includeDefaults=True
```

salt.modules.sysrc.**remove**(*name*, ***kwargs*)
Remove system rc configuration variables

CLI Example:

```
salt '*' sysrc.remove name=sshd_enable
```

salt.modules.sysrc.**set_**(*name*, *value*, ***kwargs*)
Set system rc configuration variables

CLI Example:

```
salt '*' sysrc.remove name=sshd_enable
```

22.16.238 salt.modules.system

Support for reboot, shutdown, etc

salt.modules.system.**halt**()
Halt a running system

CLI Example:

```
salt '*' system.halt
```

salt.modules.system.**init**(*runlevel*)
Change the system runlevel on sysV compatible systems

CLI Example:

```
salt '*' system.init 3
```

salt.modules.system.**poweroff**()
Poweroff a running system

CLI Example:

```
salt '*' system.poweroff
```

salt.modules.system.**reboot**()
Reboot the system using the `reboot' command

CLI Example:

```
salt '*' system.reboot
```

salt.modules.system.**shutdown**(*at_time=None*)
Shutdown a running system

CLI Example:

```
salt '*' system.shutdown
```

22.16.239 salt.modules.systemd

Provide the service module for systemd

salt.modules.systemd.**available**(*name*)
Check that the given service is available taking into account template units.

CLI Example:

```
salt '*' service.available sshd
```

salt.modules.systemd.**disable**(*name*, ***kwargs*)
Disable the named service to not start when the system boots

CLI Example:

```
salt '*' service.disable <service name>
```

salt.modules.systemd.**disabled**(*name*)
Return if the named service is disabled to start on boot

CLI Example:

```
salt '*' service.disabled <service name>
```

salt.modules.systemd.**enable**(*name*, ***kwargs*)
Enable the named service to start when the system boots

CLI Example:

```
salt '*' service.enable <service name>
```

salt.modules.systemd.**enabled**(*name*, ***kwargs*)
Return if the named service is enabled to start on boot

CLI Example:

```
salt '*' service.enabled <service name>
```

salt.modules.systemd.**execs**()
Return a list of all files specified as `ExecStart` for all services.

CLI Example:

salt `*' service.execs

salt.modules.systemd.**force_reload**(*name*)
Force-reload the specified service with systemd

CLI Example:

```
salt '*' service.force_reload <service name>
```

salt.modules.systemd.**get_all**()
Return a list of all available services

CLI Example:

```
salt '*' service.get_all
```

salt.modules.systemd.**get_disabled**()
Return a list of all disabled services

CLI Example:

```
salt '*' service.get_disabled
```

salt.modules.systemd.**get_enabled**()
Return a list of all enabled services

CLI Example:

```
salt '*' service.get_enabled
```

salt.modules.systemd.**mask**(*name*)
Mask the specified service with systemd

CLI Example:

```
salt '*' service.mask <service name>
```

salt.modules.systemd.**missing**(*name*)
The inverse of service.available. Returns `True` if the specified service is not available, otherwise returns `False`.

CLI Example:

```
salt '*' service.missing sshd
```

salt.modules.systemd.**reload_**(*name*)
Reload the specified service with systemd

CLI Example:

```
salt '*' service.reload <service name>
```

salt.modules.systemd.**restart**(*name*)
Restart the specified service with systemd

CLI Example:

```
salt '*' service.restart <service name>
```

salt.modules.systemd.**show**(*name*)
Show properties of one or more units/jobs or the manager

CLI Example:

salt `*' service.show <service name>

salt.modules.systemd.**start**(*name*)
Start the specified service with systemd

CLI Example:

```
salt '*' service.start <service name>
```

salt.modules.systemd.**status**(*name*, *sig=None*)
Return the status for a service via systemd, returns a bool whether the service is running.

CLI Example:

```
salt '*' service.status <service name>
```

salt.modules.systemd.**stop**(*name*)
Stop the specified service with systemd

CLI Example:

```
salt '*' service.stop <service name>
```

salt.modules.systemd.**systemctl_reload**()
Reloads systemctl, an action needed whenever unit files are updated.

CLI Example:

```
salt '*' service.systemctl_reload
```

salt.modules.systemd.**unmask**(*name*)
Unmask the specified service with systemd

CLI Example:

```
salt '*' service.unmask <service name>
```

22.16.240 salt.modules.test

Module for running arbitrary tests

salt.modules.test.**arg**(**args*, ***kwargs*)
Print out the data passed into the function `*args` and `kwargs`, this is used to both test the publication data and cli argument passing, but also to display the information available within the publication data. Returns {``args'': args, ``kwargs'': kwargs}.

CLI Example:

```
salt '*' test.arg 1 "two" 3.1 txt="hello" wow='{a: 1, b: "hello"}'
```

salt.modules.test.**arg_repr**(**args*, ***kwargs*)
Print out the data passed into the function `*args` and `kwargs`, this is used to both test the publication data and cli argument passing, but also to display the information available within the publication data. Returns {``args'': repr(args), ``kwargs'': repr(kwargs)}.

CLI Example:

```
salt '*' test.arg_repr 1 "two" 3.1 txt="hello" wow='{a: 1, b: "hello"}'
```

salt.modules.test.**arg_type**(**args*, ***kwargs*)
Print out the types of the args and kwargs. This is used to test the types of the args and kwargs passed down to the minion

CLI Example:

```
salt '*' test.arg_type 1 'int'
```

salt.modules.test.**assertion**(*assertion*)
Assert the given argument

CLI Example:

```
salt '*' test.assert False
```

salt.modules.test.**attr_call**()
Call grains.items via the attribute

CLI Example:

```
.. code-block:: bash
```

salt `*' test.attr_call

salt.modules.test.**collatz**(*start*)

Execute the collatz conjecture from the passed starting number, returns the sequence and the time it took to compute. Used for performance tests.

CLI Example:

```
salt '*' test.collatz 3
```

salt.modules.test.**conf_test**()

Return the value for test.foo in the minion configuration file, or return the default value

CLI Example:

```
salt '*' test.conf_test
```

salt.modules.test.**cross_test**(*func*, *args=None*)

Execute a minion function via the __salt__ object in the test module, used to verify that the minion functions can be called via the __salt__ module.

CLI Example:

```
salt '*' test.cross_test file.gid_to_group 0
```

salt.modules.test.**echo**(*text*)

Return a string - used for testing the connection

CLI Example:

```
salt '*' test.echo 'foo bar baz quo qux'
```

salt.modules.test.**exception**(*message='Test Exception'*)

Raise an exception

Optionally provide an error message or output the full stack.

CLI Example:

```
salt '*' test.exception 'Oh noes!'
```

salt.modules.test.**fib**(*num*)

Return a Fibonacci sequence up to the passed number, and the timeit took to compute in seconds. Used for performance tests

CLI Example:

```
salt '*' test.fib 3
```

salt.modules.test.**get_opts**()

Return the configuration options passed to this minion

CLI Example:

```
salt '*' test.get_opts
```

salt.modules.test.**kwarg**(***kwargs*)

Print out the data passed into the function `**kwargs`, this is used to both test the publication data and cli kwarg passing, but also to display the information available within the publication data.

CLI Example:

```
salt '*' test.kwarg num=1 txt="two" env='{a: 1, b: "hello"}'
```

salt.modules.test.**missing_func**()

salt.modules.test.**module_report**()
Return a dict containing all of the exeution modules with a report on the overall availability via different references

CLI Example:

```
.. code-block:: bash
```

salt `*' test.module_report

salt.modules.test.**not_loaded**()
List the modules that were not loaded by the salt loader system

CLI Example:

```
salt '*' test.not_loaded
```

salt.modules.test.**opts_pkg**()
Return an opts package with the grains and opts for this minion. This is primarily used to create the options used for master side state compiling routines

CLI Example:

```
salt '*' test.opts_pkg
```

salt.modules.test.**outputter**(*data*)
Test the outputter, pass in data to return

CLI Example:

```
salt '*' test.outputter foobar
```

salt.modules.test.**ping**()
Used to make sure the minion is up and responding. Not an ICMP ping.

Returns `True`.

CLI Example:

```
salt '*' test.ping
```

salt.modules.test.**provider**(*module*)
Pass in a function name to discover what provider is being used

CLI Example:

```
salt '*' test.provider service
```

salt.modules.test.**providers**()
Return a dict of the provider names and the files that provided them

CLI Example:

```
salt '*' test.providers
```

salt.modules.test.**rand_sleep**(*max=60*)
Sleep for a random number of seconds, used to test long-running commands and minions returning at differing intervals

CLI Example:

```
salt '*' test.rand_sleep 60
```

salt.modules.test.**rand_str**(*size=9999999999*)
Return a random string

CLI Example:

```
salt '*' test.rand_str
```

salt.modules.test.**retcode**(*code=42*)
Test that the returncode system is functioning correctly

CLI Example:

```
salt '*' test.retcode 42
```

salt.modules.test.**sleep**(*length*)
Instruct the minion to initiate a process that will sleep for a given period of time.

CLI Example:

```
salt '*' test.sleep 20
```

salt.modules.test.**stack**()
Return the current stack trace

CLI Example:

```
salt '*' test.stack
```

salt.modules.test.**try_**(*module*, *return_try_exception=False*, ***kwargs*)
Try to run a module command. On an exception return None. If *return_try_exception* is set True return the exception. This can be helpfull in templates where running a module might fail as expected.

CLI Example:

```
<pre>
{% for i in range(0,230) %}
    {{ salt['test.try'](module='ipmi.get_users', bmc_host='172.2.2.'+i)|yaml(False) }}
{% endfor %}
</pre>
```

salt.modules.test.**tty**(**args*, ***kwargs*)
Deprecated! Moved to cmdmod.

CLI Example:

```
salt '*' test.tty tty0 'This is a test'
salt '*' test.tty pts3 'This is a test'
```

salt.modules.test.**version**()
Return the version of salt on the minion

CLI Example:

```
salt '*' test.version
```

salt.modules.test.**versions_information**()
Returns versions of components used by salt as a dict

CLI Example:

```
salt '*' test.versions_information
```

salt.modules.test.**versions_report**()
Returns versions of components used by salt

CLI Example:

```
salt '*' test.versions_report
```

22.16.241 salt.modules.test_virtual

Module for running arbitrary tests with a __virtual__ function

salt.modules.test_virtual.**ping**()

22.16.242 salt.modules.timezone

Module for managing timezone on POSIX-like systems.

salt.modules.timezone.**get_hwclock**()
Get current hardware clock setting (UTC or localtime)

CLI Example:

```
salt '*' timezone.get_hwclock
```

salt.modules.timezone.**get_offset**()
Get current numeric timezone offset from UCT (i.e. -0700)

CLI Example:

```
salt '*' timezone.get_offset
```

salt.modules.timezone.**get_zone**()
Get current timezone (i.e. America/Denver)

CLI Example:

```
salt '*' timezone.get_zone
```

salt.modules.timezone.**get_zonecode**()
Get current timezone (i.e. PST, MDT, etc)

CLI Example:

```
salt '*' timezone.get_zonecode
```

salt.modules.timezone.**set_hwclock**(*clock*)
Sets the hardware clock to be either UTC or localtime

CLI Example:

```
salt '*' timezone.set_hwclock UTC
```

salt.modules.timezone.**set_zone**(*timezone*)
Unlinks, then symlinks /etc/localtime to the set timezone.

The timezone is crucial to several system processes, each of which SHOULD be restarted (for instance, whatever you system uses as its cron and syslog daemons). This will not be magically done for you!

CLI Example:

```
salt '*' timezone.set_zone 'America/Denver'
```

salt.modules.timezone.**zone_compare**(*timezone*)

Checks the hash sum between the given timezone, and the one set in /etc/localtime. Returns True if they match, and False if not. Mostly useful for running state checks.

CLI Example:

```
salt '*' timezone.zone_compare 'America/Denver'
```

22.16.243 salt.modules.tls

A salt module for SSL/TLS. Can create a Certificate Authority (CA) or use Self-Signed certificates.

depends

- PyOpenSSL Python module

configuration Add the following values in `/etc/salt/minion` for the CA module to function properly:

```
ca.cert_base_path: '/etc/pki'
```

salt.modules.tls.**ca_exists**(*ca_name*, *cacert_path=None*)

Verify whether a Certificate Authority (CA) already exists

ca_name name of the CA

cacert_path absolute path to ca certificates root directory

CLI Example:

```
salt '*' tls.ca_exists test_ca /etc/certs
```

salt.modules.tls.**cert_base_path**(*cacert_path=None*)

Return the base path for certs from CLI or from options

CLI Example:

```
salt '*' tls.cert_base_path
```

salt.modules.tls.**cert_info**(*cert_path*, *digest='sha256'*)

Return information for a particular certificate

cert_path path to the cert file

digest what digest to use for fingerprinting

CLI Example:

```
salt '*' tls.cert_info /dir/for/certs/cert.pem
```

salt.modules.tls.**create_ca**(*ca_name*, *bits=2048*, *days=365*, *CN='localhost'*, *C='US'*, *ST='Utah'*, *L='Salt Lake City'*, *O='SaltStack'*, *OU=None*, *emailAddress='xyz@pdq.net'*, *fixmode=False*, *cacert_path=None*, *digest='sha256'*)

Create a Certificate Authority (CA)

ca_name name of the CA

bits number of RSA key bits, Default is `2048`

days number of days the CA will be valid, Default is `365`

CN common name in the request, Default is `localhost`

C country, Default is `US`

ST state, Default is `Utah`

L locality, Default is `Salt Lake City`

O organization, Default is `SaltStack`

OU organizational unit, Default is `None`

emailAddress email address for the CA owner, Default is `xyz@pdq.net`

cacert_path absolute path to ca certificates root directory

digest The message digest algorithm. Must be a string describing a digest algorithm supported by OpenSSL (by EVP_get_digestbyname, specifically). For example, ``md5'' or ``sha1''. Default: `sha256'

Writes out a CA certificate based upon defined config values. If the file already exists, the function just returns assuming the CA certificate already exists.

If the following values were set:

```
ca.cert_base_path='/etc/pki'
ca_name='koji'
```

the resulting CA, and corresponding key, would be written in the following location:

```
/etc/pki/koji/koji_ca_cert.crt
/etc/pki/koji/koji_ca_cert.key
```

CLI Example:

```
salt '*' tls.create_ca test_ca
```

salt.modules.tls.**create_ca_signed_cert**(*ca_name*, *CN*, *days=365*, *cacert_path=None*, *digest='sha256'*, ***extensions*)

Create a Certificate (CERT) signed by a named Certificate Authority (CA)

If the certificate file already exists, the function just returns assuming the CERT already exists.

The CN *must* match an existing CSR generated by create_csr. If it does not, this method does nothing.

ca_name name of the CA

CN common name matching the certificate signing request

days number of days certificate is valid, Default is `365` (1 year)

cacert_path absolute path to ca certificates root directory

digest The message digest algorithm. Must be a string describing a digest algorithm supported by OpenSSL (by EVP_get_digestbyname, specifically). For example, ``md5'' or ``sha1''. Default: `sha256'

****extensions** X509 V3 certificate extension

Writes out a Certificate (CERT). If the file already exists, the function just returns assuming the CERT already exists.

The CN *must* match an existing CSR generated by create_csr. If it does not, this method does nothing.

If the following values were set:

```
ca.cert_base_path='/etc/pki'
ca_name='koji'
CN='test.egavas.org'
```

the resulting signed certificate would be written in the following location:

```
/etc/pki/koji/certs/test.egavas.org.crt
```

CLI Example:

```
salt '*' tls.create_ca_signed_cert test localhost
```

salt.modules.tls.**create_csr**(*ca_name*, *bits=2048*, *CN='localhost'*, *C='US'*, *ST='Utah'*, *L='Salt Lake City'*, *O='SaltStack'*, *OU=None*, *emailAddress='xyz@pdq.net'*, *subjectAltName=None*, *cacert_path=None*, *digest='sha256'*)

Create a Certificate Signing Request (CSR) for a particular Certificate Authority (CA)

ca_name name of the CA

bits number of RSA key bits, Default is `2048`

CN common name in the request, Default is `localhost`

C country, Default is `US`

ST state, Default is `Utah`

L locality, Default is `Salt Lake City`

O organization. Must the same as CA certificate or an error will be raised, Default is `SaltStack`

OU organizational unit, Default is `None`

emailAddress email address for the request, Default is `xyz@pdq.net`

subjectAltName valid subjectAltNames in full form, e.g. to add DNS entry you would call this function with this value: **['DNS:myapp.foo.comm']**

cacert_path absolute path to ca certificates root directory

digest The message digest algorithm. Must be a string describing a digest algorithm supported by OpenSSL (by EVP_get_digestbyname, specifically). For example, ``md5'' or ``sha1''. Default: `sha256'

Writes out a Certificate Signing Request (CSR) If the file already exists, the function just returns assuming the CSR already exists.

If the following values were set:

```
ca.cert_base_path='/etc/pki'
ca_name='koji'
CN='test.egavas.org'
```

the resulting CSR, and corresponding key, would be written in the following location:

```
/etc/pki/koji/certs/test.egavas.org.csr
/etc/pki/koji/certs/test.egavas.org.key
```

CLI Example:

```
salt '*' tls.create_csr test
```

salt.modules.tls.**create_pkcs12**(*ca_name*, *CN*, *passphrase=''*, *cacert_path=None*)

Create a PKCS#12 browser certificate for a particular Certificate (CN)

ca_name name of the CA

CN common name matching the certificate signing request

passphrase used to unlock the PKCS#12 certificate when loaded into the browser

cacert_path absolute path to ca certificates root directory

If the following values were set:

```
ca.cert_base_path='/etc/pki'
ca_name='koji'
CN='test.egavas.org'
```

the resulting signed certificate would be written in the following location:

```
/etc/pki/koji/certs/test.egavas.org.p12
```

CLI Example:

```
salt '*' tls.create_pkcs12 test localhost
```

salt.modules.tls.**create_self_signed_cert**(*tls_dir='tls'*, *bits=2048*, *days=365*, *CN='localhost'*, *C='US'*, *ST='Utah'*, *L='Salt Lake City'*, *O='SaltStack'*, *OU=None*, *emailAddress='xyz@pdq.net'*, *cacert_path=None*, *digest='sha256'*)

Create a Self-Signed Certificate (CERT)

tls_dir location appended to the ca.cert_base_path, Default is `tls`

bits number of RSA key bits, Default is `2048`

days validity of certificate, Default is `365`

CN common name in the request, Default is `localhost`

C country, Default is `US`

ST state, Default is `Utah`

L locality, Default is `Salt Lake City`

O organization. Must the same as CA certificate or an error will be raised, Default is `SaltStack`

OU organizational unit, Default is `None`

emailAddress email address for the request, Default is `xyz@pdq.net`

cacert_path absolute path to ca certificates root directory

digest The message digest algorithm. Must be a string describing a digest algorithm supported by OpenSSL (by EVP_get_digestbyname, specifically). For example, ``md5'' or ``sha1''. Default: `sha256'

Writes out a Self-Signed Certificate (CERT). If the file already exists, the function just returns.

If the following values were set:

```
ca.cert_base_path='/etc/pki'
tls_dir='koji'
CN='test.egavas.org'
```

the resulting CERT, and corresponding key, would be written in the following location:

```
/etc/pki/koji/certs/test.egavas.org.crt
/etc/pki/koji/certs/test.egavas.org.key
```

CLI Examples:

```
salt '*' tls.create_self_signed_cert
salt 'minion' tls.create_self_signed_cert CN='test.mysite.org'
```

salt.modules.tls.**get_ca**(*ca_name*, *as_text=False*, *cacert_path=None*)
Get the certificate path or content

ca_name name of the CA

as_text if true, return the certificate content instead of the path

cacert_path absolute path to ca certificates root directory

CLI Example:

```
salt '*' tls.get_ca test_ca as_text=False cacert_path=/etc/certs
```

salt.modules.tls.**maybe_fix_ssl_version**(*ca_name*, *cacert_path=None*)
Check that the X509 version is correct (was incorrectly set in previous salt versions). This will fix the version if needed.

ca_name ca authority name

cacert_path absolute path to ca certificates root directory

CLI Example:

```
salt '*' tls.maybe_fix_ssl_version test_ca /etc/certs
```

salt.modules.tls.**set_ca_path**(*cacert_path*)
If wanted, store the aforementioned cacert_path in context to be used as the basepath for further operations

CLI Example:

```
salt '*' tls.set_ca_path /etc/certs
```

22.16.244 salt.modules.tomcat

Support for Tomcat

This module uses the manager webapp to manage Apache tomcat webapps. If the manager webapp is not configured some of the functions won't work.

Note: The config format was changed in 2014.7.0, but backwards compatibility for the old-style config will be in the 2014.7.1 release.

The following grains/pillar should be set:

```
tomcat-manager:
  user: <username>
  passwd: <password>
```

or the old format:

```
tomcat-manager.user: <username>
tomcat-manager.passwd: <password>
```

Also configure a user in the conf/tomcat-users.xml file:

```
<?xml version='1.0' encoding='utf-8'?>
<tomcat-users>
    <role rolename="manager-script"/>
    <user username="tomcat" password="tomcat" roles="manager-script"/>
</tomcat-users>
```

Note:

- More information about tomcat manager: http://tomcat.apache.org/tomcat-7.0-doc/manager-howto.html
- if you use only this module for deployments you've might want to strict access to the manager only from localhost for more info: http://tomcat.apache.org/tomcat-7.0-doc/manager-howto.html#Configuring_Manager_Application_Access
- Tested on:

  JVM Vendor: Sun Microsystems Inc.

  JVM Version: 1.6.0_43-b01

  OS Architecture: amd64

  OS Name: Linux

  OS Version: 2.6.32-358.el6.x86_64

  Tomcat Version: Apache Tomcat/7.0.37

`salt.modules.tomcat.`**`deploy_war`**(*war*, *context*, *force='no'*, *url='http://localhost:8080/manager'*, *saltenv='base'*, *timeout=180*, *env=None*, *temp_war_location=None*)

Deploy a WAR file

war absolute path to WAR file (should be accessible by the user running tomcat) or a path supported by the salt.modules.cp.get_file function

context the context path to deploy

force [False] set True to deploy the webapp even one is deployed in the context

url [http://localhost:8080/manager] the URL of the server manager webapp

saltenv [base] the environment for WAR file in used by salt.modules.cp.get_url function

timeout [180] timeout for HTTP request

temp_war_location [None] use another location to temporarily copy to war file by default the system's temp directory is used

CLI Examples:

cp module

```
salt '*' tomcat.deploy_war salt://application.war /api
salt '*' tomcat.deploy_war salt://application.war /api no
salt '*' tomcat.deploy_war salt://application.war /api yes http://localhost:8080/manager
```

minion local file system

```
salt '*' tomcat.deploy_war /tmp/application.war /api
salt '*' tomcat.deploy_war /tmp/application.war /api no
salt '*' tomcat.deploy_war /tmp/application.war /api yes http://localhost:8080/manager
```

salt.modules.tomcat.**fullversion**()
Return all server information from catalina.sh version

CLI Example:

```
salt '*' tomcat.fullversion
```

salt.modules.tomcat.**leaks**(*url='http://localhost:8080/manager'*, *timeout=180*)
Find memory leaks in tomcat

url [http://localhost:8080/manager] the URL of the server manager webapp

timeout [180] timeout for HTTP request

CLI Examples:

```
salt '*' tomcat.leaks
```

salt.modules.tomcat.**ls**(*url='http://localhost:8080/manager'*, *timeout=180*)
list all the deployed webapps

url [http://localhost:8080/manager] the URL of the server manager webapp

timeout [180] timeout for HTTP request

CLI Examples:

```
salt '*' tomcat.ls
salt '*' tomcat.ls http://localhost:8080/manager
```

salt.modules.tomcat.**passwd**(*passwd*, *user=''*, *alg='md5'*, *realm=None*)
This function replaces the $CATALINA_HOME/bin/digest.sh script convert a clear-text password to the $CATALINA_BASE/conf/tomcat-users.xml format

CLI Examples:

```
salt '*' tomcat.passwd secret
salt '*' tomcat.passwd secret tomcat sha1
salt '*' tomcat.passwd secret tomcat sha1 'Protected Realm'
```

salt.modules.tomcat.**reload_**(*app*, *url='http://localhost:8080/manager'*, *timeout=180*)
Reload the webapp

app the webapp context path

url [http://localhost:8080/manager] the URL of the server manager webapp

timeout [180] timeout for HTTP request

CLI Examples:

```
salt '*' tomcat.reload /jenkins
salt '*' tomcat.reload /jenkins http://localhost:8080/manager
```

salt.modules.tomcat.**serverinfo**(*url='http://localhost:8080/manager'*, *timeout=180*)
return details about the server

url [http://localhost:8080/manager] the URL of the server manager webapp

timeout [180] timeout for HTTP request

CLI Examples:

```
salt '*' tomcat.serverinfo
salt '*' tomcat.serverinfo http://localhost:8080/manager
```

salt.modules.tomcat.**sessions**(*app*, *url='http://localhost:8080/manager'*, *timeout=180*)
return the status of the webapp sessions

app the webapp context path

url [http://localhost:8080/manager] the URL of the server manager webapp

timeout [180] timeout for HTTP request

CLI Examples:

```
salt '*' tomcat.sessions /jenkins
salt '*' tomcat.sessions /jenkins http://localhost:8080/manager
```

salt.modules.tomcat.**signal**(*signal=None*)
Signals catalina to start, stop, securestart, forcestop.

CLI Example:

```
salt '*' tomcat.signal start
```

salt.modules.tomcat.**start**(*app*, *url='http://localhost:8080/manager'*, *timeout=180*)
Start the webapp

app the webapp context path

url [http://localhost:8080/manager] the URL of the server manager webapp

timeout timeout for HTTP request

CLI Examples:

```
salt '*' tomcat.start /jenkins
salt '*' tomcat.start /jenkins http://localhost:8080/manager
```

salt.modules.tomcat.**status**(*url='http://localhost:8080/manager'*, *timeout=180*)
Used to test if the tomcat manager is up

url [http://localhost:8080/manager] the URL of the server manager webapp

timeout [180] timeout for HTTP request

CLI Examples:

```
salt '*' tomcat.status
salt '*' tomcat.status http://localhost:8080/manager
```

salt.modules.tomcat.**status_webapp**(*app*, *url='http://localhost:8080/manager'*, *timeout=180*)
return the status of the webapp (stopped | running | missing)

app the webapp context path

url [http://localhost:8080/manager] the URL of the server manager webapp

timeout [180] timeout for HTTP request

CLI Examples:

```
salt '*' tomcat.status_webapp /jenkins
salt '*' tomcat.status_webapp /jenkins http://localhost:8080/manager
```

salt.modules.tomcat.**stop**(*app*, *url='http://localhost:8080/manager'*, *timeout=180*)
Stop the webapp

app the webapp context path

url [http://localhost:8080/manager] the URL of the server manager webapp

timeout [180] timeout for HTTP request

CLI Examples:

```
salt '*' tomcat.stop /jenkins
salt '*' tomcat.stop /jenkins http://localhost:8080/manager
```

salt.modules.tomcat.**undeploy**(*app*, *url='http://localhost:8080/manager'*, *timeout=180*)
Undeploy a webapp

app the webapp context path

url [http://localhost:8080/manager] the URL of the server manager webapp

timeout [180] timeout for HTTP request

CLI Examples:

```
salt '*' tomcat.undeploy /jenkins
salt '*' tomcat.undeploy /jenkins http://localhost:8080/manager
```

salt.modules.tomcat.**version**()
Return server version from catalina.sh version

CLI Example:

```
salt '*' tomcat.version
```

22.16.245 salt.modules.twilio_notify

Module for notifications via Twilio

New in version 2014.7.0.

depends

- twilio python module

configuration Configure this module by specifying the name of a configuration profile in the minion config, minion pillar, or master config.

For example:

```
my-twilio-account:
    twilio.account_sid: AC32a3c83990934481addd5ce1659f04d2
    twilio.auth_token: mytoken
```

salt.modules.twilio_notify.**send_sms**(*profile*, *body*, *to*, *from_*)
Send an sms

CLI Example:

twilio.send_sms twilio-account `Test sms' `+18019999999' `+18011111111'

22.16.246 salt.modules.upstart

Module for the management of upstart systems. The Upstart system only supports service starting, stopping and restarting.

Currently (as of Ubuntu 12.04) there is no tool available to disable Upstart services (like update-rc.d). This[1] is the recommended way to disable an Upstart service. So we assume that all Upstart services that have not been disabled in this manner are enabled.

But this is broken because we do not check to see that the dependent services are enabled. Otherwise we would have to do something like parse the output of ``initctl show-config'' to determine if all service dependencies are enabled to start on boot. For example, see the ``start on'' condition for the lightdm service below[2]. And this would be too hard. So we wait until the upstart developers have solved this problem. :) This is to say that an Upstart service that is enabled may not really be enabled.

Also, when an Upstart service is enabled, should the dependent services be enabled too? Probably not. But there should be a notice about this, at least.

[1] http://upstart.ubuntu.com/cookbook/#disabling-a-job-from-automatically-starting

[2] example upstart configuration file:

```
lightdm
emits login-session-start
emits desktop-session-start
emits desktop-shutdown
start on ((((filesystem and runlevel [!06]) and started dbus) and (drm-device-added card0 PRIMARY_DEV
stop on runlevel [016]
```

Warning: This module should not be used on Red Hat systems. For these, the `rh_service` module should be used, as it supports the hybrid upstart/sysvinit system used in RHEL/CentOS 6.

`salt.modules.upstart.`**`available`**`(`*name*`)`

Returns `True` if the specified service is available, otherwise returns `False`.

CLI Example:

```
salt '*' service.available sshd
```

`salt.modules.upstart.`**`disable`**`(`*name*, ***kwargs*`)`

Disable the named service from starting on boot

CLI Example:

```
salt '*' service.disable <service name>
```

`salt.modules.upstart.`**`disabled`**`(`*name*`)`

Check to see if the named service is disabled to start on boot

CLI Example:

```
salt '*' service.disabled <service name>
```

`salt.modules.upstart.`**`enable`**`(`*name*, ***kwargs*`)`

Enable the named service to start at boot

CLI Example:

```
salt '*' service.enable <service name>
```

`salt.modules.upstart.`**`enabled`**`(`*name*, ***kwargs*`)`

Check to see if the named service is enabled to start on boot

CLI Example:

```
salt '*' service.enabled <service name>
```

salt.modules.upstart.**force_reload**(*name*)
Force-reload the named service

CLI Example:

```
salt '*' service.force_reload <service name>
```

salt.modules.upstart.**full_restart**(*name*)
Do a full restart (stop/start) of the named service

CLI Example:

```
salt '*' service.full_restart <service name>
```

salt.modules.upstart.**get_all**()
Return all installed services

CLI Example:

```
salt '*' service.get_all
```

salt.modules.upstart.**get_disabled**()
Return the disabled services

CLI Example:

```
salt '*' service.get_disabled
```

salt.modules.upstart.**get_enabled**()
Return the enabled services

CLI Example:

```
salt '*' service.get_enabled
```

salt.modules.upstart.**missing**(*name*)
The inverse of service.available. Returns `True` if the specified service is not available, otherwise returns `False`.

CLI Example:

```
salt '*' service.missing sshd
```

salt.modules.upstart.**reload_**(*name*)
Reload the named service

CLI Example:

```
salt '*' service.reload <service name>
```

salt.modules.upstart.**restart**(*name*)
Restart the named service

CLI Example:

```
salt '*' service.restart <service name>
```

salt.modules.upstart.**start**(*name*)
Start the specified service

CLI Example:

```
salt '*' service.start <service name>
```

salt.modules.upstart.**status**(*name*, *sig=None*)
Return the status for a service, returns a bool whether the service is running.

CLI Example:

```
salt '*' service.status <service name>
```

salt.modules.upstart.**stop**(*name*)
Stop the specified service

CLI Example:

```
salt '*' service.stop <service name>
```

22.16.247 salt.modules.useradd

Manage users with the useradd command

salt.modules.useradd.**add**(*name*, *uid=None*, *gid=None*, *groups=None*, *home=None*, *shell=None*, *unique=True*, *system=False*, *fullname=''*, *roomnumber=''*, *workphone=''*, *homephone=''*, *createhome=True*, *loginclass=None*)
Add a user to the minion

CLI Example:

```
salt '*' user.add name <uid> <gid> <groups> <home> <shell>
```

salt.modules.useradd.**chfullname**(*name*, *fullname*)
Change the user's Full Name

CLI Example:

```
salt '*' user.chfullname foo "Foo Bar"
```

salt.modules.useradd.**chgid**(*name*, *gid*)
Change the default group of the user

CLI Example:

```
salt '*' user.chgid foo 4376
```

salt.modules.useradd.**chgroups**(*name*, *groups*, *append=False*)
Change the groups this user belongs to, add append to append the specified groups

CLI Example:

```
salt '*' user.chgroups foo wheel,root True
```

salt.modules.useradd.**chhome**(*name*, *home*, *persist=False*)
Change the home directory of the user, pass True for persist to move files to the new home directory if the old home directory exist.

CLI Example:

```
salt '*' user.chhome foo /home/users/foo True
```

salt.modules.useradd.**chhomephone**(*name*, *homephone*)
Change the user's Home Phone

CLI Example:

```
salt '*' user.chhomephone foo "7735551234"
```

salt.modules.useradd.**chloginclass**(*name*, *loginclass*)
Change the default login class of the user

CLI Example:

```
salt '*' user.chloginclass foo staff
```

salt.modules.useradd.**chroomnumber**(*name*, *roomnumber*)
Change the user's Room Number

CLI Example:

```
salt '*' user.chroomnumber foo 123
```

salt.modules.useradd.**chshell**(*name*, *shell*)
Change the default shell of the user

CLI Example:

```
salt '*' user.chshell foo /bin/zsh
```

salt.modules.useradd.**chuid**(*name*, *uid*)
Change the uid for a named user

CLI Example:

```
salt '*' user.chuid foo 4376
```

salt.modules.useradd.**chworkphone**(*name*, *workphone*)
Change the user's Work Phone

CLI Example:

```
salt '*' user.chworkphone foo "7735550123"
```

salt.modules.useradd.**delete**(*name*, *remove=False*, *force=False*)
Remove a user from the minion

CLI Example:

```
salt '*' user.delete name remove=True force=True
```

salt.modules.useradd.**get_loginclass**(*name*)
Get the login class of the user

CLI Example:

```
salt '*' user.get_loginclass foo
```

salt.modules.useradd.**getent**(*refresh=False*)
Return the list of all info for all users

CLI Example:

```
salt '*' user.getent
```

salt.modules.useradd.**info**(*name*)
Return user information

CLI Example:

```
salt '*' user.info root
```

salt.modules.useradd.**list_groups**(*name*)
Return a list of groups the named user belongs to

CLI Example:

```
salt '*' user.list_groups foo
```

salt.modules.useradd.**list_users**()
Return a list of all users

CLI Example:

```
salt '*' user.list_users
```

salt.modules.useradd.**rename**(*name*, *new_name*)
Change the username for a named user

CLI Example:

```
salt '*' user.rename name new_name
```

22.16.248 salt.modules.uwsgi

uWSGI stats server http://uwsgi-docs.readthedocs.org/en/latest/StatsServer.html

maintainer Peter Baumgartner <pete@lincolnloop.com>

maturity new

platform all

salt.modules.uwsgi.**stats**(*socket*)
Return the data from *uwsgi --connect-and-read* as a dictionary.

socket The socket the uWSGI stats server is listening on

CLI Example:

```
salt '*' uwsgi.stats /var/run/mystatsserver.sock

salt '*' uwsgi.stats 127.0.0.1:5050
```

22.16.249 salt.modules.varnish

Support for Varnish

New in version 2014.7.0.

Note: These functions are designed to work with all implementations of Varnish from 3.x onwards

salt.modules.varnish.**ban**(*ban_expression*)
Add ban to the varnish cache

CLI Example:

```
salt '*' varnish.ban ban_expression
```

salt.modules.varnish.**ban_list**()
List varnish cache current bans

CLI Example:

```
salt '*' varnish.ban_list
```

salt.modules.varnish.**param_set**(*param*, *value*)
Set a param in varnish cache

CLI Example:

```
salt '*' varnish.param_set param value
```

salt.modules.varnish.**param_show**(*param=None*)
Show params of varnish cache

CLI Example:

```
salt '*' varnish.param_show param
```

salt.modules.varnish.**purge**()
Purge the varnish cache

CLI Example:

```
salt '*' varnish.purge
```

salt.modules.varnish.**version**()
Return server version from varnishd -V

CLI Example:

```
salt '*' varnish.version
```

22.16.250 salt.modules.vbox_guest

VirtualBox Guest Additions installer

salt.modules.vbox_guest.**additions_install**(**args*, ***kwargs*)
Install VirtualBox Guest Additions. Uses the CD, connected by VirtualBox.

To connect VirtualBox Guest Additions via VirtualBox graphical interface press `Host+D' (`Host' is usually `Right Ctrl').

See https://www.virtualbox.org/manual/ch04.html#idp52733088 for more details.

CLI Example:

```
salt '*' vbox_guest.additions_install
salt '*' vbox_guest.additions_install reboot=True
salt '*' vbox_guest.additions_install upgrade_os=True
```

Parameters

- **reboot** (*bool*) -- reboot computer to complete installation
- **upgrade_os** (*bool*) -- upgrade OS (to ensure the latests version of kernel and developer tools are installed)

Returns version of VirtualBox Guest Additions or string with error

salt.modules.vbox_guest.**additions_mount**()
Mount VirtualBox Guest Additions CD to the temp directory.

To connect VirtualBox Guest Additions via VirtualBox graphical interface press `Host+D' (`Host' is usually `Right Ctrl').

CLI Example:

```
salt '*' vbox_guest.additions_mount
```

Returns True or OSError exception

salt.modules.vbox_guest.**additions_remove**(***kwargs*)
Remove VirtualBox Guest Additions.

Firstly it tries to uninstall itself by executing `/opt/VBoxGuestAdditions-VERSION/uninstall.run uninstall'. It uses the CD, connected by VirtualBox if it failes.

CLI Example:

```
salt '*' vbox_guest.additions_remove
salt '*' vbox_guest.additions_remove force=True
```

Parameters **force** (*bool*) -- force VirtualBox Guest Additions removing

Returns True if VirtualBox Guest Additions were removed successfully else False

salt.modules.vbox_guest.**additions_umount**(*mount_point*)
Unmount VirtualBox Guest Additions CD from the temp directory.

CLI Example:

```
salt '*' vbox_guest.additions_umount
```

Parameters **mount_point** -- directory VirtualBox Guest Additions is mounted to

Returns True or an string with error

salt.modules.vbox_guest.**additions_version**()
Check VirtualBox Guest Additions version.

CLI Example:

```
salt '*' vbox_guest.additions_version
```

Returns version of VirtualBox Guest Additions or False if they are not installed

salt.modules.vbox_guest.**grant_access_to_shared_folders_to**(*name*, *users=None*)
Grant access to auto-mounted shared folders to the users.

User is specified by it's name. To grant access for several users use argument *users*. Access will be denied to the users not listed in *users* argument.

See https://www.virtualbox.org/manual/ch04.html#sf_mount_auto for more details.

CLI Example:

```
    salt '*' vbox_guest.grant_access_to_shared_folders_to fred
    salt '*' vbox_guest.grant_access_to_shared_folders_to users ['fred', 'roman']
```

Parameters

- **name** (*str*) -- name of the user to grant access to auto-mounted shared folders to
- **users** (*list of str*) -- list of names of users to grant access to auto-mounted shared folders to (if specified, *name* will not be taken into account)

Returns list of users who have access to auto-mounted shared folders

salt.modules.vbox_guest.**list_shared_folders_users**()
List users who have access to auto-mounted shared folders.

See https://www.virtualbox.org/manual/ch04.html#sf_mount_auto for more details.

CLI Example:

```
    salt '*' vbox_guest.list_shared_folders_users
```

Returns list of users who have access to auto-mounted shared folders

22.16.251 salt.modules.virt

Work with virtual machines managed by libvirt

depends libvirt Python module

salt.modules.virt.**create**(*vm_*)
Start a defined domain

CLI Example:

```
    salt '*' virt.create <vm name>
```

salt.modules.virt.**create_xml_path**(*path*)
Start a domain based on the XML-file path passed to the function

CLI Example:

```
    salt '*' virt.create_xml_path <path to XML file on the node>
```

salt.modules.virt.**create_xml_str**(*xml*)
Start a domain based on the XML passed to the function

CLI Example:

```
    salt '*' virt.create_xml_str <XML in string format>
```

salt.modules.virt.**ctrl_alt_del**(*vm_*)
Sends CTRL+ALT+DEL to a VM

CLI Example:

```
    salt '*' virt.ctrl_alt_del <vm name>
```

salt.modules.virt.**define_vol_xml_path**(*path*)
Define a volume based on the XML-file path passed to the function

CLI Example:

```
salt '*' virt.define_vol_xml_path <path to XML file on the node>
```

salt.modules.virt.**define_vol_xml_str**(*xml*)
Define a volume based on the XML passed to the function

CLI Example:

```
salt '*' virt.define_vol_xml_str <XML in string format>
```

salt.modules.virt.**define_xml_path**(*path*)
Define a domain based on the XML-file path passed to the function

CLI Example:

```
salt '*' virt.define_xml_path <path to XML file on the node>
```

salt.modules.virt.**define_xml_str**(*xml*)
Define a domain based on the XML passed to the function

CLI Example:

```
salt '*' virt.define_xml_str <XML in string format>
```

salt.modules.virt.**destroy**(*vm_*)
Hard power down the virtual machine, this is equivalent to pulling the power

CLI Example:

```
salt '*' virt.destroy <vm name>
```

salt.modules.virt.**freecpu**()
Return an int representing the number of unallocated cpus on this hypervisor

CLI Example:

```
salt '*' virt.freecpu
```

salt.modules.virt.**freemem**()
Return an int representing the amount of memory that has not been given to virtual machines on this node

CLI Example:

```
salt '*' virt.freemem
```

salt.modules.virt.**full_info**()
Return the node_info, vm_info and freemem

CLI Example:

```
salt '*' virt.full_info
```

salt.modules.virt.**get_disks**(*vm_*)
Return the disks of a named vm

CLI Example:

```
salt '*' virt.get_disks <vm name>
```

salt.modules.virt.**get_graphics**(*vm_*)
Returns the information on vnc for a given vm

CLI Example:

```
    salt '*' virt.get_graphics <vm name>
```

salt.modules.virt.**get_macs**(*vm_*)
Return a list off MAC addresses from the named vm

CLI Example:

```
    salt '*' virt.get_macs <vm name>
```

salt.modules.virt.**get_nics**(*vm_*)
Return info about the network interfaces of a named vm

CLI Example:

```
    salt '*' virt.get_nics <vm name>
```

salt.modules.virt.**get_profiles**(*hypervisor=None*)
Return the virt profiles for hypervisor.

Currently there are profiles for:

- nic
- disk

CLI Example:

```
    salt '*' virt.get_profiles
    salt '*' virt.get_profiles hypervisor=esxi
```

salt.modules.virt.**get_xml**(*vm_*)
Returns the XML for a given vm

CLI Example:

```
    salt '*' virt.get_xml <vm name>
```

salt.modules.virt.**init**(*name*, *cpu*, *mem*, *image=None*, *nic='default'*, *hypervisor='kvm'*, *start=True*, *disk='default'*, *saltenv='base'*, ***kwargs*)
Initialize a new vm

CLI Example:

```
    salt 'hypervisor' virt.init vm_name 4 512 salt://path/to/image.raw
    salt 'hypervisor' virt.init vm_name 4 512 nic=profile disk=profile
```

salt.modules.virt.**is_hyper**()
Returns a bool whether or not this node is a hypervisor of any kind

CLI Example:

```
    salt '*' virt.is_hyper
```

salt.modules.virt.**is_kvm_hyper**()
Returns a bool whether or not this node is a KVM hypervisor

CLI Example:

```
    salt '*' virt.is_kvm_hyper
```

salt.modules.virt.**is_xen_hyper**()
Returns a bool whether or not this node is a XEN hypervisor

CLI Example:

```
salt '*' virt.is_xen_hyper
```

salt.modules.virt.**list_active_vms**()
Return a list of names for active virtual machine on the minion

CLI Example:

```
salt '*' virt.list_active_vms
```

salt.modules.virt.**list_inactive_vms**()
Return a list of names for inactive virtual machine on the minion

CLI Example:

```
salt '*' virt.list_inactive_vms
```

salt.modules.virt.**list_vms**()
Return a list of virtual machine names on the minion

CLI Example:

```
salt '*' virt.list_vms
```

salt.modules.virt.**migrate**(*vm_*, *target*, *ssh=False*)
Shared storage migration

CLI Example:

```
salt '*' virt.migrate <vm name> <target hypervisor>
```

salt.modules.virt.**migrate_non_shared**(*vm_*, *target*, *ssh=False*)
Attempt to execute non-shared storage ``all'' migration

CLI Example:

```
salt '*' virt.migrate_non_shared <vm name> <target hypervisor>
```

salt.modules.virt.**migrate_non_shared_inc**(*vm_*, *target*, *ssh=False*)
Attempt to execute non-shared storage ``all'' migration

CLI Example:

```
salt '*' virt.migrate_non_shared_inc <vm name> <target hypervisor>
```

salt.modules.virt.**node_info**()
Return a dict with information about this node

CLI Example:

```
salt '*' virt.node_info
```

salt.modules.virt.**pause**(*vm_*)
Pause the named vm

CLI Example:

```
salt '*' virt.pause <vm name>
```

salt.modules.virt.**purge**(*vm_*, *dirs=False*)
Recursively destroy and delete a virtual machine, pass True for dir's to also delete the directories containing the virtual machine disk images - USE WITH EXTREME CAUTION!

CLI Example:

```
salt '*' virt.purge <vm name>
```

salt.modules.virt.**reboot**(*vm_*)
Reboot a domain via ACPI request

CLI Example:

```
salt '*' virt.reboot <vm name>
```

salt.modules.virt.**reset**(*vm_*)
Reset a VM by emulating the reset button on a physical machine

CLI Example:

```
salt '*' virt.reset <vm name>
```

salt.modules.virt.**resume**(*vm_*)
Resume the named vm

CLI Example:

```
salt '*' virt.resume <vm name>
```

salt.modules.virt.**seed_non_shared_migrate**(*disks*, *force=False*)
Non shared migration requires that the disks be present on the migration destination, pass the disks information via this function, to the migration destination before executing the migration.

CLI Example:

```
salt '*' virt.seed_non_shared_migrate <disks>
```

salt.modules.virt.**set_autostart**(*vm_*, *state='on'*)
Set the autostart flag on a VM so that the VM will start with the host system on reboot.

CLI Example:

```
salt "*" virt.set_autostart <vm name> <on | off>
```

salt.modules.virt.**setmem**(*vm_*, *memory*, *config=False*)
Changes the amount of memory allocated to VM. The VM must be shutdown for this to work.

memory is to be specified in MB If config is True then we ask libvirt to modify the config as well

CLI Example:

```
salt '*' virt.setmem myvm 768
```

salt.modules.virt.**setvcpus**(*vm_*, *vcpus*, *config=False*)
Changes the amount of vcpus allocated to VM. The VM must be shutdown for this to work.

vcpus is an int representing the number to be assigned If config is True then we ask libvirt to modify the config as well

CLI Example:

```
salt '*' virt.setvcpus myvm 2
```

salt.modules.virt.**shutdown**(*vm_*)
Send a soft shutdown signal to the named vm

CLI Example:

```
salt '*' virt.shutdown <vm name>
```

salt.modules.virt.**start**(*vm_*)
Alias for the obscurely named `create' function

CLI Example:

```
salt '*' virt.start <vm name>
```

salt.modules.virt.**stop**(*vm_*)
Alias for the obscurely named `destroy' function

CLI Example:

```
salt '*' virt.stop <vm name>
```

salt.modules.virt.**undefine**(*vm_*)
Remove a defined vm, this does not purge the virtual machine image, and this only works if the vm is powered down

CLI Example:

```
salt '*' virt.undefine <vm name>
```

salt.modules.virt.**virt_type**()
Returns the virtual machine type as a string

CLI Example:

```
salt '*' virt.virt_type
```

salt.modules.virt.**vm_cputime**(*vm_=None*)
Return cputime used by the vms on this hyper in a list of dicts:

```
[
    'your-vm': {
        'cputime' <int>
        'cputime_percent' <int>
        },
    ...
    ]
```

If you pass a VM name in as an argument then it will return info for just the named VM, otherwise it will return all VMs.

CLI Example:

```
salt '*' virt.vm_cputime
```

salt.modules.virt.**vm_diskstats**(*vm_=None*)
Return disk usage counters used by the vms on this hyper in a list of dicts:

```
[
    'your-vm': {
        'rd_req'   : 0,
        'rd_bytes' : 0,
        'wr_req'   : 0,
        'wr_bytes' : 0,
        'errs'     : 0
        },
```

```
        ...
        ]
```

If you pass a VM name in as an argument then it will return info for just the named VM, otherwise it will return all VMs.

CLI Example:

```
salt '*' virt.vm_blockstats
```

salt.modules.virt.**vm_info**(*vm_=None*)

Return detailed information about the vms on this hyper in a list of dicts:

```
[
    'your-vm': {
        'cpu': <int>,
        'maxMem': <int>,
        'mem': <int>,
        'state': '<state>',
        'cputime' <int>
        },
    ...
    ]
```

If you pass a VM name in as an argument then it will return info for just the named VM, otherwise it will return all VMs.

CLI Example:

```
salt '*' virt.vm_info
```

salt.modules.virt.**vm_netstats**(*vm_=None*)

Return combined network counters used by the vms on this hyper in a list of dicts:

```
[
    'your-vm': {
        'rx_bytes'   : 0,
        'rx_packets' : 0,
        'rx_errs'    : 0,
        'rx_drop'    : 0,
        'tx_bytes'   : 0,
        'tx_packets' : 0,
        'tx_errs'    : 0,
        'tx_drop'    : 0
        },
    ...
    ]
```

If you pass a VM name in as an argument then it will return info for just the named VM, otherwise it will return all VMs.

CLI Example:

```
salt '*' virt.vm_netstats
```

salt.modules.virt.**vm_state**(*vm_=None*)

Return list of all the vms and their state.

If you pass a VM name in as an argument then it will return info for just the named VM, otherwise it will return all VMs.

CLI Example:

```
salt '*' virt.vm_state <vm name>
```

22.16.252 salt.modules.virtualenv

Create virtualenv environments

salt.modules.virtualenv_mod.**create**(*path*, *venv_bin=None*, *system_site_packages=False*, *distribute=False*, *clear=False*, *python=None*, *extra_search_dir=None*, *never_download=None*, *prompt=None*, *pip=False*, *symlinks=None*, *upgrade=None*, *user=None*, *use_vt=False*, *saltenv='base'*)

Create a virtualenv

path The path to create the virtualenv

venv_bin [None (default `virtualenv')] The name (and optionally path) of the virtualenv command. This can also be set globally in the minion config file as `virtualenv.venv_bin`.

system_site_packages [False] Passthrough argument given to virtualenv or pyvenv

distribute [False] Passthrough argument given to virtualenv

pip [False] Install pip after creating a virtual environment, implies distribute=True

clear [False] Passthrough argument given to virtualenv or pyvenv

python [None (default)] Passthrough argument given to virtualenv

extra_search_dir [None (default)] Passthrough argument given to virtualenv

never_download [None (default)] Passthrough argument given to virtualenv if True

prompt [None (default)] Passthrough argument given to virtualenv if not None

symlinks [None] Passthrough argument given to pyvenv if True

upgrade [None] Passthrough argument given to pyvenv if True

user [None] Set ownership for the virtualenv

runas [None] Set ownership for the virtualenv

use_vt Use VT terminal emulation (see ouptut while installing)

Note: The `runas` argument is deprecated as of 2014.1.0. `user` should be used instead.

CLI Example:

```
salt '*' virtualenv.create /path/to/new/virtualenv
```

salt.modules.virtualenv_mod.**get_resource_content**(*venv*, *package_or_requirement*, *resource_name*)

Returns the content of a resource of a package or a distribution inside a virtualenv

CLI Example:

```
salt '*' virtualenv.get_resource_content /path/to/my/venv my_package my/resource.xml
```

salt.modules.virtualenv_mod.**get_resource_path**(*venv*, *package_or_requirement*, *resource_name*)

Returns the path to a resource of a package or a distribution inside a virtualenv

CLI Example:

```
salt '*' virtualenv.get_resource_path /path/to/my/venv my_package my/resource.xml
```

salt.modules.virtualenv_mod.**get_site_packages**(*venv*)
Returns the path to the site-packages directory inside a virtualenv

CLI Example:

```
salt '*' virtualenv.get_site_packages /path/to/my/venv
```

22.16.253 salt.modules.win_autoruns

Module for listing programs that automatically run on startup (very alpha...not tested on anything but my Win 7x64)

salt.modules.win_autoruns.**list_**()
Get a list of automatically running programs

CLI Example:

```
salt '*' autoruns.list
```

22.16.254 salt.modules.win_disk

Module for gathering disk information on Windows

depends

- win32api Python module

salt.modules.win_disk.**usage**()
Return usage information for volumes mounted on this minion

CLI Example:

```
salt '*' disk.usage
```

22.16.255 salt.modules.win_dns_client

Module for configuring DNS Client on Windows systems

salt.modules.win_dns_client.**add_dns**(*ip*, *interface='Local Area Connection'*, *index=1*)
Add the DNS server to the network interface (index starts from 1)

Note: if the interface DNS is configured by DHCP, all the DNS servers will be removed from the interface and the requested DNS will be the only one

CLI Example:

```
salt '*' win_dns_client.add_dns <ip> <interface> <index>
```

salt.modules.win_dns_client.**dns_dhcp**(*interface='Local Area Connection'*)
Configure the interface to get its DNS servers from the DHCP server

CLI Example:

```
salt '*' win_dns_client.dns_dhcp <interface>
```

salt.modules.win_dns_client.**get_dns_config**(*interface='Local Area Connection'*)
Get the type of DNS configuration (dhcp / static)

CLI Example:

```
salt '*' win_dns_client.get_dns_config 'Local Area Connection'
```

salt.modules.win_dns_client.**get_dns_servers**(*interface='Local Area Connection'*)
Return a list of the configured DNS servers of the specified interface

CLI Example:

```
salt '*' win_dns_client.get_dns_servers 'Local Area Connection'
```

salt.modules.win_dns_client.**rm_dns**(*ip*, *interface='Local Area Connection'*)
Remove the DNS server from the network interface

CLI Example:

```
salt '*' win_dns_client.rm_dns <ip> <interface>
```

22.16.256 salt.modules.win_file

Manage information about files on the minion, set/read user, group data

depends

- win32api
- win32file
- win32security

salt.modules.win_file.**chgrp**(*path*, *group*)
Change the group of a file

Under Windows, this will do nothing.

While a file in Windows does have a `primary group', this rarely used attribute generally has no bearing on permissions unless intentionally configured and is only used to support Unix compatibility features (e.g. Services For Unix, NFS services).

Salt, therefore, remaps this function to do nothing while still being compatible with Unix behavior. When managing Windows systems, this function is superfluous and will generate an info level log entry if used directly.

If you do actually want to set the `primary group' of a file, use *file .chpgrp*.

CLI Example:

```
salt '*' file.chpgrp c:\temp\test.txt administrators
```

salt.modules.win_file.**chown**(*path*, *user*, *group=None*, *pgroup=None*, *follow_symlinks=True*)
Chown a file, pass the file the desired user and group

Under Windows, the group parameter will be ignored.

This is because while files in Windows do have a `primary group' property, this is rarely used. It generally has no bearing on permissions unless intentionally configured and is most commonly used to provide Unix compatibility (e.g. Services For Unix, NFS services).

If you do want to change the `primary group' property and understand the implications, pass the Windows only parameter, pgroup, instead.

To set the primary group to `None', it must be specified in quotes. Otherwise Salt will interpret it as the Python value of None and no primary group changes will occur. See the example below.

CLI Example:

```
salt '*' file.chown c:\temp\test.txt myusername
salt '*' file.chown c:\temp\test.txt myusername pgroup=Administrators
salt '*' file.chown c:\temp\test.txt myusername "pgroup='None'"
```

salt.modules.win_file.**chpgrp**(*path*, *group*)

Change the group of a file

Under Windows, this will set the rarely used primary group of a file. This generally has no bearing on permissions unless intentionally configured and is most commonly used to provide Unix compatibility (e.g. Services For Unix, NFS services).

Ensure you know what you are doing before using this function.

To set the primary group to `None', it must be specified in quotes. Otherwise Salt will interpret it as the Python value of None and no primary group changes will occur. See the example below.

CLI Example:

```
salt '*' file.chpgrp c:\temp\test.txt Administrators
salt '*' file.chpgrp c:\temp\test.txt "'None'"
```

salt.modules.win_file.**get_attributes**(*path*)

Return a dictionary object with the Windows file attributes for a file.

CLI Example:

```
salt '*' file.get_attributes c:\temp\a.txt
```

salt.modules.win_file.**get_gid**(*path*, *follow_symlinks=True*)

Return the id of the group that owns a given file

Under Windows, this will return the uid of the file.

While a file in Windows does have a `primary group', this rarely used attribute generally has no bearing on permissions unless intentionally configured and is only used to support Unix compatibility features (e.g. Services For Unix, NFS services).

Salt, therefore, remaps this function to provide functionality that somewhat resembles Unix behavior for API compatibility reasons. When managing Windows systems, this function is superfluous and will generate an info level log entry if used directly.

If you do actually want to access the `primary group' of a file, use *file.get_pgid*.

CLI Example:

```
salt '*' file.get_gid c:\temp\test.txt
```

salt.modules.win_file.**get_group**(*path*, *follow_symlinks=True*)

Return the group that owns a given file

Under Windows, this will return the user (owner) of the file.

While a file in Windows does have a `primary group', this rarely used attribute generally has no bearing on permissions unless intentionally configured and is only used to support Unix compatibility features (e.g. Services For Unix, NFS services).

Salt, therefore, remaps this function to provide functionality that somewhat resembles Unix behavior for API compatibility reasons. When managing Windows systems, this function is superfluous and will generate an info level log entry if used directly.

If you do actually want to access the `primary group' of a file, use *file.get_pgroup*.

CLI Example:

```
salt '*' file.get_group c:\temp\test.txt
```

salt.modules.win_file.**get_mode**(*path*)
Return the mode of a file

Right now we're just returning None because Windows' doesn't have a mode like Linux

CLI Example:

```
salt '*' file.get_mode /etc/passwd
```

salt.modules.win_file.**get_pgid**(*path*, *follow_symlinks=True*)
Return the id of the primary group that owns a given file (Windows only)

This function will return the rarely used primary group of a file. This generally has no bearing on permissions unless intentionally configured and is most commonly used to provide Unix compatibility (e.g. Services For Unix, NFS services).

Ensure you know what you are doing before using this function.

CLI Example:

```
salt '*' file.get_pgid c:\temp\test.txt
```

salt.modules.win_file.**get_pgroup**(*path*, *follow_symlinks=True*)
Return the name of the primary group that owns a given file (Windows only)

This function will return the rarely used primary group of a file. This generally has no bearing on permissions unless intentionally configured and is most commonly used to provide Unix compatibility (e.g. Services For Unix, NFS services).

Ensure you know what you are doing before using this function.

The return value may be `None', e.g. if the user is not on a domain. This is a valid group - do not confuse this with the Salt/Python value of None which means no value was returned. To be certain, use the *get_pgid* function which will return the SID, including for the system `None' group.

CLI Example:

```
salt '*' file.get_pgroup c:\temp\test.txt
```

salt.modules.win_file.**get_uid**(*path*, *follow_symlinks=True*)
Return the id of the user that owns a given file

Symlinks are followed by default to mimic Unix behavior. Specify *follow_symlinks=False* to turn off this behavior.

CLI Example:

```
salt '*' file.get_uid c:\temp\test.txt
salt '*' file.get_uid c:\temp\test.txt follow_symlinks=False
```

salt.modules.win_file.**get_user**(*path*, *follow_symlinks=True*)
Return the user that owns a given file

Symlinks are followed by default to mimic Unix behavior. Specify *follow_symlinks=False* to turn off this behavior.

CLI Example:

```
salt '*' file.get_user c:\temp\test.txt
salt '*' file.get_user c:\temp\test.txt follow_symlinks=False
```

salt.modules.win_file.**gid_to_group**(*gid*)

Convert the group id to the group name on this system

Under Windows, because groups are just another ACL entity, this function behaves the same as uid_to_user.

For maintaining Windows systems, this function is superfluous and only exists for API compatibility with Unix. Use the uid_to_user function instead; an info level log entry will be generated if this function is used directly.

CLI Example:

```
salt '*' file.gid_to_group S-1-5-21-626487655-2533044672-482107328-1010
```

salt.modules.win_file.**group_to_gid**(*group*)

Convert the group to the gid on this system

Under Windows, because groups are just another ACL entity, this function behaves the same as user_to_uid, except if None is given, `' is returned.

For maintaining Windows systems, this function is superfluous and only exists for API compatibility with Unix. Use the user_to_uid function instead; an info level log entry will be generated if this function is used directly.

CLI Example:

```
salt '*' file.group_to_gid administrators
```

salt.modules.win_file.**is_link**(*path*)

Return the path that a symlink points to

This is only supported on Windows Vista or later.

Inline with Unix behavior, this function will raise an error if the path is not a symlink, however, the error raised will be a SaltInvocationError, not an OSError.

CLI Example:

```
salt '*' file.is_link /path/to/link
```

salt.modules.win_file.**lchown**(*path*, *user*, *group=None*, *pgroup=None*)

Chown a file, pass the file the desired user and group without following any symlinks.

Under Windows, the group parameter will be ignored.

This is because while files in Windows do have a `primary group' property, this is rarely used. It generally has no bearing on permissions unless intentionally configured and is most commonly used to provide Unix compatibility (e.g. Services For Unix, NFS services).

If you do want to change the `primary group' property and understand the implications, pass the Windows only parameter, pgroup, instead.

To set the primary group to `None', it must be specified in quotes. Otherwise Salt will interpret it as the Python value of None and no primary group changes will occur. See the example below.

CLI Example:

```
    salt '*' file.lchown c:\temp\test.txt myusername
    salt '*' file.lchown c:\temp\test.txt myusername pgroup=Administrators
    salt '*' file.lchown c:\temp\test.txt myusername "pgroup='None'"
```

salt.modules.win_file.**readlink**(*path*)

Return the path that a symlink points to

This is only supported on Windows Vista or later.

Inline with Unix behavior, this function will raise an error if the path is not a symlink, however, the error raised will be a SaltInvocationError, not an OSError.

CLI Example:

```
    salt '*' file.readlink /path/to/link
```

salt.modules.win_file.**set_attributes**(*path*, *archive=None*, *hidden=None*, *normal=None*, *notIndexed=None*, *readonly=None*, *system=None*, *temporary=None*)

Set file attributes for a file. Note that the normal attribute means that all others are false. So setting it will clear all others.

CLI Example:

```
    salt '*' file.set_attributes c:\temp\a.txt normal=True
    salt '*' file.set_attributes c:\temp\a.txt readonly=True hidden=True
```

salt.modules.win_file.**set_mode**(*path*, *mode*)

Set the mode of a file

This just calls get_mode, which returns None because we don't use mode on Windows

CLI Example:

```
    salt '*' file.set_mode /etc/passwd 0644
```

salt.modules.win_file.**stats**(*path*, *hash_type='md5'*, *follow_symlinks=True*)

Return a dict containing the stats for a given file

Under Windows, *gid* will equal *uid* and *group* will equal *user*.

While a file in Windows does have a `primary group', this rarely used attribute generally has no bearing on permissions unless intentionally configured and is only used to support Unix compatibility features (e.g. Services For Unix, NFS services).

Salt, therefore, remaps these properties to keep some kind of compatibility with Unix behavior. If the `primary group' is required, it can be accessed in the *pgroup* and *pgid* properties.

CLI Example:

```
    salt '*' file.stats /etc/passwd
```

salt.modules.win_file.**symlink**(*src*, *link*)

Create a symbolic link to a file

This is only supported with Windows Vista or later and must be executed by a user with the SeCreateSymbolicLink privilege.

The behavior of this function matches the Unix equivalent, with one exception - invalid symlinks cannot be created. The source path must exist. If it doesn't, an error will be raised.

CLI Example:

```
salt '*' file.symlink /path/to/file /path/to/link
```

salt.modules.win_file.**uid_to_user**(*uid*)
Convert a uid to a user name

CLI Example:

```
salt '*' file.uid_to_user S-1-5-21-626487655-2533044672-482107328-1010
```

salt.modules.win_file.**user_to_uid**(*user*)
Convert user name to a uid

CLI Example:

```
salt '*' file.user_to_uid myusername
```

22.16.257 salt.modules.win_firewall

Module for configuring Windows Firewall

salt.modules.win_firewall.**add_rule**(*name*, *localport*, *protocol='tcp'*, *action='allow'*, *dir='in'*)
New in version 2015.5.0.

Add a new firewall rule

CLI Example:

```
salt '*' firewall.add_rule "test" "tcp" "8080"
```

salt.modules.win_firewall.**disable**(*profile='allprofiles'*)
Disable all the firewall profiles

CLI Example:

```
salt '*' firewall.disable
```

salt.modules.win_firewall.**enable**(*profile='allprofiles'*)
Enable firewall profile :param profile: (default: allprofiles)

New in version 2015.5.0.

CLI Example:

```
salt '*' firewall.enable
```

salt.modules.win_firewall.**get_config**()
Get the status of all the firewall profiles

CLI Example:

```
salt '*' firewall.get_config
```

salt.modules.win_firewall.**get_rule**(*name='all'*)
New in version 2015.5.0.

Get firewall rule(s) info

CLI Example:

```
salt '*' firewall.get_rule "MyAppPort"
```

22.16.258 salt.modules.win_groupadd

Manage groups on Windows

salt.modules.win_groupadd.**add**(*name*, *gid=None*, *system=False*)
Add the specified group

CLI Example:

```
salt '*' group.add foo
```

salt.modules.win_groupadd.**adduser**(*name*, *username*)
add a user to a group

CLI Example:

```
salt '*' group.adduser foo username
```

salt.modules.win_groupadd.**delete**(*name*)
Remove the named group

CLI Example:

```
salt '*' group.delete foo
```

salt.modules.win_groupadd.**deluser**(*name*, *username*)
remove a user from a group

CLI Example:

```
salt '*' group.deluser foo username
```

salt.modules.win_groupadd.**getent**(*refresh=False*)
Return info on all groups

CLI Example:

```
salt '*' group.getent
```

salt.modules.win_groupadd.**info**(*name*)
Return information about a group

CLI Example:

```
salt '*' group.info foo
```

salt.modules.win_groupadd.**members**(*name*, *members_list*)
remove a user from a group

CLI Example:

```
salt '*' group.members foo 'user1,user2,user3'
```

22.16.259 salt.modules.win_ip

The networking module for Windows based systems

salt.modules.win_ip.**disable**(*iface*)
Disable an interface

CLI Example:

```
salt -G 'os_family:Windows' ip.disable 'Local Area Connection #2'
```

salt.modules.win_ip.**enable**(*iface*)
Enable an interface

CLI Example:

```
salt -G 'os_family:Windows' ip.enable 'Local Area Connection #2'
```

salt.modules.win_ip.**get_all_interfaces**()
Return configs for all interfaces

CLI Example:

```
salt -G 'os_family:Windows' ip.get_all_interfaces
```

salt.modules.win_ip.**get_default_gateway**()
Set DNS source to DHCP on Windows

CLI Example:

```
salt -G 'os_family:Windows' ip.get_default_gateway
```

salt.modules.win_ip.**get_interface**(*iface*)
Return the configuration of a network interface

CLI Example:

```
salt -G 'os_family:Windows' ip.get_interface 'Local Area Connection'
```

salt.modules.win_ip.**get_subnet_length**(*mask*)
Convenience function to convert the netmask to the CIDR subnet length

CLI Example:

```
salt -G 'os_family:Windows' ip.get_subnet_length 255.255.255.0
```

salt.modules.win_ip.**is_disabled**(*iface*)
Returns `True` if interface is disabled, otherwise `False`

CLI Example:

```
salt -G 'os_family:Windows' ip.is_disabled 'Local Area Connection #2'
```

salt.modules.win_ip.**is_enabled**(*iface*)
Returns `True` if interface is enabled, otherwise `False`

CLI Example:

```
salt -G 'os_family:Windows' ip.is_enabled 'Local Area Connection #2'
```

salt.modules.win_ip.**raw_interface_configs**()
Return raw configs for all interfaces

CLI Example:

```
salt -G 'os_family:Windows' ip.raw_interface_configs
```

salt.modules.win_ip.**set_dhcp_all**(*iface*)
Set both IP Address and DNS to DHCP

CLI Example:

```
salt -G 'os_family:Windows' ip.set_dhcp_all 'Local Area Connection'
```

salt.modules.win_ip.**set_dhcp_dns**(*iface*)
Set DNS source to DHCP on Windows

CLI Example:

```
salt -G 'os_family:Windows' ip.set_dhcp_dns 'Local Area Connection'
```

salt.modules.win_ip.**set_dhcp_ip**(*iface*)
Set Windows NIC to get IP from DHCP

CLI Example:

```
salt -G 'os_family:Windows' ip.set_dhcp_ip 'Local Area Connection'
```

salt.modules.win_ip.**set_static_dns**(*iface*, **addrs*)
Set static DNS configuration on a Windows NIC

CLI Example:

```
salt -G 'os_family:Windows' ip.set_static_dns 'Local Area Connection' '192.168.1.1'
salt -G 'os_family:Windows' ip.set_static_dns 'Local Area Connection' '192.168.1.252' '192.168.1
```

salt.modules.win_ip.**set_static_ip**(*iface*, *addr*, *gateway=None*, *append=False*)
Set static IP configuration on a Windows NIC

iface The name of the interface to manage

addr IP address with subnet length (ex. `10.1.2.3/24`). The `ip.get_subnet_length` function can be used to calculate the subnet length from a netmask.

gateway [None] If specified, the default gateway will be set to this value.

append [False] If `True`, this IP address will be added to the interface. Default is `False`, which overrides any existing configuration for the interface and sets `addr` as the only address on the interface.

CLI Example:

```
salt -G 'os_family:Windows' ip.set_static_ip 'Local Area Connection' 10.1.2.3/24 gateway=10.1.2.
salt -G 'os_family:Windows' ip.set_static_ip 'Local Area Connection' 10.1.2.4/24 append=True
```

22.16.260 salt.modules.win_network

Module for gathering and managing network information

salt.modules.win_network.**dig**(*host*)
Performs a DNS lookup with dig

Note: dig must be installed on the Windows minion

CLI Example:

```
salt '*' network.dig archlinux.org
```

salt.modules.win_network.**hw_addr**(*iface*)
Return the hardware address (a.k.a. MAC address) for a given interface

CLI Example:

```
salt '*' network.hw_addr 'Wireless Connection #1'
```

salt.modules.win_network.**hwaddr**(*iface*)
Return the hardware address (a.k.a. MAC address) for a given interface

CLI Example:

```
salt '*' network.hw_addr 'Wireless Connection #1'
```

salt.modules.win_network.**in_subnet**(*cidr*)
Returns True if host is within specified subnet, otherwise False

CLI Example:

```
salt '*' network.in_subnet 10.0.0.0/16
```

salt.modules.win_network.**interfaces**()
Return a dictionary of information about all the interfaces on the minion

CLI Example:

```
salt '*' network.interfaces
```

salt.modules.win_network.**interfaces_names**()
Return a list of all the interfaces names

CLI Example:

```
salt '*' network.interfaces_names
```

salt.modules.win_network.**ip_addrs**(*interface=None*, *include_loopback=False*)
Returns a list of IPv4 addresses assigned to the host. 127.0.0.1 is ignored, unless `include_loopback=True' is indicated. If `interface' is provided, then only IP addresses from that interface will be returned.

CLI Example:

```
salt '*' network.ip_addrs
```

salt.modules.win_network.**ip_addrs6**(*interface=None*, *include_loopback=False*)
Returns a list of IPv6 addresses assigned to the host. ::1 is ignored, unless `include_loopback=True' is indicated. If `interface' is provided, then only IP addresses from that interface will be returned.

CLI Example:

```
salt '*' network.ip_addrs6
```

salt.modules.win_network.**ipaddrs**(*interface=None*, *include_loopback=False*)
Returns a list of IPv4 addresses assigned to the host. 127.0.0.1 is ignored, unless `include_loopback=True' is indicated. If `interface' is provided, then only IP addresses from that interface will be returned.

CLI Example:

```
salt '*' network.ip_addrs
```

salt.modules.win_network.**ipaddrs6**(*interface=None*, *include_loopback=False*)
Returns a list of IPv6 addresses assigned to the host. ::1 is ignored, unless `include_loopback=True' is indicated. If `interface' is provided, then only IP addresses from that interface will be returned.

CLI Example:

```
salt '*' network.ip_addrs6
```

salt.modules.win_network.**netstat**()
Return information on open ports and states

CLI Example:

```
salt '*' network.netstat
```

salt.modules.win_network.**nslookup**(*host*)
Query DNS for information about a domain or ip address

CLI Example:

```
salt '*' network.nslookup archlinux.org
```

salt.modules.win_network.**ping**(*host*)
Performs a ping to a host

CLI Example:

```
salt '*' network.ping archlinux.org
```

salt.modules.win_network.**subnets**()
Returns a list of subnets to which the host belongs

CLI Example:

```
salt '*' network.subnets
```

salt.modules.win_network.**traceroute**(*host*)
Performs a traceroute to a 3rd party host

CLI Example:

```
salt '*' network.traceroute archlinux.org
```

22.16.261 salt.modules.win_ntp

Management of NTP servers on Windows

New in version 2014.1.0.

salt.modules.win_ntp.**get_servers**()
Get list of configured NTP servers

CLI Example:

```
salt '*' ntp.get_servers
```

salt.modules.win_ntp.**set_servers**(**servers*)
Set Windows to use a list of NTP servers

CLI Example:

```
salt '*' ntp.set_servers 'pool.ntp.org' 'us.pool.ntp.org'
```

22.16.262 salt.modules.win_path

Manage the Windows System PATH

Note that not all Windows applications will rehash the PATH environment variable, Only the ones that listen to the WM_SETTINGCHANGE message http://support.microsoft.com/kb/104011

salt.modules.win_path.**add**(*path*, *index=0*)
Add the directory to the SYSTEM path in the index location

CLI Example:

```
# Will add to the beginning of the path
salt '*' win_path.add 'c:\python27' 0

# Will add to the end of the path
salt '*' win_path.add 'c:\python27' index='-1'
```

salt.modules.win_path.**exists**(*path*)
Check if the directory is configured in the SYSTEM path Case-insensitive and ignores trailing backslash

CLI Example:

```
salt '*' win_path.exists 'c:\python27'
salt '*' win_path.exists 'c:\python27\'
salt '*' win_path.exists 'C:\pyThon27'
```

salt.modules.win_path.**get_path**()
Returns the system path

salt.modules.win_path.**rehash**()
Send a WM_SETTINGCHANGE Broadcast to Windows to rehash the Environment variables

salt.modules.win_path.**remove**(*path*)
Remove the directory from the SYSTEM path

22.16.263 salt.modules.win_pkg

A module to manage software on Windows

depends

- win32com
- win32con
- win32api
- pywintypes

salt.modules.win_pkg.**get_repo_data**(*saltenv='base'*)
Returns the cached winrepo data

CLI Example:

```
salt '*' pkg.get_repo_data
```

salt.modules.win_pkg.**install**(*name=None*, *refresh=False*, *pkgs=None*, *saltenv='base'*, ***kwargs*)
Install the passed package

Return a dict containing the new package names and versions:

```
{'<package>': {'old': '<old-version>',
               'new': '<new-version>'}}
```

CLI Example:

```
    salt '*' pkg.install <package name>
```

salt.modules.win_pkg.**latest_version**(**names*, ***kwargs*)

Return the latest version of the named package available for upgrade or installation. If more than one package name is specified, a dict of name/version pairs is returned.

If the latest version of a given package is already installed, an empty string will be returned for that package.

CLI Example:

```
    salt '*' pkg.latest_version <package name>
    salt '*' pkg.latest_version <package1> <package2> <package3> ...
```

salt.modules.win_pkg.**list_available**(**names*)

Return a list of available versions of the specified package.

CLI Example:

```
    salt '*' pkg.list_available <package name>
    salt '*' pkg.list_available <package name01> <package name02>
```

salt.modules.win_pkg.**list_pkgs**(*versions_as_list=False*, ***kwargs*)

List the packages currently installed in a dict:

```
    {'<package_name>': '<version>'}
```

CLI Example:

```
    salt '*' pkg.list_pkgs
    salt '*' pkg.list_pkgs versions_as_list=True
```

salt.modules.win_pkg.**list_upgrades**(*refresh=True*)

List all available package upgrades on this system

CLI Example:

```
    salt '*' pkg.list_upgrades
```

salt.modules.win_pkg.**purge**(*name=None*, *pkgs=None*, *version=None*, ***kwargs*)

Package purges are not supported, this function is identical to `remove()`.

name The name of the package to be deleted.

version The version of the package to be deleted. If this option is used in combination with the `pkgs` option below, then this version will be applied to all targeted packages.

Multiple Package Options:

pkgs A list of packages to delete. Must be passed as a python list. The `name` parameter will be ignored if this option is passed.

New in version 0.16.0.

Returns a dict containing the changes.

CLI Example:

```
    salt '*' pkg.purge <package name>
    salt '*' pkg.purge <package1>,<package2>,<package3>
    salt '*' pkg.purge pkgs='["foo", "bar"]'
```

salt.modules.win_pkg.**refresh_db**(*saltenv='base'*)

Just recheck the repository and return a dict:

```
{'<database name>': Bool}
```

CLI Example:

```
salt '*' pkg.refresh_db
```

salt.modules.win_pkg.**remove**(*name=None*, *pkgs=None*, *version=None*, *extra_uninstall_flags=None*, ***kwargs*)

Remove packages.

name The name of the package to be deleted.

version The version of the package to be deleted. If this option is used in combination with the `pkgs` option below, then this version will be applied to all targeted packages.

Multiple Package Options:

pkgs A list of packages to delete. Must be passed as a python list. The `name` parameter will be ignored if this option is passed.

New in version 0.16.0.

Returns a dict containing the changes.

CLI Example:

```
salt '*' pkg.remove <package name>
salt '*' pkg.remove <package1>,<package2>,<package3>
salt '*' pkg.remove pkgs='["foo", "bar"]'
```

salt.modules.win_pkg.**upgrade**(*refresh=True*)

Run a full system upgrade

Return a dict containing the new package names and versions:

```
{'<package>': {'old': '<old-version>',
               'new': '<new-version>'}}
```

CLI Example:

```
salt '*' pkg.upgrade
```

salt.modules.win_pkg.**upgrade_available**(*name*)

Check whether or not an upgrade is available for a given package

CLI Example:

```
salt '*' pkg.upgrade_available <package name>
```

salt.modules.win_pkg.**version**(**names*, ***kwargs*)

Returns a version if the package is installed, else returns an empty string

CLI Example:

```
salt '*' pkg.version <package name>
```

22.16.264 salt.modules.win_repo

Module to manage Windows software repo on a Standalone Minion

The following options must be set in the Minion config: file_client: local win_repo_cachefile: c:saltfile_rootswinrepowinrepo.p win_repo: c:saltfile_rootswinrepo

Place all Windows package files in the `win_repo' directory.

`salt.modules.win_repo.`**`genrepo`**`()`
Generate win_repo_cachefile based on sls files in the win_repo

CLI Example:

```
salt-call winrepo.genrepo -c c:\salt\conf
```

`salt.modules.win_repo.`**`update_git_repos`**`()`
Checkout git repos containing Windows Software Package Definitions

---

Note: This function will not work unless git is installed and the git module is further updated to work on Windows. In the meantime just place all Windows package files in the `win_repo` directory.

22.16.265 salt.modules.win_servermanager

Manage Windows features via the ServerManager powershell module

`salt.modules.win_servermanager.`**`install`**`(`*feature*, *recurse=False*`)`
Install a feature

Note: Some features requires reboot after un/installation, if so until the server is restarted Other features can not be installed !

Note: Some features takes a long time to complete un/installation, set -t with a long timeout

CLI Example:

```
salt '*' win_servermanager.install Telnet-Client
salt '*' win_servermanager.install SNMP-Service True
```

`salt.modules.win_servermanager.`**`list_available`**`()`
List available features to install

CLI Example:

```
salt '*' win_servermanager.list_available
```

`salt.modules.win_servermanager.`**`list_installed`**`()`
List installed features

CLI Example:

```
salt '*' win_servermanager.list_installed
```

`salt.modules.win_servermanager.`**`remove`**`(`*feature*`)`
Remove an installed feature

Note: Some features require a reboot after installation/uninstallation. If one of these features are modified, then other features cannot be installed until the server is restarted. Additionally, some features take a while to complete installation/uninstallation, so it is a good idea to use the `-t` option to set a longer timeout.

CLI Example:

```
salt -t 600 '*' win_servermanager.remove Telnet-Client
```

22.16.266 salt.modules.win_service

Windows Service module.

salt.modules.win_service.**available**(*name*)
Returns `True` if the specified service is available, otherwise returns `False`.

CLI Example:

```
salt '*' service.available <service name>
```

salt.modules.win_service.**create_win_salt_restart_task**()
Create a task in Windows task scheduler to enable restarting the salt-minion

CLI Example:

```
salt '*' service.create_win_salt_restart_task()
```

salt.modules.win_service.**disable**(*name*, ***kwargs*)
Disable the named service to start at boot

CLI Example:

```
salt '*' service.disable <service name>
```

salt.modules.win_service.**disabled**(*name*)
Check to see if the named service is disabled to start on boot

CLI Example:

```
salt '*' service.disabled <service name>
```

salt.modules.win_service.**enable**(*name*, ***kwargs*)
Enable the named service to start at boot

CLI Example:

```
salt '*' service.enable <service name>
```

salt.modules.win_service.**enabled**(*name*, ***kwargs*)
Check to see if the named service is enabled to start on boot

CLI Example:

```
salt '*' service.enabled <service name>
```

salt.modules.win_service.**execute_salt_restart_task**()
Run the Windows Salt restart task

CLI Example:

```
salt '*' service.execute_salt_restart_task()
```

salt.modules.win_service.**get_all**()
Return all installed services

CLI Example:

```
salt '*' service.get_all
```

salt.modules.win_service.**get_disabled**()
Return the disabled services

CLI Example:

```
salt '*' service.get_disabled
```

salt.modules.win_service.**get_enabled**()
Return the enabled services

CLI Example:

```
salt '*' service.get_enabled
```

salt.modules.win_service.**get_service_name**(**args*)
The Display Name is what is displayed in Windows when services.msc is executed. Each Display Name has an associated Service Name which is the actual name of the service. This function allows you to discover the Service Name by returning a dictionary of Display Names and Service Names, or filter by adding arguments of Display Names.

If no args are passed, return a dict of all services where the keys are the service Display Names and the values are the Service Names.

If arguments are passed, create a dict of Display Names and Service Names

CLI Example:

```
salt '*' service.get_service_name
salt '*' service.get_service_name 'Google Update Service (gupdate)' 'DHCP Client'
```

salt.modules.win_service.**getsid**(*name*)
Return the sid for this windows service

CLI Example:

```
salt '*' service.getsid <service name>
```

salt.modules.win_service.**has_powershell**()
Confirm if Powershell is available

CLI Example:

```
salt '*' service.has_powershell
```

salt.modules.win_service.**missing**(*name*)
The inverse of service.available. Returns `True` if the specified service is not available, otherwise returns `False`.

CLI Example:

```
salt '*' service.missing <service name>
```

salt.modules.win_service.**restart**(*name*)
Restart the named service

CLI Example:

```
salt '*' service.restart <service name>
```

salt.modules.win_service.**start**(*name*)
Start the specified service

CLI Example:

```
salt '*' service.start <service name>
```

salt.modules.win_service.**status**(*name*, *sig=None*)
Return the status for a service, returns the PID or an empty string if the service is running or not, pass a signature to use to find the service via ps

CLI Example:

```
salt '*' service.status <service name> [service signature]
```

salt.modules.win_service.**stop**(*name*)
Stop the specified service

CLI Example:

```
salt '*' service.stop <service name>
```

22.16.267 salt.modules.win_shadow

Manage the shadow file

salt.modules.win_shadow.**info**(*name*)
Return information for the specified user This is just returns dummy data so that salt states can work.

CLI Example:

```
salt '*' shadow.info root
```

salt.modules.win_shadow.**set_password**(*name*, *password*)
Set the password for a named user.

CLI Example:

```
salt '*' shadow.set_password root mysecretpassword
```

22.16.268 salt.modules.win_status

Module for returning various status data about a minion. These data can be useful for compiling into stats later.

depends

- pythoncom
- wmi

salt.modules.win_status.**master**(*master=None*, *connected=True*)
New in version 2015.5.0.

Fire an event if the minion gets disconnected from its master. This function is meant to be run via a scheduled job from the minion. If master_ip is an FQDN/Hostname, is must be resolvable to a valid IPv4 address.

CLI Example:

```
salt '*' status.master
```

salt.modules.win_status.**procs**()
Return the process data

CLI Example:

```
salt '*' status.procs
```

22.16.269 salt.modules.win_system

Support for reboot, shutdown, etc

salt.modules.win_system.**get_computer_desc**()
Get the Windows computer description

CLI Example:

```
salt 'minion-id' system.get_computer_desc
```

salt.modules.win_system.**get_computer_name**()
Get the Windows computer name

CLI Example:

```
salt 'minion-id' system.get_computer_name
```

salt.modules.win_system.**get_pending_computer_name**()
Get a pending computer name. If the computer name has been changed, and the change is pending a system reboot, this function will return the pending computer name. Otherwise, `None` will be returned. If there was an error retrieving the pending computer name, `False` will be returned, and an error message will be logged to the minion log.

CLI Example:

```
salt 'minion-id' system.get_pending_computer_name
```

salt.modules.win_system.**get_system_date**()
Get the Windows system date

CLI Example:

```
salt '*' system.get_system_date
```

salt.modules.win_system.**get_system_time**()
Get the Windows system time

CLI Example:

```
salt '*' system.get_system_time
```

salt.modules.win_system.**halt**(*timeout=5*)
Halt a running system

CLI Example:

```
salt '*' system.halt
```

salt.modules.win_system.**init**(*runlevel*)
Change the system runlevel on sysV compatible systems

CLI Example:

```
salt '*' system.init 3
```

salt.modules.win_system.**join_domain**(*domain=None, username=None, password=None, account_ou=None, account_exists=False*)

Join a computer to an Active Directory domain

domain The domain to which the computer should be joined, e.g. `my-company.com`

username Username of an account which is authorized to join computers to the specified domain. Need to be either fully qualified like `user@domain.tld` or simply `user`

password Password of the specified user

account_ou [None] The DN of the OU below which the account for this computer should be created when joining the domain, e.g. `ou=computers,ou=departm_432,dc=my-company,dc=com`

account_exists [False] Needs to be set to `True` to allow re-using an existing account

CLI Example:

```
salt 'minion-id' system.join_domain domain='domain.tld' \
                 username='joinuser' password='joinpassword' \
                 account_ou='ou=clients,ou=org,dc=domain,dc=tld' \
                 account_exists=False
```

salt.modules.win_system.**poweroff**(*timeout=5*)

Poweroff a running system

CLI Example:

```
salt '*' system.poweroff
```

salt.modules.win_system.**reboot**(*timeout=5*)

Reboot the system

CLI Example:

```
salt '*' system.reboot
```

salt.modules.win_system.**set_computer_desc**(*desc*)

Set the Windows computer description

CLI Example:

```
salt 'minion-id' system.set_computer_desc 'This computer belongs to Dave!'
```

salt.modules.win_system.**set_computer_name**(*name*)

Set the Windows computer name

CLI Example:

```
salt 'minion-id' system.set_computer_name 'DavesComputer'
```

salt.modules.win_system.**set_system_date**(*newdate*)

Set the Windows system date. Use <mm-dd-yy> format for the date.

CLI Example:

```
salt '*' system.set_system_date '03-28-13'
```

salt.modules.win_system.**set_system_time**(*newtime*)

Set the Windows system time

CLI Example:

```
salt '*' system.set_system_time '11:31:15 AM'
```

salt.modules.win_system.**shutdown**(*timeout=5*)
Shutdown a running system

CLI Example:

```
salt '*' system.shutdown
```

salt.modules.win_system.**shutdown_hard**()
Shutdown a running system with no timeout or warning

CLI Example:

```
salt '*' system.shutdown_hard
```

salt.modules.win_system.**start_time_service**()
Start the Windows time service

CLI Example:

```
salt '*' system.start_time_service
```

salt.modules.win_system.**stop_time_service**()
Stop the Windows time service

CLI Example:

```
salt '*' system.stop_time_service
```

22.16.270 salt.modules.win_timezone

Module for managing timezone on Windows systems.

salt.modules.win_timezone.**get_hwclock**()
Get current hardware clock setting (UTC or localtime)

CLI Example:

```
salt '*' timezone.get_hwclock
```

salt.modules.win_timezone.**get_offset**()
Get current numeric timezone offset from UCT (i.e. -0700)

CLI Example:

```
salt '*' timezone.get_offset
```

salt.modules.win_timezone.**get_zone**()
Get current timezone (i.e. America/Denver)

CLI Example:

```
salt '*' timezone.get_zone
```

salt.modules.win_timezone.**get_zonecode**()
Get current timezone (i.e. PST, MDT, etc)

CLI Example:

```
salt '*' timezone.get_zonecode
```

salt.modules.win_timezone.**set_hwclock**(*clock*)
Sets the hardware clock to be either UTC or localtime

CLI Example:

```
salt '*' timezone.set_hwclock UTC
```

salt.modules.win_timezone.**set_zone**(*timezone*)
Unlinks, then symlinks /etc/localtime to the set timezone.

The timezone is crucial to several system processes, each of which SHOULD be restarted (for instance, whatever you system uses as its cron and syslog daemons). This will not be magically done for you!

CLI Example:

```
salt '*' timezone.set_zone 'America/Denver'
```

salt.modules.win_timezone.**zone_compare**(*timezone*)
Checks the md5sum between the given timezone, and the one set in /etc/localtime. Returns True if they match, and False if not. Mostly useful for running state checks.

Example:

```
salt '*' timezone.zone_compare 'America/Denver'
```

22.16.271 salt.modules.win_update

Module for running windows updates.

depends

- win32com
- win32con
- win32api
- pywintypes

New in version 2014.7.0.

class salt.modules.win_update.**PyWinUpdater**(*categories=None*, *skipUI=True*, *skipDownloaded=False*, *skipInstalled=True*, *skipReboot=False*, *skipPresent=False*, *softwareUpdates=True*, *driverUpdates=False*, *skipHidden=True*)

AutoSearch()
this function generates a search string. simplifying the search function while still providing as many features as possible.

Download()

GetAvailableCategories()

GetCategories()

GetDownloadResults()

GetInstallationResults()
this gets results of installation process.

GetInstallationResultsPretty()
converts the installation results into a pretty print.

GetSearchResults()

GetSearchResultsPretty()

Install()

Search(*searchString*)

SetCategories(*categories*)

SetInclude(*include*, *state*)

SetIncludes(*includes*)

salt.modules.win_update.**download_updates**(*includes=None*, *retries=5*, *categories=None*)
Downloads all available updates, skipping those that require user interaction.

Various aspects of the updates can be included or excluded. this feature is still in development.

retries Number of retries to make before giving up. This is total, not per step.

categories Specify the categories to update. Must be passed as a list.

```
salt '*' win_update.download_updates categories="['Updates']"
```

Categories include the following:

- Updates
- Windows 7
- Critical Updates
- Security Updates
- Update Rollups

CLI Examples:

```
# Normal Usage
salt '*' win_update.download_updates

# Download critical updates only
salt '*' win_update.download_updates categories="['Critical Updates']"
```

salt.modules.win_update.**install_updates**(*includes=None*, *retries=5*, *categories=None*)
Downloads and installs all available updates, skipping those that require user interaction.

Add `cached` to only install those updates which have already been downloaded.

you can set the maximum number of retries to `n` in the search process by adding: `retries=n`

various aspects of the updates can be included or excluded. This function is still under development.

retries Number of retries to make before giving up. This is total, not per step.

categories Specify the categories to install. Must be passed as a list.

```
salt '*' win_update.install_updates categories="['Updates']"
```

Categories include the following:

- Updates
- Windows 7
- Critical Updates
- Security Updates
- Update Rollups

CLI Examples:

```
# Normal Usage
salt '*' win_update.install_updates

# Install all critical updates
salt '*' win_update.install_updates categories="['Critical Updates']"
```

salt.modules.win_update.**list_updates**(*verbose=False, includes=None, retries=5, categories=None*)

Returns a summary of available updates, grouped into their non-mutually exclusive categories.

verbose Print results in greater detail

retries Number of retries to make before giving up. This is total, not per step.

categories Specify the categories to list. Must be passed as a list.

```
salt '*' win_update.list_updates categories="['Updates']"
```

Categories include the following:

- Updates
- Windows 7
- Critical Updates
- Security Updates
- Update Rollups

CLI Examples:

```
# Normal Usage
salt '*' win_update.list_updates

# List all critical updates list in verbose detail
salt '*' win_update.list_updates categories=['Critical Updates'] verbose=True
```

22.16.272 salt.modules.win_useradd

Manage Windows users with the net user command

NOTE: This currently only works with local user accounts, not domain accounts

salt.modules.win_useradd.**add**(*name, password=None, uid=None, gid=None, groups=None, home=False, shell=None, unique=False, system=False, fullname=False, roomnumber=False, workphone=False, homephone=False, loginclass=False, createhome=False*)

Add a user to the minion

CLI Example:

```
salt '*' user.add name password
```

salt.modules.win_useradd.**addgroup**(*name*, *group*)
Add user to a group

CLI Example:

```
salt '*' user.addgroup username groupname
```

salt.modules.win_useradd.**chfullname**(*name*, *fullname*)
Change the full name of the user

CLI Example:

```
salt '*' user.chfullname user 'First Last'
```

salt.modules.win_useradd.**chgroups**(*name*, *groups*, *append=False*)
Change the groups this user belongs to, add append to append the specified groups

CLI Example:

```
salt '*' user.chgroups foo wheel,root True
```

salt.modules.win_useradd.**chhome**(*name*, *home*)
Change the home directory of the user

CLI Example:

```
salt '*' user.chhome foo \\fileserver\home\foo
```

salt.modules.win_useradd.**chprofile**(*name*, *profile*)
Change the profile directory of the user

CLI Example:

```
salt '*' user.chprofile foo \\fileserver\profiles\foo
```

salt.modules.win_useradd.**delete**(*name*, *purge=False*, *force=False*)
Remove a user from the minion NOTE: purge and force have not been implemented on Windows yet

CLI Example:

```
salt '*' user.delete name
```

salt.modules.win_useradd.**getent**(*refresh=False*)
Return the list of all info for all users

CLI Example:

```
salt '*' user.getent
```

salt.modules.win_useradd.**info**(*name*)
Return user information

CLI Example:

```
salt '*' user.info root
```

salt.modules.win_useradd.**list_groups**(*name*)
Return a list of groups the named user belongs to

CLI Example:

```
salt '*' user.list_groups foo
```

salt.modules.win_useradd.**list_users**()
Return a list of users on Windows

salt.modules.win_useradd.**removegroup**(*name*, *group*)
Remove user from a group

CLI Example:

```
salt '*' user.removegroup username groupname
```

salt.modules.win_useradd.**rename**(*name*, *new_name*)
Change the username for a named user

CLI Example:

```
salt '*' user.rename name new_name
```

salt.modules.win_useradd.**setpassword**(*name*, *password*)
Set a user's password

CLI Example:

```
salt '*' user.setpassword name password
```

22.16.273 salt.modules.xapi

This module (mostly) uses the XenAPI to manage Xen virtual machines.

Big fat warning: the XenAPI used in this file is the one bundled with Xen Source, NOT XenServer nor Xen Cloud Platform. As a matter of fact it *will* fail under those platforms. From what I've read, little work is needed to adapt this code to XS/XCP, mostly playing with XenAPI version, but as XCP is not taking precedence on Xen Source on many platforms, please keep compatibility in mind.

Useful documentation:

. http://downloads.xen.org/Wiki/XenAPI/xenapi-1.0.6.pdf . http://docs.vmd.citrix.com/XenServer/6.0.0/1.0/en_gb/api/ . https://github.com/xapi-project/xen-api/tree/master/scripts/examples/python . http://xenbits.xen.org/gitweb/?p=xen.git;a=tree;f=tools/python/xen/xm;hb=HEAD

salt.modules.xapi.**create**(*config_*)
Start a defined domain

CLI Example:

```
salt '*' virt.create <path to Xen cfg file>
```

salt.modules.xapi.**destroy**(*vm_*)
Hard power down the virtual machine, this is equivalent to pulling the power

CLI Example:

```
salt '*' virt.destroy <vm name>
```

salt.modules.xapi.**freecpu**()
Return an int representing the number of unallocated cpus on this hypervisor

CLI Example:

```
salt '*' virt.freecpu
```

salt.modules.xapi.**freemem**()
Return an int representing the amount of memory that has not been given to virtual machines on this node

CLI Example:

```
salt '*' virt.freemem
```

salt.modules.xapi.**full_info**()
Return the node_info, vm_info and freemem

CLI Example:

```
salt '*' virt.full_info
```

salt.modules.xapi.**get_disks**(*vm_*)
Return the disks of a named vm

CLI Example:

```
salt '*' virt.get_disks <vm name>
```

salt.modules.xapi.**get_macs**(*vm_*)
Return a list off MAC addresses from the named vm

CLI Example:

```
salt '*' virt.get_macs <vm name>
```

salt.modules.xapi.**get_nics**(*vm_*)
Return info about the network interfaces of a named vm

CLI Example:

```
salt '*' virt.get_nics <vm name>
```

salt.modules.xapi.**is_hyper**()
Returns a bool whether or not this node is a hypervisor of any kind

CLI Example:

```
salt '*' virt.is_hyper
```

salt.modules.xapi.**list_vms**()
Return a list of virtual machine names on the minion

CLI Example:

```
salt '*' virt.list_vms
```

salt.modules.xapi.**migrate**(*vm_*, *target*, *live=1*, *port=0*, *node=-1*, *ssl=None*, *change_home_server=0*)
Migrates the virtual machine to another hypervisor

CLI Example:

```
salt '*' virt.migrate <vm name> <target hypervisor> [live] [port] [node] [ssl] [change_home_serv
```

Optional values:

live Use live migration

port Use a specified port

node Use specified NUMA node on target

ssl use ssl connection for migration

change_home_server change home server for managed domains

salt.modules.xapi.**node_info**()
Return a dict with information about this node

CLI Example:

```
salt '*' virt.node_info
```

salt.modules.xapi.**pause**(*vm_*)
Pause the named vm

CLI Example:

```
salt '*' virt.pause <vm name>
```

salt.modules.xapi.**reboot**(*vm_*)
Reboot a domain via ACPI request

CLI Example:

```
salt '*' virt.reboot <vm name>
```

salt.modules.xapi.**reset**(*vm_*)
Reset a VM by emulating the reset button on a physical machine

CLI Example:

```
salt '*' virt.reset <vm name>
```

salt.modules.xapi.**resume**(*vm_*)
Resume the named vm

CLI Example:

```
salt '*' virt.resume <vm name>
```

salt.modules.xapi.**setmem**(*vm_*, *memory*)
Changes the amount of memory allocated to VM.

Memory is to be specified in MB

CLI Example:

```
salt '*' virt.setmem myvm 768
```

salt.modules.xapi.**setvcpus**(*vm_*, *vcpus*)
Changes the amount of vcpus allocated to VM.

vcpus is an int representing the number to be assigned

CLI Example:

```
salt '*' virt.setvcpus myvm 2
```

salt.modules.xapi.**shutdown**(*vm_*)
Send a soft shutdown signal to the named vm

CLI Example:

```
salt '*' virt.shutdown <vm name>
```

salt.modules.xapi.**start**(*config_*)
Alias for the obscurely named `create' function

CLI Example:

```
salt '*' virt.start <path to Xen cfg file>
```

salt.modules.xapi.**vcpu_pin**(*vm_*, *vcpu*, *cpus*)
Set which CPUs a VCPU can use.

CLI Example:

```
salt 'foo' virt.vcpu_pin domU-id 2 1
salt 'foo' virt.vcpu_pin domU-id 2 2-6
```

salt.modules.xapi.**vm_cputime**(*vm_=None*)
Return cputime used by the vms on this hyper in a list of dicts:

```
[
    'your-vm': {
        'cputime' <int>
        'cputime_percent' <int>
        },
    ...
    ]
```

If you pass a VM name in as an argument then it will return info for just the named VM, otherwise it will return all VMs.

CLI Example:

```
salt '*' virt.vm_cputime
```

salt.modules.xapi.**vm_diskstats**(*vm_=None*)
Return disk usage counters used by the vms on this hyper in a list of dicts:

```
[
    'your-vm': {
        'io_read_kbs'   : 0,
        'io_write_kbs'  : 0
        },
    ...
    ]
```

If you pass a VM name in as an argument then it will return info for just the named VM, otherwise it will return all VMs.

CLI Example:

```
salt '*' virt.vm_diskstats
```

salt.modules.xapi.**vm_info**(*vm_=None*)
Return detailed information about the vms.

If you pass a VM name in as an argument then it will return info for just the named VM, otherwise it will return all VMs.

CLI Example:

```
salt '*' virt.vm_info
```

salt.modules.xapi.**vm_netstats**(*vm_=None*)

Return combined network counters used by the vms on this hyper in a list of dicts:

```
[
    'your-vm': {
        'io_read_kbs'           : 0,
        'io_total_read_kbs'     : 0,
        'io_total_write_kbs'    : 0,
        'io_write_kbs'          : 0
        },
    ...
    ]
```

If you pass a VM name in as an argument then it will return info for just the named VM, otherwise it will return all VMs.

CLI Example:

```
salt '*' virt.vm_netstats
```

salt.modules.xapi.**vm_state**(*vm_=None*)

Return list of all the vms and their state.

If you pass a VM name in as an argument then it will return info for just the named VM, otherwise it will return all VMs.

CLI Example:

```
salt '*' virt.vm_state <vm name>
```

22.16.274 salt.modules.xfs

Module for managing XFS file systems.

salt.modules.xfs.**defragment**(*device*)

Defragment mounted XFS filesystem. In order to mount a filesystem, device should be properly mounted and writable.

CLI Example:

```
salt '*' xfs.defragment /dev/sda1
```

salt.modules.xfs.**devices**()

Get known XFS formatted devices on the system.

CLI Example:

```
salt '*' xfs.devices
```

salt.modules.xfs.**dump**(*device*, *destination*, *level=0*, *label=None*, *noerase=None*)

Dump filesystem device to the media (file, tape etc).

Required parameters:

- **device**: XFS device, content of which to be dumped.
- **destination**: Specifies a dump destination.

Valid options are:

- **label**: Label of the dump. Otherwise automatically generated label is used.
- **level**: Specifies a dump level of 0 to 9.
- **noerase**: Pre-erase media.

Other options are not used in order to let `xfsdump` use its default values, as they are most optimal. See the `xfsdump(8)` manpage for a more complete description of these options.

CLI Example:

```
salt '*' xfs.dump /dev/sda1 /detination/on/the/client
salt '*' xfs.dump /dev/sda1 /detination/on/the/client label='Company accountancy'
salt '*' xfs.dump /dev/sda1 /detination/on/the/client noerase=True
```

`salt.modules.xfs.`**`estimate`**`(`*path*`)`

Estimate the space that an XFS filesystem will take. For each directory estimate the space that directory would take if it were copied to an XFS filesystem. Estimation does not cross mount points.

CLI Example:

```
salt '*' xfs.estimate /path/to/file
salt '*' xfs.estimate /path/to/dir/*
```

`salt.modules.xfs.`**`info`**`(`*device*`)`

Get filesystem geometry information.

CLI Example:

```
salt '*' xfs.info /dev/sda1
```

`salt.modules.xfs.`**`inventory`**`()`

Display XFS dump inventory without restoration.

CLI Example:

```
salt '*' xfs.inventory
```

`salt.modules.xfs.`**`mkfs`**`(`*device*, *label=None*, *ssize=None*, *noforce=None*, *bso=None*, *gmo=None*, *ino=None*, *lso=None*, *rso=None*, *nmo=None*, *dso=None*`)`

Create a file system on the specified device. By default wipes out with force.

General options:

- **label**: Specify volume label.
- **ssize**: Specify the fundamental sector size of the filesystem.
- **noforce**: Do not force create filesystem, if disk is already formatted.

Filesystem geometry options:

- **bso**: Block size options.
- **gmo**: Global metadata options.
- **dso**: **Data section options. These options specify the location, size**, and other parameters of the data section of the filesystem.
- **ino**: Inode options to specify the inode size of the filesystem, and other inode allocation parameters.
- **lso**: Log section options.
- **nmo**: Naming options.
- **rso**: Realtime section options.

See the `mkfs.xfs(8)` manpage for a more complete description of corresponding options description.

CLI Example:

```
salt '*' xfs.mkfs /dev/sda1
salt '*' xfs.mkfs /dev/sda1 dso='su=32k,sw=6' noforce=True
salt '*' xfs.mkfs /dev/sda1 dso='su=32k,sw=6' lso='logdev=/dev/sda2,size=10000b'
```

`salt.modules.xfs.`**`modify`**(*device*, *label=None*, *lazy_counting=None*, *uuid=None*)

Modify parameters of an XFS filesystem.

CLI Example:

```
salt '*' xfs.modify /dev/sda1 label='My backup' lazy_counting=False
salt '*' xfs.modify /dev/sda1 uuid=False
salt '*' xfs.modify /dev/sda1 uuid=True
```

`salt.modules.xfs.`**`prune_dump`**(*sessionid*)

Prunes the dump session identified by the given session id.

CLI Example:

```
salt '*' xfs.prune_dump b74a3586-e52e-4a4a-8775-c3334fa8ea2c
```

22.16.275 salt.modules.xmpp

Module for Sending Messages via XMPP (a.k.a. Jabber)

New in version 2014.1.0.

depends

- sleekxmpp python module

configuration This module can be used by either passing a jid and password directly to send_message, or by specifying the name of a configuration profile in the minion config, minion pillar, or master config.

For example:

```
my-xmpp-login:
    xmpp.jid: myuser@jabber.example.org/resourcename
    xmpp.password: verybadpass
```

The resourcename refers to the resource that is using this account. It is user-definable, and optional. The following configurations are both valid:

```
my-xmpp-login:
    xmpp.jid: myuser@jabber.example.org/salt
    xmpp.password: verybadpass

my-xmpp-login:
    xmpp.jid: myuser@jabber.example.org
    xmpp.password: verybadpass
```

class `salt.modules.xmpp.`**`SendMsgBot`**(*jid*, *password*, *recipient*, *msg*)

`start`(*event*)

salt.modules.xmpp.**send_msg**(*recipient*, *message*, *jid=None*, *password=None*, *profile=None*)
Send a message to an XMPP recipient. Designed for use in states.

CLI Examples:

```
xmpp.send_msg 'admins@xmpp.example.com' 'This is a salt module test'             profile='my-xmp
xmpp.send_msg 'admins@xmpp.example.com' 'This is a salt module test'             jid='myuser@xmp
```

22.16.276 salt.modules.yumpkg

Support for YUM

Note: This module makes heavy use of the **repoquery** utility, from the yum-utils package. This package will be installed as a dependency if salt is installed via EPEL. However, if salt has been installed using pip, or a host is being managed using salt-ssh, then as of version 2014.7.0 yum-utils will be installed automatically to satisfy this dependency.

---

salt.modules.yumpkg.**check_db**(**names*, ***kwargs*)
New in version 0.17.0.

Returns a dict containing the following information for each specified package:

1. A key `found`, which will be a boolean value denoting if a match was found in the package database.
2. If `found` is `False`, then a second key called `suggestions` will be present, which will contain a list of possible matches.

The `fromrepo`, `enablerepo` and `disablerepo` arguments are supported, as used in pkg states, and the `disableexcludes` option is also supported.

New in version 2014.7.0: Support for the `disableexcludes` option

CLI Examples:

```
salt '*' pkg.check_db <package1> <package2> <package3>
salt '*' pkg.check_db <package1> <package2> <package3> fromrepo=epel-testing
salt '*' pkg.check_db <package1> <package2> <package3> disableexcludes=main
```

salt.modules.yumpkg.**clean_metadata**(***kwargs*)
New in version 2014.1.0.

Cleans local yum metadata. Functionally identical to `refresh_db()`.

CLI Example:

```
salt '*' pkg.clean_metadata
```

salt.modules.yumpkg.**del_repo**(*repo*, *basedir=None*, ***kwargs*)
Delete a repo from <basedir> (default basedir: all dirs in *reposdir* yum option).

If the .repo file that the repo exists in does not contain any other repo configuration, the file itself will be deleted.

CLI Examples:

```
salt '*' pkg.del_repo myrepo
salt '*' pkg.del_repo myrepo basedir=/path/to/dir
salt '*' pkg.del_repo myrepo basedir=/path/to/dir,/path/to/another/dir
```

salt.modules.yumpkg.**diff**(**paths*)

Return a formatted diff between current files and original in a package. NOTE: this function includes all files (configuration and not), but does not work on binary content.

Parameters **path** -- Full path to the installed file

Returns Difference string or raises and exception if examined file is binary.

CLI example:

```
salt '*' pkg.diff /etc/apache2/httpd.conf /etc/sudoers
```

salt.modules.yumpkg.**download**(**packages*)

New in version 2015.5.0.

Download packages to the local disk. Requires `yumdownloader` from `yum-utils` package.

Note: `yum-utils` will already be installed on the minion if the package was installed from the Fedora / EPEL repositories.

CLI example:

```
salt '*' pkg.download httpd
salt '*' pkg.download httpd postfix
```

salt.modules.yumpkg.**expand_repo_def**(*repokwargs*)

Take a repository definition and expand it to the full pkg repository dict that can be used for comparison. This is a helper function to make certain repo managers sane for comparison in the pkgrepo states.

There is no use to calling this function via the CLI.

salt.modules.yumpkg.**file_dict**(**packages*)

New in version 2014.1.0.

List the files that belong to a package, grouped by package. Not specifying any packages will return a list of *every* file on the system's rpm database (not generally recommended).

CLI Examples:

```
salt '*' pkg.file_list httpd
salt '*' pkg.file_list httpd postfix
salt '*' pkg.file_list
```

salt.modules.yumpkg.**file_list**(**packages*)

New in version 2014.1.0.

List the files that belong to a package. Not specifying any packages will return a list of *every* file on the system's rpm database (not generally recommended).

CLI Examples:

```
salt '*' pkg.file_list httpd
salt '*' pkg.file_list httpd postfix
salt '*' pkg.file_list
```

salt.modules.yumpkg.**get_locked_packages**(*pattern=None*, *full=True*)

Get packages that are currently locked `yum -q versionlock list`.

CLI Example:

```
salt '*' pkg.get_locked_packages
```

salt.modules.yumpkg.**get_repo**(*repo*, *basedir=None*, ***kwargs*)
Display a repo from <basedir> (default basedir: all dirs in *reposdir* yum option).

CLI Examples:

```
salt '*' pkg.get_repo myrepo
salt '*' pkg.get_repo myrepo basedir=/path/to/dir
salt '*' pkg.get_repo myrepo basedir=/path/to/dir,/path/to/another/dir
```

salt.modules.yumpkg.**group_diff**(*name*)
New in version 2014.1.0.

Lists packages belonging to a certain group, and which are installed

CLI Example:

```
salt '*' pkg.group_diff 'Perl Support'
```

salt.modules.yumpkg.**group_info**(*name*)
New in version 2014.1.0.

Lists packages belonging to a certain group

CLI Example:

```
salt '*' pkg.group_info 'Perl Support'
```

salt.modules.yumpkg.**group_install**(*name*, *skip=()*, *include=()*, ***kwargs*)
New in version 2014.1.0.

Install the passed package group(s). This is basically a wrapper around pkg.install, which performs package group resolution for the user. This function is currently considered experimental, and should be expected to undergo changes.

name Package group to install. To install more than one group, either use a comma-separated list or pass the value as a python list.

CLI Examples:

```
salt '*' pkg.group_install 'Group 1'
salt '*' pkg.group_install 'Group 1,Group 2'
salt '*' pkg.group_install '["Group 1", "Group 2"]'
```

skip The name(s), in a list, of any packages that would normally be installed by the package group (``default'' packages), which should not be installed. Can be passed either as a comma-separated list or a python list.

CLI Examples:

```
salt '*' pkg.group_install 'My Group' skip='foo,bar'
salt '*' pkg.group_install 'My Group' skip='["foo", "bar"]'
```

include The name(s), in a list, of any packages which are included in a group, which would not normally be installed (``optional'' packages). Note that this will not enforce group membership; if you include packages which are not members of the specified groups, they will still be installed. Can be passed either as a comma-separated list or a python list.

CLI Examples:

```
salt '*' pkg.group_install 'My Group' include='foo,bar'
salt '*' pkg.group_install 'My Group' include='["foo", "bar"]'
```

Note: Because this is essentially a wrapper around pkg.install, any argument which can be passed to pkg.install may also be included here, and it will be passed along wholesale.

salt.modules.yumpkg.**group_list**()

New in version 2014.1.0.

Lists all groups known by yum on this system

CLI Example:

```
salt '*' pkg.group_list
```

salt.modules.yumpkg.**hold**(*name=None*, *pkgs=None*, *sources=None*, ***kwargs*)

New in version 2014.7.0.

Hold packages with `yum -q versionlock`.

name The name of the package to be deleted.

Multiple Package Options:

pkgs A list of packages to hold. Must be passed as a python list. The `name` parameter will be ignored if this option is passed.

Returns a dict containing the changes.

CLI Example:

```
salt '*' pkg.hold <package name>
salt '*' pkg.hold pkgs='["foo", "bar"]'
```

salt.modules.yumpkg.**install**(*name=None*, *refresh=False*, *fromrepo=None*, *skip_verify=False*, *pkgs=None*, *sources=None*, *reinstall=False*, *normalize=True*, ***kwargs*)

Install the passed package(s), add refresh=True to clean the yum database before package is installed.

name The name of the package to be installed. Note that this parameter is ignored if either ``pkgs'' or ``sources'' is passed. Additionally, please note that this option can only be used to install packages from a software repository. To install a package file manually, use the ``sources'' option.

32-bit packages can be installed on 64-bit systems by appending the architecture designation (`.i686`, `.i586`, etc.) to the end of the package name.

CLI Example:

```
salt '*' pkg.install <package name>
```

refresh Whether or not to update the yum database before executing.

reinstall Specifying reinstall=True will use `yum reinstall` rather than `yum install` for requested packages that are already installed.

If a version is specified with the requested package, then `yum reinstall` will only be used if the installed version matches the requested version.

Works with sources when the package header of the source can be matched to the name and version of an installed package.

New in version 2014.7.0.

skip_verify Skip the GPG verification check (e.g., `--nogpgcheck`)

version Install a specific version of the package, e.g. 1.2.3-4.el5. Ignored if ``pkgs'' or ``sources'' is passed.

Repository Options:

fromrepo Specify a package repository (or repositories) from which to install. (e.g., `yum --disablerepo='*' --enablerepo='somerepo'`)

enablerepo (ignored if `fromrepo` is specified) Specify a disabled package repository (or repositories) to enable. (e.g., `yum --enablerepo='somerepo'`)

disablerepo (ignored if `fromrepo` is specified) Specify an enabled package repository (or repositories) to disable. (e.g., `yum --disablerepo='somerepo'`)

disableexcludes Disable exclude from main, for a repo or for everything. (e.g., `yum --disableexcludes='main'`)

New in version 2014.7.0.

Multiple Package Installation Options:

pkgs A list of packages to install from a software repository. Must be passed as a python list. A specific version number can be specified by using a single-element dict representing the package and its version.

CLI Examples:

```
salt '*' pkg.install pkgs='["foo", "bar"]'
salt '*' pkg.install pkgs='["foo", {"bar": "1.2.3-4.el5"}]'
```

sources A list of RPM packages to install. Must be passed as a list of dicts, with the keys being package names, and the values being the source URI or local path to the package.

CLI Example:

```
salt '*' pkg.install sources='[{"foo": "salt://foo.rpm"}, {"bar": "salt://bar.rpm"}]'
```

normalize Normalize the package name by removing the architecture. Default is True. This is useful for poorly created packages which might include the architecture as an actual part of the name such as kernel modules which match a specific kernel version.

New in version 2014.7.0.

Example:

```
salt -G role:nsd pkg.install gpfs.gplbin-2.6.32-279.31.1.el6.x86_64 normalize=False
```

Returns a dict containing the new package names and versions:

```
{'<package>': {'old': '<old-version>',
               'new': '<new-version>'}}
```

`salt.modules.yumpkg.`**`latest_version`**(**names*, ***kwargs*)

Return the latest version of the named package available for upgrade or installation. If more than one package name is specified, a dict of name/version pairs is returned.

If the latest version of a given package is already installed, an empty string will be returned for that package.

A specific repo can be requested using the `fromrepo` keyword argument, and the `disableexcludes` option is also supported.

New in version 2014.7.0: Support for the `disableexcludes` option

CLI Example:

```
salt '*' pkg.latest_version <package name>
salt '*' pkg.latest_version <package name> fromrepo=epel-testing
```

```
    salt '*' pkg.latest_version <package name> disableexcludes=main
    salt '*' pkg.latest_version <package1> <package2> <package3> ...
```

salt.modules.yumpkg.**list_pkgs**(*versions_as_list=False*, ***kwargs*)

List the packages currently installed in a dict:

```
    {'<package_name>': '<version>'}
```

CLI Example:

```
    salt '*' pkg.list_pkgs
```

salt.modules.yumpkg.**list_repo_pkgs**(**args*, ***kwargs*)

New in version 2014.1.0.

Changed in version 2014.7.0: All available versions of each package are now returned. This required a slight modification to the structure of the return dict. The return data shown below reflects the updated return dict structure.

Returns all available packages. Optionally, package names (and name globs) can be passed and the results will be filtered to packages matching those names. This is recommended as it speeds up the function considerably.

This function can be helpful in discovering the version or repo to specify in a `pkg.installed` state.

The return data is a dictionary of repo names, with each repo containing a dictionary in which the keys are package names, and the values are a list of version numbers. Here is an example of the return data:

```
    {
        'base': {
            'bash': ['4.1.2-15.el6_4'],
            'kernel': ['2.6.32-431.el6']
        },
        'updates': {
            'bash': ['4.1.2-15.el6_5.2', '4.1.2-15.el6_5.1'],
            'kernel': ['2.6.32-431.29.2.el6',
                       '2.6.32-431.23.3.el6',
                       '2.6.32-431.20.5.el6',
                       '2.6.32-431.20.3.el6',
                       '2.6.32-431.17.1.el6',
                       '2.6.32-431.11.2.el6',
                       '2.6.32-431.5.1.el6',
                       '2.6.32-431.3.1.el6',
                       '2.6.32-431.1.2.0.1.el6']
        }
    }
```

fromrepo [None] Only include results from the specified repo(s). Multiple repos can be specified, comma-separated.

CLI Example:

```
    salt '*' pkg.list_repo_pkgs
    salt '*' pkg.list_repo_pkgs foo bar baz
    salt '*' pkg.list_repo_pkgs 'samba4*' fromrepo=base,updates
```

salt.modules.yumpkg.**list_repos**(*basedir=None*)

Lists all repos in <basedir> (default: all dirs in *reposdir* yum option).

CLI Example:

```
    salt '*' pkg.list_repos
    salt '*' pkg.list_repos basedir=/path/to/dir
    salt '*' pkg.list_repos basedir=/path/to/dir,/path/to/another/dir
```

salt.modules.yumpkg.**list_upgrades**(*refresh=True*, ***kwargs*)

Check whether or not an upgrade is available for all packages

The `fromrepo`, `enablerepo`, and `disablerepo` arguments are supported, as used in pkg states, and the `disableexcludes` option is also supported.

New in version 2014.7.0: Support for the `disableexcludes` option

CLI Example:

```
    salt '*' pkg.list_upgrades
```

salt.modules.yumpkg.**mod_repo**(*repo*, *basedir=None*, ***kwargs*)

Modify one or more values for a repo. If the repo does not exist, it will be created, so long as the following values are specified:

repo name by which the yum refers to the repo

name a human-readable name for the repo

baseurl the URL for yum to reference

mirrorlist the URL for yum to reference

Key/Value pairs may also be removed from a repo's configuration by setting a key to a blank value. Bear in mind that a name cannot be deleted, and a baseurl can only be deleted if a mirrorlist is specified (or vice versa).

CLI Examples:

```
    salt '*' pkg.mod_repo reponame enabled=1 gpgcheck=1
    salt '*' pkg.mod_repo reponame basedir=/path/to/dir enabled=1
    salt '*' pkg.mod_repo reponame baseurl= mirrorlist=http://host.com/
```

salt.modules.yumpkg.**modified**(**packages*, ***flags*)

List the modified files that belong to a package. Not specifying any packages will return a list of _all_ modified files on the system's RPM database.

New in version 2015.5.0.

Filtering by flags (True or False):

size Include only files where size changed.

mode Include only files which file's mode has been changed.

checksum Include only files which MD5 checksum has been changed.

device Include only files which major and minor numbers has been changed.

symlink Include only files which are symbolic link contents.

owner Include only files where owner has been changed.

group Include only files where group has been changed.

time Include only files where modification time of the file has been changed.

capabilities Include only files where capabilities differ or not. Note: supported only on newer RPM versions.

CLI Examples:

```
    salt '*' pkg.modified
    salt '*' pkg.modified httpd
    salt '*' pkg.modified httpd postfix
    salt '*' pkg.modified httpd owner=True group=False
```

salt.modules.yumpkg.**normalize_name**(*name*)

Strips the architecture from the specified package name, if necessary. Circumstances where this would be done include:

- If the arch is 32 bit and the package name ends in a 32-bit arch.
- If the arch matches the OS arch, or is `noarch`.

CLI Example:

```
    salt '*' pkg.normalize_name zsh.x86_64
```

salt.modules.yumpkg.**owner**(**paths*)

New in version 2014.7.0.

Return the name of the package that owns the file. Multiple file paths can be passed. Like `pkg.version <salt.modules.yumpkg.version`, if a single path is passed, a string will be returned, and if multiple paths are passed, a dictionary of file/package name pairs will be returned.

If the file is not owned by a package, or is not present on the minion, then an empty string will be returned for that path.

CLI Examples:

```
    salt '*' pkg.owner /usr/bin/apachectl
    salt '*' pkg.owner /usr/bin/apachectl /etc/httpd/conf/httpd.conf
```

salt.modules.yumpkg.**purge**(*name=None*, *pkgs=None*, ***kwargs*)

Package purges are not supported by yum, this function is identical to `pkg.remove`.

name The name of the package to be deleted.

Multiple Package Options:

pkgs A list of packages to delete. Must be passed as a python list. The `name` parameter will be ignored if this option is passed.

New in version 0.16.0.

Returns a dict containing the changes.

CLI Example:

```
    salt '*' pkg.purge <package name>
    salt '*' pkg.purge <package1>,<package2>,<package3>
    salt '*' pkg.purge pkgs='["foo", "bar"]'
```

salt.modules.yumpkg.**refresh_db**(*branch_arg*, *repo_arg*, *exclude_arg*)

Check the yum repos for updated packages

Returns:

- `True`: Updates are available
- `False`: An error occurred
- `None`: No updates are available

CLI Example:

```
salt '*' pkg.refresh_db
```

salt.modules.yumpkg.**remove**(*name=None, pkgs=None, **kwargs*)

Remove packages with `yum -q -y remove`.

name The name of the package to be deleted.

Multiple Package Options:

pkgs A list of packages to delete. Must be passed as a python list. The `name` parameter will be ignored if this option is passed.

New in version 0.16.0.

Returns a dict containing the changes.

CLI Example:

```
salt '*' pkg.remove <package name>
salt '*' pkg.remove <package1>,<package2>,<package3>
salt '*' pkg.remove pkgs='["foo", "bar"]'
```

salt.modules.yumpkg.**unhold**(*name=None, pkgs=None, sources=None, **kwargs*)

New in version 2014.7.0.

Hold packages with `yum -q versionlock`.

name The name of the package to be deleted.

Multiple Package Options:

pkgs A list of packages to unhold. Must be passed as a python list. The `name` parameter will be ignored if this option is passed.

Returns a dict containing the changes.

CLI Example:

```
salt '*' pkg.unhold <package name>
salt '*' pkg.unhold pkgs='["foo", "bar"]'
```

salt.modules.yumpkg.**upgrade**(*refresh=True, fromrepo=None, skip_verify=False, **kwargs*)

Run a full system upgrade, a yum upgrade

Changed in version 2014.7.0.

Return a dict containing the new package names and versions:

```
{'<package>': {'old': '<old-version>',
               'new': '<new-version>'}}
```

CLI Example:

```
salt '*' pkg.upgrade
```

Repository Options:

fromrepo Specify a package repository (or repositories) from which to install. (e.g., `yum --disablerepo='*' --enablerepo='somerepo'`)

enablerepo (ignored if `fromrepo` is specified) Specify a disabled package repository (or repositories) to enable. (e.g., `yum --enablerepo='somerepo'`)

disablerepo (ignored if `fromrepo` is specified) Specify an enabled package repository (or repositories) to disable. (e.g., `yum --disablerepo='somerepo'`)

disableexcludes Disable exclude from main, for a repo or for everything. (e.g., `yum --disableexcludes='main'`)

New in version 2014.7.0.

`salt.modules.yumpkg.`**`upgrade_available`**(*name*)
Check whether or not an upgrade is available for a given package

CLI Example:

```
salt '*' pkg.upgrade_available <package name>
```

`salt.modules.yumpkg.`**`verify`**(**names*, ***kwargs*)
New in version 2014.1.0.

Runs an rpm -Va on a system, and returns the results in a dict

Files with an attribute of config, doc, ghost, license or readme in the package header can be ignored using the `ignore_types` keyword argument

CLI Example:

```
salt '*' pkg.verify
salt '*' pkg.verify httpd
salt '*' pkg.verify 'httpd postfix'
salt '*' pkg.verify 'httpd postfix' ignore_types=['config','doc']
```

`salt.modules.yumpkg.`**`version`**(**names*, ***kwargs*)
Returns a string representing the package version or an empty string if not installed. If more than one package name is specified, a dict of name/version pairs is returned.

CLI Example:

```
salt '*' pkg.version <package name>
salt '*' pkg.version <package1> <package2> <package3> ...
```

22.16.277 salt.modules.zcbuildout

Management of zc.buildout

New in version 2014.1.0.

This module is inspired by minitage's buildout maker

Note: The zc.buildout integration is still in beta; the API is subject to change

General notes

You have those following methods:

- upgrade_bootstrap
- bootstrap
- run_buildout
- buildout

`salt.modules.zcbuildout.`**`bootstrap`**(**a*, ***kw*)
Run the buildout bootstrap dance (python bootstrap.py).

directory directory to execute in

config alternative buildout configuration file to use

runas User used to run buildout as

env environment variables to set when running

buildout_ver force a specific buildout version (1 | 2)

test_release buildout accept test release

offline are we executing buildout in offline mode

distribute Forcing use of distribute

new_st Forcing use of setuptools >= 0.7

python path to a python executable to use in place of default (salt one)

onlyif Only execute cmd if statement on the host return 0

unless Do not execute cmd if statement on the host return 0

use_vt Use the new salt VT to stream output [experimental]

CLI Example:

```
salt '*' buildout.bootstrap /srv/mybuildout
```

salt.modules.zcbuildout.**buildout**(*a, **kw)

Run buildout in a directory.

directory directory to execute in

config buildout config to use

parts specific buildout parts to run

runas user used to run buildout as

env environment variables to set when running

buildout_ver force a specific buildout version (1 | 2)

test_release buildout accept test release

new_st Forcing use of setuptools >= 0.7

distribute use distribute over setuptools if possible

offline does buildout run offline

python python to use

debug run buildout with -D debug flag

onlyif Only execute cmd if statement on the host return 0

unless Do not execute cmd if statement on the host return 0

newest run buildout in newest mode

verbose run buildout in verbose mode (-vvvvv)

use_vt Use the new salt VT to stream output [experimental]

CLI Example:

```
salt '*' buildout.buildout /srv/mybuildout
```

salt.modules.zcbuildout.**run_buildout**(**a*, ***kw*)

Run a buildout in a directory.

directory directory to execute in

config alternative buildout configuration file to use

offline are we executing buildout in offline mode

runas user used to run buildout as

env environment variables to set when running

onlyif Only execute cmd if statement on the host return 0

unless Do not execute cmd if statement on the host return 0

newest run buildout in newest mode

force run buildout unconditionally

verbose run buildout in verbose mode (-vvvvv)

use_vt Use the new salt VT to stream output [experimental]

CLI Example:

```
salt '*' buildout.run_buildout /srv/mybuildout
```

salt.modules.zcbuildout.**upgrade_bootstrap**(**a*, ***kw*)

Upgrade current bootstrap.py with the last released one.

Indeed, when we first run a buildout, a common source of problem is to have a locally stale bootstrap, we just try to grab a new copy

directory directory to execute in

offline are we executing buildout in offline mode

buildout_ver forcing to use a specific buildout version (1 | 2)

onlyif Only execute cmd if statement on the host return 0

unless Do not execute cmd if statement on the host return 0

CLI Example:

```
salt '*' buildout.upgrade_bootstrap /srv/mybuildout
```

22.16.278 salt.modules.zfs

Salt interface to ZFS commands

codeauthor Nitin Madhok <nmadhok@clemson.edu>

salt.modules.zfs.**create**(*name*, ***kwargs*)

New in version 2015.5.0.

Create a ZFS File System.

CLI Example:

```
salt '*' zfs.create myzpool/mydataset [create_parent=True|False]
```

Note: ZFS properties can be specified at the time of creation of the filesystem by passing an additional argument called ``properties'' and specifying the properties with their respective values in the form of a python dictionary:

```
properties="{'property1': 'value1', 'property2': 'value2'}"
```

Example:

```
salt '*' zfs.create myzpool/mydataset properties="{'mountpoint': '/export/zfs', 'sharenfs': 'on'
```

`salt.modules.zfs.`**`destroy`**(*name*, ***kwargs*)

New in version 2015.5.0.

Destroy a ZFS File System.

CLI Example:

```
salt '*' zfs.destroy myzpool/mydataset [force=True|False]
```

`salt.modules.zfs.`**`exists`**(*name*)

New in version 2015.5.0.

Check if a ZFS filesystem or volume or snapshot exists.

CLI Example:

```
salt '*' zfs.exists myzpool/mydataset
```

`salt.modules.zfs.`**`list_`**(*name=''*, ***kwargs*)

New in version 2015.5.0.

Return a list of all datasets or a specified dataset on the system and the values of their used, available, referenced, and mountpoint properties.

Note: Information about the dataset and all of it's descendent datasets can be displayed by passing `recursive=True` on the CLI.

CLI Example:

```
salt '*' zfs.list [recursive=True|False]
salt '*' zfs.list /myzpool/mydataset [recursive=True|False]
```

Note: Dataset property value output can be customized by passing an additional argument called ``properties'' in the form of a python list:

```
properties="[property1, property2, property3]"
```

Example:

```
salt '*' zfs.list /myzpool/mydataset properties="[name, sharenfs, mountpoint]"
```

`salt.modules.zfs.`**`rename`**(*name*, *new_name*)

New in version 2015.5.0.

Rename or Relocate a ZFS File System.

CLI Example:

```
salt '*' zfs.rename myzpool/mydataset myzpool/renameddataset
```

22.16.279 salt.modules.zk_concurrency

Concurrency controls in zookeeper

This module allows you to acquire and release a slot. This is primarily useful for ensureing that no more than N hosts take a specific action at once. This can also be used to coordinate between masters.

`salt.modules.zk_concurrency.`**`lock`**(*path*, *zk_hosts*, *identifier=None*, *max_concurrency=1*, *timeout=None*, *ephemeral_lease=False*, *force=False*)

Get lock (with optional timeout)

path The path in zookeeper where the lock is

zk_hosts zookeeper connect string

identifier Name to identify this minion

max_concurrency Maximum number of lock holders

timeout timeout to wait for the lock. A None timeout will block forever

ephemeral_lease Whether the locks in zookeper should be ephemeral

force Forcibly acquire the lock regardless of available slots

Example:

... code-block: bash

salt minion zk_concurrency.lock /lock/path host1:1234,host2:1234

`salt.modules.zk_concurrency.`**`lock_holders`**(*path*, *zk_hosts*, *identifier=None*, *max_concurrency=1*, *timeout=None*, *ephemeral_lease=False*)

Return an un-ordered list of lock holders

path The path in zookeeper where the lock is

zk_hosts zookeeper connect string

identifier Name to identify this minion

max_concurrency Maximum number of lock holders

timeout timeout to wait for the lock. A None timeout will block forever

ephemeral_lease Whether the locks in zookeper should be ephemeral

Example:

... code-block: bash

salt minion zk_concurrency.lock_holders /lock/path host1:1234,host2:1234

`salt.modules.zk_concurrency.`**`unlock`**(*path*, *zk_hosts=None*, *identifier=None*, *max_concurrency=1*, *ephemeral_lease=False*)

Remove lease from semaphore

path The path in zookeeper where the lock is

zk_hosts zookeeper connect string

identifier Name to identify this minion

max_concurrency Maximum number of lock holders

timeout timeout to wait for the lock. A None timeout will block forever

ephemeral_lease Whether the locks in zookeper should be ephemeral

Example:

... code-block: bash

salt minion zk_concurrency.unlock /lock/path host1:1234,host2:1234

22.16.280 salt.modules.znc

znc - An advanced IRC bouncer

New in version 2014.7.0.

Provides an interface to basic ZNC functionality

`salt.modules.znc.`**`buildmod`**`(*modules)`

Build module using znc-buildmod

CLI Example:

```
salt '*' znc.buildmod module.cpp [...]
```

`salt.modules.znc.`**`dumpconf`**`()`

Write the active configuration state to config file

CLI Example:

```
salt '*' znc.dumpconf
```

`salt.modules.znc.`**`rehashconf`**`()`

Rehash the active configuration state from config file

CLI Example:

```
salt '*' znc.rehashconf
```

`salt.modules.znc.`**`version`**`()`

Return server version from znc --version

CLI Example:

```
salt '*' znc.version
```

22.16.281 salt.modules.zpool

Module for running ZFS zpool command

codeauthor Nitin Madhok <nmadhok@clemson.edu>

`salt.modules.zpool.`**`add`**`(pool_name, *vdevs)`

Add the specified vdev's to the given pool

CLI Example:

```
salt '*' zpool.add myzpool /path/to/vdev1 /path/to/vdev2 [...]
```

salt.modules.zpool.**create**(*pool_name*, **vdevs*, ***kwargs*)

New in version 2015.5.0.

Create a simple zpool, a mirrored zpool, a zpool having nested VDEVs, a hybrid zpool with cache, spare and log drives or a zpool with RAIDZ-1, RAIDZ-2 or RAIDZ-3

CLI Example:

```
salt '*' zpool.create myzpool /path/to/vdev1 [...] [force=True|False]
salt '*' zpool.create myzpool mirror /path/to/vdev1 /path/to/vdev2 [...] [force=True|False]
salt '*' zpool.create myzpool raidz1 /path/to/vdev1 /path/to/vdev2 raidz2 /path/to/vdev3 /path/t
salt '*' zpool.create myzpool mirror /path/to/vdev1 [...] mirror /path/to/vdev2 /path/to/vdev3 [
salt '*' zpool.create myhybridzpool mirror /tmp/file1 [...] log mirror /path/to/vdev1 [...] cach
```

Note: Zpool properties can be specified at the time of creation of the pool by passing an additional argument called ``properties'' and specifying the properties with their respective values in the form of a python dictionary:

```
properties="{'property1': 'value1', 'property2': 'value2'}"
```

Example:

```
salt '*' zpool.create myzpool /path/to/vdev1 [...] properties="{'property1': 'value1', 'property
```

---

salt.modules.zpool.**create_file_vdev**(*size*, **vdevs*)

Creates file based `virtual devices` for a zpool

`*vdevs` is a list of full paths for mkfile to create

CLI Example:

```
salt '*' zpool.create_file_vdev 7g /path/to/vdev1 [/path/to/vdev2] [...]
```

Note: Depending on file size, the above command may take a while to return.

---

salt.modules.zpool.**destroy**(*pool_name*)

Destroys a storage pool

CLI Example:

```
salt '*' zpool.destroy myzpool
```

salt.modules.zpool.**exists**(*pool_name*)

Check if a ZFS storage pool is active

CLI Example:

```
salt '*' zpool.exists myzpool
```

salt.modules.zpool.**export**(**pools*, ***kwargs*)

New in version 2015.5.0.

Export storage pools

CLI Example:

```
    salt '*' zpool.export myzpool ... [force=True|False]
    salt '*' zpool.export myzpool2 myzpool2 ... [force=True|False]
```

salt.modules.zpool.**import_**(*pool_name=''*, *new_name=''*, ***kwargs*)
New in version 2015.5.0.

Import storage pools or list pools available for import

CLI Example:

```
    salt '*' zpool.import [all=True|False]
    salt '*' zpool.import myzpool [mynewzpool] [force=True|False]
```

salt.modules.zpool.**iostat**(*name=''*)
Display I/O statistics for the given pools

CLI Example:

```
    salt '*' zpool.iostat myzpool
```

salt.modules.zpool.**list_**()
New in version 2015.5.0.

Return a list of all pools in the system with health status and space usage

CLI Example:

```
    salt '*' zpool.list
```

salt.modules.zpool.**offline**(*pool_name*, **vdevs*, ***kwargs*)
New in version 2015.5.0.

Ensure that the specified devices are offline

> **Warning:** By default, the OFFLINE state is persistent. The device remains offline when the system is rebooted. To temporarily take a device offline, use `temporary=True`.

CLI Example:

```
    salt '*' zpool.offline myzpool /path/to/vdev1 [...] [temporary=True|False]
```

salt.modules.zpool.**online**(*pool_name*, **vdevs*, ***kwargs*)
New in version 2015.5.0.

Ensure that the specified devices are online

CLI Example:

```
    salt '*' zpool.online myzpool /path/to/vdev1 [...]
```

salt.modules.zpool.**replace**(*pool_name*, *old*, *new*)
Replaces old device with new device.

CLI Example:

```
    salt '*' zpool.replace myzpool /path/to/vdev1 /path/to/vdev2
```

salt.modules.zpool.**scrub**(*pool_name=None*)
Begin a scrub

CLI Example:

```
salt '*' zpool.scrub myzpool
```

salt.modules.zpool.**status**(*name=''*)
Return the status of the named zpool

CLI Example:

```
salt '*' zpool.status myzpool
```

salt.modules.zpool.**zpool_list**()
Deprecated since version 2014.7.0: Use `list_()` instead.

Return a list of all pools in the system with health status and space usage

CLI Example:

```
salt '*' zpool.zpool_list
```

22.16.282 salt.modules.zypper

Package support for openSUSE via the zypper package manager

depends

- `zypp` Python module. Install with `zypper install python-zypp`

salt.modules.zypper.**add_lock**(*name=None*, *pkgs=None*, ***kwargs*)
Add a package lock. Specify packages to lock by exact name.

CLI Example:

```
salt '*' pkg.add_lock <package name>
salt '*' pkg.add_lock <package1>,<package2>,<package3>
salt '*' pkg.add_lock pkgs='["foo", "bar"]'
```

salt.modules.zypper.**clean_locks**()
Remove unused locks that do not currently (with regard to repositories used) lock any package.

CLI Example:

```
salt '*' pkg.clean_locks
```

salt.modules.zypper.**del_repo**(*repo*)
Delete a repo.

CLI Examples:

```
salt '*' pkg.del_repo alias
```

salt.modules.zypper.**diff**(**paths*)
Return a formatted diff between current files and original in a package. NOTE: this function includes all files (configuration and not), but does not work on binary content.

Parameters **path** -- Full path to the installed file

Returns Difference string or raises and exception if examined file is binary.

CLI example:

```
salt '*' pkg.diff /etc/apache2/httpd.conf /etc/sudoers
```

salt.modules.zypper.**download**(**packages*)

Download packages to the local disk.

CLI example:

```
salt '*' pkg.download httpd
salt '*' pkg.download httpd postfix
```

salt.modules.zypper.**file_dict**(**packages*)

List the files that belong to a package, grouped by package. Not specifying any packages will return a list of *every* file on the system's rpm database (not generally recommended).

CLI Examples:

```
salt '*' pkg.file_list httpd
salt '*' pkg.file_list httpd postfix
salt '*' pkg.file_list
```

salt.modules.zypper.**file_list**(**packages*)

List the files that belong to a package. Not specifying any packages will return a list of *every* file on the system's rpm database (not generally recommended).

CLI Examples:

```
salt '*' pkg.file_list httpd
salt '*' pkg.file_list httpd postfix
salt '*' pkg.file_list
```

salt.modules.zypper.**get_repo**(*repo*, ***kwargs*)

Display a repo.

CLI Example:

```
salt '*' pkg.get_repo alias
```

salt.modules.zypper.**install**(*name=None*, *refresh=False*, *fromrepo=None*, *pkgs=None*, *sources=None*, ***kwargs*)

Install the passed package(s), add refresh=True to run `zypper refresh' before package is installed.

name The name of the package to be installed. Note that this parameter is ignored if either ``pkgs'' or ``sources'' is passed. Additionally, please note that this option can only be used to install packages from a software repository. To install a package file manually, use the ``sources'' option.

CLI Example:

```
salt '*' pkg.install <package name>
```

refresh Whether or not to refresh the package database before installing.

fromrepo Specify a package repository to install from.

version Can be either a version number, or the combination of a comparison operator (<, >, <=, >=, =) and a version number (ex. `>1.2.3-4'). This parameter is ignored if ``pkgs'' or ``sources'' is passed.

Multiple Package Installation Options:

pkgs A list of packages to install from a software repository. Must be passed as a python list. A specific version number can be specified by using a single-element dict representing the package and its version. As with the `version` parameter above, comparison operators can be used to target a specific version of a package.

CLI Examples:

```
        salt '*' pkg.install pkgs='["foo", "bar"]'
        salt '*' pkg.install pkgs='["foo", {"bar": "1.2.3-4"}]'
        salt '*' pkg.install pkgs='["foo", {"bar": "<1.2.3-4"}]'
```

sources A list of RPM packages to install. Must be passed as a list of dicts, with the keys being package names, and the values being the source URI or local path to the package.

CLI Example:

```
        salt '*' pkg.install sources='[{"foo": "salt://foo.rpm"},{"bar": "salt://bar.rpm"}]'
```

Returns a dict containing the new package names and versions:

```
    {'<package>': {'old': '<old-version>',
                   'new': '<new-version>'}}
```

salt.modules.zypper.**latest_version**(**names*, ***kwargs*)

Return the latest version of the named package available for upgrade or installation. If more than one package name is specified, a dict of name/version pairs is returned.

If the latest version of a given package is already installed, an empty string will be returned for that package.

CLI Example:

```
    salt '*' pkg.latest_version <package name>
    salt '*' pkg.latest_version <package1> <package2> <package3> ...
```

salt.modules.zypper.**list_installed_patterns**()

List installed patterns on the system.

CLI Examples:

```
    salt '*' pkg.list_installed_patterns
```

salt.modules.zypper.**list_locks**()

List current package locks.

Return a dict containing the locked package with attributes:

```
    {'<package>': {'case_sensitive': '<case_sensitive>',
                   'match_type': '<match_type>'
                   'type': '<type>'}}
```

CLI Example:

```
    salt '*' pkg.list_locks
```

salt.modules.zypper.**list_patterns**()

List all known patterns from available repos.

CLI Examples:

```
    salt '*' pkg.list_patterns
```

salt.modules.zypper.**list_pkgs**(*versions_as_list=False*, ***kwargs*)

List the packages currently installed as a dict:

```
    {'<package_name>': '<version>'}
```

CLI Example:

```
salt '*' pkg.list_pkgs
```

salt.modules.zypper.**list_products**()
List all installed SUSE products.

CLI Examples:

```
salt '*' pkg.list_products
```

salt.modules.zypper.**list_repos**()
Lists all repos.

CLI Example:

```
salt '*' pkg.list_repos
```

salt.modules.zypper.**list_upgrades**(*refresh=True*)
List all available package upgrades on this system

CLI Example:

```
salt '*' pkg.list_upgrades
```

salt.modules.zypper.**mod_repo**(*repo*, ***kwargs*)
Modify one or more values for a repo. If the repo does not exist, it will be created, so long as the following values are specified:

repo or alias alias by which the zypper refers to the repo

url or mirrorlist the URL for zypper to reference

enabled enable or disable (True or False) repository, but do not remove if disabled.

refresh enable or disable (True or False) auto-refresh of the repository.

cache Enable or disable (True or False) RPM files caching.

gpgcheck Enable or disable (True or False) GOG check for this repository.

Key/Value pairs may also be removed from a repo's configuration by setting a key to a blank value. Bear in mind that a name cannot be deleted, and a url can only be deleted if a mirrorlist is specified (or vice versa).

CLI Examples:

```
salt '*' pkg.mod_repo alias alias=new_alias
salt '*' pkg.mod_repo alias url= mirrorlist=http://host.com/
```

salt.modules.zypper.**modified**(**packages*, ***flags*)
List the modified files that belong to a package. Not specifying any packages will return a list of _all_ modified files on the system's RPM database.

New in version 2015.5.0.

Filtering by flags (True or False):

size Include only files where size changed.

mode Include only files which file's mode has been changed.

checksum Include only files which MD5 checksum has been changed.

device Include only files which major and minor numbers has been changed.

symlink Include only files which are symbolic link contents.

owner Include only files where owner has been changed.

group Include only files where group has been changed.

time Include only files where modification time of the file has been changed.

capabilities Include only files where capabilities differ or not. Note: supported only on newer RPM versions.

CLI Examples:

```
salt '*' pkg.modified
salt '*' pkg.modified httpd
salt '*' pkg.modified httpd postfix
salt '*' pkg.modified httpd owner=True group=False
```

`salt.modules.zypper.`**`owner`**(**paths*)

Return the name of the package that owns the file. Multiple file paths can be passed. If a single path is passed, a string will be returned, and if multiple paths are passed, a dictionary of file/package name pairs will be returned.

If the file is not owned by a package, or is not present on the minion, then an empty string will be returned for that path.

CLI Examples:

```
salt '*' pkg.owner /usr/bin/apachectl
salt '*' pkg.owner /usr/bin/apachectl /etc/httpd/conf/httpd.conf
```

`salt.modules.zypper.`**`purge`**(*name=None*, *pkgs=None*, ***kwargs*)

Recursively remove a package and all dependencies which were installed with it, this will call a `zypper -n remove -u`

name The name of the package to be deleted.

Multiple Package Options:

pkgs A list of packages to delete. Must be passed as a python list. The `name` parameter will be ignored if this option is passed.

New in version 0.16.0.

Returns a dict containing the changes.

CLI Example:

```
salt '*' pkg.purge <package name>
salt '*' pkg.purge <package1>,<package2>,<package3>
salt '*' pkg.purge pkgs='["foo", "bar"]'
```

`salt.modules.zypper.`**`refresh_db`**()

Just run a `zypper refresh`, return a dict:

```
{'<database name>': Bool}
```

CLI Example:

```
salt '*' pkg.refresh_db
```

`salt.modules.zypper.`**`remove`**(*name=None*, *pkgs=None*, ***kwargs*)

Remove packages with `zypper -n remove`

name The name of the package to be deleted.

Multiple Package Options:

pkgs A list of packages to delete. Must be passed as a python list. The `name` parameter will be ignored if this option is passed.

New in version 0.16.0.

Returns a dict containing the changes.

CLI Example:

```
salt '*' pkg.remove <package name>
salt '*' pkg.remove <package1>,<package2>,<package3>
salt '*' pkg.remove pkgs='["foo", "bar"]'
```

`salt.modules.zypper.`**`remove_lock`**(*name=None, pkgs=None, **kwargs*)
Remove specified package lock.

CLI Example:

```
salt '*' pkg.remove_lock <package name>
salt '*' pkg.remove_lock <package1>,<package2>,<package3>
salt '*' pkg.remove_lock pkgs='["foo", "bar"]'
```

`salt.modules.zypper.`**`search`**(*criteria*)
List known packags, available to the system.

CLI Examples:

```
salt '*' pkg.search <criteria>
```

`salt.modules.zypper.`**`upgrade`**(*refresh=True*)
Run a full system upgrade, a zypper upgrade

Return a dict containing the new package names and versions:

```
{'<package>': {'old': '<old-version>',
               'new': '<new-version>'}}
```

CLI Example:

```
salt '*' pkg.upgrade
```

`salt.modules.zypper.`**`upgrade_available`**(*name*)
Check whether or not an upgrade is available for a given package

CLI Example:

```
salt '*' pkg.upgrade_available <package name>
```

`salt.modules.zypper.`**`verify`**(**names, **kwargs*)
Runs an rpm -Va on a system, and returns the results in a dict

Files with an attribute of config, doc, ghost, license or readme in the package header can be ignored using the `ignore_types` keyword argument

CLI Example:

```
salt '*' pkg.verify
salt '*' pkg.verify httpd
salt '*' pkg.verify 'httpd postfix'
salt '*' pkg.verify 'httpd postfix' ignore_types=['config','doc']
```

salt.modules.zypper.**version**(**names*, ***kwargs*)

Returns a string representing the package version or an empty string if not installed. If more than one package name is specified, a dict of name/version pairs is returned.

CLI Example:

```
salt '*' pkg.version <package name>
salt '*' pkg.version <package1> <package2> <package3> ...
```

22.17 Full list of netapi modules

22.17.1 rest_cherrypy

A REST API for Salt

New in version 2014.7.0.

depends

- CherryPy Python module. Versions 3.2.{2,3,4} are strongly

recommended due to a known SSL error introduced in version 3.2.5. The issue was reportedly resolved with CherryPy milestone 3.3, but the patch was committed for version 3.6.1.

optdepends

- ws4py Python module for websockets support.

configuration All authentication is done through Salt's *external auth* system which requires additional configuration not described here.

Example production-ready configuration; add to the Salt master config file and restart the `salt-master` and `salt-api` daemons:

```
rest_cherrypy:
  port: 8000
  ssl_crt: /etc/pki/tls/certs/localhost.crt
  ssl_key: /etc/pki/tls/certs/localhost.key
```

Using only a secure HTTPS connection is strongly recommended since Salt authentication credentials will be sent over the wire.

A self-signed certificate can be generated using the `create_self_signed_cert()` execution function. Running this function requires pyOpenSSL and the `salt-call` script is available in the `salt-minion` package.

```
salt-call --local tls.create_self_signed_cert
```

All available configuration options are detailed below. These settings configure the CherryPy HTTP server and do not apply when using an external server such as Apache or Nginx.

port **Required**

The port for the webserver to listen on.

host `[0.0.0.0]` The socket interface for the HTTP server to listen on.

debug `[False]` Starts the web server in development mode. It will reload itself when the underlying code is changed and will output more debugging info.

ssl_crt The path to a SSL certificate. (See below)

ssl_key The path to the private key for your SSL certificate. (See below)

disable_ssl A flag to disable SSL. Warning: your Salt authentication credentials will be sent in the clear!

webhook_disable_auth [False] The `Webhook` URL requires authentication by default but external services cannot always be configured to send authentication. See the Webhook documentation for suggestions on securing this interface.

webhook_url [/hook] Configure the URL endpoint for the `Webhook` entry point.

thread_pool [`100`] The number of worker threads to start up in the pool.

socket_queue_size [`30`] Specify the maximum number of HTTP connections to queue.

expire_responses [True] Whether to check for and kill HTTP responses that have exceeded the default timeout.

max_request_body_size [`1048576`] Maximum size for the HTTP request body.

collect_stats [False] Collect and report statistics about the CherryPy server

Reports are available via the `Stats` URL.

static A filesystem path to static HTML/JavaScript/CSS/image assets.

static_path [`/static`] The URL prefix to use when serving static assets out of the directory specified in the `static` setting.

app A filesystem path to an HTML file that will be served as a static file. This is useful for bootstrapping a single-page JavaScript app.

app_path [`/app`] The URL prefix to use for serving the HTML file specified in the `app` setting. This should be a simple name containing no slashes.

Any path information after the specified path is ignored; this is useful for apps that utilize the HTML5 history API.

root_prefix [`/`] A URL path to the main entry point for the application. This is useful for serving multiple applications from the same URL.

Authentication

Authentication is performed by passing a session token with each request. Tokens are generated via the `Login` URL.

The token may be sent in one of two ways:

- Include a custom header named `X-Auth-Token`.

  For example, using curl:

```
curl -sSk https://localhost:8000/login                -H 'Accept: application/x-yaml'
```

  Copy the `token` value from the output and include it in subsequent requests:

```
```

curl -sSk https://localhost:8000 -H `Accept: application/x-yaml' -H `X-Auth-Token: 697adbdc8fe971d09ae4c2a3add7248859c87079' -d client=local -d tgt='*' -d fun=test.ping

- Sent via a cookie. This option is a convenience for HTTP clients that automatically handle cookie support (such as browsers).

  For example, using curl:

```
# Write the cookie file:
curl -sSk https://localhost:8000/login             -c ~/cookies.txt             -H 'Accept: appl

# Read the cookie file:
curl -sSk https://localhost:8000             -b ~/cookies.txt             -H 'Accept: applicatio
```

See also:

You can bypass the session handling via the Run URL.

Usage

Commands are sent to a running Salt master via this module by sending HTTP requests to the URLs detailed below.

Content negotiation

This REST interface is flexible in what data formats it will accept as well as what formats it will return (e.g., JSON, YAML, x-www-form-urlencoded).

- Specify the format of data in the request body by including the `Content-Type` header.
- Specify the desired data format for the response body with the `Accept` header.

Data sent in POST and PUT requests must be in the format of a list of lowstate dictionaries. This allows multiple commands to be executed in a single HTTP request. The order of commands in the request corresponds to the return for each command in the response.

Lowstate, broadly, is a dictionary of values that are mapped to a function call. This pattern is used pervasively throughout Salt. The functions called from netapi modules are described in *Client Interfaces*.

The following example (in JSON format) causes Salt to execute two commands, a command sent to minions as well as a runner function on the master:

```
[{
    "client": "local",
    "tgt": "*",
    "fun": "test.fib",
    "arg": ["10"]
},
{
    "client": "runner",
    "fun": "jobs.lookup_jid",
    "jid": "20130603122505459265"
}]
```

---

x-www-form-urlencoded

Sending JSON or YAML in the request body is simple and most flexible, however sending data in urlencoded format is also supported with the caveats below. It is the default format for HTML forms, many JavaScript libraries, and the **`curl`** command.

For example, the equivalent to running `salt '*' test.ping` is sending `fun=test.ping&arg&client=local&tgt=*` in the HTTP request body.

Caveats:

- Only a single command may be sent per HTTP request.
- Repeating the `arg` parameter multiple times will cause those parameters to be combined into a single list.

  Note, some popular frameworks and languages (notably jQuery, PHP, and Ruby on Rails) will automatically append empty brackets onto repeated parameters. E.g., `arg=one`, `arg=two` will be sent as `arg[]=one`, `arg[]=two`. This is not supported; send JSON or YAML instead.

Deployment

The `rest_cherrypy` netapi module is a standard Python WSGI app. It can be deployed one of two ways.

`salt-api` using the CherryPy server

The default configuration is to run this module using **`salt-api`** to start the Python-based CherryPy server. This server is lightweight, multi-threaded, encrypted with SSL, and should be considered production-ready.

Using a WSGI-compliant web server

This module may be deployed on any WSGI-compliant server such as Apache with mod_wsgi or Nginx with FastCGI, to name just two (there are many).

Note, external WSGI servers handle URLs, paths, and SSL certs directly. The `rest_cherrypy` configuration options are ignored and the `salt-api` daemon does not need to be running at all. Remember Salt authentication credentials are sent in the clear unless SSL is being enforced!

An example Apache virtual host configuration:

```
<VirtualHost *:80>
    ServerName example.com
    ServerAlias *.example.com

    ServerAdmin webmaster@example.com

    LogLevel warn
    ErrorLog /var/www/example.com/logs/error.log
    CustomLog /var/www/example.com/logs/access.log combined

    DocumentRoot /var/www/example.com/htdocs

    WSGIScriptAlias / /path/to/salt/netapi/rest_cherrypy/wsgi.py
</VirtualHost>
```

REST URI Reference

- /
- /login
- /logout
- /minions
- /jobs
- /run
- /events
- /hook
- /keys
- /ws
- /stats

/

class salt.netapi.rest_cherrypy.app.**LowDataAdapter**

The primary entry point to Salt's REST API

GET()

An explanation of the API with links of where to go next

GET /

Request Headers

- **Accept** -- the desired response format.

Status Codes

- **200** -- success
- **401** -- authentication required
- **406** -- requested Content-Type not available

Example request:

```
curl -i localhost:8000
```

```
GET / HTTP/1.1
Host: localhost:8000
Accept: application/json
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json
```

POST

Mock out specified imports

This allows autodoc to do its thing without having oodles of req'd installed libs. This doesn't work with `import *` imports.

http://read-the-docs.readthedocs.org/en/latest/faq.html#i-get-import-errors-on-libraries-that-depend-on-c-modules

/login

class salt.netapi.rest_cherrypy.app.**Login**(**args*, ***kwargs*)

Log in to receive a session token

Authentication information.

GET()

Present the login interface

GET /login

An explanation of how to log in.

Status Codes

- **200** -- success
- **401** -- authentication required
- **406** -- requested Content-Type not available

Example request:

```
curl -i localhost:8000/login
```

```
GET /login HTTP/1.1
Host: localhost:8000
Accept: text/html
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: text/html
```

POST(***kwargs*)

Authenticate against Salt's eauth system

POST /login

Request Headers

- **X-Auth-Token** -- a session token from Login.
- **Accept** -- the desired response format.
- **Content-Type** -- the format of the request body.

Form Parameters

- **eauth** -- the eauth backend configured for the user
- **username** -- username
- **password** -- password

Status Codes

- **200** -- success
- **401** -- authentication required
- **406** -- requested Content-Type not available

Example request:

```
curl -si localhost:8000/login \
        -H "Accept: application/json" \
        -d username='saltuser' \
        -d password='saltpass' \
        -d eauth='pam'
```

```
POST / HTTP/1.1
Host: localhost:8000
Content-Length: 42
Content-Type: application/x-www-form-urlencoded
Accept: application/json

username=saltuser&password=saltpass&eauth=pam
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 206
X-Auth-Token: 6d1b722e
Set-Cookie: session_id=6d1b722e; expires=Sat, 17 Nov 2012 03:23:52 GMT; Path=/

{"return": {
    "token": "6d1b722e",
    "start": 1363805943.776223,
    "expire": 1363849143.776224,
    "user": "saltuser",
    "eauth": "pam",
    "perms": [
        "grains.*",
        "status.*",
        "sys.*",
        "test.*"
    ]
}}
```

/logout

class salt.netapi.rest_cherrypy.app.**Logout**

Class to remove or invalidate sessions

POST()

Destroy the currently active session and expire the session cookie

/minions

class salt.netapi.rest_cherrypy.app.**Minions**

Convenience URLs for working with minions

GET(*mid=None*)

A convenience URL for getting lists of minions or getting minion details

GET /minions/(*mid*)

Request Headers

- **X-Auth-Token** -- a session token from Login.
- **Accept** -- the desired response format.

Status Codes

- **200** -- success
- **401** -- authentication required
- **406** -- requested Content-Type not available

Example request:

```
curl -i localhost:8000/minions/ms-3
```

```
GET /minions/ms-3 HTTP/1.1
Host: localhost:8000
Accept: application/x-yaml
```

Example response:

```
HTTP/1.1 200 OK
Content-Length: 129005
Content-Type: application/x-yaml

return:
- ms-3:
    grains.items:
        ...
```

POST(***kwargs*)

Start an execution command and immediately return the job id

POST /minions

Request Headers

- **X-Auth-Token** -- a session token from `Login`.
- **Accept** -- the desired response format.
- **Content-Type** -- the format of the request body.

Response Headers

- **Content-Type** -- the format of the response body; depends on the `Accept` request header.

Status Codes

- **200** -- success
- **401** -- authentication required
- **406** -- requested Content-Type not available

lowstate data describing Salt commands must be sent in the request body. The `client` option will be set to `local_async()`.

Example request:

```
curl -sSi localhost:8000/minions \
    -H "Accept: application/x-yaml" \
    -d tgt='*' \
    -d fun='status.diskusage'
```

```
POST /minions HTTP/1.1
Host: localhost:8000
Accept: application/x-yaml
Content-Length: 26
Content-Type: application/x-www-form-urlencoded

tgt=*&fun=status.diskusage
```

Example response:

```
HTTP/1.1 202 Accepted
Content-Length: 86
Content-Type: application/x-yaml

return:
- jid: '20130603122505459265'
  minions: [ms-4, ms-3, ms-2, ms-1, ms-0]
_links:
  jobs:
    - href: /jobs/20130603122505459265
```

/jobs

class salt.netapi.rest_cherrypy.app.**Jobs**

GET(*jid=None*)

A convenience URL for getting lists of previously run jobs or getting the return from a single job

GET /jobs/(*jid*)

List jobs or show a single job from the job cache.

Status Codes

- **200** -- success
- **401** -- authentication required
- **406** -- requested Content-Type not available

Example request:

```
curl -i localhost:8000/jobs
```

```
GET /jobs HTTP/1.1
Host: localhost:8000
Accept: application/x-yaml
```

Example response:

```
HTTP/1.1 200 OK
Content-Length: 165
Content-Type: application/x-yaml

return:
- '20121130104633606931':
    Arguments:
    - '3'
    Function: test.fib
    Start Time: 2012, Nov 30 10:46:33.606931
    Target: jerry
    Target-type: glob
```

Example request:

```
curl -i localhost:8000/jobs/20121130104633606931
```

```
GET /jobs/20121130104633606931 HTTP/1.1
Host: localhost:8000
Accept: application/x-yaml
```

Example response:

```
HTTP/1.1 200 OK
Content-Length: 73
Content-Type: application/x-yaml

info:
- Arguments:
    - '3'
    Function: test.fib
    Minions:
    - jerry
    Start Time: 2012, Nov 30 10:46:33.606931
```

```
            Target: '*'
            Target-type: glob
            User: saltdev
            jid: '20121130104633606931'
        return:
        - jerry:
            - - 0
            - 1
            - 1
            - 2
            - 6.9141387939453125e-06
```

/run

class salt.netapi.rest_cherrypy.app.**Run**

Class to run commands without normal session handling

POST(***kwargs*)

Run commands bypassing the *normal session handling*

POST /run

This entry point is primarily for ``one-off'' commands. Each request must pass full Salt authentication credentials. Otherwise this URL is identical to the `root URL (/)`.

lowstate data describing Salt commands must be sent in the request body.

Status Codes

- **200** -- success
- **401** -- authentication required
- **406** -- requested Content-Type not available

Example request:

```
curl -sS localhost:8000/run \
    -H 'Accept: application/x-yaml' \
    -d client='local' \
    -d tgt='*' \
    -d fun='test.ping' \
    -d username='saltdev' \
    -d password='saltdev' \
    -d eauth='pam'
```

```
POST /run HTTP/1.1
Host: localhost:8000
Accept: application/x-yaml
Content-Length: 75
Content-Type: application/x-www-form-urlencoded

client=local&tgt=*&fun=test.ping&username=saltdev&password=saltdev&eauth=pam
```

Example response:

```
HTTP/1.1 200 OK
Content-Length: 73
Content-Type: application/x-yaml

return:
- ms-0: true
    ms-1: true
```

```
            ms-2: true
            ms-3: true
            ms-4: true
```

The /run enpoint can also be used to issue commands using the salt-ssh subsystem.

When using salt-ssh, eauth credentials should not be supplied. Instad, authentication should be handled by the SSH layer itself. The use of the salt-ssh client does not require a salt master to be running. Instead, only a roster file must be present in the salt configuration directory.

All SSH client requests are synchronous.

** Example SSH client request:**

```
curl -sS localhost:8000/run \
    -H 'Accept: application/x-yaml' \
    -d client='ssh' \
    -d tgt='*' \
    -d fun='test.ping'
```

```
POST /run HTTP/1.1
Host: localhost:8000
Accept: application/x-yaml
Content-Length: 75
Content-Type: application/x-www-form-urlencoded

client=ssh&tgt=*&fun=test.ping
```

Example SSH response:

```
return:
- silver:
  fun: test.ping
  fun_args: []
  id: silver
  jid: '20141203103525666185'
  retcode: 0
  return: true
  success: true
```

/events

class salt.netapi.rest_cherrypy.app.**Events**

Expose the Salt event bus

The event bus on the Salt master exposes a large variety of things, notably when executions are started on the master and also when minions ultimately return their results. This URL provides a real-time window into a running Salt infrastructure.

See also:

events

GET(*token=None*, *salt_token=None*)

An HTTP stream of the Salt master event bus

This stream is formatted per the Server Sent Events (SSE) spec. Each event is formatted as JSON.

GET /events

Status Codes

- **200** -- success
- **401** -- authentication required
- **406** -- requested Content-Type not available

Query Parameters

- **token** -- **optional** parameter containing the token ordinarily supplied via the X-Auth-Token header in order to allow cross-domain requests in browsers that do not include CORS support in the EventSource API. E.g., `curl -NsS localhost:8000/events?token=308650d`
- **salt_token** -- **optional** parameter containing a raw Salt *eauth token* (not to be confused with the token returned from the /login URL). E.g., `curl -NsS localhost:8000/events?salt_token=30742765`

Example request:

```
curl -NsS localhost:8000/events
```

```
GET /events HTTP/1.1
Host: localhost:8000
```

Example response:

Note, the `tag` field is not part of the spec. SSE compliant clients should ignore unknown fields. This addition allows non-compliant clients to only watch for certain tags without having to deserialze the JSON object each time.

```
HTTP/1.1 200 OK
Connection: keep-alive
Cache-Control: no-cache
Content-Type: text/event-stream;charset=utf-8

retry: 400

tag: salt/job/20130802115730568475/new
data: {'tag': 'salt/job/20130802115730568475/new', 'data': {'minions': ['ms-4', 'ms-3', 'ms-

tag: salt/job/20130802115730568475/ret/jerry
data: {'tag': 'salt/job/20130802115730568475/ret/jerry', 'data': {'jid': '20130802115730568
```

The event stream can be easily consumed via JavaScript:

```
var source = new EventSource('/events');
source.onopen = function() { console.debug('opening') };
source.onerror = function(e) { console.debug('error!', e) };
source.onmessage = function(e) {
    console.debug('Tag: ', e.data.tag)
    console.debug('Data: ', e.data.data)
};
```

Or using CORS:

```
var source = new EventSource('/events?token=ecd589e4e01912cf3c4035afad73426dbb8dba75', {with
```

It is also possible to consume the stream via the shell.

Records are separated by blank lines; the `data:` and `tag:` prefixes will need to be removed manually before attempting to unserialize the JSON.

curl's `-N` flag turns off input buffering which is required to process the stream incrementally.

Here is a basic example of printing each event as it comes in:

```
curl -NsS localhost:8000/events |\
        while IFS= read -r line ; do
            echo $line
        done
```

Here is an example of using awk to filter events based on tag:

```
curl -NsS localhost:8000/events |\
        awk '
            BEGIN { RS=""; FS="\\n" }
            $1 ~ /^tag: salt\/job\/[0-9]+\/new$/ { print $0 }
        '
tag: salt/job/20140112010149808995/new
data: {"tag": "salt/job/20140112010149808995/new", "data": {"tgt_type": "glob", "jid": "2014
tag: 20140112010149808995
data: {"tag": "20140112010149808995", "data": {"fun_args": [], "jid": "20140112010149808995"
```

/hook

class salt.netapi.rest_cherrypy.app.**Webhook**

A generic web hook entry point that fires an event on Salt's event bus

External services can POST data to this URL to trigger an event in Salt. For example, Amazon SNS, Jenkins-CI or Travis-CI, or GitHub web hooks.

Note: Be mindful of security

Salt's Reactor can run any code. A Reactor SLS that responds to a hook event is responsible for validating that the event came from a trusted source and contains valid data.

This is a generic interface and securing it is up to you!

This URL requires authentication however not all external services can be configured to authenticate. For this reason authentication can be selectively disabled for this URL. Follow best practices -- always use SSL, pass a secret key, configure the firewall to only allow traffic from a known source, etc.

The event data is taken from the request body. The `Content-Type` header is respected for the payload.

The event tag is prefixed with `salt/netapi/hook` and the URL path is appended to the end. For example, a `POST` request sent to `/hook/mycompany/myapp/mydata` will produce a Salt event with the tag `salt/netapi/hook/mycompany/myapp/mydata`.

The following is an example `.travis.yml` file to send notifications to Salt of successful test runs:

```
language: python
script: python -m unittest tests
after_success:
    - |
        curl -sSk https://saltapi-url.example.com:8000/hook/travis/build/success
```

See also:

events, *reactor*

POST(**args*, ***kwargs*)

Fire an event in Salt with a custom event tag and data

POST /hook

Status Codes

- **200** -- success
- **401** -- authentication required
- **406** -- requested Content-Type not available
- **413** -- request body is too large

Example request:

```
curl -sS localhost:8000/hook -d foo='Foo!' -d bar='Bar!'
```

```
POST /hook HTTP/1.1
Host: localhost:8000
Content-Length: 16
Content-Type: application/x-www-form-urlencoded

foo=Foo&bar=Bar!
```

Example response:

```
HTTP/1.1 200 OK
Content-Length: 14
Content-Type: application/json

{"success": true}
```

As a practical example, an internal continuous-integration build server could send an HTTP POST request to the URL `https://localhost:8000/hook/mycompany/build/success` which contains the result of a build and the SHA of the version that was built as JSON. That would then produce the following event in Salt that could be used to kick off a deployment via Salt's Reactor:

```
Event fired at Fri Feb 14 17:40:11 2014
*************************
Tag: salt/netapi/hook/mycompany/build/success
Data:
{'_stamp': '2014-02-14_17:40:11.440996',
    'headers': {
        'X-My-Secret-Key': 'F0fAgoQjIT@W',
        'Content-Length': '37',
        'Content-Type': 'application/json',
        'Host': 'localhost:8000',
        'Remote-Addr': '127.0.0.1'},
    'post': {'revision': 'aa22a3c4b2e7', 'result': True}}
```

Salt's Reactor could listen for the event:

```
reactor:
  - 'salt/netapi/hook/mycompany/build/*':
    - /srv/reactor/react_ci_builds.sls
```

And finally deploy the new build:

```
{% set secret_key = data.get('headers', {}).get('X-My-Secret-Key') %}
{% set build = data.get('post', {}) %}

{% if secret_key == 'F0fAgoQjIT@W' and build.result == True %}
deploy_my_app:
  cmd.state.sls:
    - tgt: 'application*'
    - arg:
      - myapp.deploy
    - kwarg:
```

```
                pillar:
                  revision: {{ revision }}
        {% endif %}
```

/keys

class salt.netapi.rest_cherrypy.app.**Keys**

Convenience URLs for working with minion keys

New in version 2014.7.0.

These URLs wrap the functionality provided by the `key wheel module` functions.

GET(*mid=None*)

Show the list of minion keys or detail on a specific key

New in version 2014.7.0.

GET /keys/(*mid*)

List all keys or show a specific key

Status Codes

- **200** -- success
- **401** -- authentication required
- **406** -- requested Content-Type not available

Example request:

```
curl -i localhost:8000/keys
```

```
GET /keys HTTP/1.1
Host: localhost:8000
Accept: application/x-yaml
```

Example response:

```
HTTP/1.1 200 OK
Content-Length: 165
Content-Type: application/x-yaml

return:
  local:
  - master.pem
  - master.pub
  minions:
  - jerry
  minions_pre: []
  minions_rejected: []
```

Example request:

```
curl -i localhost:8000/keys/jerry
```

```
GET /keys/jerry HTTP/1.1
Host: localhost:8000
Accept: application/x-yaml
```

Example response:

```
HTTP/1.1 200 OK
Content-Length: 73
Content-Type: application/x-yaml

return:
  minions:
    jerry: 51:93:b3:d0:9f:3a:6d:e5:28:67:c2:4b:27:d6:cd:2b
```

POST (*mid*, *keysize=None*, *force=None*, ***kwargs*)

Easily generate keys for a minion and auto-accept the new key

New in version 2014.7.0.

Example partial kickstart script to bootstrap a new minion:

```
%post
mkdir -p /etc/salt/pki/minion
curl -sSk https://localhost:8000/keys \
        -d mid=jerry \
        -d username=kickstart \
        -d password=kickstart \
        -d eauth=pam \
    | tar -C /etc/salt/pki/minion -xf -

mkdir -p /etc/salt/minion.d
printf 'master: 10.0.0.5\nid: jerry' > /etc/salt/minion.d/id.conf
%end
```

POST /keys

Generate a public and private key and return both as a tarball

Authentication credentials must be passed in the request.

Status Codes

- **200** -- success
- **401** -- authentication required
- **406** -- requested Content-Type not available

Example request:

```
curl -sSk https://localhost:8000/keys \
        -d mid=jerry \
        -d username=kickstart \
        -d password=kickstart \
        -d eauth=pam \
        -o jerry-salt-keys.tar
```

```
POST /keys HTTP/1.1
Host: localhost:8000
```

Example response:

```
HTTP/1.1 200 OK
Content-Length: 10240
Content-Disposition: attachment; filename="saltkeys-jerry.tar"
Content-Type: application/x-tar

jerry.pub0000644000000000000000000000070300000000000010730 0ustar  00000000000000
```

/ws

class salt.netapi.rest_cherrypy.app.**WebsocketEndpoint**

Open a WebSocket connection to Salt's event bus

The event bus on the Salt master exposes a large variety of things, notably when executions are started on the master and also when minions ultimately return their results. This URL provides a real-time window into a running Salt infrastructure. Uses websocket as the transport mechanism.

See also:

events

GET(*token=None*, ***kwargs*)

Return a websocket connection of Salt's event stream

GET /ws/(*token*)

Query Parameters

- **format_events** -- The event stream will undergo server-side formatting if the `format_events` URL parameter is included in the request. This can be useful to avoid formatting on the client-side:

```
curl -NsS <...snip...> localhost:8000/ws?format_events
```

Request Headers

- **X-Auth-Token** -- an authentication token from `Login`.

Status Codes

- **101** -- switching to the websockets protocol
- **401** -- authentication required
- **406** -- requested Content-Type not available

Example request:

curl -NsSk -H `X-Auth-Token: ffedf49d' -H `Host: localhost:8000' -H `Connection: Upgrade' -H `Upgrade: websocket' -H `Origin: https://localhost:8000' -H `Sec-WebSocket-Version: 13' -H `Sec-WebSocket-Key: `"$(echo -n $RANDOM | base64)" localhost:8000/ws

```
GET /ws HTTP/1.1
Connection: Upgrade
Upgrade: websocket
Host: localhost:8000
Origin: https://localhost:8000
Sec-WebSocket-Version: 13
Sec-WebSocket-Key: s65VsgHigh7v/Jcf4nXHnA==
X-Auth-Token: ffedf49d
```

Example response:

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: mWZjBV9FCglzn1rIKJAxrTFlnJE=
Sec-WebSocket-Version: 13
```

An authentication token **may optionally** be passed as part of the URL for browsers that cannot be configured to send the authentication header or cookie:

```
curl -NsS <...snip...> localhost:8000/ws/ffedf49d
```

The event stream can be easily consumed via JavaScript:

```
// Note, you must be authenticated!
var source = new Websocket('ws://localhost:8000/ws/d0ce6c1a');
source.onerror = function(e) { console.debug('error!', e); };
source.onmessage = function(e) { console.debug(e.data); };

source.send('websocket client ready')

source.close();
```

Or via Python, using the Python module websocket-client for example.

```
# Note, you must be authenticated!

from websocket import create_connection

ws = create_connection('ws://localhost:8000/ws/d0ce6c1a')
ws.send('websocket client ready')

# Look at https://pypi.python.org/pypi/websocket-client/ for more
# examples.
while listening_to_events:
    print ws.recv()

ws.close()
```

Above examples show how to establish a websocket connection to Salt and activating real time updates from Salt's event stream by signaling `websocket client ready`.

/stats

class salt.netapi.rest_cherrypy.app.**Stats**

Expose statistics on the running CherryPy server

GET()

Return a dump of statistics collected from the CherryPy server

GET /stats

Request Headers

- **X-Auth-Token** -- a session token from Login.
- **Accept** -- the desired response format.

Response Headers

- **Content-Type** -- the format of the response body; depends on the Accept request header.

Status Codes

- **200** -- success
- **401** -- authentication required
- **406** -- requested Content-Type not available

22.17.2 rest_tornado

A non-blocking REST API for Salt

depends

- tornado Python module

configuration All authentication is done through Salt's *external auth* system which requires additional configuration not described here.

In order to run rest_tornado with the salt-master add the following to the Salt master config file.

```
rest_tornado:
    # can be any port
    port: 8000
    # address to bind to (defaults to 0.0.0.0)
    address: 0.0.0.0
    # socket backlog
    backlog: 128
    ssl_crt: /etc/pki/api/certs/server.crt
    # no need to specify ssl_key if cert and key
    # are in one single file
    ssl_key: /etc/pki/api/certs/server.key
    debug: False
    disable_ssl: False
```

Authentication

Authentication is performed by passing a session token with each request. Tokens are generated via the `SaltAuthHandler` URL.

The token may be sent in one of two ways:

- Include a custom header named `X-Auth-Token`.
- Sent via a cookie. This option is a convenience for HTTP clients that automatically handle cookie support (such as browsers).

See also:

You can bypass the session handling via the `RunSaltAPIHandler` URL.

Usage

Commands are sent to a running Salt master via this module by sending HTTP requests to the URLs detailed below.

Content negotiation

This REST interface is flexible in what data formats it will accept as well as what formats it will return (e.g., JSON, YAML, x-www-form-urlencoded).

- Specify the format of data in the request body by including the `Content-Type` header.
- Specify the desired data format for the response body with the `Accept` header.

Data sent in POST and PUT requests must be in the format of a list of lowstate dictionaries. This allows multiple commands to be executed in a single HTTP request.

lowstate A dictionary containing various keys that instruct Salt which command to run, where that command lives, any parameters for that command, any authentication credentials, what returner to use, etc.

Salt uses the lowstate data format internally in many places to pass command data between functions. Salt also uses lowstate for the *LocalClient()* Python API interface.

The following example (in JSON format) causes Salt to execute two commands:

```
[{
    "client": "local",
    "tgt": "*",
    "fun": "test.fib",
    "arg": ["10"]
},
{
    "client": "runner",
    "fun": "jobs.lookup_jid",
    "jid": "20130603122505459265"
}]
```

Multiple commands in a Salt API request will be executed in serial and makes no gaurantees that all commands will run. Meaning that if test.fib (from the example above) had an exception, the API would still execute ``jobs.lookup_jid''.

Responses to these lowstates are an in-order list of dicts containing the return data, a yaml response could look like:

```
- ms-1: true
  ms-2: true
- ms-1: foo
  ms-2: bar
```

In the event of an exception while executing a command the return for that lowstate will be a string, for example if no minions matched the first lowstate we would get a return like:

```
- No minions matched the target. No command was sent, no jid was assigned.
- ms-1: true
  ms-2: true
```

---

x-www-form-urlencoded

Sending JSON or YAML in the request body is simple and most flexible, however sending data in urlencoded format is also supported with the caveats below. It is the default format for HTML forms, many JavaScript libraries, and the **`curl`** command.

For example, the equivalent to running `salt '*' test.ping` is sending `fun=test.ping&arg&client=local&tgt=*` in the HTTP request body.

Caveats:

- Only a single command may be sent per HTTP request.
- Repeating the `arg` parameter multiple times will cause those parameters to be combined into a single list.

  Note, some popular frameworks and languages (notably jQuery, PHP, and Ruby on Rails) will automatically append empty brackets onto repeated parameters. E.g., `arg=one`, `arg=two` will be sent as `arg[]=one`, `arg[]=two`. This is not supported; send JSON or YAML instead.

A Websockets add-on to saltnado

depends

- tornado Python module

In order to enable saltnado_websockets you must add websockets: True to your saltnado config block.

```
rest_tornado:
    # can be any port
    port: 8000
    ssl_crt: /etc/pki/api/certs/server.crt
    # no need to specify ssl_key if cert and key
    # are in one single file
    ssl_key: /etc/pki/api/certs/server.key
    debug: False
    disable_ssl: False
    websockets: True
```

All Events

Exposes `all` ``real-time'' events from Salt's event bus on a websocket connection. It should be noted that ``Real-time'' here means these events are made available to the server as soon as any salt related action (changes to minions, new jobs etc) happens. Clients are however assumed to be able to tolerate any network transport related latencies. Functionality provided by this endpoint is similar to the `/events` end point.

The event bus on the Salt master exposes a large variety of things, notably when executions are started on the master and also when minions ultimately return their results. This URL provides a real-time window into a running Salt infrastructure. Uses websocket as the transport mechanism.

Exposes GET method to return websocket connections. All requests should include an auth token. A way to obtain obtain authentication tokens is shown below.

```
% curl -si localhost:8000/login \
    -H "Accept: application/json" \
    -d username='salt' \
    -d password='salt' \
    -d eauth='pam'
```

Which results in the response

```
{
    "return": [{
        "perms": [".*", "@runner", "@wheel"],
        "start": 1400556492.277421,
        "token": "d0ce6c1a37e99dcc0374392f272fe19c0090cca7",
        "expire": 1400599692.277422,
        "user": "salt",
        "eauth": "pam"
    }]
}
```

In this example the `token` returned is `d0ce6c1a37e99dcc0374392f272fe19c0090cca7` and can be included in subsequent websocket requests (as part of the URL).

The event stream can be easily consumed via JavaScript:

```
// Note, you must be authenticated!

// Get the Websocket connection to Salt
var source = new Websocket('wss://localhost:8000/all_events/d0ce6c1a37e99dcc0374392f272fe19c0090cca7

// Get Salt's "real time" event stream.
source.onopen = function() { source.send('websocket client ready'); };

// Other handlers
source.onerror = function(e) { console.debug('error!', e); };

// e.data represents Salt's "real time" event data as serialized JSON.
source.onmessage = function(e) { console.debug(e.data); };

// Terminates websocket connection and Salt's "real time" event stream on the server.
source.close();
```

Or via Python, using the Python module websocket-client for example. Or the tornado client.

```
# Note, you must be authenticated!

from websocket import create_connection

# Get the Websocket connection to Salt
ws = create_connection('wss://localhost:8000/all_events/d0ce6c1a37e99dcc0374392f272fe19c0090cca7')

# Get Salt's "real time" event stream.
ws.send('websocket client ready')


# Simple listener to print results of Salt's "real time" event stream.
# Look at https://pypi.python.org/pypi/websocket-client/ for more examples.
while listening_to_events:
    print ws.recv()       #  Salt's "real time" event data as serialized JSON.

# Terminates websocket connection and Salt's "real time" event stream on the server.
ws.close()

# Please refer to https://github.com/liris/websocket-client/issues/81 when using a self signed cert
```

Above examples show how to establish a websocket connection to Salt and activating real time updates from Salt's event stream by signaling `websocket client ready`.

Formatted Events

Exposes `formatted` ``real-time'' events from Salt's event bus on a websocket connection. It should be noted that ``Real-time'' here means these events are made available to the server as soon as any salt related action (changes to minions, new jobs etc) happens. Clients are however assumed to be able to tolerate any network transport related latencies. Functionality provided by this endpoint is similar to the `/events` end point.

The event bus on the Salt master exposes a large variety of things, notably when executions are started on the master and also when minions ultimately return their results. This URL provides a real-time window into a running Salt infrastructure. Uses websocket as the transport mechanism.

Formatted events parses the raw ``real time'' event stream and maintains a current view of the following:

- minions
- jobs

A change to the minions (such as addition, removal of keys or connection drops) or jobs is processed and clients are updated. Since we use salt's presence events to track minions, please enable `presence_events` and set a small value for the `loop_interval` in the salt master config file.

Exposes GET method to return websocket connections. All requests should include an auth token. A way to obtain obtain authentication tokens is shown below.

```
% curl -si localhost:8000/login \
    -H "Accept: application/json" \
    -d username='salt' \
    -d password='salt' \
    -d eauth='pam'
```

Which results in the response

```
{
    "return": [{
        "perms": [".*", "@runner", "@wheel"],
        "start": 1400556492.277421,
        "token": "d0ce6c1a37e99dcc0374392f272fe19c0090cca7",
        "expire": 1400599692.277422,
        "user": "salt",
        "eauth": "pam"
    }]
}
```

In this example the `token` returned is `d0ce6c1a37e99dcc0374392f272fe19c0090cca7` and can be included in subsequent websocket requests (as part of the URL).

The event stream can be easily consumed via JavaScript:

```
// Note, you must be authenticated!

// Get the Websocket connection to Salt
var source = new Websocket('wss://localhost:8000/formatted_events/d0ce6c1a37e99dcc0374392f272fe19c009

// Get Salt's "real time" event stream.
source.onopen = function() { source.send('websocket client ready'); };

// Other handlers
source.onerror = function(e) { console.debug('error!', e); };

// e.data represents Salt's "real time" event data as serialized JSON.
source.onmessage = function(e) { console.debug(e.data); };

// Terminates websocket connection and Salt's "real time" event stream on the server.
source.close();
```

Or via Python, using the Python module websocket-client for example. Or the tornado client.

```
# Note, you must be authenticated!

from websocket import create_connection
```

```
# Get the Websocket connection to Salt
ws = create_connection('wss://localhost:8000/formatted_events/d0ce6c1a37e99dcc0374392f272fe19c0090cca

# Get Salt's "real time" event stream.
ws.send('websocket client ready')


# Simple listener to print results of Salt's "real time" event stream.
# Look at https://pypi.python.org/pypi/websocket-client/ for more examples.
while listening_to_events:
    print ws.recv()       #  Salt's "real time" event data as serialized JSON.

# Terminates websocket connection and Salt's "real time" event stream on the server.
ws.close()

# Please refer to https://github.com/liris/websocket-client/issues/81 when using a self signed cert
```

Above examples show how to establish a websocket connection to Salt and activating real time updates from Salt's event stream by signaling `websocket client ready`.

Example responses

`Minion information` is a dictionary keyed by each connected minion's `id` (`mid`), grains information for each minion is also included.

Minion information is sent in response to the following minion events:

- **connection drops**
  - requires running `manage.present` periodically every `loop_interval` seconds
- minion addition
- minon removal

```
# Not all grains are shown
data: {
    "minions": {
        "minion1": {
            "id": "minion1",
            "grains": {
                "kernel": "Darwin",
                "domain": "local",
                "zmqversion": "4.0.3",
                "kernelrelease": "13.2.0"
            }
        }
    }
}
```

`Job information` is also tracked and delivered.

Job information is also a dictionary in which each job's information is keyed by salt's `jid`.

```
data: {
    "jobs": {
        "20140609153646699137": {
```

```
            "tgt_type": "glob",
            "jid": "20140609153646699137",
            "tgt": "*",
            "start_time": "2014-06-09T15:36:46.700315",
            "state": "complete",
            "fun": "test.ping",
            "minions": {
                "minion1": {
                    "return": true,
                    "retcode": 0,
                    "success": true
                }
            }
        }
    }
}
```

Setup

REST URI Reference

- /
- /login
- /minions
- /jobs
- /run
- /events
- /hook

/

salt.netapi.rest_tornado.saltnado.**SaltAPIHandler**
alias of <Mock object at 0x2b97d709e410>

/login

salt.netapi.rest_tornado.saltnado.**SaltAuthHandler**
alias of <Mock object at 0x2b97d709e450>

/minions

salt.netapi.rest_tornado.saltnado.**MinionSaltAPIHandler**
alias of <Mock object at 0x2b97d709e310>

/jobs

salt.netapi.rest_tornado.saltnado.**JobsSaltAPIHandler**
alias of <Mock object at 0x2b97d709e050>

/run

salt.netapi.rest_tornado.saltnado.**RunSaltAPIHandler**
alias of <Mock object at 0x2b97daf43190>

/events

salt.netapi.rest_tornado.saltnado.**EventsSaltAPIHandler**
alias of <Mock object at 0x2b97dd62a250>

/hook

salt.netapi.rest_tornado.saltnado.**WebhookSaltAPIHandler**
alias of <Mock object at 0x2b97dd62a990>

22.17.3 rest_wsgi

A minimalist REST API for Salt

This rest_wsgi module provides a no-frills REST interface for sending commands to the Salt master. There are no dependencies.

Extra care must be taken when deploying this module into production. Please read this documentation in entirety.

All authentication is done through Salt's *external auth* system.

Usage

- All requests must be sent to the root URL (/).
- All requests must be sent as a POST request with JSON content in the request body.
- All responses are in JSON.

See also:

rest_cherrypy

The rest_cherrypy module is more full-featured, production-ready, and has builtin security features.

Deployment

The rest_wsgi netapi module is a standard Python WSGI app. It can be deployed one of two ways.

Using a WSGI-compliant web server

This module may be run via any WSGI-compliant production server such as Apache with mod_wsgi or Nginx with FastCGI.

It is strongly recommended that this app be used with a server that supports HTTPS encryption since raw Salt authentication credentials must be sent with every request. Any apps that access Salt through this interface will need to manually manage authentication credentials (either username and password or a Salt token). Tread carefully.

`salt-api` using a development-only server

If run directly via the salt-api daemon it uses the wsgiref.simple_server() that ships in the Python standard library. This is a single-threaded server that is intended for testing and development. **This server does not use encryption**; please note that raw Salt authentication credentials must be sent with every HTTP request.

Running this module via salt-api is not recommended!

In order to start this module via the `salt-api` daemon the following must be put into the Salt master config:

```
rest_wsgi:
    port: 8001
```

Usage examples

`POST /`

Example request for a basic `test.ping`:

```
% curl -sS -i \
        -H 'Content-Type: application/json' \
        -d '[{"eauth":"pam","username":"saltdev","password":"saltdev","client":"local","tgt":"*"
```

Example response:

```
HTTP/1.0 200 OK
Content-Length: 89
Content-Type: application/json

{"return": [{"ms--4": true, "ms--3": true, "ms--2": true, "ms--1": true, "ms--0": true}]}
```

Example request for an asynchronous `test.ping`:

```
% curl -sS -i \
        -H 'Content-Type: application/json' \
        -d '[{"eauth":"pam","username":"saltdev","password":"saltdev","client":"local_async","tg
```

Example response:

```
HTTP/1.0 200 OK
Content-Length: 103
Content-Type: application/json

{"return": [{"jid": "20130412192112593739", "minions": ["ms--4", "ms--3", "ms--2", "ms--1", "ms-
```

Example request for looking up a job ID:

```
% curl -sS -i \
        -H 'Content-Type: application/json' \
        -d '[{"eauth":"pam","username":"saltdev","password":"saltdev","client":"runner","fun":"j
```

Example response:

```
HTTP/1.0 200 OK
Content-Length: 89
Content-Type: application/json

{"return": [{"ms--4": true, "ms--3": true, "ms--2": true, "ms--1": true, "ms--0": true}]}
```

form lowstate A list of *lowstate* data appropriate for the *client* interface you are calling.

status 200 success

status 401 authentication required

22.18 Full list of builtin output modules

Follow one of the below links for further information and examples

compact	Display compact output data structure
highstate	Outputter for displaying results of state runs
json_out	Display return data in JSON format
key	Display salt-key output
nested	Recursively display nested data
newline_values_only	Display values only, separated by newlines
no_out	Display no output
no_return	Display output for minions that did not return
overstatestage	Display clean output of an overstate stage
pprint_out	Python pretty-print (pprint)
progress	Display return data as a progress bar
raw	Display raw output data structure
txt	Simple text outputter
virt_query	virt.query outputter
yaml_out	Display return data in YAML format

22.18.1 salt.output.compact

Display compact output data structure

Example output:: `saltdev': {`test_|-always-passes_|-foo_|-succeed_without_changes': {`comment': `Success!', `name': `foo', `start_time': `05:16:26.111814', `result': True, `duration': 1, `__run_num__': 0, `changes': {}}, `test_|-my-custom-combo_|-foo_|-configurable_test_state': {`comment': `bar.baz', `name': `foo', `start_time': `05:16:26.117177', `result': False, `duration': 1, `__run_num__': 4, `changes': {`testing': {`new': `Something pretended to change', `old': `Unchanged'}}}, `test_|-always-fails_|-foo_|-fail_without_changes': {`comment': `Failure!', `name': `foo', `start_time': `05:16:26.113124', `result': False, `duration': 1, `__run_num__': 1, `changes': {}}, `test_|-always-changes-and-succeeds_|-foo_|-succeed_with_changes': {`comment': `Success!', `name': `foo', `start_time': `05:16:26.114570', `result': True, `duration': 0, `__run_num__': 2, `changes': {`testing': {`new': `Something pretended to change', `old': `Unchanged'}}}, `test_|-always-changes-and-fails_|-foo_|-fail_with_changes': {`comment': `Failure!', `name': `foo', `start_time': `05:16:26.115561', `result': False, `duration': 1, `__run_num__': 3, `changes': {`testing': {`new': `Something pretended to change', `old': `Unchanged'}}}}}

{`myminion': {`foo': {`list': ['Hello', `World'], `bar': `baz', `dictionary': {`abc': 123, `def': 456}}}}

salt.output.compact.**output**(*data*)

Rather basic....

22.18.2 salt.output.highstate

Outputter for displaying results of state runs

The return data from the Highstate command is a standard data structure which is parsed by the highstate outputter to deliver a clean and readable set of information about the HighState run on minions.

Two configurations can be set to modify the highstate outputter. These values can be set in the master config to change the output of the `salt` command or set in the minion config to change the output of the `salt-call` command.

state_verbose: By default *state_verbose* is set to *True*, setting this to *False* will instruct the highstate outputter to omit displaying anything in green, this means that nothing with a result of True and no changes will not be printed

state_output: The highstate outputter has five output modes, `full`, `terse`, `mixed`, `changes` and `filter`.

- The default is set to `full`, which will display many lines of detailed information for each executed chunk.
- If `terse` is used, then the output is greatly simplified and shown in only one line.
- If `mixed` is used, then terse output will be used unless a state failed, in which case full output will be used.
- If `changes` is used, then terse output will be used if there was no error and no changes, otherwise full output will be used.
- If `filter` is used, then either or both of two different filters can be used: `exclude` or `terse`. These can be set as such from the command line, or in the Salt config as *state_output_exclude* or *state_output_terse*, respectively. The values to exclude must be a comma-separated list of *True*, *False* and/or *None*. Because of parsing nuances, if only one of these is used, it must still contain a comma. For instance: *exclude=True,*.

state_tabular: If *state_output* uses the terse output, set this to *True* for an aligned output format. If you wish to use a custom format, this can be set to a string.

Example output:

```
myminion:
----------
          ID: test.ping
    Function: module.run
      Result: True
     Comment: Module function test.ping executed
     Changes:
              ----------
              ret:
                  True

Summary
------------
Succeeded: 1
Failed:    0
------------
Total:     0
```

`salt.output.highstate.`**`output`**`(`*`data`*`)`

The HighState Outputter is only meant to be used with the state.highstate function, or a function that returns highstate return data.

22.18.3 salt.output.json_out

Display return data in JSON format

configuration The output format can be configured in two ways: Using the `--out-indent` CLI flag and specifying a positive integer or a negative integer to group JSON from each minion to a single line.

Or setting the `output_indent` setting in the Master or Minion configuration file with one of the following values:

- `Null`: put each minion return on a single line.
- `pretty`: use four-space indents and sort the keys.
- An integer: specify the indentation level.

Salt's outputters operate on a per-minion basis. Each minion return will be output as a single JSON object once it comes in to the master.

Some JSON parsers can guess when an object ends and a new one begins but many can not. A good way to differentiate between each minion return is to use the single-line output format and to parse each line individually. Example output (truncated):

```
{"dave": {"en0": {"hwaddr": "02:b0:26:32:4c:69", ...}}}
{"jerry": {"en0": {"hwaddr": "02:26:ab:0d:b9:0d", ...}}}
{"kevin": {"en0": {"hwaddr": "02:6d:7f:ce:9f:ee", ...}}}
{"mike": {"en0": {"hwaddr": "02:48:a2:4b:70:a0", ...}}}
{"phill": {"en0": {"hwaddr": "02:1d:cc:a2:33:55", ...}}}
{"stuart": {"en0": {"hwaddr": "02:9a:e0:ea:9e:3c", ...}}}
```

`salt.output.json_out.`**`output`**`(`*`data`*`)`
Print the output data in JSON

22.18.4 salt.output.key

Display salt-key output

The `salt-key` command makes use of this outputter to format its output.

`salt.output.key.`**`output`**`(`*`data`*`)`
Read in the dict structure generated by the salt key API methods and print the structure.

22.18.5 salt.output.nested

Recursively display nested data

This is the default outputter for most execution functions.

Example output:

```
myminion:
    ----------
    foo:
        ----------
        bar:
```

```
            baz
        dictionary:
            ----------
            abc:
                123
            def:
                456
        list:
            - Hello
            - World
```

class salt.output.nested.**NestDisplay**
Manage the nested display contents

display(*ret*, *indent*, *prefix*, *out*)
Recursively iterate down through data structures to determine output

ustring(*indent*, *color*, *msg*, *prefix=''*, *suffix=''*, *endc=None*)

salt.output.nested.**output**(*ret*)
Display ret data

22.18.6 salt.output.newline_values_only

Display values only, separated by newlines

New in version 2015.5.0.

This outputter is designed for Salt CLI return data. It will do the following to the return dict:

1. Get just the values (ignoring the minion IDs).
2. Each value, if it is iterable, is split a separate line.
3. Each minion's values are separated by newlines.

This results in a single string of return data containing all the values from the various minions.

Warning: As noted above, this outputter will discard the minion ID. If the minion ID is important, then an outputter that returns the full return dictionary in a parsable format (such as `json`, `pprint`,, or `yaml`) may be more suitable.

Example 1

Input

```
{
    'myminion': ['127.0.0.1', '10.0.0.1'],
    'second-minion': ['127.0.0.1', '10.0.0.2']
}
```

Output

```
127.0.0.1
10.0.0.1
127.0.0.1
10.0.0.2
```

Example 2

Input

```
{
    'myminion': 8,
    'second-minion': 10
}
```

Output

```
8
10
```

salt.output.newline_values_only.**output**(*data*)
Display modified ret data

22.18.7 salt.output.no_out

Display no output

No output is produced when this outputter is selected

salt.output.no_out.**output**(*ret*)
Don't display data. Used when you only are interested in the return.

22.18.8 salt.output.no_return

Display output for minions that did not return

This outputter is used to display notices about which minions failed to return when a salt function is run with `-v` or `--verbose`. It should not be called directly from the CLI.

Example output:

```
virtucentos:
    Minion did not return
```

class salt.output.no_return.**NestDisplay**
Create generator for nested output

display(*ret*, *indent*, *prefix*, *out*)
Recursively iterate down through data structures to determine output

salt.output.no_return.**output**(*ret*)
Display ret data

22.18.9 salt.output.overstatestage

Display clean output of an overstate stage

This outputter is used to display *OverState* stages, and should not be called directly.

`salt.output.overstatestage.`**`output`**(*data*)
Format the data for printing stage information from the overstate system

22.18.10 salt.output.pprint_out

Python pretty-print (pprint)

The python pretty-print system was once the default outputter. It simply passes the return data through to `pprint.pformat` and prints the results.

Example output:

```
{'saltmine': {'foo': {'bar': 'baz',
                      'dictionary': {'abc': 123, 'def': 456},
                      'list': ['Hello', 'World']}}}
```

`salt.output.pprint_out.`**`output`**(*data*)
Print out via pretty print

22.18.11 salt.output.progress

Display return data as a progress bar

`salt.output.progress.`**`output`**(*ret*, *bar*)
Update the progress bar

`salt.output.progress.`**`progress_iter`**(*progress*)
Initialize and return a progress bar iter

22.18.12 salt.output.raw

Display raw output data structure

This outputter simply displays the output as a python data structure, by printing a string representation of it. It is similar to the `pprint` outputter, only the data is not nicely formatted/indented.

This was the original outputter used by Salt before the outputter system was developed.

Example output:

```
{'myminion': {'foo': {'list': ['Hello', 'World'], 'bar': 'baz', 'dictionary': {'abc': 123, 'def': 456
```

`salt.output.raw.`**`output`**(*data*)
Rather basic....

22.18.13 salt.output.txt

Simple text outputter

The txt outputter has been developed to make the output from shell commands on minions appear as they do when the command is executed on the minion.

`salt.output.txt.`**`output`**`(`*`data`*`)`
Output the data in lines, very nice for running commands

22.18.14 salt.output.virt_query

virt.query outputter

Used to display the output from the `virt.query` runner.

`salt.output.virt_query.`**`output`**`(`*`data`*`)`
Display output for the salt-run virt.query function

22.18.15 salt.output.yaml_out

Display return data in YAML format

This outputter defaults to printing in YAML block mode for better readability.

Example output:

```
saltmine:
  foo:
    bar: baz
    dictionary:
      abc: 123
      def: 456
    list:
      - Hello
      - World
```

`salt.output.yaml_out.`**`output`**`(`*`data`*`)`
Print out YAML using the block mode

22.19 Peer Communication

Salt 0.9.0 introduced the capability for Salt minions to publish commands. The intent of this feature is not for Salt minions to act as independent brokers one with another, but to allow Salt minions to pass commands to each other.

In Salt 0.10.0 the ability to execute runners from the master was added. This allows for the master to return collective data from runners back to the minions via the peer interface.

The peer interface is configured through two options in the master configuration file. For minions to send commands from the master the `peer` configuration is used. To allow for minions to execute runners from the master the `peer_run` configuration is used.

Since this presents a viable security risk by allowing minions access to the master publisher the capability is turned off by default. The minions can be allowed access to the master publisher on a per minion basis based on regular expressions. Minions with specific ids can be allowed access to certain Salt modules and functions.

22.19.1 Peer Configuration

The configuration is done under the `peer` setting in the Salt master configuration file, here are a number of configuration possibilities.

The simplest approach is to enable all communication for all minions, this is only recommended for very secure environments.

```
peer:
  .*:
    - .*
```

This configuration will allow minions with IDs ending in example.com access to the test, ps, and pkg module functions.

```
peer:
  .*example.com:
    - test.*
    - ps.*
    - pkg.*
```

The configuration logic is simple, a regular expression is passed for matching minion ids, and then a list of expressions matching minion functions is associated with the named minion. For instance, this configuration will also allow minions ending with foo.org access to the publisher.

```
peer:
  .*example.com:
    - test.*
    - ps.*
    - pkg.*
  .*foo.org:
    - test.*
    - ps.*
    - pkg.*
```

22.19.2 Peer Runner Communication

Configuration to allow minions to execute runners from the master is done via the `peer_run` option on the master. The `peer_run` configuration follows the same logic as the `peer` option. The only difference is that access is granted to runner modules.

To open up access to all minions to all runners:

```
peer_run:
  .*:
    - .*
```

This configuration will allow minions with IDs ending in example.com access to the manage and jobs runner functions.

```
peer_run:
  .*example.com:
    - manage.*
    - jobs.*
```

22.19.3 Using Peer Communication

The publish module was created to manage peer communication. The publish module comes with a number of functions to execute peer communication in different ways. Currently there are three functions in the publish module. These examples will show how to test the peer system via the salt-call command.

To execute test.ping on all minions:

```
# salt-call publish.publish \* test.ping
```

To execute the manage.up runner:

```
# salt-call publish.runner manage.up
```

To match minions using other matchers, use `expr_form`:

```
# salt-call publish.publish 'webserv* and not G@os:Ubuntu' test.ping expr_form='compound'
```

22.20 Pillars

Salt includes a number of built-in external pillars, listed at *Full list of builtin pillar modules*.

You may also wish to look at the standard pillar documentation, at *Pillar Configuration*

The source for the built-in Salt pillars can be found here: https://github.com/saltstack/salt/blob/develop/salt/pillar

22.21 Full list of builtin pillar modules

cmd_json	Execute a command and read the output as JSON.
cmd_yaml	Execute a command and read the output as YAML.
cmd_yamlex	Execute a command and read the output as YAMLEX.
cobbler	A module to pull data from Cobbler via its API into the Pillar dictionary
django_orm	Generate Pillar data from Django models through the Django ORM
ec2_pillar	Retrieve EC2 instance data for minions.
etcd_pillar	Use etcd data as a Pillar source
file_encode	
file_tree	Recursively iterate over directories and add all files as Pillar data.
foreman	A module to pull data from Foreman via its API into the Pillar dictionary
git_pillar	Clone a remote git repository and use the filesystem as a Pillar source
hiera	Use hiera data as a Pillar source
libvirt	Load up the libvirt keys into Pillar for a given minion if said keys have been generated using the libvirt ke

Continued on n

Table 22.8 -- continued from previous page

mongo	Read Pillar data from a mongodb collection
mysql	Retrieve Pillar data by doing a MySQL query
pepa	Pepa
pillar_ldap	Use LDAP data as a Pillar source
puppet	Execute an unmodified puppet_node_classifier and read the output as YAML.
reclass_adapter	Use the ``reclass'' database as a Pillar source
redismod	Read pillar data from a Redis backend
s3	Copy pillar data from a bucket in Amazon S3
svn_pillar	Clone a remote SVN repository and use the filesystem as a Pillar source
varstack_pillar	Use varstack data as a Pillar source
virtkey	Accept a key from a hypervisor if the virt runner has already submitted an authorization request

22.21.1 salt.pillar.cmd_json

Execute a command and read the output as JSON. The JSON data is then directly overlaid onto the minion's Pillar data.

`salt.pillar.cmd_json.`**`ext_pillar`**(*minion_id*, *pillar*, *command*)
Execute a command and read the output as JSON

22.21.2 salt.pillar.cmd_yaml

Execute a command and read the output as YAML. The YAML data is then directly overlaid onto the minion's Pillar data

`salt.pillar.cmd_yaml.`**`ext_pillar`**(*minion_id*, *pillar*, *command*)
Execute a command and read the output as YAML

22.21.3 salt.pillar.cmd_yamlex

Execute a command and read the output as YAMLEX. The YAMLEX data is then directly overlaid onto the minion's Pillar data

`salt.pillar.cmd_yamlex.`**`ext_pillar`**(*minion_id*, *pillar*, *command*)
Execute a command and read the output as YAMLEX

22.21.4 salt.pillar.cobbler

A module to pull data from Cobbler via its API into the Pillar dictionary

Configuring the Cobbler ext_pillar

The same cobbler* parameters are used for both the Cobbler tops and Cobbler pillar modules.

```
ext_pillar:
- cobbler:
    key: cobbler # Nest results within this key. By default, values are not nested.
    only: [parameters] # Add only these keys to pillar.

cobbler.url: https://example.com/cobbler_api #default is http://localhost/cobbler_api
```

```
cobbler.user: username # default is no username
cobbler.password: password # default is no password
```

Module Documentation

salt.pillar.cobbler.**ext_pillar**(*minion_id*, *pillar*, *key=None*, *only=()*)
Read pillar data from Cobbler via its API.

22.21.5 salt.pillar.django_orm

Generate Pillar data from Django models through the Django ORM

maintainer Micah Hausler <micah.hausler@gmail.com>

maturity new

Configuring the django_orm ext_pillar

To use this module, your Django project must be on the salt master server with database access. This assumes you are using virtualenv with all the project's requirements installed.

```
ext_pillar:
  - django_orm:
      pillar_name: my_application
      project_path: /path/to/project/
      settings_module: my_application.settings
      env_file: /path/to/env/file.sh
      # Optional: If your project is not using the system python,
      # add your virtualenv path below.
      env: /path/to/virtualenv/

      django_app:

        # Required: the app that is included in INSTALLED_APPS
        my_application.clients:

          # Required: the model name
          Client:

            # Required: model field to use as the key in the rendered
            # Pillar. Must be unique; must also be included in the
            # ``fields`` list below.
            name: shortname

            # Optional:
            # See Django's QuerySet documentation for how to use .filter()
            filter:  {'kw': 'args'}

            # Required: a list of field names
            # List items will be used as arguments to the .values() method.
            # See Django's QuerySet documentation for how to use .values()
            fields:
              - field_1
              - field_2
```

This would return pillar data that would look like

```
my_application:
  my_application.clients:
    Client:
      client_1:
        field_1: data_from_field_1
        field_2: data_from_field_2
      client_2:
        field_1: data_from_field_1
        field_2: data_from_field_2
```

As another example, data from multiple database tables can be fetched using Django's regular lookup syntax. Note, using ManyToManyFields will not currently work since the return from values() changes if a ManyToMany is present.

```
ext_pillar:
  - django_orm:
      pillar_name: djangotutorial
      project_path: /path/to/mysite
      settings_module: mysite.settings

      django_app:
        mysite.polls:
          Choices:
            name: poll__question
            fields:
              - poll__question
              - poll__id
              - choice_text
              - votes
```

Module Documentation

salt.pillar.django_orm.**ext_pillar**(*minion_id*, *pillar*, *pillar_name*, *project_path*, *settings_module*, *django_app*, *env=None*, *env_file=None*, **args*, ***kwargs*)

Connect to a Django database through the ORM and retrieve model fields

Parameters

- **pillar_name** (*str*) -- The name of the pillar to be returned
- **project_path** (*str*) -- The full path to your Django project (the directory manage.py is in)
- **settings_module** (*str*) -- The settings module for your project. This can be found in your manage.py file
- **django_app** (*str*) -- A dictionary containing your apps, models, and fields
- **env** (*str*) -- The full path to the virtualenv for your Django project
- **env_file** (*str*) -- An optional bash file that sets up your environment. The file is run in a subprocess and the changed variables are then added

22.21.6 salt.pillar.ec2_pillar

Retrieve EC2 instance data for minions.

The minion id must be the instance-id retrieved from AWS. As an option, use_grain can be set to True. This allows the use of an instance-id grain instead of the minion-id. Since this is a potential security risk, the configuration can be further expanded to include a list of minions that are trusted to only allow the alternate id of the instances to specific hosts. There is no glob matching at this time.

```
ext_pillar:
  - ec2_pillar:
      use_grain: True
      minion_ids:
        - trusted-minion-1
        - trusted-minion-2
        - trusted-minion-3
```

This is a very simple pillar that simply retrieves the instance data from AWS. Currently the only portion implemented are EC2 tags, which returns a list of key/value pairs for all of the EC2 tags assigned to the instance.

`salt.pillar.ec2_pillar.`**`ext_pillar`**(*minion_id*, *pillar*, *use_grain=False*, *minion_ids=None*)
Execute a command and read the output as YAML

22.21.7 salt.pillar.etcd_pillar

Use etcd data as a Pillar source

New in version 2014.7.0.

depends

- python-etcd

In order to use an etcd server, a profile must be created in the master configuration file:

```
my_etcd_config:
  etcd.host: 127.0.0.1
  etcd.port: 4001
```

After the profile is created, configure the external pillar system to use it. Optionally, a root may be specified.

```
ext_pillar:
  - etcd: my_etcd_config

ext_pillar:
  - etcd: my_etcd_config root=/salt
```

Using these configuration profiles, multiple etcd sources may also be used:

```
ext_pillar:
  - etcd: my_etcd_config
  - etcd: my_other_etcd_config
```

The `minion_id` may be used in the `root` path to expose minion-specific information stored in etcd.

```
ext_pillar:
  - etcd: my_etcd_config root=/salt/%(minion_id)s
```

Minion-specific values may override shared values when the minion-specific root appears after the shared root:

```
ext_pillar:
  - etcd: my_etcd_config root=/salt-shared
  - etcd: my_other_etcd_config root=/salt-private/%(minion_id)s
```

Using the configuration above, the following commands could be used to share a key with all minions but override its value for a specific minion:

```
etcdctl set /salt-shared/mykey my_value
etcdctl set /salt-private/special_minion_id/mykey my_other_value
```

salt.pillar.etcd_pillar.**ext_pillar**(*minion_id*, *pillar*, *conf*)
Check etcd for all data

22.21.8 salt.pillar.file_tree

Recursively iterate over directories and add all files as Pillar data.

Example configuration:

```
ext_pillar:
  - file_tree:
      root_dir: /path/to/root/directory
      follow_dir_links: False
      raw_data: False
```

The `root_dir` parameter is required and points to the directory where files for each host are stored. The `follow_dir_links` parameter is optional and defaults to False. If `follow_dir_links` is set to True, file_tree will follow symbolic links to other directories. Be careful when using `follow_dir_links`, the current implementation is dumb and will run into infinite recursion if a recursive symlink chain exists in the root_dir!

If `raw_data` is set to True, it will revert the behavior of the python open() function, which adds a line break character at the end of the file, in this case, the pillar data.

To fill pillar data for each host, file_tree recursively iterates over `root_dir`/hosts/`id` (where `id` is a minion ID), and constructs the same directory tree with contents of all the files inside the pillar tree.

For example, the following `root_dir` tree:

```
./hosts/
./hosts/test-host/
./hosts/test-host/files/
./hosts/test-host/files/testdir/
./hosts/test-host/files/testdir/file1.txt
./hosts/test-host/files/testdir/file2.txt
./hosts/test-host/files/another-testdir/
./hosts/test-host/files/another-testdir/symlink-to-file1.txt
```

will result in the following pillar tree for minion with ID ``test-host'':

```
test-host:
    ----------
    files:
        ----------
        another-testdir:
            ----------
            symlink-to-file1.txt:
                Contents of file #1.

        testdir:
            ----------
            file1.txt:
                Contents of file #1.

            file2.txt:
                Contents of file #2.
```

To fill pillar data for minion in a node group, file_tree recursively iterates over `root_dir`/nodegroups/`nodegroup` (where `nodegroup` is a minion node group), and constructs the same directory tree with contents of all the files inside the pillar tree. **IMPORTANT**: The host data take precedence over the node group data

For example, the following `root_dir` tree:

```
./nodegroups/
./nodegroups/test-group/
./nodegroups/test-group/files/
./nodegroups/test-group/files/testdir/
./nodegroups/test-group/files/testdir/file1.txt
./nodegroups/test-group/files/testdir/file2.txt
./nodegroups/test-group/files/another-testdir/
./nodegroups/test-group/files/another-testdir/symlink-to-file1.txt
```

will result in the following pillar tree for minion in the node group ``test-group'':

```
test-host:
    ----------
    files:
        ----------
        another-testdir:
            ----------
            symlink-to-file1.txt:
                Contents of file #1.

        testdir:
            ----------
            file1.txt:
                Contents of file #1.

            file2.txt:
                Contents of file #2.
```

salt.pillar.file_tree.**ext_pillar**(*minion_id*, *pillar*, *root_dir=None*, *follow_dir_links=False*, *debug=False*, *raw_data=False*)

Find pillar data for specified ID.

22.21.9 salt.pillar.foreman

A module to pull data from Foreman via its API into the Pillar dictionary

Configuring the Foreman ext_pillar

Set the following Salt config to setup Foreman as external pillar source:

```
ext_pillar:
  - foreman:
      key: foreman # Nest results within this key
      only: ['hostgroup_name', 'parameters'] # Add only these keys to pillar

foreman.url: https://example.com/foreman_api
foreman.user: username # default is admin
foreman.password: password # default is changeme
```

The following options are optional:

```
foreman.api: apiversion # default is 2 (1 is not supported yet)
foreman.verifyssl: False # default is True
foreman.certfile: /etc/ssl/certs/mycert.pem # default is None
foreman.keyfile: /etc/ssl/private/mykey.pem # default is None
foreman.cafile: /etc/ssl/certs/mycert.ca.pem # default is None
foreman.lookup_parameters: True # default is True
```

An alternative would be to use the Foreman modules integrating Salt features in the Smart Proxy and the webinterface.

Further information can be found on Github.

Module Documentation

`salt.pillar.foreman.`**`ext_pillar`**(*minion_id*, *pillar*, *key=None*, *only=()*)

Read pillar data from Foreman via its API.

22.21.10 salt.pillar.git_pillar

Clone a remote git repository and use the filesystem as a Pillar source

Currently GitPython is the only supported provider for git Pillars

This external Pillar source can be configured in the master config file like so:

```
ext_pillar:
  - git: master git://gitserver/git-pillar.git root=subdirectory
```

The *root=* parameter is optional and used to set the subdirectory from where to look for Pillar files (such as `top.sls`).

Changed in version 2014.7.0: The optional `root` parameter will be added.

Changed in version 2015.5.0: The special branch name `__env__' will be replace by the environment ({{env}})

Note that this is not the same thing as configuring pillar data using the `pillar_roots` parameter. The branch referenced in the `ext_pillar` entry above (`master`), would evaluate to the `base` environment, so this branch needs to contain a `top.sls` with a `base` section in it, like this:

```
base:
  '*':
    - foo
```

To use other environments from the same git repo as git_pillar sources, just add additional lines, like so:

```
ext_pillar:
  - git: master git://gitserver/git-pillar.git
  - git: dev git://gitserver/git-pillar.git
```

To remap a specific branch to a specific environment separate the branch name and the environment name with a colon:

```
ext_pillar:
  - git: develop:dev git://gitserver/git-pillar.git
  - git: master:prod git://gitserver/git-pillar.git
```

In this case, the `dev` branch would need its own `top.sls` with a `dev` section in it, like this:

```
dev:
  '*':
    - bar
```

In a gitfs base setup with pillars from the same repository as the states, the `ext_pillar:` configuration would be like:

```
ext_pillar:
  - git: _ git://gitserver/git-pillar.git root=pillar
```

The (optinal) root=pillar defines the directory that contains the pillar data. The corresponding `top.sls` would be like:

```
{{env}}:
  '*':
    - bar
```

class salt.pillar.git_pillar.**GitPillar**(*branch*, *repo_location*, *opts*)

Deal with the remote git repository for Pillar

envs()

Return a list of refs that can be used as environments

map_branch(*branch*, *opts=None*)

update()

Ensure you are following the latest changes on the remote

Return boolean whether it worked

salt.pillar.git_pillar.**envs**(*branch*, *repo_location*)

Return a list of refs that can be used as environments

salt.pillar.git_pillar.**ext_pillar**(*minion_id*, *repo_string*, *pillar_dirs*)
Execute a command and read the output as YAML

salt.pillar.git_pillar.**update**(*branch*, *repo_location*)
Ensure you are following the latest changes on the remote

return boolean whether it worked

22.21.11 salt.pillar.hiera

Use hiera data as a Pillar source

salt.pillar.hiera.**ext_pillar**(*minion_id*, *pillar*, *conf*)
Execute hiera and return the data

22.21.12 salt.pillar.libvirt

Load up the libvirt keys into Pillar for a given minion if said keys have been generated using the libvirt key runner

salt.pillar.libvirt.**ext_pillar**(*minion_id*, *pillar*, *command*)
Read in the generated libvirt keys

salt.pillar.libvirt.**gen_hyper_keys**(*minion_id*, *country='US'*, *state='Utah'*, *locality='Salt Lake City'*, *organization='Salted'*)
Generate the keys to be used by libvirt hypervisors, this routine gens the keys and applies them to the pillar for the hypervisor minions

22.21.13 salt.pillar.mongo

Read Pillar data from a mongodb collection

depends pymongo (for salt-master)

This module will load a node-specific pillar dictionary from a mongo collection. It uses the node's id for lookups and can load either the whole document, or just a specific field from that document as the pillar dictionary.

Salt Master Mongo Configuration

The module shares the same base mongo connection variables as `salt.returners.mongo_return`. These variables go in your master config file.

- `mongo.db` - The mongo database to connect to. Defaults to `'salt'`.
- `mongo.host` - The mongo host to connect to. Supports replica sets by specifying all hosts in the set, comma-delimited. Defaults to `'salt'`.
- `mongo.port` - The port that the mongo database is running on. Defaults to `27017`.
- `mongo.user` - The username for connecting to mongo. Only required if you are using mongo authentication. Defaults to `''`.
- `mongo.password` - The password for connecting to mongo. Only required if you are using mongo authentication. Defaults to `''`.

Configuring the Mongo ext_pillar

The Mongo ext_pillar takes advantage of the fact that the Salt Master configuration file is yaml. It uses a sub-dictionary of values to adjust specific features of the pillar. This is the explicit single-line dictionary notation for yaml. One may be able to get the easier-to-read multi-line dict to work correctly with some experimentation.

```
ext_pillar:
  - mongo: {collection: vm, id_field: name, re_pattern: \.example\.com, fields: [customer_id, softwa
```

In the example above, we've decided to use the `vm` collection in the database to store the data. Minion ids are stored in the `name` field on documents in that collection. And, since minion ids are FQDNs in most cases, we'll need to trim the domain name in order to find the minion by hostname in the collection. When we find a minion, return only the `customer_id`, `software`, and `apache_vhosts` fields, as that will contain the data we want for a given node. They will be available directly inside the `pillar` dict in your SLS templates.

Module Documentation

`salt.pillar.mongo.`**`ext_pillar`**(*minion_id*, *pillar*, *collection='pillar'*, *id_field='_id'*, *re_pattern=None*, *re_replace=''*, *fields=None*)

Connect to a mongo database and read per-node pillar information.

Parameters:

- *collection*: The mongodb collection to read data from. Defaults to `'pillar'`.
- *id_field*: The field in the collection that represents an individual minion id. Defaults to `'_id'`.
- *re_pattern*: If your naming convention in the collection is shorter than the minion id, you can use this to trim the name. *re_pattern* will be used to match the name, and *re_replace* will be used to replace it. Backrefs are supported as they are in the Python standard library. If `None`, no mangling of the name will be performed - the collection will be searched with the entire minion id. Defaults to `None`.
- *re_replace*: Use as the replacement value in node ids matched with *re_pattern*. Defaults to ''. Feel free to use backreferences here.
- *fields*: The specific fields in the document to use for the pillar data. If `None`, will use the entire document. If using the entire document, the `_id` field will be converted to string. Be careful with other fields in the document as they must be string serializable. Defaults to `None`.

22.21.14 salt.pillar.mysql

Retrieve Pillar data by doing a MySQL query

MariaDB provides Python support through the MySQL Python package. Therefore, you may use this module with both MySQL or MariaDB.

maturity new

depends python-mysqldb

platform all

Theory of mysql ext_pillar

Ok, here's the theory for how this works...

- If there's a keyword arg of mysql_query, that'll go first.
- Then any non-keyword args are processed in order.
- Finally, remaining keywords are processed.

We do this so that it's backward compatible with older configs. Keyword arguments are sorted before being appended, so that they're predictable, but they will always be applied last so overall it's moot.

For each of those items we process, it depends on the object type:

- Strings are executed as is and the pillar depth is determined by the number of fields returned.
- A list has the first entry used as the query, the second as the pillar depth.
- A mapping uses the keys ``query'' and ``depth'' as the tuple

You can retrieve as many fields as you like, how they get used depends on the exact settings.

Configuring the mysql ext_pillar

First an example of how legacy queries were specified.

```
ext_pillar:
  - mysql:
      mysql_query: "SELECT pillar,value FROM pillars WHERE minion_id = %s"
```

Alternatively, a list of queries can be passed in

```
ext_pillar:
  - mysql:
      - "SELECT pillar,value FROM pillars WHERE minion_id = %s"
      - "SELECT pillar,value FROM more_pillars WHERE minion_id = %s"
```

Or you can pass in a mapping

```
ext_pillar:
  - mysql:
      main: "SELECT pillar,value FROM pillars WHERE minion_id = %s"
      extras: "SELECT pillar,value FROM more_pillars WHERE minion_id = %s"
```

The query can be provided as a string as we have just shown, but they can be provided as lists

```
ext_pillar:
  - mysql:
      - "SELECT pillar,value FROM pillars WHERE minion_id = %s"
        2
```

Or as a mapping

```
ext_pillar:
  - mysql:
      - query: "SELECT pillar,value FROM pillars WHERE minion_id = %s"
        depth: 2
```

The depth defines how the dicts are constructed. Essentially if you query for fields a,b,c,d for each row you'll get:

- With depth 1: {a: {``b'': b, ``c'': c, ``d'': d}}
- With depth 2: {a: {b: {``c'': c, ``d'': d}}}
- With depth 3: {a: {b: {c: d}}}

Depth greater than 3 wouldn't be different from 3 itself. Depth of 0 translates to the largest depth needed, so 3 in this case. (max depth == key count - 1)

The legacy compatibility translates to depth 1.

Then they are merged in a similar way to plain pillar data, in the order returned by MySQL.

Thus subsequent results overwrite previous ones when they collide.

The ignore_null option can be used to change the overwrite behavior so that only non-NULL values in subsequent results will overwrite. This can be used to selectively overwrite default values.

```
ext_pillar:
  - mysql:
      - query: "SELECT pillar,value FROM pillars WHERE minion_id = 'default' and minion_id != %s"
        depth: 2
      - query: "SELECT pillar,value FROM pillars WHERE minion_id = %s"
        depth: 2
        ignore_null: True
```

If you specify *as_list: True* in the mapping expression it will convert collisions to lists.

If you specify *with_lists: `...'* in the mapping expression it will convert the specified depths to list. The string provided is a sequence numbers that are comma separated. The string `1,3' will result in:

```
a,b,c,d,e,1  # field 1 same, field 3 differs
a,b,c,f,g,2  # ^^^^
a,z,h,y,j,3  # field 1 same, field 3 same
a,z,h,y,k,4  # ^^^^
  ^   ^
```

These columns define list grouping

```
{a: [
      {c: [
          {e: 1},
          {g: 2}
          ]
      },
      {h: [
          {j: 3, k: 4 }
          ]
      }
]}
```

The range for with_lists is 1 to number_of_fields, inclusive. Numbers outside this range are ignored.

Finally, if you pass the queries in via a mapping, the key will be the first level name where as passing them in as a list will place them in the root. This isolates the query results into their own subtrees. This may be a help or hindrance to your aims and can be used as such.

You can basically use any SELECT query that gets you the information, you could even do joins or subqueries in case your minion_id is stored elsewhere. It is capable of handling single rows or multiple rows per minion.

MySQL configuration of the MySQL returner is being used (mysql.db, mysql.user, mysql.pass, mysql.port, mysql.host)

Required python modules: MySQLdb

More complete example

```
mysql:
  user: 'salt'
  pass: 'super_secret_password'
  db: 'salt_db'

ext_pillar:
  - mysql:
      fromdb:
        query: 'SELECT col1,col2,col3,col4,col5,col6,col7
                  FROM some_random_table
                 WHERE minion_pattern LIKE %s'
        depth: 5
        as_list: True
        with_lists: [1,3]
```

salt.pillar.mysql.**ext_pillar**(*minion_id*, *pillar*, **args*, ***kwargs*)
Execute queries, merge and return as a dict

class salt.pillar.mysql.**merger**
This class receives and processes the database rows in a database agnostic way.

as_list = False

depth = 0

enter_root(*root*)
Set self.focus for kwarg queries

extract_queries(*args*, *kwargs*)
This function normalizes the config block into a set of queries we can use. The return is a list of consistently laid out dicts.

field_names = None

focus = None

ignore_null = False

num_fields = 0

process_fields(*field_names*, *depth*)
The primary purpose of this function is to store the sql field list and the depth to which we process.

process_results(*rows*)
This function takes a list of database results and iterates over, merging them into a dict form.

result = None

with_lists = None

22.21.15 salt.pillar.pepa

Pepa

Configuration templating for SaltStack using Hierarchical substitution and Jinja.

Configuring Pepa

```
extension_modules: /srv/salt/ext

ext_pillar:
  - pepa:
      resource: host                # Name of resource directory and sub-key in pillars
      sequence:                     # Sequence used for hierarchical substitution
        - hostname:                 # Name of key
            name: input             # Alias used for template directory
            base_only: True         # Only use templates from Base environment, i.e. no staging
        - default:
        - environment:
        - location..region:
            name: region
        - location..country:
            name: country
        - location..datacenter:
            name: datacenter
        - roles:
        - osfinger:
            name: os
        - hostname:
            name: override
            base_only: True
      subkey: True                  # Create a sub-key in pillars, named after the resource in this c
      subkey_only: True             # Only create a sub-key, and leave the top level untouched

pepa_roots:                         # Base directory for each environment
  base: /srv/pepa/base              # Path for base environment
  dev: /srv/pepa/base               # Associate dev with base
  qa: /srv/pepa/qa
  prod: /srv/pepa/prod

# Use a different delimiter for nested dictionaries, defaults to '..' since some keys may use '.' in
#pepa_delimiter: ..

# Supply Grains for Pepa, this should **ONLY** be used for testing or validation
#pepa_grains:
#  environment: dev

# Supply Pillar for Pepa, this should **ONLY** be used for testing or validation
#pepa_pillars:
#  saltversion: 0.17.4

# Enable debug for Pepa, and keep Salt on warning
#log_level: debug

#log_granular_levels:
```

```
#  salt: warning
#  salt.loaded.ext.pillar.pepa: debug
```

Pepa can also be used in Master-less SaltStack setup.

Command line

```
usage: pepa.py [-h] [-c CONFIG] [-d] [-g GRAINS] [-p PILLAR] [-n] [-v]
               hostname

positional arguments:
  hostname              Hostname

optional arguments:
  -h, --help            show this help message and exit
  -c CONFIG, --config CONFIG
                        Configuration file
  -d, --debug           Print debug info
  -g GRAINS, --grains GRAINS
                        Input Grains as YAML
  -p PILLAR, --pillar PILLAR
                        Input Pillar as YAML
  -n, --no-color        No color output
  -v, --validate        Validate output
```

Templates

Templates is configuration for a host or software, that can use information from Grains or Pillars. These can then be used for hierarchically substitution.

Example File: host/input/test_example_com.yaml

```
location..region: emea
location..country: nl
location..datacenter: foobar
environment: dev
roles:
  - salt.master
network..gateway: 10.0.0.254
network..interfaces..eth0..hwaddr: 00:20:26:a1:12:12
network..interfaces..eth0..dhcp: False
network..interfaces..eth0..ipv4: 10.0.0.3
network..interfaces..eth0..netmask: 255.255.255.0
network..interfaces..eth0..fqdn: {{ hostname }}
cobbler..profile: fedora-19-x86_64
```

As you see in this example you can use Jinja directly inside the template.

Example File: host/region/amer.yaml

```
network..dns..servers:
  - 10.0.0.1
  - 10.0.0.2
time..ntp..servers:
```

```
  - ntp1.amer.example.com
  - ntp2.amer.example.com
  - ntp3.amer.example.com
time..timezone: America/Chihuahua
yum..mirror: yum.amer.example.com
```

Each template is named after the value of the key using lowercase and all extended characters are replaced with underscore.

Example:

osfinger: Fedora-19

Would become:

fedora_19.yaml

Nested dictionaries

In order to create nested dictionaries as output you can use double dot ".." as a delimiter. You can change this using ``pepa_delimiter'' we choose double dot since single dot is already used by key names in some modules, and using ":" requires quoting in the YAML.

Example:

```
network..dns..servers:
  - 10.0.0.1
  - 10.0.0.2
network..dns..options:
  - timeout:2
  - attempts:1
  - ndots:1
network..dns..search:
  - example.com
```

Would become:

```
network:
  dns:
    servers:
      - 10.0.0.1
      - 10.0.0.2
    options:
      - timeout:2
      - attempts:1
      - ndots:1
    search:
      - example.com
```

Operators

Operators can be used to merge/unset a list/hash or set the key as immutable, so it can't be changed.

Operator	Description
merge()	Merge list or hash
unset()	Unset key
immutable()	Set the key as immutable, so it can't be changed
imerge()	Set immutable and merge
iunset()	Set immutable and unset

Example:

```
network..dns..search..merge():
  - foobar.com
  - dummy.nl
owner..immutable(): Operations
host..printers..unset():
```

Validation

Since it's very hard to test Jinja as is, the best approach is to run all the permutations of input and validate the output, i.e. Unit Testing.

To facilitate this in Pepa we use YAML, Jinja and Cerberus <https://github.com/nicolaiarocci/cerberus>.

Schema

So this is a validation schema for network configuration, as you see it can be customized with Jinja just as Pepa templates.

This was designed to be run as a build job in Jenkins or similar tool. You can provide Grains/Pillar input using either the config file or command line arguments.

File Example: **host/validation/network.yaml**

```
network..dns..search:
  type: list
  allowed:
    - example.com

network..dns..options:
  type: list
  allowed: ['timeout:2', 'attempts:1', 'ndots:1']

network..dns..servers:
  type: list
  schema:
    regex: ^([0-9]{1,3}\.){3}[0-9]{1,3}$

network..gateway:
  type: string
  regex: ^([0-9]{1,3}\.){3}[0-9]{1,3}$

{% if network.interfaces is defined %}
{% for interface in network.interfaces %}

network..interfaces..{{ interface }}..dhcp:
  type: boolean
```

```
network..interfaces..{{ interface }}..fqdn:
  type: string
  regex: ^([a-z0-9]([a-z0-9-]{0,61}[a-z0-9])?\.)+[a-zA-Z]{2,6}$

network..interfaces..{{ interface }}..hwaddr:
  type: string
  regex: ^([0-9a-f]{1,2}\:){5}[0-9a-f]{1,2}$

network..interfaces..{{ interface }}..ipv4:
  type: string
  regex: ^([0-9]{1,3}\.){3}[0-9]{1,3}$

network..interfaces..{{ interface }}..netmask:
  type: string
  regex: ^([0-9]{1,3}\.){3}[0-9]{1,3}$

{% endfor %}
{% endif %}
```

Links

For more examples and information see <https://github.com/mickep76/pepa>.

salt.pillar.pepa.**ext_pillar**(*minion_id*, *pillar*, *resource*, *sequence*, *subkey=False*, *subkey_only=False*)
Evaluate Pepa templates

salt.pillar.pepa.**key_value_to_tree**(*data*)
Convert key/value to tree

salt.pillar.pepa.**validate**(*output*, *resource*)
Validate Pepa templates

22.21.16 salt.pillar.pillar_ldap

Use LDAP data as a Pillar source

This pillar module parses a config file (specified in the salt master config), and executes a series of LDAP searches based on that config. Data returned by these searches is aggregated, with data items found later in the LDAP search order overriding data found earlier on.

The final result set is merged with the pillar data.

salt.pillar.pillar_ldap.**ext_pillar**(*minion_id*, *pillar*, *config_file*)
Execute LDAP searches and return the aggregated data

22.21.17 salt.pillar.puppet

Execute an unmodified puppet_node_classifier and read the output as YAML. The YAML data is then directly overlaid onto the minion's Pillar data.

salt.pillar.puppet.**ext_pillar**(*minion_id*, *pillar*, *command*)
Execute an unmodified puppet_node_classifier and read the output as YAML

22.21.18 salt.pillar.reclass_adapter

Use the ``reclass'' database as a Pillar source

This `ext_pillar` plugin provides access to the **reclass** database, such that Pillar data for a specific minion are fetched using **reclass**.

You can find more information about **reclass** at http://reclass.pantsfullofunix.net.

To use the plugin, add it to the `ext_pillar` list in the Salt master config and tell **reclass** by way of a few options how and where to find the inventory:

```
ext_pillar:
    - reclass:
        storage_type: yaml_fs
        inventory_base_uri: /srv/salt
```

This would cause **reclass** to read the inventory from YAML files in `/srv/salt/nodes` and `/srv/salt/classes`.

If you are also using **reclass** as `master_tops` plugin, and you want to avoid having to specify the same information for both, use YAML anchors (take note of the differing data types for `ext_pillar` and `master_tops`):

```
reclass: &reclass
    storage_type: yaml_fs
    inventory_base_uri: /srv/salt
    reclass_source_path: ~/code/reclass

ext_pillar:
    - reclass: *reclass

master_tops:
    reclass: *reclass
```

If you want to run reclass from source, rather than installing it, you can either let the master know via the `PYTHONPATH` environment variable, or by setting the configuration option, like in the example above.

`salt.pillar.reclass_adapter.`**`ext_pillar`**`(`*minion_id*, *pillar*, ***kwargs*`)`
Obtain the Pillar data from **reclass** for the given `minion_id`.

22.21.19 salt.pillar.redismod

Read pillar data from a Redis backend

New in version 2014.7.0.

depends

- redis Python module (on master)

Salt Master Redis Configuration

The module shares the same base Redis connection variables as `salt.returners.redis_return`. These variables go in your master config file.

- `redis.db` - The Redis database to use. Defaults to `0`.

- `redis.host` - The Redis host to connect to. Defaults to `'salt'`.
- `redis.port` - The port that the Redis database is listening on. Defaults to `6379`.
- `redis.password` - The password for authenticating with Redis. Only required if you are using master auth. Defaults to `None`.

Configuring the Redis ext_pillar

```
    ext_pillar:
      - redis: {function: key_value}
```

`salt.pillar.redismod.`**`ext_pillar`**(*minion_id*, *pillar*, *function*, ***kwargs*)
Grabs external pillar data based on configured function

`salt.pillar.redismod.`**`key_json`**(*minion_id*, *pillar*, *pillar_key=None*)
Pulls a string from redis and deserializes it from json. Deserialized dictionary data loaded directly into top level if pillar_key is not set.

pillar_key Pillar key to return data into

`salt.pillar.redismod.`**`key_value`**(*minion_id*, *pillar*, *pillar_key='redis_pillar'*)
Looks for key in redis matching minion_id, returns a structure based on the data type of the redis key. String for string type, dict for hash type and lists for lists, sets and sorted sets.

pillar_key Pillar key to return data into

22.21.20 salt.pillar.s3

Copy pillar data from a bucket in Amazon S3

The S3 pillar can be configured in the master config file with the following options

```
ext_pillar:
  - s3:
      bucket: my.fancy.pillar.bucket
      keyid: KASKFJWAKJASJKDAJKSD
      key: ksladfDLKDALSFKSD93q032sdDasdfasdflsadkf
      multiple_env: False
      environment: base
      prefix: somewhere/overthere
      verify_ssl: True
      service_url: s3.amazonaws.com
```

The `bucket` parameter specifies the target S3 bucket. It is required.

The `keyid` parameter specifies the key id to use when access the S3 bucket. It is required.

The `key` parameter specifies the key to use when access the S3 bucket. It is required.

The `multiple_env` defaults to False. It specifies whether the pillar should interpret top level folders as pillar environments (see mode section below).

The `environment` defaults to `base'. It specifies which environment the bucket represents when in single environments mode (see mode section below). It is ignored if multiple_env is True.

The `prefix` defaults to ``. It specifies a key prefix to use when searching for data in the bucket for the pillar. It works when multiple_env is True or False. Essentially it tells ext_pillar to look for your pillar data in a `subdirectory' of your S3 bucket

The `verify_ssl` parameter defaults to True. It specifies whether to check for valid S3 SSL certificates. *NOTE* If you use bucket names with periods, this must be set to False else an invalid certificate error will be thrown (issue #12200).

The `service_url` parameter defaults to `s3.amazonaws.com'. It specifies the base url to use for accessing S3.

This pillar can operate in two modes, single environment per bucket or multiple environments per bucket.

Single environment mode must have this bucket structure:

```
s3://<bucket name>/<prefix>/<files>
```

Multiple environment mode must have this bucket structure:

```
s3://<bucket name>/<prefix>/<environment>/<files>
```

If you wish to define your pillar data entirely within S3 it's recommended that you use the *prefix=* parameter and specify one entry in ext_pillar for each environment rather than specifying multiple_env. This is due to issue #22471 (https://github.com/saltstack/salt/issues/22471)

class `salt.pillar.s3.`**`S3Credentials`**(*key*, *keyid*, *bucket*, *service_url*, *verify_ssl=True*)

`salt.pillar.s3.`**`ext_pillar`**(*minion_id*, *pillar*, *bucket*, *key*, *keyid*, *verify_ssl*, *multiple_env=False*, *environment='base'*, *prefix=''*, *service_url=None*)

Execute a command and read the output as YAML

22.21.21 salt.pillar.svn_pillar

Clone a remote SVN repository and use the filesystem as a Pillar source

This external Pillar source can be configured in the master config file like so:

```
ext_pillar:
  - svn: trunk svn://svnserver/repo root=subdirectory
```

The *root=* parameter is optional and used to set the subdirectory from where to look for Pillar files (such as `top.sls`).

Changed in version 2014.7.0: The optional `root` parameter will be added.

Note that this is not the same thing as configuring pillar data using the `pillar_roots` parameter. The branch referenced in the `ext_pillar` entry above (`master`), would evaluate to the `base` environment, so this branch needs to contain a `top.sls` with a `base` section in it, like this:

```
base:
  '*':
    - foo
```

To use other environments from the same SVN repo as svn_pillar sources, just add additional lines, like so:

```
ext_pillar:
  - svn: trunk svn://svnserver/repo
  - svn: dev svn://svnserver/repo
```

In this case, the `dev` branch would need its own `top.sls` with a `dev` section in it, like this:

```
dev:
  '*':
    - bar
```

class salt.pillar.svn_pillar.**SvnPillar**(*branch*, *repo_location*, *root*, *opts*)
Deal with the remote SVN repository for Pillar

pillar_dir()
Returns the directory of the pillars (repo cache + branch + root)

update()

salt.pillar.svn_pillar.**ext_pillar**(*minion_id*, *pillar*, *repo_string*)
Execute a command and read the output as YAML

22.21.22 salt.pillar.varstack_pillar

Use varstack data as a Pillar source

salt.pillar.varstack_pillar.**ext_pillar**(*minion_id*, *pillar*, *conf*)
Parse varstack data and return the result

22.21.23 salt.pillar.virtkey

Accept a key from a hypervisor if the virt runner has already submitted an authorization request

salt.pillar.virtkey.**ext_pillar**(*hyper_id*, *pillar*, *name*, *key*)
Accept the key for the VM on the hyper, if authorized.

22.22 Renderers

The Salt state system operates by gathering information from common data types such as lists, dictionaries, and strings that would be familiar to any developer.

SLS files are translated from whatever data templating format they are written in back into Python data types to be consumed by Salt.

By default SLS files are rendered as Jinja templates and then parsed as YAML documents. But since the only thing the state system cares about is raw data, the SLS files can be any structured format that can be dreamed up.

Currently there is support for `Jinja + YAML`, `Mako + YAML`, `Wempy + YAML`, `Jinja + json`, `Mako + json` and `Wempy + json`.

Renderers can be written to support any template type. This means that the Salt states could be managed by XML files, HTML files, Puppet files, or any format that can be translated into the Pythonic data structure used by the state system.

22.22.1 Multiple Renderers

A default renderer is selected in the master configuration file by providing a value to the `renderer` key.

When evaluating an SLS, more than one renderer can be used.

When rendering SLS files, Salt checks for the presence of a Salt-specific shebang line.

The shebang line directly calls the name of the renderer as it is specified within Salt. One of the most common reasons to use multiple renderers is to use the Python or `py` renderer.

Below, the first line is a shebang that references the `py` renderer.

```
#!py

def run():
    '''
    Install the python-mako package
    '''
    return {'include': ['python'],
            'python-mako': {'pkg': ['installed']}}
```

22.22.2 Composing Renderers

A renderer can be composed from other renderers by connecting them in a series of pipes(`|`).

In fact, the default `Jinja + YAML` renderer is implemented by connecting a YAML renderer to a Jinja renderer. Such renderer configuration is specified as: `jinja | yaml`.

Other renderer combinations are possible:

`yaml` i.e, just YAML, no templating.

`mako | yaml` pass the input to the `mako` renderer, whose output is then fed into the `yaml` renderer.

`jinja | mako | yaml` This one allows you to use both jinja and mako templating syntax in the input and then parse the final rendered output as YAML.

The following is a contrived example SLS file using the `jinja | mako | yaml` renderer:

```
#!jinja|mako|yaml

An_Example:
  cmd.run:
    - name: |
        echo "Using Salt ${grains['saltversion']}" \
             "from path {{grains['saltpath']}}."
    - cwd: /

<%doc> ${...} is Mako's notation, and so is this comment. </%doc>
{#     Similarly, {{...}} is Jinja's notation, and so is this comment. #}
```

For backward compatibility, `jinja | yaml` can also be written as `yaml_jinja`, and similarly, the `yaml_mako`, `yaml_wempy`, `json_jinja`, `json_mako`, and `json_wempy` renderers are all supported.

Keep in mind that not all renderers can be used alone or with any other renderers. For example, the template renderers shouldn't be used alone as their outputs are just strings, which still need to be parsed by another renderer to turn them into highstate data structures.

For example, it doesn't make sense to specify `yaml | jinja` because the output of the YAML renderer is a highstate data structure (a dict in Python), which cannot be used as the input to a template renderer. Therefore, when combining renderers, you should know what each renderer accepts as input and what it returns as output.

22.22.3 Writing Renderers

A custom renderer must be a Python module placed in the renderers directory and the module implement the `render` function.

The `render` function will be passed the path of the SLS file as an argument.

The purpose of of `render` function is to parse the passed file and to return the Python data structure derived from the file.

Custom renderers must be placed in a `_renderers` directory within the `file_roots` specified by the master config file.

Custom renderers are distributed when any of the following are run: `state.highstate`

`saltutil.sync_renderers`

`saltutil.sync_all`

Any custom renderers which have been synced to a minion, that are named the same as one of Salt's default set of renderers, will take the place of the default renderer with the same name.

22.22.4 Examples

The best place to find examples of renderers is in the Salt source code.

Documentation for renderers included with Salt can be found here:

https://github.com/saltstack/salt/blob/develop/salt/renderers

Here is a simple YAML renderer example:

```
import yaml
def render(yaml_data, env='', sls='', **kws):
    if not isinstance(yaml_data, basestring):
        yaml_data = yaml_data.read()
    data = yaml.load(yaml_data)
    return data if data else {}
```

22.22.5 Full List of Renderers

Full list of builtin renderer modules

`cheetah`	Cheetah Renderer for Salt
`genshi`	Genshi Renderer for Salt
`gpg`	Renderer that will decrypt GPG ciphers
`jinja`	Jinja loading utils to enable a more powerful backend for jinja templates
`json`	JSON Renderer for Salt
`mako`	Mako Renderer for Salt
`msgpack`	
`py`	Pure python state renderer
`pydsl`	A Python-based DSL
`pyobjects`	Python renderer that includes a Pythonic Object based interface
`stateconf`	A flexible renderer that takes a templating engine and a data format

Continued on next page

Table 22.9 -- continued from previous page

wempy	
yaml	YAML Renderer for Salt
yamlex	

salt.renderers.cheetah

Cheetah Renderer for Salt

salt.renderers.cheetah.**render**(*cheetah_data*, *saltenv='base'*, *sls=''*, *method='xml'*, ***kws*)

Render a Cheetah template.

Return type A Python data structure

salt.renderers.genshi

Genshi Renderer for Salt

salt.renderers.genshi.**render**(*genshi_data*, *saltenv='base'*, *sls=''*, *method='xml'*, ***kws*)

Render a Genshi template. A method should be passed in as part of the kwargs. If no method is passed in, xml is assumed. Valid methods are:

Note that the `text` method will call `NewTextTemplate`. If `oldtext` is desired, it must be called explicitly

Return type A Python data structure

salt.renderers.gpg

Renderer that will decrypt GPG ciphers

Any key in the SLS file can be a GPG cipher, and this renderer will decrypt it before passing it off to Salt. This allows you to safely store secrets in source control, in such a way that only your Salt master can decrypt them and distribute them only to the minions that need them.

The typical use-case would be to use ciphers in your pillar data, and keep a secret key on your master. You can put the public key in source control so that developers can add new secrets quickly and easily.

This renderer requires the python-gnupg package. Be careful to install the `python-gnupg` package, not the `gnupg` package, or you will get errors.

To set things up, you will first need to generate a keypair. On your master, run:

```
# gpg --gen-key --homedir /etc/salt/gpgkeys
```

Do not supply a password for your keypair, and use a name that makes sense for your application. Be sure to back up your gpg directory someplace safe!

To retrieve the public key:

```
# gpg --armor --homedir /etc/salt/gpgkeys --armor --export <KEY-NAME>           > exported_pubkey.gpg
```

Now, to encrypt secrets, copy the public key to your local machine and run:

```
$ gpg --import exported_pubkey.gpg
```

To generate a cipher from a secret:

```
$ echo -n "supersecret" | gpg --homedir --armor --encrypt -r <KEY-name>
```

Set up the renderer on your master by adding something like this line to your config:

```
renderer: jinja | yaml | gpg
```

Now you can include your ciphers in your pillar data like so:

```
a-secret: |
  -----BEGIN PGP MESSAGE-----
  Version: GnuPG v1

  hQEMAweRHKaPCfNeAQf9GLTN16hCfXAbPwU6BbBK0unOc7i9/etGuVc5CyU9Q6um
  QuetdvQVLFO/HkrC4lgeNQdM6D9E8PKonMlgJPyUvC8ggxhj0/IPFEKmrsnv2k6+
  cnEfmVexS7o/U1VOVjoyUeliMCJlAz/30RXaME49Cpi6No2+vKD8a4q4nZN1UZcG
  RhkhC0S22zNxOXQ38TBkmtJcqxnqT6YWKTUsjVubW3bVC+u2HGqJHu79wmwuN8tz
  m4wBkfCAd8Eyo2jEnWQcM4TcXiF01XPL4z4g1/9AAxh+Q4d8RIRP4fbw7ct4nCJv
  Gr9v2DTF7HNigIMl4ivMIn9fp+EZurJNiQskLgNbktJGAeEKYkqX5iCuB1b693hJ
  FKlwHiJt5yA8X2dDtfk8/Ph1Jx2TwGS+lGjlZaNqp3R1xuAZzXzZMLyZDe5+i3RJ
  skqmFTbOiA==
  =Eqsm
  -----END PGP MESSAGE-----
```

`salt.renderers.gpg.`**`decrypt_ciphertext`**(*c*, *gpg*)
: Given a block of ciphertext as a string, and a gpg object, try to decrypt the cipher and return the decrypted string. If the cipher cannot be decrypted, log the error, and return the ciphertext back out.

`salt.renderers.gpg.`**`decrypt_object`**(*o*, *gpg*)
: Recursively try to decrypt any object. If the object is a string, and it contains a valid GPG header, decrypt it, otherwise keep going until a string is found.

`salt.renderers.gpg.`**`render`**(*gpg_data*, *saltenv='base'*, *sls=''*, *argline=''*, ***kwargs*)
: Create a gpg object given a gpg_keydir, and then use it to try to decrypt the data to be rendered.

salt.renderers.jinja

Jinja loading utils to enable a more powerful backend for jinja templates

Jinja in States The most basic usage of Jinja in state files is using control structures to wrap conditional or redundant state elements:

```
{% if grains['os'] != 'FreeBSD' %}
tcsh:
    pkg:
        - installed
{% endif %}

motd:
  file.managed:
    {% if grains['os'] == 'FreeBSD' %}
    - name: /etc/motd
    {% elif grains['os'] == 'Debian' %}
```

```
    - name: /etc/motd.tail
    {% endif %}
    - source: salt://motd
```

In this example, the first if block will only be evaluated on minions that aren't running FreeBSD, and the second block changes the file name based on the *os* grain.

Writing **if-else** blocks can lead to very redundant state files however. In this case, using *pillars*, or using a previously defined variable might be easier:

```
{% set motd = ['/etc/motd'] %}
{% if grains['os'] == 'Debian' %}
  {% set motd = ['/etc/motd.tail', '/var/run/motd'] %}
{% endif %}

{% for motdfile in motd %}
{{ motdfile }}:
  file.managed:
    - source: salt://motd
{% endfor %}
```

Using a variable set by the template, the for loop will iterate over the list of MOTD files to update, adding a state block for each file.

Include and Import Includes and imports can be used to share common, reusable state configuration between state files and between files.

```
{% from 'lib.sls' import test %}
```

This would import the `test` template variable or macro, not the `test` state element, from the file `lib.sls`. In the case that the included file performs checks again grains, or something else that requires context, passing the context into the included file is required:

```
{% from 'lib.sls' import test with context %}
```

Macros Macros are helpful for eliminating redundant code, however stripping whitespace from the template block, as well as contained blocks, may be necessary to emulate a variable return from the macro.

```
# init.sls
{% from 'lib.sls' import pythonpkg with context %}

python-virtualenv:
  pkg.installed:
    - name: {{ pythonpkg('virtualenv') }}

python-fabric:
  pkg.installed:
    - name: {{ pythonpkg('fabric') }}
```

```
# lib.sls
{% macro pythonpkg(pkg) -%}
  {%- if grains['os'] == 'FreeBSD' -%}
    py27-{{ pkg }}
  {%- elif grains['os'] == 'Debian' -%}
    python-{{ pkg }}
  {%- endif -%}
{%- endmacro %}
```

This would define a macro that would return a string of the full package name, depending on the packaging system's naming convention. The whitespace of the macro was eliminated, so that the macro would return a string without line breaks, using whitespace control.

Template Inheritance Template inheritance works fine from state files and files. The search path starts at the root of the state tree or pillar.

Filters Saltstack extends builtin filters with these custom filters:

strftime Converts any time related object into a time based string. It requires a valid *strftime directives*. An *exhaustive list* can be found in the official Python documentation.

```
    {% set curtime = None | strftime() %}
```

Fuzzy dates require the timelib Python module is installed.

```
    {{ "2002/12/25"|strftime("%y") }}
    {{ "1040814000"|strftime("%Y-%m-%d") }}
    {{ datetime|strftime("%u") }}
    {{ "tomorrow"|strftime }}
```

sequence Ensure that parsed data is a sequence.

yaml_encode Serializes a single object into a YAML scalar with any necessary handling for escaping special characters. This will work for any scalar YAML data type: ints, floats, timestamps, booleans, strings, unicode. It will *not* work for multi-objects such as sequences or maps.

```
       {%- set bar = 7 %}
       {%- set baz = none %}
       {%- set zip = true %}
       {%- set zap = 'The word of the day is "salty"' %}

       {%- load_yaml as foo %}
       bar: {{ bar|yaml_encode }}
       baz: {{ baz|yaml_encode }}
       baz: {{ zip|yaml_encode }}
       baz: {{ zap|yaml_encode }}
       {%- endload %}

    In the above case ``{{ bar }}`` and ``{{ foo.bar }}`` should be
    identical and ``{{ baz }}`` and ``{{ foo.baz }}`` should be
    identical.
```

yaml_dquote Serializes a string into a properly-escaped YAML double-quoted string. This is useful when the contents of a string are unknown and may contain quotes or unicode that needs to be preserved. The resulting string will be emitted with opening and closing double quotes.

```
        {%- set bar = '"The quick brown fox . . ."' %}
        {%- set baz = 'The word of the day is "salty".' %}

        {%- load_yaml as foo %}
        bar: {{ bar|yaml_dquote }}
        baz: {{ baz|yaml_dquote }}
        {%- endload %}

    In the above case ``{{ bar }}`` and ``{{ foo.bar }}`` should be
    identical and ``{{ baz }}`` and ``{{ foo.baz }}`` should be
    identical.  If variable contents are not guaranteed to be a string
    then it is better to use ``yaml_encode`` which handles all YAML
    scalar types.
```

yaml_squote Similar to the `yaml_dquote` filter but with single quotes. Note that YAML only allows special escapes inside double quotes so `yaml_squote` is not nearly as useful (viz. you likely want to use `yaml_encode` or `yaml_dquote`).

Jinja in Files Jinja can be used in the same way in managed files:

```
# redis.sls
/etc/redis/redis.conf:
    file.managed:
        - source: salt://redis.conf
        - template: jinja
        - context:
            bind: 127.0.0.1
```

```
# lib.sls
{% set port = 6379 %}
```

```
# redis.conf
{% from 'lib.sls' import port with context %}
port {{ port }}
bind {{ bind }}
```

As an example, configuration was pulled from the file context and from an external template file.

Note: Macros and variables can be shared across templates. They should not be starting with one or more underscores, and should be managed by one of the following tags: *macro*, *set*, *load_yaml*, *load_json*, *import_yaml* and *import_json*.

Calling Salt Functions The Jinja renderer provides a shorthand lookup syntax for the `salt` dictionary of *execution function*.

New in version 2014.7.0.

```
# The following two function calls are equivalent.
{{ salt['cmd.run']('whoami') }}
{{ salt.cmd.run('whoami') }}
```

Debugging The `show_full_context` function can be used to output all variables present in the current Jinja context.

New in version 2014.7.0.

```
Context is: {{ show_full_context() }}
```

`salt.renderers.jinja.`**`render`**(*template_file*, *saltenv='base'*, *sls=''*, *argline=''*, *context=None*, *tmplpath=None*, ***kws*)

Render the template_file, passing the functions and grains into the Jinja rendering system.

Return type string

class `salt.utils.jinja.`**`SerializerExtension`**(*environment*)

Yaml and Json manipulation.

Format filters

Allows to jsonify or yamlify any data structure. For example, this dataset:

```
data = {
    'foo': True,
    'bar': 42,
    'baz': [1, 2, 3],
    'qux': 2.0
}
```

```
yaml = {{ data|yaml }}
json = {{ data|json }}
python = {{ data|python }}
```

will be rendered as:

```
yaml = {bar: 42, baz: [1, 2, 3], foo: true, qux: 2.0}
json = {"baz": [1, 2, 3], "foo": true, "bar": 42, "qux": 2.0}
python = {'bar': 42, 'baz': [1, 2, 3], 'foo': True, 'qux': 2.0}
```

The yaml filter takes an optional flow_style parameter to control the default-flow-style parameter of the YAML dumper.

```
{{ data|yaml(False) }}
```

will be rendered as:

```
bar: 42
baz:
  - 1
  - 2
  - 3
foo: true
qux: 2.0
```

Load filters

Strings and variables can be deserialized with **load_yaml** and **load_json** tags and filters. It allows one to manipulate data directly in templates, easily:

```
{%- set yaml_src = "{foo: it works}"|load_yaml %}
{%- set json_src = "{'bar': 'for real'}"|load_json %}
Dude, {{ yaml_src.foo }} {{ json_src.bar }}!
```

will be rendered has:

```
Dude, it works for real!
```

Load tags

Salt implements **import_yaml** and **import_json** tags. They work like the import tag, except that the document is also deserialized.

Syntaxes are {% load_yaml as [VARIABLE] %}[YOUR DATA]{% endload %} and {% load_json as [VARIABLE] %}[YOUR DATA]{% endload %}

For example:

```
{% load_yaml as yaml_src %}
    foo: it works
{% endload %}
{% load_json as json_src %}
    {
        "bar": "for real"
    }
{% endload %}
Dude, {{ yaml_src.foo }} {{ json_src.bar }}!
```

will be rendered has:

```
Dude, it works for real!
```

Import tags

External files can be imported and made available as a Jinja variable.

```
{% import_yaml "myfile.yml" as myfile %}
{% import_json "defaults.json" as defaults %}
{% import_text "completeworksofshakespeare.txt" as poems %}
```

Catalog

`import_*` and `load_*` tags will automatically expose their target variable to import. This feature makes catalog of data to handle.

for example:

```
# doc1.sls
{% load_yaml as var1 %}
    foo: it works
{% endload %}
{% load_yaml as var2 %}
    bar: for real
{% endload %}
```

```
# doc2.sls
{% from "doc1.sls" import var1, var2 as local2 %}
{{ var1.foo }} {{ local2.bar }}
```

salt.renderers.json

JSON Renderer for Salt

salt.renderers.json.**render**(*json_data*, *saltenv='base'*, *sls=''*, ***kws*)
Accepts JSON as a string or as a file object and runs it through the JSON parser.

Return type A Python data structure

salt.renderers.mako

Mako Renderer for Salt

salt.renderers.mako.**render**(*template_file*, *saltenv='base'*, *sls=''*, *context=None*, *tmplpath=None*, ***kws*)
Render the template_file, passing the functions and grains into the Mako rendering system.

Return type string

salt.renderers.msgpack

salt.renderers.msgpack.**render**(*msgpack_data*, *saltenv='base'*, *sls=''*, ***kws*)
Accepts a message pack string or a file object, renders said data back to a python dict.

Return type A Python data structure

salt.renderers.py

Pure python state renderer

The SLS file should contain a function called `run` which returns high state data.

In this module, a few objects are defined for you, giving access to Salt's execution functions, grains, pillar, etc. They are:

- `__salt__` - *Execution functions* (i.e. `__salt__['test.echo']('foo')`)
- `__grains__` - *Grains* (i.e. `__grains__['os']`)
- `__pillar__` - *Pillar data* (i.e. `__pillar__['foo']`)
- `__opts__` - Minion configuration options
- `__env__` - The effective salt fileserver environment (i.e. `base`). Also referred to as a ``saltenv''. `__env__` should not be modified in a pure python SLS file. To use a different environment, the environment should be set when executing the state. This can be done in a couple different ways:
  - Using the `saltenv` argument on the salt CLI (i.e. `salt '*' state.sls foo.bar.baz saltenv=env_name`).
  - By adding a `saltenv` argument to an individual state within the SLS file. In other words, adding a line like this to the state's data structure: `{'saltenv': 'env_name'}`
- `__sls__` - The SLS path of the file. For example, if the root of the base environment is `/srv/salt`, and the SLS file is `/srv/salt/foo/bar/baz.sls`, then `__sls__` in that file will be `foo.bar.baz`.

```
#!py

def run():
    config = {}

    if __grains__['os'] == 'Ubuntu':
        user = 'ubuntu'
```

```
        group = 'ubuntu'
        home = '/home/{0}'.format(user)
    else:
        user = 'root'
        group = 'root'
        home = '/root/'

    config['s3cmd'] = {
        'pkg': [
            'installed',
            {'name': 's3cmd'},
        ],
    }

    config[home + '/.s3cfg'] = {
        'file.managed': [
            {'source': 'salt://s3cfg/templates/s3cfg'},
            {'template': 'jinja'},
            {'user': user},
            {'group': group},
            {'mode': 600},
            {'context': {
                'aws_key': __pillar__['AWS_ACCESS_KEY_ID'],
                'aws_secret_key': __pillar__['AWS_SECRET_ACCESS_KEY'],
                },
            },
        ],
    }

    return config
```

salt.renderers.py.**render**(*template*, *saltenv='base'*, *sls=''*, *tmplpath=None*, ***kws*)

Render the python module's components

Return type string

salt.renderers.pydsl

A Python-based DSL

maintainer Jack Kuan <kjkuan@gmail.com>

maturity new

platform all

The *pydsl* renderer allows one to author salt formulas (.sls files) in pure Python using a DSL that's easy to write and easy to read. Here's an example:

```
#!pydsl

apache = state('apache')
apache.pkg.installed()
apache.service.running()
state('/var/www/index.html') \
    .file('managed',
          source='salt://webserver/index.html') \
    .require(pkg='apache')
```

Notice that any Python code is allow in the file as it's really a Python module, so you have the full power of Python at your disposal. In this module, a few objects are defined for you, including the usual (with `__` added) `__salt__` dictionary, `__grains__`, `__pillar__`, `__opts__`, `__env__`, and `__sls__`, plus a few more:

`__file__`

local file system path to the sls module.

`__pydsl__`

Salt PyDSL object, useful for configuring DSL behavior per sls rendering.

`include`

Salt PyDSL function for creating *Include declaration*`s.

`extend`

Salt PyDSL function for creating *Extend declaration*`s.

`state`

Salt PyDSL function for creating *ID declaration*`s.

A state *ID declaration* is created with a `state(id)` function call. Subsequent `state(id)` call with the same id returns the same object. This singleton access pattern applies to all declaration objects created with the DSL.

```
state('example')
assert state('example') is state('example')
assert state('example').cmd is state('example').cmd
assert state('example').cmd.running is state('example').cmd.running
```

The *id* argument is optional. If omitted, an UUID will be generated and used as the *id*.

`state(id)` returns an object under which you can create a *State declaration* object by accessing an attribute named after *any* state module available in Salt.

```
state('example').cmd
state('example').file
state('example').pkg
...
```

Then, a *Function declaration* object can be created from a *State declaration* object by one of the following two ways:

1. by calling a method named after the state function on the *State declaration* object.

```
state('example').file.managed(...)
```

2. by directly calling the attribute named for the *State declaration*, and supplying the state function name as the first argument.

```
state('example').file('managed', ...)
```

With either way of creating a *Function declaration* object, any *Function arg declaration*`s can be passed as keyword arguments to the call. Subsequent calls of a *Function declaration* will update the arg declarations.

```
state('example').file('managed', source='salt://webserver/index.html')
state('example').file.managed(source='salt://webserver/index.html')
```

As a shortcut, the special *name* argument can also be passed as the first or second positional argument depending on the first or second way of calling the *State declaration* object. In the following two examples *ls -la* is the *name* argument.

```
state('example').cmd.run('ls -la', cwd='/')
state('example').cmd('run', 'ls -la', cwd='/')
```

Finally, a *Requisite declaration* object with its *Requisite reference*`s can be created by invoking one of the requisite methods (see *State Requisites*) on either a *Function declaration* object or a *State declaration* object. The return value of a requisite call is also a *Function declaration* object, so you can chain several requisite calls together.

Arguments to a requisite call can be a list of *State declaration* objects and/or a set of keyword arguments whose names are state modules and values are IDs of *ID declaration*`s or names of *Name declaration*`s.

```
apache2 = state('apache2')
apache2.pkg.installed()
state('libapache2-mod-wsgi').pkg.installed()

# you can call requisites on function declaration
apache2.service.running() \
               .require(apache2.pkg,
                        pkg='libapache2-mod-wsgi') \
               .watch(file='/etc/apache2/httpd.conf')

# or you can call requisites on state declaration.
# this actually creates an anonymous function declaration object
# to add the requisites.
apache2.service.require(state('libapache2-mod-wsgi').pkg,
                        pkg='apache2') \
               .watch(file='/etc/apache2/httpd.conf')

# we still need to set the name of the function declaration.
apache2.service.running()
```

Include declaration objects can be created with the `include` function, while *Extend declaration* objects can be created with the `extend` function, whose arguments are just *Function declaration* objects.

```
include('edit.vim', 'http.server')
extend(state('apache2').service.watch(file='/etc/httpd/httpd.conf')
```

The `include` function, by default, causes the included sls file to be rendered as soon as the `include` function is called. It returns a list of rendered module objects; sls files not rendered with the pydsl renderer return `None`s. This behavior creates no *Include declaration*`s in the resulting high state data structure.

```
import types

# including multiple sls returns a list.
_, mod = include('a-non-pydsl-sls', 'a-pydsl-sls')

assert _ is None
assert isinstance(slsmods[1], types.ModuleType)
```

```
# including a single sls returns a single object
mod = include('a-pydsl-sls')

# myfunc is a function that calls state(...) to create more states.
mod.myfunc(1, 2, "three")
```

Notice how you can define a reusable function in your pydsl sls module and then call it via the module returned by `include`.

It's still possible to do late includes by passing the `delayed=True` keyword argument to `include`.

```
include('edit.vim', 'http.server', delayed=True)
```

Above will just create a *Include declaration* in the rendered result, and such call always returns `None`.

Special integration with the *cmd* state Taking advantage of rendering a Python module, PyDSL allows you to declare a state that calls a pre-defined Python function when the state is executed.

```
greeting = "hello world"
def helper(something, *args, **kws):
    print greeting                  # hello world
    print something, args, kws      # test123 ['a', 'b', 'c'] {'x': 1, 'y': 2}

state().cmd.call(helper, "test123", 'a', 'b', 'c', x=1, y=2)
```

The *cmd.call* state function takes care of calling our `helper` function with the arguments we specified in the states, and translates the return value of our function into a structure expected by the state system. See `salt.states.cmd.call()` for more information.

Implicit ordering of states Salt states are explicitly ordered via *Requisite declaration*`s. However, with *pydsl* it's possible to let the renderer track the order of creation for *Function declaration* objects, and implicitly add `require` requisites for your states to enforce the ordering. This feature is enabled by setting the `ordered` option on `__pydsl__`.

Note: this feature is only available if your minions are using Python >= 2.7.

```
include('some.sls.file')

A = state('A').cmd.run(cwd='/var/tmp')
extend(A)

__pydsl__.set(ordered=True)

for i in range(10):
    i = str(i)
    state(i).cmd.run('echo '+i, cwd='/')
state('1').cmd.run('echo one')
state('2').cmd.run(name='echo two')
```

Notice that the `ordered` option needs to be set after any `extend` calls. This is to prevent *pydsl* from tracking the creation of a state function that's passed to an `extend` call.

Above example should create states from `0` to `9` that will output `0`, `one`, `two`, `3`, ... `9`, in that order.

It's important to know that *pydsl* tracks the *creations* of *Function declaration* objects, and automatically adds a `require` requisite to a *Function declaration* object that requires the last *Function declaration* object created before it in the sls file.

This means later calls (perhaps to update the function's *Function arg declaration*) to a previously created function declaration will not change the order.

Render time state execution When Salt processes a salt formula file, the file is rendered to salt's high state data representation by a renderer before the states can be executed. In the case of the *pydsl* renderer, the .sls file is executed as a python module as it is being rendered which makes it easy to execute a state at render time. In *pydsl*, executing one or more states at render time can be done by calling a configured *ID declaration* object.

```
#!pydsl

s = state() # save for later invocation

# configure it
s.cmd.run('echo at render time', cwd='/')
s.file.managed('target.txt', source='salt://source.txt')

s() # execute the two states now
```

Once an *ID declaration* is called at render time it is detached from the sls module as if it was never defined.

Note: If *implicit ordering* is enabled (i.e., via `__pydsl__.set(ordered=True)`) then the *first* invocation of a *ID declaration* object must be done before a new *Function declaration* is created.

Integration with the stateconf renderer The *salt.renderers.stateconf* renderer offers a few interesting features that can be leveraged by the *pydsl* renderer. In particular, when using with the *pydsl* renderer, we are interested in *stateconf*`s sls namespacing feature (via dot-prefixed id declarations), as well as, the automatic *start* and *goal* states generation.

Now you can use *pydsl* with *stateconf* like this:

```
#!pydsl|stateconf -ps

include('xxx', 'yyy')

# ensure that states in xxx run BEFORE states in this file.
extend(state('.start').stateconf.require(stateconf='xxx::goal'))

# ensure that states in yyy run AFTER states in this file.
extend(state('.goal').stateconf.require_in(stateconf='yyy::start'))

__pydsl__.set(ordered=True)

...
```

`-s` enables the generation of a stateconf *start* state, and `-p` lets us pipe high state data rendered by *pydsl* to *stateconf*. This example shows that by `require`-ing or `require_in`-ing the included sls' *start* or *goal* states, it's possible to ensure that the included sls files can be made to execute before or after a state in the including sls file.

Importing custom Python modules To use a custom Python module inside a PyDSL state, place the module somewhere that it can be loaded by the Salt loader, such as *_modules* in the */srv/salt* directory.

Then, copy it to any minions as necessary by using *saltutil.sync_modules*.

To import into a PyDSL SLS, one must bypass the Python importer and insert it manually by getting a reference from Python's *sys.modules* dictionary.

For example:

```
#!pydsl|stateconf -ps

def main():
    my_mod = sys.modules['salt.loaded.ext.module.my_mod']
```

`salt.renderers.pydsl.`**`render`**(*template*, *saltenv='base'*, *sls=''*, *tmplpath=None*, *rendered_sls=None*, ***kws*)

salt.renderers.pyobjects

Python renderer that includes a Pythonic Object based interface

maintainer Evan Borgstrom <evan@borgstrom.ca>

Let's take a look at how you use pyobjects in a state file. Here's a quick example that ensures the `/tmp` directory is in the correct state.

```
1 #!pyobjects
2
3 File.managed("/tmp", user='root', group='root', mode='1777')
```

Nice and Pythonic!

By using the ``shebang'' syntax to switch to the pyobjects renderer we can now write our state data using an object based interface that should feel at home to python developers. You can import any module and do anything that you'd like (with caution, importing sqlalchemy, django or other large frameworks has not been tested yet). Using the pyobjects renderer is exactly the same as using the built-in Python renderer with the exception that pyobjects provides you with an object based interface for generating state data.

Creating state data Pyobjects takes care of creating an object for each of the available states on the minion. Each state is represented by an object that is the CamelCase version of its name (i.e. `File`, `Service`, `User`, etc), and these objects expose all of their available state functions (i.e. `File.managed`, `Service.running`, etc).

The name of the state is split based upon underscores (_), then each part is capitalized and finally the parts are joined back together.

Some examples:

- `postgres_user` becomes `PostgresUser`
- `ssh_known_hosts` becomes `SshKnownHosts`

Context Managers and requisites How about something a little more complex. Here we're going to get into the core of how to use pyobjects to write states.

```
#!pyobjects

with Pkg.installed("nginx"):
    Service.running("nginx", enable=True)

    with Service("nginx", "watch_in"):
        File.managed("/etc/nginx/conf.d/mysite.conf",
                     owner='root', group='root', mode='0444',
                     source='salt://nginx/mysite.conf')
```

The objects that are returned from each of the magic method calls are setup to be used a Python context managers (`with`) and when you use them as such all declarations made within the scope will **automatically** use the enclosing state as a requisite!

The above could have also been written use direct requisite statements as.

```
#!pyobjects

Pkg.installed("nginx")
Service.running("nginx", enable=True, require=Pkg("nginx"))
File.managed("/etc/nginx/conf.d/mysite.conf",
             owner='root', group='root', mode='0444',
             source='salt://nginx/mysite.conf',
             watch_in=Service("nginx"))
```

You can use the direct requisite statement for referencing states that are generated outside of the current file.

```
#!pyobjects

# some-other-package is defined in some other state file
Pkg.installed("nginx", require=Pkg("some-other-package"))
```

The last thing that direct requisites provide is the ability to select which of the SaltStack requisites you want to use (require, require_in, watch, watch_in, use & use_in) when using the requisite as a context manager.

```
#!pyobjects

with Service("my-service", "watch_in"):
    ...
```

The above example would cause all declarations inside the scope of the context manager to automatically have their `watch_in` set to `Service("my-service")`.

Including and Extending To include other states use the `include()` function. It takes one name per state to include.

To extend another state use the `extend()` function on the name when creating a state.

```
#!pyobjects

include('http', 'ssh')

Service.running(extend('apache'),
                watch=[File('/etc/httpd/extra/httpd-vhosts.conf')])
```

Importing from other state files Like any Python project that grows you will likely reach a point where you want to create reusability in your state tree and share objects between state files, Map Data (described below) is a perfect example of this.

To facilitate this Python's `import` statement has been augmented to allow for a special case when working with a Salt state tree. If you specify a Salt url (`salt://...`) as the target for importing from then the pyobjects renderer will take care of fetching the file for you, parsing it with all of the pyobjects features available and then place the requested objects in the global scope of the template being rendered.

This works for both types of import statements, `import X` and `from X import Y`.

```
#!pyobjects

import salt://myfile.sls
from salt://something/data.sls import Object
```

See the Map Data section for a more practical use.

Caveats:

- You cannot use the `as` syntax, you can only import objects using their existing name.
- Imported objects are ALWAYS put into the global scope of your template, regardless of where your import statement is.

Salt object In the spirit of the object interface for creating state data pyobjects also provides a simple object interface to the `__salt__` object.

A function named `salt` exists in scope for your sls files and will dispatch its attributes to the `__salt__` dictionary.

The following lines are functionally equivalent:

```
#!pyobjects

ret = salt.cmd.run(bar)
ret = __salt__['cmd.run'](bar)
```

Pillar, grain, mine & config data Pyobjects provides shortcut functions for calling `pillar.get`, `grains.get`, `mine.get` & `config.get` on the `__salt__` object. This helps maintain the readability of your state files.

Each type of data can be access by a function of the same name: `pillar()`, `grains()`, `mine()` and `config()`.

The following pairs of lines are functionally equivalent:

```
#!pyobjects

value = pillar('foo:bar:baz', 'qux')
value = __salt__['pillar.get']('foo:bar:baz', 'qux')

value = grains('pkg:apache')
value = __salt__['grains.get']('pkg:apache')

value = mine('os:Fedora', 'network.interfaces', 'grain')
value = __salt__['mine.get']('os:Fedora', 'network.interfaces', 'grain')

value = config('foo:bar:baz', 'qux')
value = __salt__['config.get']('foo:bar:baz', 'qux')
```

Map Data When building complex states or formulas you often need a way of building up a map of data based on grain data. The most common use of this is tracking the package and service name differences between distributions.

To build map data using pyobjects we provide a class named Map that you use to build your own classes with inner classes for each set of values for the different grain matches.

```
#!pyobjects

class Samba(Map):
    merge = 'samba:lookup'

    class Debian:
        server = 'samba'
        client = 'samba-client'
        service = 'samba'

    class Ubuntu:
        __grain__ = 'os'
        service = 'smbd'

    class RedHat:
        server = 'samba'
        client = 'samba'
        service = 'smb'
```

To use this new data you can import it into your state file and then access your attributes. To access the data in the map you simply access the attribute name on the base class that is extending Map. Assuming the above Map was in the file `samba/map.sls`, you could do the following.

```
#!pyobjects

from salt://samba/map.sls import Samba

with Pkg.installed("samba", names=[Samba.server, Samba.client]):
    Service.running("samba", name=Samba.service)
```

TODO

- Interface for working with reactor files

`salt.renderers.pyobjects.`**`load_states`**`()`
This loads our states into the salt __context__

`salt.renderers.pyobjects.`**`render`**`(`*template*, *saltenv='base'*, *sls=''*, *salt_data=True*, ***kwargs*`)`

salt.renderers.stateconf

maintainer Jack Kuan <kjkuan@gmail.com>

maturity new

platform all

This module provides a custom renderer that processes a salt file with a specified templating engine (e.g. Jinja) and a chosen data renderer (e.g. YAML), extracts arguments for any `stateconf.set` state, and provides the extracted arguments (including Salt-specific args, such as `require`, etc) as template context. The goal is to make writing reusable/configurable/parameterized salt files easier and cleaner.

To use this renderer, either set it as the default renderer via the `renderer` option in master/minion's config, or use the shebang line in each individual sls file, like so: `#!stateconf`. Note, due to the way this renderer works, it must be specified as the first renderer in a render pipeline. That is, you cannot specify `#!mako|yaml|stateconf`, for example. Instead, you specify them as renderer arguments: `#!stateconf mako . yaml`.

Here's a list of features enabled by this renderer.

- Prefixes any state id (declaration or reference) that starts with a dot (`.`) to avoid duplicated state ids when the salt file is included by other salt files.

  For example, in the *salt://some/file.sls*, a state id such as `.sls_params` will be turned into `some.file::sls_params`. Example:

```
#!stateconf yaml . jinja

.vim:
  pkg.installed
```

  Above will be translated into:

```
some.file::vim:
  pkg.installed:
    - name: vim
```

  Notice how that if a state under a dot-prefixed state id has no `name` argument then one will be added automatically by using the state id with the leading dot stripped off.

  The leading dot trick can be used with extending state ids as well, so you can include relatively and extend relatively. For example, when extending a state in *salt://some/other_file.sls*, e.g.:

```
#!stateconf yaml . jinja

include:
  - .file

extend:
  .file::sls_params:
    stateconf.set:
      - name1: something
```

  Above will be pre-processed into:

```
include:
  - some.file

extend:
  some.file::sls_params:
    stateconf.set:
      - name1: something
```

- Adds a `sls_dir` context variable that expands to the directory containing the rendering salt file. So, you can write `salt://{{sls_dir}}/...` to reference templates files used by your salt file.
- Recognizes the special state function, `stateconf.set`, that configures a default list of named arguments usable within the template context of the salt file. Example:

```
#!stateconf yaml . jinja

.sls_params:
  stateconf.set:
    - name1: value1
```

```
        - name2: value2
        - name3:
          - value1
          - value2
          - value3
        - require_in:
          - cmd: output

    # --- end of state config ---

    .output:
      cmd.run:
        - name: |
            echo 'name1={{sls_params.name1}}
                  name2={{sls_params.name2}}
                  name3[1]={{sls_params.name3[1]}}
            '
```

This even works with `include` + `extend` so that you can override the default configured arguments by including the salt file and then `extend` the `stateconf.set` states that come from the included salt file. (*IMPORTANT: Both the included and the extending sls files must use the stateconf renderer for this ``extend`` to work!*)

Notice that the end of configuration marker (`# --- end of state config --`) is needed to separate the use of `stateconf.set' form the rest of your salt file. The regex that matches such marker can be configured via the `stateconf_end_marker` option in your master or minion config file.

Sometimes, it is desirable to set a default argument value that's based on earlier arguments in the same `stateconf.set`. For example, it may be tempting to do something like this:

```
    #!stateconf yaml . jinja

    .apache:
      stateconf.set:
        - host: localhost
        - port: 1234
        - url: 'http://{{host}}:{{port}}/'

    # --- end of state config ---

    .test:
      cmd.run:
        - name: echo '{{apache.url}}'
        - cwd: /
```

However, this won't work. It can however be worked around like so:

```
    #!stateconf yaml . jinja

    .apache:
      stateconf.set:
        - host: localhost
        - port: 1234
    {#  - url: 'http://{{host}}:{{port}}/' #}

    # --- end of state config ---
    # {{ apache.setdefault('url', "http://%(host)s:%(port)s/" % apache) }}

    .test:
```

```
    cmd.run:
      - name: echo '{{apache.url}}'
      - cwd: /
```

- Adds support for relative include and exclude of .sls files. Example:

```
#!stateconf yaml . jinja

include:
  - .apache
  - .db.mysql

exclude:
  - sls: .users
```

  If the above is written in a salt file at *salt://some/where.sls* then it will include *salt://some/apache.sls* and *salt://some/db/mysql.sls*, and exclude *salt://some/users.ssl*. Actually, it does that by rewriting the above `include` and `exclude` into:

```
include:
  - some.apache
  - some.db.mysql

exclude:
  - sls: some.users
```

- Optionally (enabled by default, *disable* via the *-G* renderer option, e.g. in the shebang line: `#!stateconf -G`), generates a `stateconf.set` goal state (state id named as `.goal` by default, configurable via the master/minion config option, `stateconf_goal_state`) that requires all other states in the salt file. Note, the `.goal` state id is subject to dot-prefix rename rule mentioned earlier.

  Such goal state is intended to be required by some state in an including salt file. For example, in your webapp salt file, if you include a sls file that is supposed to setup Tomcat, you might want to make sure that all states in the Tomcat sls file will be executed before some state in the webapp sls file.

- Optionally (enable via the *-o* renderer option, e.g. in the shebang line: `#!stateconf -o`), orders the states in a sls file by adding a `require` requisite to each state such that every state requires the state defined just before it. The order of the states here is the order they are defined in the sls file. (Note: this feature is only available if your minions are using Python >= 2.7. For Python2.6, it should also work if you install the *ordereddict* module from PyPI)

  By enabling this feature, you are basically agreeing to author your sls files in a way that gives up the explicit (or implicit?) ordering imposed by the use of `require`, `watch`, `require_in` or `watch_in` requisites, and instead, you rely on the order of states you define in the sls files. This may or may not be a better way for you. However, if there are many states defined in a sls file, then it tends to be easier to see the order they will be executed with this feature.

  You are still allowed to use all the requisites, with a few restrictions. You cannot `require` or `watch` a state defined *after* the current state. Similarly, in a state, you cannot `require_in` or `watch_in` a state defined *before* it. Breaking any of the two restrictions above will result in a state loop. The renderer will check for such incorrect uses if this feature is enabled.

  Additionally, `names` declarations cannot be used with this feature because the way they are compiled into low states make it impossible to guarantee the order in which they will be executed. This is also checked by the renderer. As a workaround for not being able to use `names`, you can achieve the same effect, by generate your states with the template engine available within your sls file.

  Finally, with the use of this feature, it becomes possible to easily make an included sls file execute all its states *after* some state (say, with id `X`) in the including sls file. All you have to do is to make state, `X`, `require_in`

the first state defined in the included sls file.

When writing sls files with this renderer, one should avoid using what can be defined in a `name` argument of a state as the state's id. That is, avoid writing states like this:

```
/path/to/some/file:
  file.managed:
    - source: salt://some/file

cp /path/to/some/file file2:
  cmd.run:
    - cwd: /
    - require:
      - file: /path/to/some/file
```

Instead, define the state id and the `name` argument separately for each state. Also, the ID should be something meaningful and easy to reference within a requisite (which is a good habit anyway, and such extra indirection would also makes the sls file easier to modify later). Thus, the above states should be written like this:

```
add-some-file:
  file.managed:
    - name: /path/to/some/file
    - source: salt://some/file

copy-files:
  cmd.run:
    - name: cp /path/to/some/file file2
    - cwd: /
    - require:
      - file: add-some-file
```

Moreover, when referencing a state from a requisite, you should reference the state's id plus the state name rather than the state name plus its `name` argument. (Yes, in the above example, you can actually `require` the `file: /path/to/some/file`, instead of the `file: add-some-file`). The reason is that this renderer will re-write or rename state id's and their references for state id's prefixed with `.`. So, if you reference `name` then there's no way to reliably rewrite such reference.

salt.renderers.wempy

`salt.renderers.wempy.`**`render`**(*template_file*, *saltenv='base'*, *sls=''*, *argline=''*, *context=None*, ***kws*)

Render the data passing the functions and grains into the rendering system

Return type string

salt.renderers.yaml

Understanding YAML The default renderer for SLS files is the YAML renderer. YAML is a markup language with many powerful features. However, Salt uses a small subset of YAML that maps over very commonly used data structures, like lists and dictionaries. It is the job of the YAML renderer to take the YAML data structure and compile it into a Python data structure for use by Salt.

Though YAML syntax may seem daunting and terse at first, there are only three very simple rules to remember when writing YAML for SLS files.

Rule One: Indentation YAML uses a fixed indentation scheme to represent relationships between data layers. Salt requires that the indentation for each level consists of exactly two spaces. Do not use tabs.

Rule Two: Colons Python dictionaries are, of course, simply key-value pairs. Users from other languages may recognize this data type as hashes or associative arrays.

Dictionary keys are represented in YAML as strings terminated by a trailing colon. Values are represented by either a string following the colon, separated by a space:

```
my_key: my_value
```

In Python, the above maps to:

```
{'my_key': 'my_value'}
```

Dictionaries can be nested:

```
first_level_dict_key:
  second_level_dict_key: value_in_second_level_dict
```

And in Python:

```
{'first_level_dict_key': {'second_level_dict_key': 'value_in_second_level_dict' }
```

Rule Three: Dashes To represent lists of items, a single dash followed by a space is used. Multiple items are a part of the same list as a function of their having the same level of indentation.

```
- list_value_one
- list_value_two
- list_value_three
```

Lists can be the value of a key-value pair. This is quite common in Salt:

```
my_dictionary:
  - list_value_one
  - list_value_two
  - list_value_three
```

Reference YAML Renderer for Salt

`salt.renderers.yaml.`**`get_yaml_loader`**(*argline*)
Return the ordered dict yaml loader

`salt.renderers.yaml.`**`render`**(*yaml_data*, *saltenv='base'*, *sls=''*, *argline=''*, ***kws*)
Accepts YAML as a string or as a file object and runs it through the YAML parser.

Return type A Python data structure

salt.renderers.yamlex

YAMLEX renderer is a replacement of the YAML renderer. It's 100% YAML with a pinch of Salt magic:

- All mappings are automatically OrderedDict
- All strings are automatically str obj
- data aggregation with !aggregation yaml tag, based on the `salt.utils.aggregation` module.
- data aggregation over documents for pillar

Instructed aggregation within the `!aggregation` and the `!reset` tags:

```
#!yamlex
foo: !aggregate first
foo: !aggregate second
bar: !aggregate {first: foo}
bar: !aggregate {second: bar}
baz: !aggregate 42
qux: !aggregate default
!reset qux: !aggregate my custom data
```

is roughly equivalent to

```
foo: [first, second]
bar: {first: foo, second: bar}
baz: [42]
qux: [my custom data]
```

Reference

`salt.renderers.yamlex.`**`render`**(*sls_data*, *saltenv='base'*, *sls=''*, ***kws*)

Accepts YAML_EX as a string or as a file object and runs it through the YAML_EX parser.

Return type A Python data structure

22.23 Returners

By default the return values of the commands sent to the Salt minions are returned to the Salt master, however anything at all can be done with the results data.

By using a Salt returner, results data can be redirected to external data-stores for analysis and archival.

Returners pull their configuration values from the Salt minions. Returners are only configured once, which is generally at load time.

The returner interface allows the return data to be sent to any system that can receive data. This means that return data can be sent to a Redis server, a MongoDB server, a MySQL server, or any system.

See also:

Full list of builtin returners

22.23.1 Using Returners

All Salt commands will return the command data back to the master. Specifying returners will ensure that the data is _also_ sent to the specified returner interfaces.

Specifying what returners to use is done when the command is invoked:

```
salt '*' test.ping --return redis_return
```

This command will ensure that the redis_return returner is used.

It is also possible to specify multiple returners:

```
salt '*' test.ping --return mongo_return,redis_return,cassandra_return
```

In this scenario all three returners will be called and the data from the test.ping command will be sent out to the three named returners.

22.23.2 Writing a Returner

A returner is a Python module containing at minimum a `returner` function. Other optional functions can be included to add support for *master_job_cache*, *external_job_cache*, and Event Returners.

`returner` The `returner` function must accept a single argument. The argument contains return data from the called minion function. If the minion function `test.ping` is called, the value of the argument will be a dictionary. Run the following command from a Salt master to get a sample of the dictionary:

```
salt-call --local --metadata test.ping --out=pprint
```

```
import redis
import json

def returner(ret):
    '''
    Return information to a redis server
    '''
    # Get a redis connection
    serv = redis.Redis(
                host='redis-serv.example.com',
                port=6379,
                db='0')
    serv.sadd("%(id)s:jobs" % ret, ret['jid'])
    serv.set("%(jid)s:%(id)s" % ret, json.dumps(ret['return']))
    serv.sadd('jobs', ret['jid'])
    serv.sadd(ret['jid'], ret['id'])
```

The above example of a returner set to send the data to a Redis server serializes the data as JSON and sets it in redis.

Master Job Cache Support

master_job_cache, *external_job_cache*, and Event Returners. Salt's *master_job_cache* allows returners to be used as a pluggable replacement for the *default_job_cache*. In order to do so, a returner must implement the following functions:

Note: The code samples contained in this section were taken from the cassandra_cql returner.

`prep_jid` Ensures that job ids (jid) don't collide, unless passed_jid is provided.

`nochache` is an optional boolean that indicates if return data should be cached. `passed_jid` is a caller provided jid which should be returned unconditionally.

```
def prep_jid(nocache, passed_jid=None):  # pylint: disable=unused-argument
    '''
    Do any work necessary to prepare a JID, including sending a custom id
    '''
    return passed_jid if passed_jid is not None else salt.utils.jid.gen_jid()
```

`save_load` Save job information. The `jid` is generated by `prep_jid` and should be considered a unique identifier for the job. The jid, for example, could be used as the primary/unique key in a database. The `load` is what is returned to a Salt master by a minion. The following code example stores the load as a JSON string in the salt.jids table.

```
def save_load(jid, load):
    '''
    Save the load to the specified jid id
    '''
    query = '''INSERT INTO salt.jids (
                jid, load
              ) VALUES (
                '{0}', '{1}'
              );'''.format(jid, json.dumps(load))

    # cassandra_cql.cql_query may raise a CommandExecutionError
    try:
        __salt__['cassandra_cql.cql_query'](query)
    except CommandExecutionError:
        log.critical('Could not save load in jids table.')
        raise
    except Exception as e:
        log.critical('''Unexpected error while inserting into
         jids: {0}'''.format(str(e)))
        raise
```

`get_load` must accept a job id (jid) and return the job load stored by `save_load`, or an empty dictionary when not found.

```
def get_load(jid):
    '''
    Return the load data that marks a specified jid
    '''
    query = '''SELECT load FROM salt.jids WHERE jid = '{0}';'''.format(jid)

    ret = {}

    # cassandra_cql.cql_query may raise a CommandExecutionError
    try:
        data = __salt__['cassandra_cql.cql_query'](query)
        if data:
            load = data[0].get('load')
```

```
            if load:
                ret = json.loads(load)
    except CommandExecutionError:
        log.critical('Could not get load from jids table.')
        raise
    except Exception as e:
        log.critical('''Unexpected error while getting load from
         jids: {0}'''.format(str(e)))
        raise

    return ret
```

External Job Cache Support

Salt's *external_job_cache* extends the *master_job_cache*. External Job Cache support requires the following functions in addition to what is required for Master Job Cache support:

get_jid Return a dictionary containing the information (load) returned by each minion when the specified job id was executed.

Sample:

```
{
    "local": {
        "master_minion": {
            "fun_args": [],
            "jid": "20150330121011408195",
            "return": true,
            "retcode": 0,
            "success": true,
            "cmd": "_return",
            "_stamp": "2015-03-30T12:10:12.708663",
            "fun": "test.ping",
            "id": "master_minion"
        }
    }
}
```

get_fun Return a dictionary of minions that called a given Salt function as their last function call.

Sample:

```
{
    "local": {
        "minion1": "test.ping",
        "minion3": "test.ping",
        "minion2": "test.ping"
    }
}
```

get_jids Return a list of all job ids.

Sample:

```
{
    "local": [
        "20150330121011408195",
        "20150330195922139916"
    ]
}
```

get_minions Returns a list of minions

Sample:

```
{
     "local": [
         "minion3",
         "minion2",
         "minion1",
         "master_minion"
     ]
}
```

Please refer to one or more of the existing returners (i.e. mysql, cassandra_cql) if you need further clarification.

Event Support

An `event_return` function must be added to the returner module to allow events to be logged from a master via the returner. A list of events are passed to the function by the master.

The following example was taken from the MySQL returner. In this example, each event is inserted into the salt_events table keyed on the event tag. The tag contains the jid and therefore is guaranteed to be unique.

```
def event_return(events):
 '''
 Return event to mysql server

 Requires that configuration be enabled via 'event_return'
 option in master config.
 '''
 with _get_serv(events, commit=True) as cur:
     for event in events:
         tag = event.get('tag', '')
         data = event.get('data', '')
         sql = '''INSERT INTO `salt_events` (`tag`, `data`, `master_id` )
                  VALUES (%s, %s, %s)'''
         cur.execute(sql, (tag, json.dumps(data), __opts__['id']))
```

Custom Returners

Place custom returners in a `_returners` directory within the `file_roots` specified by the master config file.

Custom returners are distributed when any of the following are called: `state.highstate`

`saltutil.sync_returners`

`saltutil.sync_all`

Any custom returners which have been synced to a minion that are named the same as one of Salt's default set of returners will take the place of the default returner with the same name.

Naming the Returner

Note that a returner's default name is its filename (i.e. `foo.py` becomes returner `foo`), but that its name can be overridden by using a *__virtual__ function*. A good example of this can be found in the redis returner, which is named `redis_return.py` but is loaded as simply `redis`:

```
try:
    import redis
    HAS_REDIS = True
except ImportError:
    HAS_REDIS = False

__virtualname__ = 'redis'

def __virtual__():
    if not HAS_REDIS:
        return False
    return __virtualname__
```

Testing the Returner

The `returner`, `prep_jid`, `save_load`, `get_load`, and `event_return` functions can be tested by configuring the *master_job_cache* and Event Returners in the master config file and submitting a job to `test.ping` each minion from the master.

Once you have successfully exercised the Master Job Cache functions, test the External Job Cache functions using the `ret` execution module.

```
salt-call ret.get_jids cassandra_cql --output=json
salt-call ret.get_fun cassandra_cql test.ping --output=json
salt-call ret.get_minions cassandra_cql --output=json
salt-call ret.get_jid cassandra_cql 20150330121011408195 --output=json
```

22.23.3 Event Returners

For maximimum visibility into the history of events across a Salt infrastructure, all events seen by a salt master may be logged to a returner.

To enable event logging, set the `event_return` configuration option in the master config to returner which should be designated as the handler for event returns.

Note: Not all returners support event returns. Verify a returner has an `event_return()` function before using.

Note: On larger installations, many hundreds of events may be generated on a busy master every second. Be certain to closely monitor the storage of a given returner as Salt can easily overwhelm an underpowered server with thousands of returns.

22.23.4 Full List of Returners

Full list of builtin returner modules

carbon_return	Take data from salt and ``return'' it into a carbon receiver
cassandra_return	Return data to a Cassandra ColumnFamily
couchbase_return	Simple returner for Couchbase.
couchdb_return	Simple returner for CouchDB.
django_return	A returner that will infor a Django system that returns are available using Django's signal system.
elasticsearch_return	Return data to an elasticsearch server for indexing.
etcd_return	Return data to an etcd server or cluster
hipchat_return	
kafka_return	Return data to a Kafka topic
local	The local returner is used to test the returner interface, it just prints the
local_cache	Return data to local job cache
memcache_return	Return data to a memcache server
mongo_future_return	Return data to a mongodb server
mongo_return	Return data to a mongodb server
multi_returner	Read/Write multiple returners
mysql	Return data to a mysql server
odbc	Return data to an ODBC compliant server.
postgres	Return data to a postgresql server
postgres_local_cache	Use a postgresql server for the master job cache.
redis_return	Return data to a redis server
sentry_return	Salt returner that report execution results back to sentry.
slack_returner	
sms_return	Return data by SMS.
smtp_return	Return salt data via email
sqlite3_return	Insert minion return data into a sqlite3 database
syslog_return	Return data to the host operating system's syslog facility
xmpp_return	Return salt data via xmpp

salt.returners.carbon_return

Take data from salt and ``return'' it into a carbon receiver

Add the following configuration to the minion configuration file:

```
carbon.host: <server ip address>
carbon.port: 2003
```

Errors when trying to convert data to numbers may be ignored by setting `carbon.skip_on_error` to *True*:

```
carbon.skip_on_error: True
```

By default, data will be sent to carbon using the plaintext protocol. To use the pickle protocol, set `carbon.mode` to `pickle`:

```
carbon.mode: pickle
```

Carbon settings may also be configured as:

```
carbon:
  host: <server IP or hostname>
  port: <carbon port>
  skip_on_error: True
  mode: (pickle|text)
```

Alternative configuration values can be used by prefacing the configuration. Any values not found in the alternative configuration will be pulled from the default location:

```
alternative.carbon:
  host: <server IP or hostname>
  port: <carbon port>
  skip_on_error: True
  mode: (pickle|text)
```

To use the carbon returner, append `--return carbon' to the salt command.

```
salt '*' test.ping --return carbon
```

To use the alternative configuration, append `--return_config alternative' to the salt command.

New in version 2015.5.0.

```
salt '*' test.ping --return carbon --return_config alternative
```

`salt.returners.carbon_return.`**`prep_jid`**(*nocache=False*, *passed_jid=None*)
Do any work necessary to prepare a JID, including sending a custom id

`salt.returners.carbon_return.`**`returner`**(*ret*)
Return data to a remote carbon server using the text metric protocol

Each metric will look like:

```
[module].[function].[minion_id].[metric path [...]].[metric name]
```

salt.returners.cassandra_return

Return data to a Cassandra ColumnFamily

Here's an example Keyspace / ColumnFamily setup that works with this returner:

```
create keyspace salt;
use salt;
create column family returns
  with key_validation_class='UTF8Type'
  and comparator='UTF8Type'
  and default_validation_class='UTF8Type';
```

Required python modules: pycassa

To use the cassandra returner, append `--return cassandra' to the salt command. ex:

salt `*' test.ping --return cassandra

salt.returners.cassandra_return.**prep_jid**(*nocache=False*, *passed_jid=None*)
Do any work necessary to prepare a JID, including sending a custom id

salt.returners.cassandra_return.**returner**(*ret*)
Return data to a Cassandra ColumnFamily

salt.returners.couchbase_return

Simple returner for Couchbase. Optional configuration settings are listed below, along with sane defaults.

couchbase.host: `salt' couchbase.port: 8091 couchbase.bucket: `salt' couchbase.skip_verify_views: False

To use the couchbase returner, append `--return couchbase' to the salt command. ex:

salt `*' test.ping --return couchbase

All of the return data will be stored in documents as follows:

JID load: load obj tgt_minions: list of minions targeted nocache: should we not cache the return data

JID/MINION_ID return: return_data out: out_data

salt.returners.couchbase_return.**get_jid**(*jid*)
Return the information returned when the specified job id was executed

salt.returners.couchbase_return.**get_jids**()
Return a list of all job ids

salt.returners.couchbase_return.**get_load**(*jid*)
Return the load data that marks a specified jid

salt.returners.couchbase_return.**prep_jid**(*nocache=False*, *passed_jid=None*)
Return a job id and prepare the job id directory This is the function responsible for making sure jids don't collide (unless its passed a jid) So do what you have to do to make sure that stays the case

salt.returners.couchbase_return.**returner**(*load*)
Return data to the local job cache

salt.returners.couchbase_return.**save_load**(*jid*, *clear_load*)
Save the load to the specified jid

salt.returners.couchdb_return

Simple returner for CouchDB. Optional configuration settings are listed below, along with sane defaults:

```
couchdb.db: 'salt'
couchdb.url: 'http://salt:5984/'
```

Alternative configuration values can be used by prefacing the configuration. Any values not found in the alternative configuration will be pulled from the default location:

```
alternative.couchdb.db: 'salt'
alternative.couchdb.url: 'http://salt:5984/'
```

To use the couchdb returner, append `--return couchdb` to the salt command. Example:

```
salt '*' test.ping --return couchdb
```

To use the alternative configuration, append `--return_config alternative` to the salt command.

New in version 2015.5.0.

```
salt '*' test.ping --return couchdb --return_config alternative
```

On concurrent database access As this returner creates a couchdb document whith the salt job id as document id and as only one document with a given id can exist in a given couchdb database, it is advised for most setups that every minion be configured to write to it own database (the value of `couchdb.db` may be suffixed with the minion id), otherwise multi-minion targetting can lead to losing output:

- the first returning minion is able to create a document in the database
- other minions fail with `{'error': 'HTTP Error 409: Conflict'}`

`salt.returners.couchdb_return.`**`ensure_views`**`()`
This function makes sure that all the views that should exist in the design document do exist.

`salt.returners.couchdb_return.`**`get_fun`**`(`*`fun`*`)`
Return a dict with key being minion and value being the job details of the last run of function `fun'.

`salt.returners.couchdb_return.`**`get_jid`**`(`*`jid`*`)`
Get the document with a given JID.

`salt.returners.couchdb_return.`**`get_jids`**`()`
List all the jobs that we have..

`salt.returners.couchdb_return.`**`get_minions`**`()`
Return a list of minion identifiers from a request of the view.

`salt.returners.couchdb_return.`**`get_valid_salt_views`**`()`
Returns a dict object of views that should be part of the salt design document.

`salt.returners.couchdb_return.`**`prep_jid`**`(`*`nocache=False, passed_jid=None`*`)`
Do any work necessary to prepare a JID, including sending a custom id

`salt.returners.couchdb_return.`**`returner`**`(`*`ret`*`)`
Take in the return and shove it into the couchdb database.

`salt.returners.couchdb_return.`**`set_salt_view`**`()`
Helper function that sets the salt design document. Uses get_valid_salt_views and some hardcoded values.

salt.returners.django_return

A returner that will infor a Django system that returns are available using Django's signal system.

https://docs.djangoproject.com/en/dev/topics/signals/

It is up to the Django developer to register necessary handlers with the signals provided by this returner and process returns as necessary.

The easiest way to use signals is to import them from this returner directly and then use a decorator to register them.

An example Django module that registers a function called `returner_callback' with this module's `returner' function:

```
import salt.returners.django_return from django.dispatch import receiver

@receiver(salt.returners.django_return, sender=returner) def returner_callback(sender, ret):

    print(`I received {0} from {1}'.format(ret, sender))
```

salt.returners.django_return.**prep_jid**(*nocache=False*, *passed_jid=None*)
Do any work necessary to prepare a JID, including sending a custom ID

salt.returners.django_return.**returner**(*ret*)
Signal a Django server that a return is available

salt.returners.django_return.**save_load**(*jid*, *load*)
Save the load to the specified jid

salt.returners.elasticsearch_return

Return data to an elasticsearch server for indexing.

maintainer Jurnell Cockhren <jurnell.cockhren@sophicware.com>

maturity New

depends elasticsearch-py, jsonpickle

platform all

To enable this returner the elasticsearch python client must be installed on the desired minions (all or some subset).

The required configuration is as follows:

```
elasticsearch:
  host: 'somehost.example.com:9200'
  index: 'salt'
  number_of_shards: 1 (optional)
  number_of_replicas: 0 (optional)
```

or to specify multiple elasticsearch hosts for resiliency:

```
elasticsearch:
  host:
    - 'somehost.example.com:9200'
    - 'anotherhost.example.com:9200'
    - 'yetanotherhost.example.com:9200'
  index: 'salt'
  number_of_shards: 1 (optional)
  number_of_replicas: 0 (optional)
```

The above configuration can be placed in a targeted pillar, minion or master configurations.

To use the returner per salt call:

```
salt '*' test.ping --return elasticsearch
```

In order to have the returner apply to all minions:

```
ext_job_cache: elasticsearch
```

salt.returners.elasticsearch_return.**prep_jid**(*nocache=False*, *passed_jid=None*)
Do any work necessary to prepare a JID, including sending a custom id

salt.returners.elasticsearch_return.**returner**(*ret*)
Process the return from Salt

salt.returners.etcd_return

Return data to an etcd server or cluster

depends

- python-etcd

In order to return to an etcd server, a profile should be created in the master configuration file:

```
my_etcd_config:
  etcd.host: 127.0.0.1
  etcd.port: 4001
```

It is technically possible to configure etcd without using a profile, but this is not considered to be a best practice, especially when multiple etcd servers or clusters are available.

```
etcd.host: 127.0.0.1
etcd.port: 4001
```

Additionally, two more options must be specified in the top-level configuration in order to use the etcd returner:

```
etcd.returner: my_etcd_config
etcd.returner_root: /salt/return
```

The `etcd.returner` option specifies which configuration profile to use. The `etcd.returner_root` option specifies the path inside etcd to use as the root of the returner system.

Once the etcd options are configured, the returner may be used:

CLI Example:

salt `*' test.ping --return etcd

salt.returners.etcd_return.**get_fun**()
Return a dict of the last function called for all minions

salt.returners.etcd_return.**get_jid**(*jid*)
Return the information returned when the specified job id was executed

salt.returners.etcd_return.**get_jids**()
Return a list of all job ids

salt.returners.etcd_return.**get_load**(*jid*)
Return the load data that marks a specified jid

salt.returners.etcd_return.**get_minions**()
Return a list of minions

salt.returners.etcd_return.**prep_jid**(*nocache=False*, *passed_jid=None*)
Do any work necessary to prepare a JID, including sending a custom id

salt.returners.etcd_return.**returner**(*ret*)
Return data to an etcd server or cluster

salt.returners.etcd_return.**save_load**(*jid*, *load*)
Save the load to the specified jid

salt.returners.kafka_return

Return data to a Kafka topic

maintainer Christer Edwards (christer.edwards@gmail.com)

maturity 0.1

depends kafka-python

platform all

To enable this returner install kafka-python and enable the following settings in the minion config:

returner.kafka.hostnames:

- ``server1''
- ``server2''
- ``server3''

returner.kafka.topic: `topic'

To use the kafka returner, append `--return kafka' to the Salt command, eg;

salt `*' test.ping --return kafka

salt.returners.kafka_return.**returner**(*ret*)
Return information to a Kafka server

salt.returners.local

The local returner is used to test the returner interface, it just prints the return data to the console to verify that it is being passed properly

To use the local returner, append `--return local' to the salt command. ex:

salt `*' test.ping --return local

salt.returners.local.**event_return**(*event*)
Print event return data to the terminal to verify functionality

salt.returners.local.**returner**(*ret*)
Print the return data to the terminal to verify functionality

salt.returners.local_cache

Return data to local job cache

salt.returners.local_cache.**clean_old_jobs**()
Clean out the old jobs from the job cache

salt.returners.local_cache.**get_jid**(*jid*)
Return the information returned when the specified job id was executed

salt.returners.local_cache.**get_jids**()
Return a list of all job ids

salt.returners.local_cache.**get_load**(*jid*)
Return the load data that marks a specified jid

salt.returners.local_cache.**prep_jid**(*nocache=False*, *passed_jid=None*)
Return a job id and prepare the job id directory This is the function responsible for making sure jids don't collide (unless its passed a jid) So do what you have to do to make sure that stays the case

salt.returners.local_cache.**returner**(*load*)
Return data to the local job cache

salt.returners.local_cache.**save_load**(*jid*, *clear_load*)
Save the load to the specified jid

salt.returners.memcache_return

Return data to a memcache server

To enable this returner the minion will need the python client for memcache installed and the following values configured in the minion or master config, these are the defaults.

```
memcache.host: 'localhost'
memcache.port: '11211'
```

Alternative configuration values can be used by prefacing the configuration. Any values not found in the alternative configuration will be pulled from the default location.

```
alternative.memcache.host: 'localhost'
alternative.memcache.port: '11211'
```

python2-memcache uses `localhost' and `11211' as syntax on connection.

To use the memcache returner, append `--return memcache' to the salt command.

```
salt '*' test.ping --return memcache
```

To use the alternative configuration, append `--return_config alternative' to the salt command.

New in version 2015.5.0.

```
salt '*' test.ping --return memcache --return_config alternative
```

salt.returners.memcache_return.**get_fun**(*fun*)
Return a dict of the last function called for all minions

salt.returners.memcache_return.**get_jid**(*jid*)
Return the information returned when the specified job id was executed

salt.returners.memcache_return.**get_jids**()
Return a list of all job ids

salt.returners.memcache_return.**get_load**(*jid*)
Return the load data that marks a specified jid

salt.returners.memcache_return.**get_minions**()
Return a list of minions

salt.returners.memcache_return.**prep_jid**(*nocache=False*, *passed_jid=None*)
Do any work necessary to prepare a JID, including sending a custom id

salt.returners.memcache_return.**returner**(*ret*)
Return data to a memcache data store

salt.returners.memcache_return.**save_load**(*jid*, *load*)
Save the load to the specified jid

salt.returners.mongo_future_return

Return data to a mongodb server

Required python modules: pymongo

This returner will send data from the minions to a MongoDB server. To configure the settings for your MongoDB server, add the following lines to the minion config files:

You can also ask for indexes creation on the most common used fields, which should greatly improve performance. Indexes are not created by default.

mongo.indexes: true

Alternative configuration values can be used by prefacing the configuration. Any values not found in the alternative configuration will be pulled from the default location:

```
alternative.mongo.db: <database name>
alternative.mongo.host: <server ip address>
alternative.mongo.user: <MongoDB username>
alternative.mongo.password: <MongoDB user password>
alternative.mongo.port: 27017
```

This mongo returner is being developed to replace the default mongodb returner in the future and should not be considered API stable yet.

To use the mongo returner, append `--return mongo' to the salt command.

```
salt '*' test.ping --return mongo
```

To use the alternative configuration, append `--return_config alternative' to the salt command.

New in version 2015.5.0.

```
salt '*' test.ping --return mongo --return_config alternative
```

salt.returners.mongo_future_return.**get_fun**(*fun*)
Return the most recent jobs that have executed the named function

salt.returners.mongo_future_return.**get_jid**(*jid*)
Return the return information associated with a jid

salt.returners.mongo_future_return.**get_jids**()
Return a list of job ids

salt.returners.mongo_future_return.**get_load**(*jid*)
Return the load associated with a given job id

salt.returners.mongo_future_return.**get_minions**()
Return a list of minions

salt.returners.mongo_future_return.**prep_jid**(*nocache=False, passed_jid=None*)
Do any work necessary to prepare a JID, including sending a custom id

salt.returners.mongo_future_return.**returner**(*ret*)
Return data to a mongodb server

salt.returners.mongo_future_return.**save_load**(*jid, load*)
Save the load for a given job id

salt.returners.mongo_return

Return data to a mongodb server

Required python modules: pymongo

This returner will send data from the minions to a MongoDB server. To configure the settings for your MongoDB server, add the following lines to the minion config files.

```
mongo.db: <database name>
mongo.host: <server ip address>
mongo.user: <MongoDB username>
mongo.password: <MongoDB user password>
mongo.port: 27017
```

Alternative configuration values can be used by prefacing the configuration. Any values not found in the alternative configuration will be pulled from the default location.

```
alternative.mongo.db: <database name>
alternative.mongo.host: <server ip address>
alternative.mongo.user: <MongoDB username>
alternative.mongo.password: <MongoDB user password>
alternative.mongo.port: 27017
```

To use the mongo returner, append `--return mongo' to the salt command.

```
salt '*' test.ping --return mongo_return
```

To use the alternative configuration, append `--return_config alternative' to the salt command.

New in version 2015.5.0.

```
salt '*' test.ping --return mongo_return --return_config alternative
```

salt.returners.mongo_return.**get_fun**(*fun*)
Return the most recent jobs that have executed the named function

salt.returners.mongo_return.**get_jid**(*jid*)
Return the return information associated with a jid

salt.returners.mongo_return.**prep_jid**(*nocache=False, passed_jid=None*)
Do any work necessary to prepare a JID, including sending a custom id

salt.returners.mongo_return.**returner**(*ret*)
Return data to a mongodb server

salt.returners.multi_returner

Read/Write multiple returners

salt.returners.multi_returner.**clean_old_jobs**()
Clean out the old jobs from all returners (if you have it)

salt.returners.multi_returner.**get_jid**(*jid*)
Merge the return data from all returners

salt.returners.multi_returner.**get_jids**()
Return all job data from all returners

salt.returners.multi_returner.**get_load**(*jid*)
Merge the load data from all returners

salt.returners.multi_returner.**prep_jid**(*nocache=False*, *passed_jid=None*)
Call both with prep_jid on all returners in multi_returner

TODO: finish this, what do do when you get different jids from 2 returners... since our jids are time based, this make this problem hard, because they aren't unique, meaning that we have to make sure that no one else got the jid and if they did we spin to get a new one, which means ``locking'' the jid in 2 returners is non-trivial

salt.returners.multi_returner.**returner**(*load*)
Write return to all returners in multi_returner

salt.returners.multi_returner.**save_load**(*jid*, *clear_load*)
Write load to all returners in multi_returner

salt.returners.mysql

Return data to a mysql server

maintainer Dave Boucha <dave@saltstack.com>, Seth House <shouse@saltstack.com>

maturity new

depends python-mysqldb

platform all

To enable this returner, the minion will need the python client for mysql installed and the following values configured in the minion or master config. These are the defaults:

```
mysql.host: 'salt'
mysql.user: 'salt'
mysql.pass: 'salt'
mysql.db: 'salt'
mysql.port: 3306
```

SSL is optional. The defaults are set to None. If you do not want to use SSL, either exclude these options or set them to None.

```
mysql.ssl_ca: None
mysql.ssl_cert: None
mysql.ssl_key: None
```

Alternative configuration values can be used by prefacing the configuration with *alternative.*. Any values not found in the alternative configuration will be pulled from the default location. As stated above, SSL configuration is optional. The following ssl options are simply for illustration purposes:

```
alternative.mysql.host: 'salt'
alternative.mysql.user: 'salt'
alternative.mysql.pass: 'salt'
alternative.mysql.db: 'salt'
alternative.mysql.port: 3306
alternative.mysql.ssl_ca: '/etc/pki/mysql/certs/localhost.pem'
alternative.mysql.ssl_cert: '/etc/pki/mysql/certs/localhost.crt'
alternative.mysql.ssl_key: '/etc/pki/mysql/certs/localhost.key'
```

Use the following mysql database schema:

```
CREATE DATABASE  `salt`
  DEFAULT CHARACTER SET utf8
  DEFAULT COLLATE utf8_general_ci;

USE `salt`;

--
-- Table structure for table `jids`
--

DROP TABLE IF EXISTS `jids`;
CREATE TABLE `jids` (
  `jid` varchar(255) NOT NULL,
  `load` mediumtext NOT NULL,
  UNIQUE KEY `jid` (`jid`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

--
-- Table structure for table `salt_returns`
--

DROP TABLE IF EXISTS `salt_returns`;
CREATE TABLE `salt_returns` (
  `fun` varchar(50) NOT NULL,
  `jid` varchar(255) NOT NULL,
  `return` mediumtext NOT NULL,
  `id` varchar(255) NOT NULL,
  `success` varchar(10) NOT NULL,
  `full_ret` mediumtext NOT NULL,
  `alter_time` TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  KEY `id` (`id`),
  KEY `jid` (`jid`),
  KEY `fun` (`fun`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

--
-- Table structure for table `salt_events`
--

DROP TABLE IF EXISTS `salt_events`;
```

```
CREATE TABLE `salt_events` (
`id` BIGINT NOT NULL AUTO_INCREMENT,
`tag` varchar(255) NOT NULL,
`data` varchar(1024) NOT NULL,
`alter_time` TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
`master_id` varchar(255) NOT NULL,
PRIMARY KEY (`id`),
KEY `tag` (`tag`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Required python modules: MySQLdb

To use the mysql returner, append `--return mysql' to the salt command.

```
salt '*' test.ping --return mysql
```

To use the alternative configuration, append `--return_config alternative' to the salt command.

New in version 2015.5.0.

```
salt '*' test.ping --return mysql --return_config alternative
```

salt.returners.mysql.**event_return**(*events*)
Return event to mysql server

Requires that configuration be enabled via `event_return' option in master config.

salt.returners.mysql.**get_fun**(*fun*)
Return a dict of the last function called for all minions

salt.returners.mysql.**get_jid**(*jid*)
Return the information returned when the specified job id was executed

salt.returners.mysql.**get_jids**()
Return a list of all job ids

salt.returners.mysql.**get_load**(*jid*)
Return the load data that marks a specified jid

salt.returners.mysql.**get_minions**()
Return a list of minions

salt.returners.mysql.**prep_jid**(*nocache=False*, *passed_jid=None*)
Do any work necessary to prepare a JID, including sending a custom id

salt.returners.mysql.**returner**(*ret*)
Return data to a mysql server

salt.returners.mysql.**save_load**(*jid*, *load*)
Save the load to the specified jid id

salt.returners.odbc

Return data to an ODBC compliant server. This driver was developed with Microsoft SQL Server in mind, but theoretically could be used to return data to any compliant ODBC database as long as there is a working ODBC driver for it on your minion platform.

maintainer

3. (a) Oldham (cr@saltstack.com)

maturity New

depends unixodbc, pyodbc, freetds (for SQL Server)

platform all

To enable this returner the minion will need

On Linux:

unixodbc (http://www.unixodbc.org) pyodbc (*pip install pyodbc*) The FreeTDS ODBC driver for SQL Server (http://www.freetds.org) or another compatible ODBC driver

On Windows:

TBD

unixODBC and FreeTDS need to be configured via /etc/odbcinst.ini and /etc/odbc.ini.

/etc/odbcinst.ini:

```
[TDS]
Description=TDS
Driver=/usr/lib/x86_64-linux-gnu/odbc/libtdsodbc.so
```

(Note the above Driver line needs to point to the location of the FreeTDS shared library. This example is for Ubuntu 14.04.)

/etc/odbc.ini:

```
[TS]
Description = "Salt Returner"
Driver=TDS
Server = <your server ip or fqdn>
Port = 1433
Database = salt
Trace = No
```

Also you need the following values configured in the minion or master config. Configure as you see fit:

```
returner.odbc.dsn: 'TS'
returner.odbc.user: 'salt'
returner.odbc.passwd: 'salt'
```

Alternative configuration values can be used by prefacing the configuration. Any values not found in the alternative configuration will be pulled from the default location:

```
alternative.returner.odbc.dsn: 'TS'
alternative.returner.odbc.user: 'salt'
alternative.returner.odbc.passwd: 'salt'
```

Running the following commands against Microsoft SQL Server in the desired database as the appropriate user should create the database tables correctly. Replace with equivalent SQL for other ODBC-compliant servers

```
  --
  -- Table structure for table 'jids'
  --

  if OBJECT_ID('dbo.jids', 'U') is not null
      DROP TABLE dbo.jids

  CREATE TABLE dbo.jids (
     jid   varchar(255) PRIMARY KEY,
     load  varchar(MAX) NOT NULL
   );

  --
  -- Table structure for table 'salt_returns'
  --
  IF OBJECT_ID('dbo.salt_returns', 'U') IS NOT NULL
      DROP TABLE dbo.salt_returns;

  CREATE TABLE dbo.salt_returns (
     added     datetime not null default (getdate()),
     fun       varchar(100) NOT NULL,
     jid       varchar(255) NOT NULL,
     retval    varchar(MAX) NOT NULL,
     id        varchar(255) NOT NULL,
     success   bit default(0) NOT NULL,
     full_ret  varchar(MAX)
   );

  CREATE INDEX salt_returns_added on dbo.salt_returns(added);
  CREATE INDEX salt_returns_id on dbo.salt_returns(id);
  CREATE INDEX salt_returns_jid on dbo.salt_returns(jid);
  CREATE INDEX salt_returns_fun on dbo.salt_returns(fun);

To use this returner, append '--return odbc' to the salt command.

.. code-block:: bash

  salt '*' status.diskusage --return odbc

To use the alternative configuration, append '--return_config alternative' to the salt command.

.. versionadded:: 2015.5.0

.. code-block:: bash

  salt '*' test.ping --return odbc --return_config alternative
```

salt.returners.odbc.**get_fun**(*fun*)
Return a dict of the last function called for all minions

salt.returners.odbc.**get_jid**(*jid*)
Return the information returned when the specified job id was executed

salt.returners.odbc.**get_jids**()
Return a list of all job ids

salt.returners.odbc.**get_load**(*jid*)
Return the load data that marks a specified jid

salt.returners.odbc.**get_minions**()
Return a list of minions

salt.returners.odbc.**prep_jid**(*nocache=False, passed_jid=None*)
Do any work necessary to prepare a JID, including sending a custom id

salt.returners.odbc.**returner**(*ret*)
Return data to an odbc server

salt.returners.odbc.**save_load**(*jid, load*)
Save the load to the specified jid id

salt.returners.postgres

Return data to a postgresql server

maintainer None

maturity New

depends psycopg2

platform all

To enable this returner the minion will need the psycopg2 installed and the following values configured in the minion or master config:

```
returner.postgres.host: 'salt'
returner.postgres.user: 'salt'
returner.postgres.passwd: 'salt'
returner.postgres.db: 'salt'
returner.postgres.port: 5432
```

Alternative configuration values can be used by prefacing the configuration. Any values not found in the alternative configuration will be pulled from the default location:

```
alternative.returner.postgres.host: 'salt'
alternative.returner.postgres.user: 'salt'
alternative.returner.postgres.passwd: 'salt'
alternative.returner.postgres.db: 'salt'
alternative.returner.postgres.port: 5432
```

Running the following commands as the postgres user should create the database correctly:

```
psql << EOF
CREATE ROLE salt WITH PASSWORD 'salt';
CREATE DATABASE salt WITH OWNER salt;
EOF

psql -h localhost -U salt << EOF
--
-- Table structure for table 'jids'
--

DROP TABLE IF EXISTS jids;
CREATE TABLE jids (
  jid   varchar(20) PRIMARY KEY,
  load  text NOT NULL
```

```
);

--
-- Table structure for table 'salt_returns'
--

DROP TABLE IF EXISTS salt_returns;
CREATE TABLE salt_returns (
  added     TIMESTAMP WITH TIME ZONE DEFAULT now(),
  fun       text NOT NULL,
  jid       varchar(20) NOT NULL,
  return    text NOT NULL,
  id        text NOT NULL,
  success   boolean
);
CREATE INDEX ON salt_returns (added);
CREATE INDEX ON salt_returns (id);
CREATE INDEX ON salt_returns (jid);
CREATE INDEX ON salt_returns (fun);
EOF
```

Required python modules: psycopg2

To use the postgres returner, append `--return postgres' to the salt command.

```
salt '*' test.ping --return postgres
```

To use the alternative configuration, append `--return_config alternative' to the salt command.

New in version 2015.5.0.

```
salt '*' test.ping --return postgres --return_config alternative
```

salt.returners.postgres.**get_fun**(*fun*)
Return a dict of the last function called for all minions

salt.returners.postgres.**get_jid**(*jid*)
Return the information returned when the specified job id was executed

salt.returners.postgres.**get_jids**()
Return a list of all job ids

salt.returners.postgres.**get_load**(*jid*)
Return the load data that marks a specified jid

salt.returners.postgres.**get_minions**()
Return a list of minions

salt.returners.postgres.**prep_jid**(*nocache=False*, *passed_jid=None*)
Do any work necessary to prepare a JID, including sending a custom id

salt.returners.postgres.**returner**(*ret*)
Return data to a postgres server

salt.returners.postgres.**save_load**(*jid*, *load*)
Save the load to the specified jid id

salt.returners.postgres_local_cache

Use a postgresql server for the master job cache. This helps the job cache to cope with scale.

maintainer gjredelinghuys@gmail.com

maturity New

depends psycopg2

platform all

To enable this returner the minion will need the psycopg2 installed and the following values configured in the master config:

```
master_job_cache: postgres_local_cache
master_job_cache.postgres.host: 'salt'
master_job_cache.postgres.user: 'salt'
master_job_cache.postgres.passwd: 'salt'
master_job_cache.postgres.db: 'salt'
master_job_cache.postgres.port: 5432
```

Running the following command as the postgres user should create the database correctly:

```
psql << EOF
CREATE ROLE salt WITH PASSWORD 'salt';
CREATE DATABASE salt WITH OWNER salt;
EOF
```

and then:

```
psql -h localhost -U salt << EOF
--
-- Table structure for table 'jids'
--

DROP TABLE IF EXISTS jids;
CREATE TABLE jids (
  jid   varchar(20) PRIMARY KEY,
  started TIMESTAMP WITH TIME ZONE DEFAULT now(),
  tgt_type text NOT NULL,
  cmd text NOT NULL,
  tgt text NOT NULL,
  kwargs text NOT NULL,
  ret text NOT NULL,
  username text NOT NULL,
  arg text NOT NULL,
  fun text NOT NULL
);

--
-- Table structure for table 'salt_returns'
--

DROP TABLE IF EXISTS salt_returns;
CREATE TABLE salt_returns (
  added     TIMESTAMP WITH TIME ZONE DEFAULT now(),
  fun       text NOT NULL,
```

```
  jid       varchar(20) NOT NULL,
  return    text NOT NULL,
  id        text NOT NULL,
  success   boolean
);
CREATE INDEX ON salt_returns (added);
CREATE INDEX ON salt_returns (id);
CREATE INDEX ON salt_returns (jid);
CREATE INDEX ON salt_returns (fun);
EOF
```

Required python modules: psycopg2

salt.returners.postgres_local_cache.**clean_old_jobs**()
Clean out the old jobs from the job cache

salt.returners.postgres_local_cache.**get_jid**(*jid*)
Return the information returned when the specified job id was executed

salt.returners.postgres_local_cache.**get_jids**()
Return a list of all job ids For master job cache this also formats the output and returns a string

salt.returners.postgres_local_cache.**get_load**(*jid*)
Return the load data that marks a specified jid

salt.returners.postgres_local_cache.**prep_jid**(*nocache=False*, *passed_jid=None*)
Return a job id and prepare the job id directory This is the function responsible for making sure jids don't collide (unless its passed a jid). So do what you have to do to make sure that stays the case

salt.returners.postgres_local_cache.**returner**(*load*)
Return data to a postgres server

salt.returners.postgres_local_cache.**save_load**(*jid*, *clear_load*)
Save the load to the specified jid id

salt.returners.redis_return

Return data to a redis server

To enable this returner the minion will need the python client for redis installed and the following values configured in the minion or master config, these are the defaults:

```
redis.db: '0'
redis.host: 'salt'
redis.port: 6379
```

Alternative configuration values can be used by prefacing the configuration. Any values not found in the alternative configuration will be pulled from the default location:

```
alternative.redis.db: '0'
alternative.redis.host: 'salt'
alternative.redis.port: 6379
```

To use the redis returner, append `--return redis' to the salt command.

```
salt '*' test.ping --return redis
```

To use the alternative configuration, append `--return_config alternative' to the salt command.

New in version 2015.5.0.

```
salt '*' test.ping --return redis --return_config alternative
```

salt.returners.redis_return.**get_fun**(*fun*)
Return a dict of the last function called for all minions

salt.returners.redis_return.**get_jid**(*jid*)
Return the information returned when the specified job id was executed

salt.returners.redis_return.**get_jids**()
Return a list of all job ids

salt.returners.redis_return.**get_load**(*jid*)
Return the load data that marks a specified jid

salt.returners.redis_return.**get_minions**()
Return a list of minions

salt.returners.redis_return.**prep_jid**(*nocache=False*, *passed_jid=None*)
Do any work necessary to prepare a JID, including sending a custom id

salt.returners.redis_return.**returner**(*ret*)
Return data to a redis data store

salt.returners.redis_return.**save_load**(*jid*, *load*)
Save the load to the specified jid

salt.returners.sentry_return

Salt returner that report execution results back to sentry. The returner will inspect the payload to identify errors and flag them as such.

Pillar needs something like:

```
raven:
  servers:
    - http://192.168.1.1
    - https://sentry.example.com
  public_key: deadbeefdeadbeefdeadbeefdeadbeef
  secret_key: beefdeadbeefdeadbeefdeadbeefdead
  project: 1
  tags:
    - os
    - master
    - saltversion
    - cpuarch
```

and https://pypi.python.org/pypi/raven installed

The tags list (optional) specifies grains items that will be used as sentry tags, allowing tagging of events in the sentry ui.

salt.returners.sentry_return.**prep_jid**(*nocache=False*, *passed_jid=None*)
Do any work necessary to prepare a JID, including sending a custom id

salt.returners.sentry_return.**returner**(*ret*)
Log outcome to sentry. The returner tries to identify errors and report them as such. All other messages will be reported at info level.

salt.returners.sms_return

Return data by SMS.

New in version 2015.5.0.

maintainer Damian Myerscough

maturity new

depends twilio

platform all

To enable this returner the minion will need the python twilio library installed and the following values configured in the minion or master config:

```
twilio.sid: 'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
twilio.token: 'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
twilio.to: '+1415XXXXXXX'
twilio.from: '+1650XXXXXXX'
```

To use the sms returner, append `--return sms' to the salt command.

```
salt '*' test.ping --return sms
```

salt.returners.sms_return.**returner**(*ret*)
Return a response in an SMS message

salt.returners.smtp_return

Return salt data via email

The following fields can be set in the minion conf file:

```
smtp.from (required)
smtp.to (required)
smtp.host (required)
smtp.port (optional, defaults to 25)
smtp.username (optional)
smtp.password (optional)
smtp.tls (optional, defaults to False)
smtp.subject (optional, but helpful)
smtp.gpgowner (optional)
smtp.fields (optional)
```

Alternative configuration values can be used by prefacing the configuration. Any values not found in the alternative configuration will be pulled from the default location:

```
alternative.smtp.from
alternative.smtp.to
alternative.smtp.host
alternative.smtp.port
alternative.smtp.username
alternative.smtp.password
alternative.smtp.tls
alternative.smtp.subject
alternative.smtp.gpgowner
alternative.smtp.fields
```

There are a few things to keep in mind:

- If a username is used, a password is also required. It is recommended (but not required) to use the TLS setting when authenticating.
- You should at least declare a subject, but you don't have to.
- The use of encryption, i.e. setting gpgowner in your settings, requires python-gnupg to be installed.
- The field gpgowner specifies a user's ~/.gpg directory. This must contain a gpg public key matching the address the mail is sent to. If left unset, no encryption will be used.
- smtp.fields lets you include the value(s) of various fields in the subject line of the email. These are comma-delimited. For instance:

```
    smtp.fields: id,fun
```

  ...will display the id of the minion and the name of the function in the subject line. You may also use `jid' (the job id), but it is generally recommended not to use `return', which contains the entire return data structure (which can be very large). Also note that the subject is always unencrypted.

To use the SMTP returner, append `--return smtp' to the salt command.

```
salt '*' test.ping --return smtp
```

To use the alternative configuration, append `--return_config alternative' to the salt command.

New in version 2015.5.0.

```
salt '*' test.ping --return smtp --return_config alternative
```

salt.returners.smtp_return.**prep_jid**(*nocache=False*, *passed_jid=None*)
Do any work necessary to prepare a JID, including sending a custom id

salt.returners.smtp_return.**returner**(*ret*)
Send an email with the data

salt.returners.sqlite3

Insert minion return data into a sqlite3 database

maintainer Mickey Malone <mickey.malone@gmail.com>

maturity New

depends None

platform All

Sqlite3 is a serverless database that lives in a single file. In order to use this returner the database file must exist, have the appropriate schema defined, and be accessible to the user whom the minion process is running as. This returner requires the following values configured in the master or minion config:

```
returner.sqlite3.database: /usr/lib/salt/salt.db
returner.sqlite3.timeout: 5.0
```

Alternative configuration values can be used by prefacing the configuration. Any values not found in the alternative configuration will be pulled from the default location:

```
alternative.returner.sqlite3.database: /usr/lib/salt/salt.db
alternative.returner.sqlite3.timeout: 5.0
```

Use the commands to create the sqlite3 database and tables:

```
sqlite3 /usr/lib/salt/salt.db << EOF
--
-- Table structure for table 'jids'
--

CREATE TABLE jids (
  jid TEXT PRIMARY KEY,
  load TEXT NOT NULL
  );

--
-- Table structure for table 'salt_returns'
--

CREATE TABLE salt_returns (
  fun TEXT KEY,
  jid TEXT KEY,
  id TEXT KEY,
  fun_args TEXT,
  date TEXT NOT NULL,
  full_ret TEXT NOT NULL,
  success TEXT NOT NULL
  );
EOF
```

To use the sqlite returner, append `--return sqlite3' to the salt command.

```
salt '*' test.ping --return sqlite3
```

To use the alternative configuration, append `--return_config alternative' to the salt command.

New in version 2015.5.0.

```
salt '*' test.ping --return sqlite3 --return_config alternative
```

salt.returners.sqlite3_return.**get_fun**(*fun*)
Return a dict of the last function called for all minions

salt.returners.sqlite3_return.**get_jid**(*jid*)
Return the information returned from a specified jid

salt.returners.sqlite3_return.**get_jids**()
Return a list of all job ids

salt.returners.sqlite3_return.**get_load**(*jid*)
Return the load from a specified jid

salt.returners.sqlite3_return.**get_minions**()
Return a list of minions

salt.returners.sqlite3_return.**prep_jid**(*nocache=False*, *passed_jid=None*)
Do any work necessary to prepare a JID, including sending a custom id

salt.returners.sqlite3_return.**returner**(*ret*)
Insert minion return data into the sqlite3 database

salt.returners.sqlite3_return.**save_load**(*jid*, *load*)
Save the load to the specified jid

salt.returners.syslog_return

Return data to the host operating system's syslog facility

Required python modules: syslog, json

The syslog returner simply reuses the operating system's syslog facility to log return data

To use the syslog returner, append `--return syslog' to the salt command.

```
salt '*' test.ping --return syslog
```

salt.returners.syslog_return.**prep_jid**(*nocache=False*, *passed_jid=None*)
Do any work necessary to prepare a JID, including sending a custom id

salt.returners.syslog_return.**returner**(*ret*)
Return data to the local syslog

salt.returners.xmpp_return

Return salt data via xmpp

The following fields can be set in the minion conf file:

```
xmpp.jid (required)
xmpp.password (required)
xmpp.recipient (required)
xmpp.profile (optional)
```

Alternative configuration values can be used by prefacing the configuration. Any values not found in the alternative configuration will be pulled from the default location:

```
xmpp.jid
xmpp.password
xmpp.recipient
xmpp.profile
```

XMPP settings may also be configured as:

```
xmpp:
    jid: user@xmpp.domain.com/resource
    password: password
    recipient: user@xmpp.example.com

alternative.xmpp:
    jid: user@xmpp.domain.com/resource
    password: password
    recipient: someone@xmpp.example.com

xmpp_profile:
    jid: user@xmpp.domain.com/resource
    password: password

xmpp:
    profile: xmpp_profile
    recipient: user@xmpp.example.com

alternative.xmpp:
    profile: xmpp_profile
    recipient: someone-else@xmpp.example.com
```

To use the XMPP returner, append `--return xmpp' to the salt command.

```
salt '*' test.ping --return xmpp
```

To use the alternative configuration, append `--return_config alternative' to the salt command.

New in version 2015.5.0.

```
salt '*' test.ping --return xmpp --return_config alternative
```

class salt.returners.xmpp_return.**SendMsgBot**(*jid*, *password*, *recipient*, *msg*)

start(*event*)

salt.returners.xmpp_return.**returner**(*ret*)

Send an xmpp message with the data

22.24 Full list of builtin roster modules

ansible	Read in an Ansible inventory file or script
cache	Use the minion cache on the master to derive IP addresses based on minion ID.
cloud	Use the cloud cache on the master to derive IPv4 addresses based on minion ID.
clustershell	This roster resolves hostname in a pdsh/clustershell style.
flat	Read in the roster from a flat file using the renderer system
scan	Scan a netmask or ipaddr for open ssh ports

22.24.1 salt.roster.ansible

Read in an Ansible inventory file or script

Flat inventory files should be in the regular ansible inventory format.

```
[servers]
salt.gtmanfred.com ansible_ssh_user=gtmanfred ansible_ssh_host=127.0.0.1 ansible_ssh_port=22 ansible_

[desktop]
home ansible_ssh_user=gtmanfred ansible_ssh_host=12.34.56.78 ansible_ssh_port=23 ansible_ssh_pass='pa

[computers:children]
desktop
servers

[names:vars]
http_port=80
```

then salt-ssh can be used to hit any of them

```
[~]# salt-ssh all test.ping
salt.gtmanfred.com:
    True
home:
    True
[~]# salt-ssh desktop test.ping
home:
    True
[~]# salt-ssh computers test.ping
salt.gtmanfred.com:
    True
home:
    True
[~]# salt-ssh salt.gtmanfred.com test.ping
salt.gtmanfred.com:
    True
```

There is also the option of specifying a dynamic inventory, and generating it on the fly

```
#!/bin/bash
echo '{
  "servers": {
    "hosts": [
      "salt.gtmanfred.com"
    ]
  },
  "desktop": {
    "hosts": [
      "home"
    ]
  },
  "computers": {
    "hosts":{},
    "children": [
      "desktop",
      "servers"
    ]
  },
  "_meta": {
    "hostvars": {
```

```
      "salt.gtmanfred.com": {
        "ansible_ssh_user": "gtmanfred",
        "ansible_ssh_host": "127.0.0.1",
        "ansible_sudo_pass": "password",
        "ansible_ssh_port": 22
      },
      "home": {
        "ansible_ssh_user": "gtmanfred",
        "ansible_ssh_host": "12.34.56.78",
        "ansible_sudo_pass": "password",
        "ansible_ssh_port": 23
      }
    }
  }
}'
```

This is the format that an inventory script needs to output to work with ansible, and thus here.

Any of the [groups] or direct hostnames will return. The `all' is special, and returns everything.

class salt.roster.ansible.**Inventory**(*tgt*, *tgt_type='glob'*, *inventory_file='/etc/salt/roster'*)
Matcher for static inventory files

class salt.roster.ansible.**Script**(*tgt*, *tgt_type='glob'*, *inventory_file='/etc/salt/roster'*)
Matcher for Inventory scripts

class salt.roster.ansible.**Target**

get_glob()
Return minions that match via glob

targets()
Execute the correct tgt_type routine and return

salt.roster.ansible.**targets**(*tgt*, *tgt_type='glob'*, ***kwargs*)
Return the targets from the ansible inventory_file Default: /etc/salt/roster

22.24.2 salt.roster.cache

Use the minion cache on the master to derive IP addresses based on minion ID.

Currently only contains logic to return an IPv4 address; does not handle IPv6, or authentication (passwords, keys, etc).

It is possible to configure this roster to prefer a particular type of IP over another. To configure the order, set the roster_order in the master config file. The default for this is:

```
roster_order:
  - public
  - private
  - local
```

salt.roster.cache.**extract_ipv4**(*roster_order*, *ipv4*)
Extract the preferred IP address from the ipv4 grain

salt.roster.cache.**targets**(*tgt*, *tgt_type='glob'*, ***kwargs*)
Return the targets from the flat yaml file, checks opts for location but defaults to /etc/salt/roster

22.24.3 salt.roster.cloud

Use the cloud cache on the master to derive IPv4 addresses based on minion ID.

This roster requires that the minion in question was created using at least the 2015.5.0 version of Salt Cloud. Starting with the 2015.5.0 release, Salt Cloud maintains an index of minions that it creates and deletes. This index tracks the provider and profile configuration used to provision the minion, including authentication information. So long as this configuration remains current, it can be used by Salt SSH to log into any minion in the index.

`salt.roster.cloud.`**`extract_ipv4`**(*roster_order*, *ipv4*)
Extract the preferred IP address from the ipv4 grain

`salt.roster.cloud.`**`targets`**(*tgt*, *tgt_type='glob'*, ***kwargs*)
Return the targets from the flat yaml file, checks opts for location but defaults to /etc/salt/roster

22.24.4 salt.roster.clustershell

This roster resolves hostname in a pdsh/clustershell style.

depends

- clustershell, https://github.com/cea-hpc/clustershell

When you want to use host globs for target matching, use --roster clustershell.

Example: salt-ssh --roster clustershell `server_[1-10,21-30],test_server[5,7,9]' test.ping

`salt.roster.clustershell.`**`targets`**(*tgt*, *tgt_type='glob'*, ***kwargs*)
Return the targets

22.24.5 salt.roster.flat

Read in the roster from a flat file using the renderer system

class `salt.roster.flat.`**`RosterMatcher`**(*raw*, *tgt*, *tgt_type*, *ipv='ipv4'*)
Matcher for the roster data structure

`get_data`(*minion*)
Return the configured ip

`ret_glob_minions`()
Return minions that match via glob

`ret_list_minions`()
Return minions that match via list

`ret_pcre_minions`()
Return minions that match via pcre

`targets`()
Execute the correct tgt_type routine and return

`salt.roster.flat.`**`targets`**(*tgt*, *tgt_type='glob'*, ***kwargs*)
Return the targets from the flat yaml file, checks opts for location but defaults to /etc/salt/roster

22.24.6 salt.roster.scan

Scan a netmask or ipaddr for open ssh ports

class salt.roster.scan.**RosterMatcher**(*tgt*, *tgt_type*)
Matcher for the roster data structure

targets()
Return ip addrs based on netmask, sitting in the ``glob'' spot because it is the default

salt.roster.scan.**targets**(*tgt*, *tgt_type='glob'*, ***kwargs*)
Return the targets from the flat yaml file, checks opts for location but defaults to /etc/salt/roster

22.25 Salt Runners

Salt runners are convenience applications executed with the salt-run command.

Salt runners work similarly to Salt execution modules however they execute on the Salt master itself instead of remote Salt minions.

A Salt runner can be a simple client call or a complex application.

See also:

The full list of runners

22.25.1 Full list of runner modules

cache	Return cached data from minions
cloud	The Salt Cloud Runner
doc	A runner module to collect and display the inline documentation from the
drac	Manage Dell DRAC from the Master
error	Error generator to enable integration testing of salt runner error handling
f5	Runner to provide F5 Load Balancer functionality
fileserver	Directly manage the Salt fileserver plugins
git_pillar	Directly manage the salt git_pillar plugin
http	Module for making various web calls.
jobs	A convenience system to manage jobs, both active and already run
launchd	Manage launchd plist files
lxc	Control Linux Containers via Salt
manage	General management functions for salt, tools like seeing what hosts are up
mine	A runner to access data from the salt mine
nacl	This runner helps create encrypted passwords that can be included in pillars.
network	Network tools to run from the Master
pagerduty	Runner Module for Firing Events via PagerDuty
pillar	Functions to interact with the pillar compiler on the master
queue	General management and processing of queues.
sdb	Runner for setting and querying data via the sdb API on the master
search	Runner frontend to search system
state	Execute overstate functions
survey	A general map/reduce style salt runner for aggregating results returned by several different minions.
test	This runner is used only for test purposes and servers no production purpose
thin	The thin runner is used to manage the salt thin systems.
virt	Control virtual machines via Salt
winrepo	Runner to manage Windows software repo

salt.runners.cache

Return cached data from minions

salt.runners.cache.**clear_all**(*tgt=None*, *expr_form='glob'*)
Clear the cached pillar, grains, and mine data of the targeted minions

CLI Example:

```
salt-run cache.clear_all
```

salt.runners.cache.**clear_grains**(*tgt=None*, *expr_form='glob'*)
Clear the cached grains data of the targeted minions

CLI Example:

```
salt-run cache.clear_grains
```

salt.runners.cache.**clear_mine**(*tgt=None*, *expr_form='glob'*)
Clear the cached mine data of the targeted minions

CLI Example:

```
salt-run cache.clear_mine
```

salt.runners.cache.**clear_mine_func**(*tgt=None*, *expr_form='glob'*, *clear_mine_func_flag=None*)
Clear the cached mine function data of the targeted minions

CLI Example:

```
salt-run cache.clear_mine_func tgt='*' clear_mine_func='network.interfaces'
```

salt.runners.cache.**clear_pillar**(*tgt=None*, *expr_form='glob'*)
Clear the cached pillar data of the targeted minions

CLI Example:

```
salt-run cache.clear_pillar
```

salt.runners.cache.**grains**(*tgt=None*, *expr_form='glob'*, *outputter=None*, ***kwargs*)
Return cached grains of the targeted minions

CLI Example:

```
salt-run cache.grains
```

salt.runners.cache.**mine**(*tgt=None*, *expr_form='glob'*, *outputter=None*, ***kwargs*)
Return cached mine data of the targeted minions

CLI Example:

```
salt-run cache.mine
```

salt.runners.cache.**pillar**(*tgt=None*, *expr_form='glob'*, *outputter=None*, ***kwargs*)
Return cached pillars of the targeted minions

CLI Example:

```
salt-run cache.pillar
```

salt.runners.cloud

The Salt Cloud Runner

This runner wraps the functionality of salt cloud making salt cloud routines available to all internal apis via the runner system

salt.runners.cloud.**action**(*fun=None, cloudmap=None, instances=None, provider=None, instance=None, **kwargs*)
Execute a single action on the given map/provider/instance

salt.runners.cloud.**create**(*provider, instances, **kwargs*)
Create an instance using Salt Cloud

CLI Example:

```
salt-run cloud.create my-ec2-config myinstance image=ami-1624987f size='t1.micro' ss
```

salt.runners.cloud.**destroy**(*instances*)
Destroy the named vm(s)

salt.runners.cloud.**full_query**(*query_type='list_nodes_full'*)
List all available cloud provider data

salt.runners.cloud.**list_images**(*provider='all'*)
List cloud provider images for the given providers

salt.runners.cloud.**list_locations**(*provider='all'*)
List cloud provider sizes for the given providers

salt.runners.cloud.**list_sizes**(*provider='all'*)
List cloud provider sizes for the given providers

salt.runners.cloud.**map_run**(*path, **kwargs*)
Execute a salt cloud map file

salt.runners.cloud.**profile**(*prof=None, instances=None, **kwargs*)
Create a cloud vm with the given profile and instances, instances can be a list or comma-delimited string

CLI Example:

```
salt-run cloud.profile prof=my-ec2 instances=node1,node2,node3
```

salt.runners.cloud.**query**(*query_type='list_nodes'*)
List cloud provider data for all providers

salt.runners.cloud.**select_query**(*query_type='list_nodes_select'*)
List selected nodes

salt.runners.doc

A runner module to collect and display the inline documentation from the various module types

salt.runners.doc.**execution**()
Collect all the sys.doc output from each minion and return the aggregate

CLI Example:

```
salt-run doc.execution
```

salt.runners.doc.**runner**()
Return all inline documentation for runner modules

CLI Example:

```
salt-run doc.runner
```

salt.runners.doc.**wheel**()
Return all inline documentation for wheel modules

CLI Example:

```
salt-run doc.wheel
```

salt.runners.drac

Manage Dell DRAC from the Master

The login credentials need to be configured in the Salt master configuration file.

salt.runners.drac.**poweroff**(*hostname*, *timeout=20*)
Power server off

CLI Example:

```
salt-run drac.poweroff example.com
```

salt.runners.drac.**poweron**(*hostname*, *timeout=20*)
Power server on

CLI Example:

```
salt-run drac.poweron example.com
```

salt.runners.drac.**pxe**(*hostname*, *timeout=20*)
Connect to the Dell DRAC and have the boot order set to PXE and power cycle the system to PXE boot

CLI Example:

```
salt-run drac.pxe example.com
```

salt.runners.drac.**reboot**(*hostname*, *timeout=20*)
Reboot a server using the Dell DRAC

CLI Example:

```
salt-run drac.reboot example.com
```

salt.runners.drac.**version**(*hostname*, *timeout=20*)
Display the version of DRAC

CLI Example:

```
salt-run drac.version example.com
```

salt.runners.error

Error generator to enable integration testing of salt runner error handling

salt.runners.error.**error**(*name=None*, *message=''*)
If name is None Then return empty dict

Otherwise raise an exception with __name__ from name, message from message

CLI Example:

```
salt-run error
salt-run error.error name="Exception" message="This is an error."
```

salt.runners.f5

Runner to provide F5 Load Balancer functionality

depends

- pycontrol Python module

configuration In order to connect to a F5 Load Balancer, you must specify in the Salt master configuration the currently avilable load balancers

```
load_balancers:
  bigip1.example.com
    username: admin
    password: secret
  bigip2.example.com:
    username: admin
    password: secret
```

class salt.runners.f5.**F5Mgmt**(*lb*, *username*, *password*)

add_pool_member(*name*, *port*, *pool_name*)
Add a node to a pool

check_member_pool(*member*, *pool_name*)
Check a pool member exists in a specific pool

check_pool(*name*)
Check to see if a pool exists

check_virtualserver(*name*)
Check to see if a virtual server exists

create_pool(*name*, *method='ROUND_ROBIN'*)
Create a pool on the F5 load balancer

create_vs(*name*, *ip*, *port*, *protocol*, *profile*, *pool_name*)
Create a virtual server

lbmethods()
List all the load balancer methods

salt.runners.f5.**add_pool_member**(*lb*, *name*, *port*, *pool_name*)
Add a node to a pool

CLI Examples:

```
salt-run f5.add_pool_member load_balancer 10.0.0.1 80 my_pool
```

salt.runners.f5.**check_member_pool**(*lb*, *member*, *pool_name*)
Check a pool member exists in a specific pool

CLI Examples:

```
salt-run f5.check_member_pool load_balancer 10.0.0.1 my_pool
```

salt.runners.f5.**check_pool**(*lb*, *name*)
Check to see if a pool exists

CLI Examples:

```
salt-run f5.check_pool load_balancer pool_name
```

salt.runners.f5.**check_virtualserver**(*lb*, *name*)
Check to see if a virtual server exists

CLI Examples:

```
salt-run f5.check_virtualserver load_balancer virtual_server
```

salt.runners.f5.**create_pool**(*lb*, *name*, *method='ROUND_ROBIN'*)
Create a pool on the F5 load balancer

CLI Examples:

```
salt-run f5.create_pool load_balancer pool_name loadbalance_method
salt-run f5.create_pool load_balancer my_pool ROUND_ROBIN
```

salt.runners.f5.**create_vs**(*lb*, *name*, *ip*, *port*, *protocol*, *profile*, *pool_name*)
Create a virtual server

CLI Examples:

```
salt-run f5.create_vs lbalancer vs_name 10.0.0.1 80 tcp http poolname
```

salt.runners.fileserver

Directly manage the Salt fileserver plugins

salt.runners.fileserver.**clear_cache**(*backend=None*)
New in version 2015.5.0.

Clear the fileserver cache from VCS fileserver backends (`git`, `hg`, `svn`). Executing this runner with no arguments will clear the cache for all enabled VCS fileserver backends, but this can be narrowed using the `backend` argument.

backend Only clear the update lock for the specified backend(s). If all passed backends start with a minus sign (`-`), then these backends will be excluded from the enabled backends. However, if there is a mix of backends with and without a minus sign (ex: `backend=-roots,git`) then the ones starting with a minus sign will be disregarded.

CLI Example:

```
salt-run fileserver.clear_cache
salt-run fileserver.clear_cache backend=git,hg
salt-run fileserver.clear_cache hg
salt-run fileserver.clear_cache -roots
```

salt.runners.fileserver.**clear_lock**(*backend=None*, *remote=None*)

New in version 2015.5.0.

Clear the fileserver update lock from VCS fileserver backends (git, hg, svn). This should only need to be done if a fileserver update was interrupted and a remote is not updating (generating a warning in the Master's log file). Executing this runner with no arguments will remove all update locks from all enabled VCS fileserver backends, but this can be narrowed by using the following arguments:

backend Only clear the update lock for the specified backend(s).

remote If not None, then any remotes which contain the passed string will have their lock cleared. For example, a `remote` value of **github** will remove the lock from all github.com remotes.

CLI Example:

```
salt-run fileserver.clear_lock
salt-run fileserver.clear_lock backend=git,hg
salt-run fileserver.clear_lock backend=git remote=github
salt-run fileserver.clear_lock remote=bitbucket
```

salt.runners.fileserver.**dir_list**(*saltenv='base'*, *backend=None*, *outputter=None*)

Return a list of directories in the given environment

saltenv [base] The salt fileserver environment to be listed

backend Narrow fileserver backends to a subset of the enabled ones. If all passed backends start with a minus sign (`-`), then these backends will be excluded from the enabled backends. However, if there is a mix of backends with and without a minus sign (ex: `backend=-roots,git`) then the ones starting with a minus sign will be disregarded.

New in version 2015.5.0.

CLI Example:

```
salt-run fileserver.dir_list
salt-run fileserver.dir_list saltenv=prod
salt-run fileserver.dir_list saltenv=dev backend=git
salt-run fileserver.dir_list base hg,roots
salt-run fileserver.dir_list -git
```

salt.runners.fileserver.**empty_dir_list**(*saltenv='base'*, *backend=None*, *outputter=None*)

New in version 2015.5.0.

Return a list of empty directories in the given environment

saltenv [base] The salt fileserver environment to be listed

backend Narrow fileserver backends to a subset of the enabled ones. If all passed backends start with a minus sign (`-`), then these backends will be excluded from the enabled backends. However, if there is a mix of backends with and without a minus sign (ex: `backend=-roots,git`) then the ones starting with a minus sign will be disregarded.

Note: Some backends (such as `git` and `hg`) do not support empty directories. So, passing `backend=git` or `backend=hg` will result in an empty list being returned.

CLI Example:

```
salt-run fileserver.empty_dir_list
salt-run fileserver.empty_dir_list saltenv=prod
salt-run fileserver.empty_dir_list backend=roots
```

salt.runners.fileserver.**envs**(*backend=None*, *sources=False*, *outputter=None*)
Return the available fileserver environments. If no backend is provided, then the environments for all configured backends will be returned.

backend Narrow fileserver backends to a subset of the enabled ones.

Changed in version 2015.5.0: If all passed backends start with a minus sign (`-`), then these backends will be excluded from the enabled backends. However, if there is a mix of backends with and without a minus sign (ex: `backend=-roots,git`) then the ones starting with a minus sign will be disregarded.

Additionally, fileserver backends can now be passed as a comma-separated list. In earlier versions, they needed to be passed as a python list (ex: `backend="['roots', 'git']"`)

CLI Example:

```
salt-run fileserver.envs
salt-run fileserver.envs backend=roots,git
salt-run fileserver.envs git
```

salt.runners.fileserver.**file_list**(*saltenv='base'*, *backend=None*, *outputter=None*)
Return a list of files from the salt fileserver

saltenv [base] The salt fileserver environment to be listed

backend Narrow fileserver backends to a subset of the enabled ones. If all passed backends start with a minus sign (`-`), then these backends will be excluded from the enabled backends. However, if there is a mix of backends with and without a minus sign (ex: `backend=-roots,git`) then the ones starting with a minus sign will be disregarded.

New in version 2015.5.0.

CLI Examples:

```
salt-run fileserver.file_list
salt-run fileserver.file_list saltenv=prod
salt-run fileserver.file_list saltenv=dev backend=git
salt-run fileserver.file_list base hg,roots
salt-run fileserver.file_list -git
```

salt.runners.fileserver.**lock**(*backend=None*, *remote=None*)
New in version 2015.5.0.

Set a fileserver update lock for VCS fileserver backends (`git`, `hg`, `svn`).

Note: This will only operate on enabled backends (those configured in :master_conf:`fileserver_backend`).

---

backend Only set the update lock for the specified backend(s).

remote If not None, then any remotes which contain the passed string will have their lock cleared. For example, a `remote` value of `*github.com*` will remove the lock from all github.com remotes.

CLI Example:

```
salt-run fileserver.lock
salt-run fileserver.lock backend=git,hg
salt-run fileserver.lock backend=git remote='*github.com*'
salt-run fileserver.lock remote=bitbucket
```

salt.runners.fileserver.**symlink_list**(*saltenv='base'*, *backend=None*, *outputter=None*)
Return a list of symlinked files and dirs

saltenv [base] The salt fileserver environment to be listed

backend Narrow fileserver backends to a subset of the enabled ones. If all passed backends start with a minus sign (`-`), then these backends will be excluded from the enabled backends. However, if there is a mix of backends with and without a minus sign (ex: `backend=-roots,git`) then the ones starting with a minus sign will be disregarded.

New in version 2015.5.0.

CLI Example:

```
salt-run fileserver.symlink_list
salt-run fileserver.symlink_list saltenv=prod
salt-run fileserver.symlink_list saltenv=dev backend=git
salt-run fileserver.symlink_list base hg,roots
salt-run fileserver.symlink_list -git
```

`salt.runners.fileserver.`**`update`**(*backend=None*)

Update the fileserver cache. If no backend is provided, then the cache for all configured backends will be updated.

backend Narrow fileserver backends to a subset of the enabled ones.

Changed in version 2015.5.0: If all passed backends start with a minus sign (`-`), then these backends will be excluded from the enabled backends. However, if there is a mix of backends with and without a minus sign (ex: `backend=-roots,git`) then the ones starting with a minus sign will be disregarded.

Additionally, fileserver backends can now be passed as a comma-separated list. In earlier versions, they needed to be passed as a python list (ex: `backend="['roots', 'git']"`)

CLI Example:

```
salt-run fileserver.update
salt-run fileserver.update backend=roots,git
```

salt.runners.git_pillar

Directly manage the salt git_pillar plugin

`salt.runners.git_pillar.`**`update`**(*branch*, *repo*)

Execute an update for the configured git fileserver backend for Pillar

CLI Example:

```
salt-run git_pillar.update branch='branch' repo='location'
```

salt.runners.http

Module for making various web calls. Primarily designed for webhooks and the like, but also useful for basic http testing.

New in version 2015.5.0.

`salt.runners.http.`**`query`**(*url*, *output=True*, ***kwargs*)

Query a resource, and decode the return data

CLI Example:

```
    salt-run http.query http://somelink.com/
    salt-run http.query http://somelink.com/ method=POST                params='key1=val1&key2=val2'
    salt-run http.query http://somelink.com/ method=POST                data='<xml>somecontent</xml>'
```

salt.runners.http.**update_ca_bundle**(*target=None*, *source=None*, *merge_files=None*)
Update the local CA bundle file from a URL

CLI Example:

```
    salt-run http.update_ca_bundle
    salt-run http.update_ca_bundle target=/path/to/cacerts.pem
    salt-run http.update_ca_bundle source=https://example.com/cacerts.pem
```

If the `target` is not specified, it will be pulled from the `ca_cert` configuration variable available to the master. If it cannot be found there, it will be placed at `<<FILE_ROOTS>>/cacerts.pem`.

If the `source` is not specified, it will be pulled from the `ca_cert_url` configuration variable available to the master. If it cannot be found, it will be downloaded from the cURL website, using an http (not https) URL. USING THE DEFAULT URL SHOULD BE AVOIDED!

`merge_files` may also be specified, which includes a string or list of strings representing a file or files to be appended to the end of the CA bundle, once it is downloaded.

CLI Example:

```
    salt-run http.update_ca_bundle merge_files=/path/to/mycert.pem
```

salt.runners.jobs

A convenience system to manage jobs, both active and already run

salt.runners.jobs.**active**(*outputter=None*, *display_progress=False*)
Return a report on all actively running jobs from a job id centric perspective

CLI Example:

```
    salt-run jobs.active
```

salt.runners.jobs.**list_job**(*jid*, *ext_source=None*, *outputter=None*)
List a specific job given by its jid

CLI Example:

```
    salt-run jobs.list_job 20130916125524463507
```

salt.runners.jobs.**list_jobs**(*ext_source=None*, *outputter=None*, *search_metadata=None*, *search_function=None*, *search_target=None*, *display_progress=False*)
List all detectable jobs and associated functions

CLI Example:

```
    salt-run jobs.list_jobs
```

salt.runners.jobs.**lookup_jid**(*jid*, *ext_source=None*, *missing=False*, *outputter=None*, *display_progress=False*)
Return the printout from a previously executed job

jid The jid to look up.

ext_source The external job cache to use. Default: *None*.

missing When set to *True*, adds the minions that did not return from the command. Default: *False*.

display_progress Displays progress events when set to *True*. Default: *False*.

New in version 2015.5.0.

CLI Example:

```
salt-run jobs.lookup_jid 20130916125524463507
salt-run jobs.lookup_jid 20130916125524463507 outputter=highstate
```

salt.runners.jobs.**print_job**(*jid*, *ext_source=None*, *outputter=None*)
Print a specific job's detail given by it's jid, including the return data.

CLI Example:

```
salt-run jobs.print_job 20130916125524463507
```

salt.runners.launchd

Manage launchd plist files

salt.runners.launchd.**write_launchd_plist**(*program*)
Write a launchd plist for managing salt-master or salt-minion

CLI Example:

```
salt-run launchd.write_launchd_plist salt-master
```

salt.runners.lxc

Control Linux Containers via Salt

depends lxc execution module

salt.runners.lxc.**cloud_init**(*names*, *host=None*, *quiet=False*, ***kwargs*)
Wrapper for using lxc.init in saltcloud compatibility mode

names Name of the containers, supports a single name or a comma delimited list of names.

host Minion to start the container on. Required.

saltcloud_mode init the container with the saltcloud opts format instead

salt.runners.lxc.**find_guest**(*name*, *quiet=False*)
Returns the host for a container.

```
salt-run lxc.find_guest name
```

salt.runners.lxc.**find_guests**(*names*)
Return a dict of hosts and named guests

salt.runners.lxc.**freeze**(*name*, *quiet=False*)
Freeze the named container

```
salt-run lxc.freeze name
```

salt.runners.lxc.**info**(*name*, *quiet=False*)
Returns information about a container.

```
salt-run lxc.info name
```

salt.runners.lxc.**init**(*names*, *host=None*, *saltcloud_mode=False*, *quiet=False*, ***kwargs*)

Initialize a new container

```
salt-run lxc.init name host=minion_id [cpuset=cgroups_cpuset] \
        [cpushare=cgroups_cpushare] [memory=cgroups_memory] \
        [template=lxc_template_name] [clone=original name] \
        [profile=lxc_profile] [network_proflile=network_profile] \
        [nic=network_profile] [nic_opts=nic_opts] \
        [start=(true|false)] [seed=(true|false)] \
        [install=(true|false)] [config=minion_config] \
        [snapshot=(true|false)]
```

names Name of the containers, supports a single name or a comma delimited list of names.

host Minion on which to initialize the container **(required)**

saltcloud_mode init the container with the saltcloud opts format instead See lxc.init_interface module documentation

cpuset cgroups cpuset.

cpushare cgroups cpu shares.

memory cgroups memory limit, in MB

Changed in version 2015.5.0: If no value is passed, no limit is set. In earlier Salt versions, not passing this value causes a 1024MB memory limit to be set, and it was necessary to pass `memory=0` to set no limit.

template Name of LXC template on which to base this container

clone Clone this container from an existing container

profile A LXC profile (defined in config or pillar).

network_profile Network profile to use for the container

New in version 2015.5.0.

nic Deprecated since version 2015.5.0: Use `network_profile` instead

nic_opts Extra options for network interfaces. E.g.:

```
{"eth0": {"mac": "aa:bb:cc:dd:ee:ff", "ipv4": "10.1.1.1", "ipv6":
"2001:db8::ff00:42:8329"}}
```

start Start the newly created container.

seed Seed the container with the minion config and autosign its key. Default: true

install If salt-minion is not already installed, install it. Default: true

config Optional config parameters. By default, the id is set to the name of the container.

salt.runners.lxc.**list_**(*host=None*, *quiet=False*)

List defined containers (running, stopped, and frozen) for the named (or all) host(s).

```
salt-run lxc.list [host=minion_id]
```

salt.runners.lxc.**purge**(*name*, *delete_key=True*, *quiet=False*)

Purge the named container and delete its minion key if present. WARNING: Destroys all data associated with the container.

```
salt-run lxc.purge name
```

salt.runners.lxc.**start**(*name*, *quiet=False*)
Start the named container.

```
salt-run lxc.start name
```

salt.runners.lxc.**stop**(*name*, *quiet=False*)
Stop the named container.

```
salt-run lxc.stop name
```

salt.runners.lxc.**unfreeze**(*name*, *quiet=False*)
Unfreeze the named container

```
salt-run lxc.unfreeze name
```

salt.runners.manage

General management functions for salt, tools like seeing what hosts are up and what hosts are down

salt.runners.manage.**bootstrap**(*version='develop'*, *script=None*, *hosts=''*, *root_user=True*)
Bootstrap minions with salt-bootstrap

version [develop] Git tag of version to install

script [https://bootstrap.saltstack.com] Script to execute

hosts Comma-separated hosts [example: hosts='host1.local,host2.local']

root_user [True] Prepend `root@` to each host.

CLI Example:

```
salt-run manage.bootstrap hosts='host1,host2'
salt-run manage.bootstrap hosts='host1,host2' version='v0.17'
salt-run manage.bootstrap hosts='host1,host2' version='v0.17' script='https://bootstrap.saltstac
salt-run manage.bootstrap hosts='ec2-user@host1,ec2-user@host2' root_user=False
```

salt.runners.manage.**bootstrap_psexec**(*hosts=''*, *master=None*, *version=None*, *arch='win32'*, *installer_url=None*, *username=None*, *password=None*)
Bootstrap Windows minions via PsExec.

hosts Comma separated list of hosts to deploy the Windows Salt minion.

master Address of the Salt master passed as an argument to the installer.

version Point release of installer to download. Defaults to the most recent.

arch Architecture of installer to download. Defaults to win32.

installer_url URL of minion installer executable. Defaults to the latest version from http://docs.saltstack.com/downloads

username Optional user name for login on remote computer.

password Password for optional username. If omitted, PsExec will prompt for one to be entered for each host.

CLI Example:

```
salt-run manage.bootstrap_psexec hosts='host1,host2'
salt-run manage.bootstrap_psexec hosts='host1,host2' version='0.17' username='DOMAIN\Administrat
salt-run manage.bootstrap_psexec hosts='host1,host2' installer_url='http://exampledomain/salt-in
```

salt.runners.manage.**down**(*removekeys=False*)

Print a list of all the down or unresponsive salt minions Optionally remove keys of down minions

CLI Example:

```
salt-run manage.down
salt-run manage.down removekeys=True
```

salt.runners.manage.**key_regen**()

This routine is used to regenerate all keys in an environment. This is invasive! ALL KEYS IN THE SALT ENVIRONMENT WILL BE REGENERATED!!

The key_regen routine sends a command out to minions to revoke the master key and remove all minion keys, it then removes all keys from the master and prompts the user to restart the master. The minions will all reconnect and keys will be placed in pending.

After the master is restarted and minion keys are in the pending directory execute a salt-key -A command to accept the regenerated minion keys.

The master *must* be restarted within 60 seconds of running this command or the minions will think there is something wrong with the keys and abort.

Only Execute this runner after upgrading minions and master to 0.15.1 or higher!

CLI Example:

```
salt-run manage.key_regen
```

salt.runners.manage.**not_present**(*subset=None*, *show_ipv4=False*)

Print a list of all minions that are NOT up according to Salt's presence detection (no commands will be sent)

subset [None] Pass in a CIDR range to filter minions by IP address.

show_ipv4 [False] Also show the IP address each minion is connecting from.

CLI Example:

```
salt-run manage.not_present
```

salt.runners.manage.**present**(*subset=None*, *show_ipv4=False*)

Print a list of all minions that are up according to Salt's presence detection (no commands will be sent)

subset [None] Pass in a CIDR range to filter minions by IP address.

show_ipv4 [False] Also show the IP address each minion is connecting from.

CLI Example:

```
salt-run manage.present
```

salt.runners.manage.**safe_accept**(*target*, *expr_form='glob'*)

Accept a minion's public key after checking the fingerprint over salt-ssh

CLI Example:

```
salt-run manage.safe_accept my_minion
salt-run manage.safe_accept minion1,minion2 expr_form=list
```

salt.runners.manage.**status**(*output=True*)

Print the status of all known salt minions

CLI Example:

```
salt-run manage.status
```

salt.runners.manage.**up**()
Print a list of all of the minions that are up

CLI Example:

```
salt-run manage.up
```

salt.runners.manage.**versions**()
Check the version of active minions

CLI Example:

```
salt-run manage.versions
```

salt.runners.mine

A runner to access data from the salt mine

salt.runners.mine.**get**(*tgt*, *fun*, *tgt_type='glob'*, *output=None*)
Gathers the data from the specified minions' mine, pass in the target, function to look up and the target type

CLI Example:

```
salt-run mine.get '*' network.interfaces
```

salt.runners.nacl

This runner helps create encrypted passwords that can be included in pillars. This is often usefull if you wish to store your pillars in source control or share your pillar data with others that you trust. I dont advise making your pillars public regardless if they are encrypted or not.

depends

- libnacl, https://github.com/saltstack/libnacl

The following configurations can be defined in the master config so your users can create encrypted passwords using the runner nacl.

cat /etc/salt/master.d/nacl.conf nacl.config:

key: None keyfile: /root/.nacl

Now with the config in the master you can use the runner nacl like:

salt-run nacl.enc `data'

salt.runners.nacl.**dec**(*data*, ***kwargs*)
Takes a key generated from *nacl.keygen* and decrypt some data.

CLI Examples:

```
salt-run nacl.dec pEXHQM6cuaF7A=
salt-run nacl.dec data='pEXHQM6cuaF7A=' keyfile=/root/.nacl
salt-run nacl.dec data='pEXHQM6cuaF7A=' key='cKEzd4kXsbeCE7/nLTIqXwnUiD1ulg4NoeeYcCFpd9k='
```

salt.runners.nacl.**enc**(*data*, ***kwargs*)
Takes a key generated from *nacl.keygen* and encrypt some data.

CLI Examples:

```
salt-run nacl.enc datatoenc
salt-run nacl.enc datatoenc keyfile=/root/.nacl
salt-run nacl.enc datatoenc key='cKEzd4kXsbeCE7/nLTIqXwnUiD1ulg4NoeeYcCFpd9k='
```

salt.runners.nacl.**keygen**(*keyfile=None*)

Use libnacl to generate a private key

CLI Examples:

```
salt-run nacl.keygen
salt-run nacl.keygen keyfile=/root/.nacl
salt-run --out=newline_values_only nacl.keygen > /root/.nacl
```

salt.runners.network

Network tools to run from the Master

salt.runners.network.**wol**(*mac*, *bcast=`255.255.255.255'*, *destport=9*)

Send a ``Magic Packet'' to wake up a Minion

CLI Example:

```
salt-run network.wol 08-00-27-13-69-77
salt-run network.wol 080027136977 255.255.255.255 7
salt-run network.wol 08:00:27:13:69:77 255.255.255.255 7
```

salt.runners.network.**wollist**(*maclist*, *bcast=`255.255.255.255'*, *destport=9*)

Send a ``Magic Packet'' to wake up a list of Minions. This list must contain one MAC hardware address per line

CLI Example:

```
salt-run network.wollist '/path/to/maclist'
salt-run network.wollist '/path/to/maclist' 255.255.255.255 7
salt-run network.wollist '/path/to/maclist' 255.255.255.255 7
```

salt.runners.pagerduty

Runner Module for Firing Events via PagerDuty

New in version 2014.1.0.

configuration This module can be used by specifying the name of a configuration profile in the master config.

For example:

```
my-pagerduty-account:
    pagerduty.api_key: F3Rbyjbve43rfFWf2214
    pagerduty.subdomain: mysubdomain
```

salt.runners.pagerduty.**create_event**(*service_key=None*, *description=None*, *details=None*, *incident_key=None*, *profile=None*)

Create an event in PagerDuty. Designed for use in states.

CLI Example:

```
salt-run pagerduty.create_event <service_key> <description> <details>          profile=my-pagerdu
```

The following parameters are required:

service_key This key can be found by using pagerduty.list_services.

description This is a short description of the event.

details This can be a more detailed description of the event.

profile This refers to the configuration profile to use to connect to the PagerDuty service.

`salt.runners.pagerduty.`**`list_escalation_policies`**(*profile=None*, *api_key=None*)
List escalation policies belonging to this account

CLI Example:

salt-run pagerduty.list_policies my-pagerduty-account salt-run pagerduty.list_escalation_policies my-pagerduty-account

`salt.runners.pagerduty.`**`list_incidents`**(*profile=None*, *api_key=None*)
List incidents belonging to this account

CLI Example:

salt-run pagerduty.list_incidents my-pagerduty-account

`salt.runners.pagerduty.`**`list_maintenance_windows`**(*profile=None*, *api_key=None*)
List maintenance windows belonging to this account

CLI Example:

salt-run pagerduty.list_windows my-pagerduty-account salt-run pagerduty.list_maintenance_windows my-pagerduty-account

`salt.runners.pagerduty.`**`list_policies`**(*profile=None*, *api_key=None*)
List escalation policies belonging to this account

CLI Example:

salt-run pagerduty.list_policies my-pagerduty-account salt-run pagerduty.list_escalation_policies my-pagerduty-account

`salt.runners.pagerduty.`**`list_schedules`**(*profile=None*, *api_key=None*)
List schedules belonging to this account

CLI Example:

salt-run pagerduty.list_schedules my-pagerduty-account

`salt.runners.pagerduty.`**`list_services`**(*profile=None*, *api_key=None*)
List services belonging to this account

CLI Example:

salt-run pagerduty.list_services my-pagerduty-account

`salt.runners.pagerduty.`**`list_users`**(*profile=None*, *api_key=None*)
List users belonging to this account

CLI Example:

salt-run pagerduty.list_users my-pagerduty-account

salt.runners.pagerduty.**list_windows**(*profile=None*, *api_key=None*)
List maintenance windows belonging to this account

CLI Example:

salt-run pagerduty.list_windows my-pagerduty-account salt-run pagerduty.list_maintenance_windows my-pagerduty-account

salt.runners.pillar

Functions to interact with the pillar compiler on the master

salt.runners.pillar.**show_pillar**(*minion='*'*, ***kwargs*)
Returns the compiled pillar either of a specific minion or just the global available pillars. I assume that no minion is using the id `*`.

CLI Example:

shows minion specific pillar:

```
salt-run pillar.show_pillar 'www.example.com'
```

shows global pillar:

```
salt-run pillar.show_pillar
```

shows global pillar for `dev' pillar environment:

```
salt-run pillar.show_pillar 'saltenv=dev'
```

API Example:

```
import salt.config
import salt.runner
opts = salt.config.master_config('/etc/salt/master')
runner = salt.runner.RunnerClient(opts)
pillar = runner.cmd('pillar.show_pillar', [])
print pillar¬
```

salt.runners.pillar.**show_top**(*minion=None*, *saltenv='base'*)
Returns the compiled top data for pillar for a specific minion. If no minion is specified, we use the first minion we find.

CLI Example:

```
salt-run pillar.show_top
```

salt.runners.queue

General management and processing of queues.

This runner facilitates interacting with various queue backends such as the included sqlite3 queue or the planned AWS SQS and Redis queues

The queue functions such as *insert*, *delete*, and *pop* can be used for typical management of the queue.

The *process_queue* function pops the requested number of items from the queue and creates a Salt Event that can then be processed by a Reactor. The *process_queue* function can be called manually, or can be configured to run on a schedule with the Salt Scheduler or regular system cron. It is also possible to use the peer system to allow a minion to call the runner.

This runner, as well as the Queues system, is not api stable at this time.

There are many things that could potentially be done with queues within Salt. For the time being the focus will be on queueing infrastructure actions on specific minions. The queues generally will be populated with minion IDs. When the *process_queue* runner function is called events are created on the Salt Event bus that indicate the queue and a list of one or more minion IDs. The reactor is set up to match on event tags for a specific queue and then take infrastructure actions on those minion IDs. These actions might be to delete the minion's key from the master, use salt-cloud to destroy the vm, or some other custom action.

`salt.runners.queue.`**`delete`**(*queue*, *items*, *backend='sqlite'*)
Delete an item or items from a queue

CLI Example:

```
salt-run queue.delete myqueue myitem
salt-run queue.delete myqueue myitem backend=sqlite
salt-run queue.delete myqueue "['item1', 'item2', 'item3']"
```

`salt.runners.queue.`**`insert`**(*queue*, *items*, *backend='sqlite'*)
Add an item or items to a queue

CLI Example:

```
salt-run queue.insert myqueue myitem
salt-run queue.insert myqueue "['item1', 'item2', 'item3']"
salt-run queue.insert myqueue myitem backend=sqlite
salt-run queue.insert myqueue "['item1', 'item2', 'item3']" backend=sqlite
```

`salt.runners.queue.`**`list_items`**(*queue*, *backend='sqlite'*)
List contents of a queue

CLI Example:

```
salt-run queue.list_items myqueue
salt-run queue.list_items myqueue backend=sqlite
```

`salt.runners.queue.`**`list_length`**(*queue*, *backend='sqlite'*)
Provide the number of items in a queue

CLI Example:

```
salt-run queue.list_length myqueue
salt-run queue.list_length myqueue backend=sqlite
```

`salt.runners.queue.`**`list_queues`**(*backend='sqlite'*)
Return a list of Salt Queues on the backend

CLI Example:

```
salt-run queue.list_queues
salt-run queue.list_queues backend=sqlite
```

`salt.runners.queue.`**`pop`**(*queue*, *quantity=1*, *backend='sqlite'*)
Pop one or more or all items from a queue

CLI Example:

```
salt-run queue.pop myqueue
salt-run queue.pop myqueue 6
salt-run queue.pop myqueue all
salt-run queue.pop myqueue 6 backend=sqlite
salt-run queue.pop myqueue all backend=sqlite
```

salt.runners.queue.**process_queue**(*queue*, *quantity=1*, *backend='sqlite'*)
Pop items off a queue and create an event on the Salt event bus to be processed by a Reactor.

CLI Example:

```
salt-run queue.process_queue myqueue
salt-run queue.process_queue myqueue 6
salt-run queue.process_queue myqueue all backend=sqlite
```

salt.runners.sdb

Runner for setting and querying data via the sdb API on the master

salt.runners.sdb.**get**(*uri*)
Get a value from a db, using a uri in the form of sdb://<profile>/<key>. If the uri provided does not start with sdb://, then it will be returned as-is.

CLI Example:

```
salt '*' sdb.get sdb://mymemcached/foo
```

salt.runners.sdb.**set_**(*uri*, *value*)
Set a value in a db, using a uri in the form of `sdb://<profile>/<key>`. If the uri provided does not start with `sdb://` or the value is not successfully set, return `False`.

CLI Example:

```
salt '*' sdb.set sdb://mymemcached/foo bar
```

salt.runners.search

Runner frontend to search system

salt.runners.search.**query**(*term*)
Query the search system

CLI Example:

```
salt-run search.query foo
```

salt.runners.state

Execute overstate functions

salt.runners.state.**event**(*tagmatch='*'*, *count=-1*, *quiet=False*, *sock_dir=None*, *pretty=False*)
Watch Salt's event bus and block until the given tag is matched

New in version 2014.7.0.

This is useful for utilizing Salt's event bus from shell scripts or for taking simple actions directly from the CLI.

Enable debug logging to see ignored events.

Parameters

- **tagmatch** -- the event is written to stdout for each tag that matches this pattern; uses the same matching semantics as Salt's Reactor.

- **count** -- this number is decremented for each event that matches the `tagmatch` parameter; pass `-1` to listen forever.
- **quiet** -- do not print to stdout; just block
- **sock_dir** -- path to the Salt master's event socket file.
- **pretty** -- Output the JSON all on a single line if `False` (useful for shell tools); pretty-print the JSON output if `True`.

CLI Examples:

```
# Reboot a minion and run highstate when it comes back online
salt 'jerry' system.reboot && \\
    salt-run state.event 'salt/minion/jerry/start' count=1 quiet=True && \\
    salt 'jerry' state.highstate

# Reboot multiple minions and run highstate when all are back online
salt -L 'kevin,stewart,dave' system.reboot && \\
    salt-run state.event 'salt/minion/*/start' count=3 quiet=True && \\
    salt -L 'kevin,stewart,dave' state.highstate

# Watch the event bus forever in a shell while-loop.
salt-run state.event | while read -r tag data; do
    echo $tag
    echo $data | jq -colour-output .
done
```

The following example monitors Salt's event bus in a background process watching for returns for a given job. Requires a POSIX environment and jq <http://stedolan.github.io/jq/>.

```
#!/bin/sh
# Usage: ./eventlisten.sh '*' test.sleep 10

# Mimic fnmatch from the Python stdlib.
fnmatch() { case "$2" in $1) return 0 ;; *) return 1 ;; esac ; }
count() { printf '%s\n' "$#" ; }

listen() {
    events='events'
    mkfifo $events
    exec 3<>$events     # Hold the fd open.

    # Start listening to events before starting the command to avoid race
    # conditions.
    salt-run state.event count=-1 >&3 &
    events_pid=$!

    (
        timeout=$(( 60 * 60 ))
        sleep $timeout
        kill -s USR2 $$
    ) &
    timeout_pid=$!

    # Give the runner a few to connect to the event bus.
    printf 'Subscribing to the Salt event bus...\n'
    sleep 4

    trap '
```

```
            excode=$?; trap - EXIT;
            exec 3>&-
            kill '"${timeout_pid}"'
            kill '"${events_pid}"'
            rm '"${events}"'
            exit
            echo $excode
        ' INT TERM EXIT

        trap '
            printf '\''Timeout reached; exiting.\n'\''
            exit 4
        ' USR2

        # Run the command and get the JID.
        jid=$(salt --async "$@")
        jid="${jid#*: }"    # Remove leading text up to the colon.

        # Create the event tags to listen for.
        start_tag="salt/job/${jid}/new"
        ret_tag="salt/job/${jid}/ret/*"

        # ``read`` will block when no events are going through the bus.
        printf 'Waiting for tag %s\n' "$ret_tag"
        while read -r tag data; do
            if fnmatch "$start_tag" "$tag"; then
                minions=$(printf '%s\n' "${data}" | jq -r '.["minions"][]')
                num_minions=$(count $minions)
                printf 'Waiting for %s minions.\n' "$num_minions"
                continue
            fi

            if fnmatch "$ret_tag" "$tag"; then
                mid="${tag##*/}"
                printf 'Got return for %s.\n' "$mid"
                printf 'Pretty-printing event: %s\n' "$tag"
                printf '%s\n' "$data" | jq .

                minions="$(printf '%s\n' "$minions" | sed -e '/'"$mid"'/d')"
                num_minions=$(count $minions)
                if [ $((num_minions)) -eq 0 ]; then
                    printf 'All minions returned.\n'
                    break
                else
                    printf 'Remaining minions: %s\n' "$num_minions"
                fi
            else
                printf 'Skipping tag: %s\n' "$tag"
                continue
            fi
        done <&3
    }

    listen "$@"
```

`salt.runners.state.`**`orchestrate`**(*mods*, *saltenv='base'*, *test=None*, *exclude=None*, *pillar=None*)

New in version 0.17.0.

Execute a state run from the master, used as a powerful orchestration system.

See also:

More Orchestrate documentation

- *Full Orchestrate Tutorial*
- `Docs for the master-side state module`

CLI Examples:

```
salt-run state.orchestrate webserver
salt-run state.orchestrate webserver saltenv=dev test=True
```

Changed in version 2014.1.1: Runner renamed from `state.sls` to `state.orchestrate`

Changed in version 2014.7.0: Runner uses the pillar variable

`salt.runners.state.`**`orchestrate_high`**(*data*, *test=None*, *queue=False*, *pillar=None*, ***kwargs*)

Execute a single state orchestration routine

New in version 2015.5.0.

CLI Example:

```
salt-run state.orchestrate_high '{
    stage_one:
        {salt.state: [{tgt: "db*"}, {sls: postgres_setup}]},
    stage_two:
        {salt.state: [{tgt: "web*"}, {sls: apache_setup}, {
            require: [{salt: stage_one}],
        }]},
    }'
```

`salt.runners.state.`**`orchestrate_single`**(*fun*, *name*, *test=None*, *queue=False*, *pillar=None*, ***kwargs*)

Execute a single state orchestration routine

New in version 2015.5.0.

CLI Example:

```
salt-run state.orchestrate_single fun=salt.wheel name=key.list_all
```

`salt.runners.state.`**`over`**(*saltenv='base'*, *os_fn=None*)

New in version 0.11.0.

> **Warning:** `state.over` is deprecated in favor of `state.orchestrate`, and will be removed in the Salt feature release codenamed Boron. (Three feature releases after the 2014.7.0 release, which is codenamed Helium)

Execute an overstate sequence to orchestrate the executing of states over a group of systems

CLI Examples:

```
salt-run state.over base /path/to/myoverstate.sls
```

`salt.runners.state.`**`show_stages`**(*saltenv='base'*, *os_fn=None*)

New in version 0.11.0.

Display the OverState's stage data

CLI Examples:

```
salt-run state.show_stages
salt-run state.show_stages saltenv=dev /root/overstate.sls
```

salt.runners.survey

A general map/reduce style salt runner for aggregating results returned by several different minions.

New in version 2014.7.0.

Aggregated results are sorted by the size of the minion pools which returned matching results.

Useful for playing the game: " some of these things are not like the others... " when identifying discrepancies in a large infrastructure managed by salt.

`salt.runners.survey.diff`(**args*, ***kwargs*)

Return the DIFFERENCE of the result sets returned by each matching minion pool

New in version 2014.7.0.

These pools are determined from the aggregated and sorted results of a salt command. This command displays the ``diffs'' as a series of 2-way differences-- namely the difference between the FIRST displayed minion pool (according to sort order) and EACH SUBSEQUENT minion pool result set. Differences are displayed according to the Python ``difflib.unified_diff()'' as in the case of the salt execution module ``file.get_diff''.

This command is submitted via a salt runner using the general form:

salt-run survey.diff [survey_sort=up/down] <target> <salt-execution-module> <salt-execution-module parameters>

Optionally accept a ``survey_sort='' parameter. Default: ``survey_sort=down''

CLI Example #1: (Example to display the ``differences of files'')

```
salt-run survey.diff survey_sort=up "*" cp.get_file_str file:///etc/hosts
```

`salt.runners.survey.hash`(**args*, ***kwargs*)

Return the MATCHING minion pools from the aggregated and sorted results of a salt command

New in version 2014.7.0.

This command is submitted via a salt runner using the general form:

salt-run survey.hash [survey_sort=up/down] <target> <salt-execution-module> <salt-execution-module parameters>

Optionally accept a ``survey_sort='' parameter. Default: ``survey_sort=down''

CLI Example #1: (functionally equivalent to ``salt-run manage.up'')

```
salt-run survey.hash "*" test.ping
```

CLI Example #2: (find an ``outlier'' minion config file)

```
salt-run survey.hash "*" file.get_hash /etc/salt/minion survey_sort=up
```

salt.runners.test

This runner is used only for test purposes and servers no production purpose

salt.runners.test.**arg**(**args*, ***kwargs*)
Output the given args and kwargs

Kwargs will be filtered for `private' keynames.

salt.runners.test.**raw_arg**(**args*, ***kwargs*)
Output the given args and kwargs

salt.runners.test.**sleep**(*s_time=10*)
Sleep t seconds, then return True

salt.runners.test.**stdout_print**()
Print `foo' and return `bar'

salt.runners.test.**stream**()
Return True

salt.runners.thin

The thin runner is used to manage the salt thin systems.

Salt Thin is a transport-less version of Salt that can be used to run routines in a standalone way. This runner has tools which generate the standalone salt system for easy consumption.

salt.runners.thin.**generate**(*extra_mods=''*, *overwrite=False*, *so_mods=''*)
Generate the salt-thin tarball and print the location of the tarball Optional additional mods to include (e.g. mako) can be supplied as a comma delimited string. Permits forcing an overwrite of the output file as well.

CLI Example:

```
salt-run thin.generate
salt-run thin.generate mako
salt-run thin.generate mako,wempy 1
salt-run thin.generate overwrite=1
```

salt.runners.virt

Control virtual machines via Salt

salt.runners.virt.**force_off**(*name*)
Force power down the named virtual machine

salt.runners.virt.**hyper_info**(*hyper=None*)
Return information about the hypervisors connected to this master

salt.runners.virt.**init**(*name*, *cpu*, *mem*, *image*, *hyper=None*, *seed=True*, *nic='default'*, *install=True*)
This routine is used to create a new virtual machine. This routines takes a number of options to determine what the newly created virtual machine will look like.

name The mandatory name of the new virtual machine. The name option is also the minion id, all minions must have an id.

cpu The number of cpus to allocate to this new virtual machine.

mem The amount of memory to allocate tot his virtual machine. The number is interpreted in megabytes.

image The network location of the virtual machine image, commonly a location on the salt fileserver, but http, https and ftp can also be used.

hyper The hypervisor to use for the new virtual machine, if this is omitted Salt will automatically detect what hypervisor to use.

seed Set to False to prevent Salt from seeding the new virtual machine.

nic The nic profile to use, defaults to the ``default'' nic profile which assumes a single network interface per vm associated with the ``br0'' bridge on the master.

install Set to False to prevent Salt from installing a minion on the new vm before it spins up.

salt.runners.virt.**list**(*hyper=None*, *quiet=False*)
List the virtual machines on each hyper, this is a simplified query, showing only the virtual machine names belonging to each hypervisor. A single hypervisor can be passed in to specify an individual hypervisor to list.

salt.runners.virt.**migrate**(*name*, *target=''*)
Migrate a vm from one hypervisor to another. This routine will just start the migration and display information on how to look up the progress.

salt.runners.virt.**next_hyper**()
Return the hypervisor to use for the next autodeployed vm. This queries the available hypervisors and executes some math the determine the most ``available'' next hypervisor.

salt.runners.virt.**pause**(*name*)
Pause the named vm

salt.runners.virt.**purge**(*name*, *delete_key=True*)
Destroy the named vm

salt.runners.virt.**query**(*hyper=None*, *quiet=False*)
Query the virtual machines. When called without options all hypervisors are detected and a full query is returned. A single hypervisor can be passed in to specify an individual hypervisor to query.

salt.runners.virt.**reset**(*name*)
Force power down and restart an existing vm

salt.runners.virt.**resume**(*name*)
Resume a paused vm

salt.runners.virt.**start**(*name*)
Start a named virtual machine

salt.runners.virt.**vm_info**(*name*, *quiet=False*)
Return the information on the named vm

salt.runners.winrepo

Runner to manage Windows software repo

salt.runners.winrepo.**genrepo**()
Generate win_repo_cachefile based on sls files in the win_repo

CLI Example:

```
salt-run winrepo.genrepo
```

salt.runners.winrepo.**update_git_repos**()
Checkout git repos containing Windows Software Package Definitions

CLI Example:

```
salt-run winrepo.update_git_repos
```

22.25.2 Writing Salt Runners

A Salt runner is written in a similar manner to a Salt execution module. Both are Python modules which contain functions and each public function is a runner which may be executed via the *salt-run* command.

For example, if a Python module named `test.py` is created in the runners directory and contains a function called `foo`, the `test` runner could be invoked with the following command:

```
# salt-run test.foo
```

To add custom runners, put them in a directory and add it to `runner_dirs` in the master configuration file.

22.25.3 Examples

Examples of runners can be found in the Salt distribution:

https://github.com/saltstack/salt/blob/develop/salt/runners

A simple runner that returns a well-formatted list of the minions that are responding to Salt calls could look like this:

```
# Import salt modules
import salt.client

def up():
    '''
    Print a list of all of the minions that are up
    '''
    client = salt.client.LocalClient(__opts__['conf_file'])
    minions = client.cmd('*', 'test.ping', timeout=1)
    for minion in sorted(minions):
        print minion
```

22.26 State Enforcement

Salt offers an optional interface to manage the configuration or ``state'' of the Salt minions. This interface is a fully capable mechanism used to enforce the state of systems from a central manager.

22.26.1 Mod Aggregate State Runtime Modifications

New in version 2014.7.0.

The mod_aggregate system was added in the 2014.7.0 release of Salt and allows for runtime modification of the executing state data. Simply put, it allows for the data used by Salt's state system to be changed on the fly at runtime, kind of like a configuration management JIT compiler or a runtime import system. All in all, it makes Salt much more dynamic.

How it Works

The best example is the `pkg` state. One of the major requests in Salt has long been adding the ability to install all packages defined at the same time. The mod_aggregate system makes this a reality. While executing Salt's state system, when a `pkg` state is reached the `mod_aggregate` function in the state module is called. For `pkg` this

function scans all of the other states that are slated to run, and picks up the references to `name` and `pkgs`, then adds them to `pkgs` in the first state. The result is a single call to yum, apt-get, pacman, etc as part of the first package install.

How to Use it

Note: Since this option changes the basic behavior of the state runtime, after it is enabled states should be executed using *test=True* to ensure that the desired behavior is preserved.

In config files

The first way to enable aggregation is with a configuration option in either the master or minion configuration files. Salt will invoke `mod_aggregate` the first time it encounters a state module that has aggregate support.

If this option is set in the master config it will apply to all state runs on all minions, if set in the minion config it will only apply to said minion.

Enable for all states:

```
state_aggregate: True
```

Enable for only specific state modules:

```
state_aggregate:
  - pkg
```

In states

The second way to enable aggregation is with the state-level `aggregate` keyword. In this configuration, Salt will invoke the `mod_aggregate` function the first time it encounters this keyword. Any additional occurances of the keyword will be ignored as the aggregation has already taken place.

The following example will trigger `mod_aggregate` when the `lamp_stack` state is processed resulting in a single call to the underlying package manager.

```
lamp_stack:
  pkg.installed:
    - pkgs:
      - php
      - mysql-client
    - aggregate: True

memcached:
  pkg.installed:
    - name: memcached
```

Adding mod_aggregate to a State Module

Adding a mod_aggregate routine to an existing state module only requires adding an additional function to the state module called mod_aggregate.

The mod_aggregate function just needs to accept three parameters and return the low data to use. Since mod_aggregate is working on the state runtime level it does need to manipulate *low data*.

The three parameters are *low*, *chunks*, and *running*. The *low* option is the low data for the state execution which is about to be called. The *chunks* is the list of all of the low data dictionaries which are being executed by the runtime and the *running* dictionary is the return data from all of the state executions which have already be executed.

This example, simplified from the pkg state, shows how to create mod_aggregate functions:

```
def mod_aggregate(low, chunks, running):
    '''
    The mod_aggregate function which looks up all packages in the available
    low chunks and merges them into a single pkgs ref in the present low data
    '''
    pkgs = []
    # What functions should we aggregate?
    agg_enabled = [
            'installed',
            'latest',
            'removed',
            'purged',
            ]
    # The `low` data is just a dict with the state, function (fun) and
    # arguments passed in from the sls
    if low.get('fun') not in agg_enabled:
        return low
    # Now look into what other things are set to execute
    for chunk in chunks:
        # The state runtime uses "tags" to track completed jobs, it may
        # look familiar with the _|-
        tag = salt.utils.gen_state_tag(chunk)
        if tag in running:
            # Already ran the pkg state, skip aggregation
            continue
        if chunk.get('state') == 'pkg':
            if '__agg__' in chunk:
                continue
            # Check for the same function
            if chunk.get('fun') != low.get('fun'):
                continue
            # Pull out the pkg names!
            if 'pkgs' in chunk:
                pkgs.extend(chunk['pkgs'])
                chunk['__agg__'] = True
            elif 'name' in chunk:
                pkgs.append(chunk['name'])
                chunk['__agg__'] = True
    if pkgs:
        if 'pkgs' in low:
            low['pkgs'].extend(pkgs)
        else:
            low['pkgs'] = pkgs
    # The low has been modified and needs to be returned to the state
```

```
    # runtime for execution
    return low
```

22.26.2 Altering States

Note: This documentation has been moved *here*.

22.26.3 File State Backups

In 0.10.2 a new feature was added for backing up files that are replaced by the file.managed and file.recurse states. The new feature is called the backup mode. Setting the backup mode is easy, but it can be set in a number of places.

The backup_mode can be set in the minion config file:

```
backup_mode: minion
```

Or it can be set for each file:

```
/etc/ssh/sshd_config:
  file.managed:
    - source: salt://ssh/sshd_config
    - backup: minion
```

Backed-up Files

The files will be saved in the minion cachedir under the directory named `file_backup`. The files will be in the location relative to where they were under the root filesystem and be appended with a timestamp. This should make them easy to browse.

Interacting with Backups

Starting with version 0.17.0, it will be possible to list, restore, and delete previously-created backups.

Listing

The backups for a given file can be listed using `file.list_backups`:

```
# salt foo.bar.com file.list_backups /tmp/foo.txt
foo.bar.com:
    ----------
    0:
        ----------
        Backup Time:
            Sat Jul 27 2013 17:48:41.738027
        Location:
            /var/cache/salt/minion/file_backup/tmp/foo.txt_Sat_Jul_27_17:48:41_738027_2013
        Size:
            13
```

```
    1:
        ----------
        Backup Time:
            Sat Jul 27 2013 17:48:28.369804
        Location:
            /var/cache/salt/minion/file_backup/tmp/foo.txt_Sat_Jul_27_17:48:28_369804_2013
        Size:
            35
```

Restoring

Restoring is easy using `file.restore_backup`, just pass the path and the numeric id found with `file.list_backups`:

```
# salt foo.bar.com file.restore_backup /tmp/foo.txt 1
foo.bar.com:
    ----------
    comment:
        Successfully restored /var/cache/salt/minion/file_backup/tmp/foo.txt_Sat_Jul_27_17:48:28_3698
    result:
        True
```

The existing file will be backed up, just in case, as can be seen if `file.list_backups` is run again:

```
# salt foo.bar.com file.list_backups /tmp/foo.txt
foo.bar.com:
    ----------
    0:
        ----------
        Backup Time:
            Sat Jul 27 2013 18:00:19.822550
        Location:
            /var/cache/salt/minion/file_backup/tmp/foo.txt_Sat_Jul_27_18:00:19_822550_2013
        Size:
            53
    1:
        ----------
        Backup Time:
            Sat Jul 27 2013 17:48:41.738027
        Location:
            /var/cache/salt/minion/file_backup/tmp/foo.txt_Sat_Jul_27_17:48:41_738027_2013
        Size:
            13
    2:
        ----------
        Backup Time:
            Sat Jul 27 2013 17:48:28.369804
        Location:
            /var/cache/salt/minion/file_backup/tmp/foo.txt_Sat_Jul_27_17:48:28_369804_2013
        Size:
            35
```

Note: Since no state is being run, restoring a file will not trigger any watches for the file. So, if you are restoring a config file for a service, it will likely still be necessary to run a `service.restart`.

Deleting

Deleting backups can be done using `file.delete_backup`:

```
# salt foo.bar.com file.delete_backup /tmp/foo.txt 0
foo.bar.com:
    ----------
    comment:
        Successfully removed /var/cache/salt/minion/file_backup/tmp/foo.txt_Sat_Jul_27_18:00:19_8225
    result:
        True
```

22.26.4 Understanding State Compiler Ordering

Note: This tutorial is an intermediate level tutorial. Some basic understanding of the state system and writing Salt Formulas is assumed.

Salt's state system is built to deliver all of the power of configuration management systems without sacrificing simplicity. This tutorial is made to help users understand in detail just how the order is defined for state executions in Salt.

This tutorial is written to represent the behavior of Salt as of version 0.17.0.

Compiler Basics

To understand ordering in depth some very basic knowledge about the state compiler is very helpful. No need to worry though, this is very high level!

High Data and Low Data

When defining Salt Formulas in YAML the data that is being represented is referred to by the compiler as High Data. When the data is initially loaded into the compiler it is a single large python dictionary, this dictionary can be viewed raw by running:

```
salt '*' state.show_highstate
```

This ``High Data'' structure is then compiled down to ``Low Data''. The Low Data is what is matched up to create individual executions in Salt's configuration management system. The low data is an ordered list of single state calls to execute. Once the low data is compiled the evaluation order can be seen.

The low data can be viewed by running:

```
salt '*' state.show_lowstate
```

Note: The state execution module contains MANY functions for evaluating the state system and is well worth a read! These routines can be very useful when debugging states or to help deepen one's understanding of Salt's state system.

As an example, a state written thusly:

```
apache:
  pkg.installed:
    - name: httpd
  service.running:
    - name: httpd
    - watch:
      - file: apache_conf
      - pkg: apache

apache_conf:
  file.managed:
    - name: /etc/httpd/conf.d/httpd.conf
    - source: salt://apache/httpd.conf
```

Will have High Data which looks like this represented in json:

```
{
    "apache": {
        "pkg": [
            {
                "name": "httpd"
            },
            "installed",
            {
                "order": 10000
            }
        ],
        "service": [
            {
                "name": "httpd"
            },
            {
                "watch": [
                    {
                        "file": "apache_conf"
                    },
                    {
                        "pkg": "apache"
                    }
                ]
            },
            "running",
            {
                "order": 10001
            }
        ],
        "__sls__": "blah",
        "__env__": "base"
    },
    "apache_conf": {
        "file": [
            {
                "name": "/etc/httpd/conf.d/httpd.conf"
            },
            {
                "source": "salt://apache/httpd.conf"
            },
```

```
            "managed",
            {
                "order": 10002
            }
        ],
        "__sls__": "blah",
        "__env__": "base"
    }
}
```

The subsequent Low Data will look like this:

```
[
    {
        "name": "httpd",
        "state": "pkg",
        "__id__": "apache",
        "fun": "installed",
        "__env__": "base",
        "__sls__": "blah",
        "order": 10000
    },
    {
        "name": "httpd",
        "watch": [
            {
                "file": "apache_conf"
            },
            {
                "pkg": "apache"
            }
        ],
        "state": "service",
        "__id__": "apache",
        "fun": "running",
        "__env__": "base",
        "__sls__": "blah",
        "order": 10001
    },
    {
        "name": "/etc/httpd/conf.d/httpd.conf",
        "source": "salt://apache/httpd.conf",
        "state": "file",
        "__id__": "apache_conf",
        "fun": "managed",
        "__env__": "base",
        "__sls__": "blah",
        "order": 10002
    }
]
```

This tutorial discusses the Low Data evaluation and the state runtime.

Ordering Layers

Salt defines 2 order interfaces which are evaluated in the state runtime and defines these orders in a number of passes.

Definition Order

Note: The Definition Order system can be disabled by turning the option *state_auto_order* to *False* in the master configuration file.

The top level of ordering is the *Definition Order*. The *Definition Order* is the order in which states are defined in salt formulas. This is very straightforward on basic states which do not contain `include` statements or a `top` file, as the states are just ordered from the top of the file, but the include system starts to bring in some simple rules for how the *Definition Order* is defined.

Looking back at the ``Low Data'' and ``High Data'' shown above, the order key has been transparently added to the data to enable the *Definition Order*.

The Include Statement Basically, if there is an include statement in a formula, then the formulas which are included will be run BEFORE the contents of the formula which is including them. Also, the include statement is a list, so they will be loaded in the order in which they are included.

In the following case:

`foo.sls`

```
include:
  - bar
  - baz
```

`bar.sls`

```
include:
  - quo
```

`baz.sls`

```
include:
  - qux
```

In the above case if *state.sls foo* were called then the formulas will be loaded in the following order:

1. quo
2. bar
3. qux
4. baz
5. foo

The *order* Flag

The *Definition Order* happens transparently in the background, but the ordering can be explicitly overridden using the *order* flag in states:

```
apache:
  pkg.installed:
    - name: httpd
    - order: 1
```

This order flag will over ride the definition order, this makes it very simple to create states that are always executed first, last or in specific stages, a great example is defining a number of package repositories that need to be set up before anything else, or final checks that need to be run at the end of a state run by using *order: last* or *order: -1*.

When the order flag is explicitly set the *Definition Order* system will omit setting an order for that state and directly use the order flag defined.

Lexicographical Fall-back

Salt states were written to ALWAYS execute in the same order. Before the introduction of *Definition Order* in version 0.17.0 everything was ordered lexicographically according to the name of the state, then function then id.

This is the way Salt has always ensured that states always run in the same order regardless of where they are deployed, the addition of the *Definition Order* method mealy makes this finite ordering easier to follow.

The lexicographical ordering is still applied but it only has any effect when two order statements collide. This means that if multiple states are assigned the same order number that they will fall back to lexicographical ordering to ensure that every execution still happens in a finite order.

Note: If running with *state_auto_order: False* the *order* key is not set automatically, since the Lexicographical order can be derived from other keys.

Requisite Ordering

Salt states are fully declarative, in that they are written to declare the state in which a system should be. This means that components can require that other components have been set up successfully. Unlike the other ordering systems, the *Requisite* system in Salt is evaluated at runtime.

The requisite system is also built to ensure that the ordering of execution never changes, but is always the same for a given set of states. This is accomplished by using a runtime that processes states in a completely predictable order instead of using an event loop based system like other declarative configuration management systems.

Runtime Requisite Evaluation

The requisite system is evaluated as the components are found, and the requisites are always evaluated in the same order. This explanation will be followed by an example, as the raw explanation may be a little dizzying at first as it creates a linear dependency evaluation sequence.

The ``Low Data'' is an ordered list or dictionaries, the state runtime evaluates each dictionary in the order in which they are arranged in the list. When evaluating a single dictionary it is checked for requisites, requisites are evaluated in order, *require* then *watch* then *prereq*.

Note: If using requisite in statements like require_in and watch_in these will be compiled down to require and

watch statements before runtime evaluation.

Each requisite contains an ordered list of requisites, these requisites are looked up in the list of dictionaries and then executed. Once all requisites have been evaluated and executed then the requiring state can safely be run (or not run if requisites have not been met).

This means that the requisites are always evaluated in the same order, again ensuring one of the core design principals of Salt's State system to ensure that execution is always finite is intact.

Simple Runtime Evaluation Example

Given the above ``Low Data'' the states will be evaluated in the following order:

1. The pkg.installed is executed ensuring that the apache package is installed, it contains no requisites and is therefore the first defined state to execute.
2. The service.running state is evaluated but NOT executed, a watch requisite is found, therefore they are read in order, the runtime first checks for the file, sees that it has not been executed and calls for the file state to be evaluated.
3. The file state is evaluated AND executed, since it, like the pkg state does not contain any requisites.
4. The evaluation of the service state continues, it next checks the pkg requisite and sees that it is met, with all requisites met the service state is now executed.

Best Practice

The best practice in Salt is to choose a method and stick with it, official states are written using requisites for all associations since requisites create clean, traceable dependency trails and make for the most portable formulas. To accomplish something similar to how classical imperative systems function all requisites can be omitted and the `failhard` option then set to *True* in the master configuration, this will stop all state runs at the first instance of a failure.

In the end, using requisites creates very tight and fine grained states, not using requisites makes full sequence runs and while slightly easier to write, and gives much less control over the executions.

22.26.5 Extending External SLS Data

Sometimes a state defined in one SLS file will need to be modified from a separate SLS file. A good example of this is when an argument needs to be overwritten or when a service needs to watch an additional state.

The Extend Declaration

The standard way to extend is via the extend declaration. The extend declaration is a top level declaration like `include` and encapsulates ID declaration data included from other SLS files. A standard extend looks like this:

```
include:
  - http
  - ssh

extend:
  apache:
    file:
```

```
      - name: /etc/httpd/conf/httpd.conf
      - source: salt://http/httpd2.conf
  ssh-server:
    service:
      - watch:
        - file: /etc/ssh/banner

/etc/ssh/banner:
  file.managed:
    - source: salt://ssh/banner
```

A few critical things happened here, first off the SLS files that are going to be extended are included, then the extend dec is defined. Under the extend dec 2 IDs are extended, the apache ID's file state is overwritten with a new name and source. Than the ssh server is extended to watch the banner file in addition to anything it is already watching.

Extend is a Top Level Declaration

This means that `extend` can only be called once in an sls, if if is used twice then only one of the extend blocks will be read. So this is WRONG:

```
include:
  - http
  - ssh

extend:
  apache:
    file:
      - name: /etc/httpd/conf/httpd.conf
      - source: salt://http/httpd2.conf
# Second extend will overwrite the first!! Only make one
extend:
  ssh-server:
    service:
      - watch:
        - file: /etc/ssh/banner
```

The Requisite ``in'' Statement

Since one of the most common things to do when extending another SLS is to add states for a service to watch, or anything for a watcher to watch, the requisite in statement was added to 0.9.8 to make extending the watch and require lists easier. The ssh-server extend statement above could be more cleanly defined like so:

```
include:
  - ssh

/etc/ssh/banner:
  file.managed:
    - source: salt://ssh/banner
    - watch_in:
      - service: ssh-server
```

Rules to Extend By

There are a few rules to remember when extending states:

1. Always include the SLS being extended with an include declaration
2. Requisites (watch and require) are appended to, everything else is overwritten
3. extend is a top level declaration, like an ID declaration, cannot be declared twice in a single SLS
4. Many IDs can be extended under the extend declaration

22.26.6 Failhard Global Option

Normally, when a state fails Salt continues to execute the remainder of the defined states and will only refuse to execute states that require the failed state.

But the situation may exist, where you would want all state execution to stop if a single state execution fails. The capability to do this is called `failing hard`.

State Level Failhard

A single state can have a failhard set, this means that if this individual state fails that all state execution will immediately stop. This is a great thing to do if there is a state that sets up a critical config file and setting a require for each state that reads the config would be cumbersome. A good example of this would be setting up a package manager early on:

```
/etc/yum.repos.d/company.repo:
  file.managed:
    - source: salt://company/yumrepo.conf
    - user: root
    - group: root
    - mode: 644
    - order: 1
    - failhard: True
```

In this situation, the yum repo is going to be configured before other states, and if it fails to lay down the config file, than no other states will be executed.

Global Failhard

It may be desired to have failhard be applied to every state that is executed, if this is the case, then failhard can be set in the master configuration file. Setting failhard in the master configuration file will result in failing hard when any minion gathering states from the master have a state fail.

This is NOT the default behavior, normally Salt will only fail states that require a failed state.

Using the global failhard is generally not recommended, since it can result in states not being executed or even checked. It can also be confusing to see states failhard if an admin is not actively aware that the failhard has been set.

To use the global failhard set failhard: True in the master configuration file.

22.26.7 Global State Arguments

Note: This documentation has been moved *here*.

22.26.8 Highstate data structure definitions

The Salt State Tree

A state tree is a collection of SLS files that live under the directory specified in `file_roots`. A state tree can be organized into `SLS modules`.

Top file

The main state file that instructs minions what environment and modules to use during state execution.

Configurable via `state_top`.

See also:

A detailed description of the top file

Include declaration

Defines a list of *Module reference* strings to include in this SLS.

Occurs only in the top level of the highstate structure.

Example:

```
include:
  - edit.vim
  - http.server
```

Module reference

The name of a SLS module defined by a separate SLS file and residing on the Salt Master. A module named `edit.vim` is a reference to the SLS file `salt://edit/vim.sls`.

ID declaration

Defines an individual highstate component. Always references a value of a dictionary containing keys referencing *State declaration* and *Requisite declaration*. Can be overridden by a *Name declaration* or a *Names declaration*.

Occurs on the top level or under the *Extend declaration*.

Must be unique across entire state tree. If the same ID declaration is used twice, only the first one matched will be used. All subsequent ID declarations with the same name will be ignored.

Note: Naming gotchas

In Salt versions earlier than 0.9.7, ID declarations containing dots would result in unpredictable highstate output.

Extend declaration

Extends a *Name declaration* from an included `SLS module`. The keys of the extend declaration always define existing :ref`ID declaration` which have been defined in included `SLS modules`.

Occurs only in the top level and defines a dictionary.

States cannot be extended more than once in a single state run.

Extend declarations are useful for adding-to or overriding parts of a *State declaration* that is defined in another `SLS` file. In the following contrived example, the shown `mywebsite.sls` file is `include` -ing and `extend` -ing the `apache.sls` module in order to add a `watch` declaration that will restart Apache whenever the Apache configuration file, `mywebsite` changes.

```
include:
  - apache

extend:
  apache:
    service:
      - watch:
        - file: mywebsite

mywebsite:
  file.managed:
    - name: /var/www/mysite
```

See also:

watch_in and require_in

Sometimes it is more convenient to use the *watch_in* or *require_in* syntax instead of extending another `SLS` file.

State Requisites

State declaration

A list which contains one string defining the *Function declaration* and any number of *Function arg declaration* dictionaries.

Can, optionally, contain a number of additional components like the name override components — *name* and *names*. Can also contain *requisite declarations*.

Occurs under an *ID declaration*.

Requisite declaration

A list containing *requisite references*.

Used to build the action dependency tree. While Salt states are made to execute in a deterministic order, this order is managed by requiring and watching other Salt states.

Occurs as a list component under a *State declaration* or as a key under an *ID declaration*.

Requisite reference

A single key dictionary. The key is the name of the referenced *State declaration* and the value is the ID of the referenced *ID declaration*.

Occurs as a single index in a *Requisite declaration* list.

Function declaration

The name of the function to call within the state. A state declaration can contain only a single function declaration.

For example, the following state declaration calls the `installed` function in the `pkg` state module:

```
httpd:
  pkg.installed: []
```

The function can be declared inline with the state as a shortcut. The actual data structure is compiled to this form:

```
httpd:
  pkg:
    - installed
```

Where the function is a string in the body of the state declaration. Technically when the function is declared in dot notation the compiler converts it to be a string in the state declaration list. Note that the use of the first example more than once in an ID declaration is invalid yaml.

INVALID:

```
httpd:
  pkg.installed
  service.running
```

When passing a function without arguments and another state declaration within a single ID declaration, then the long or ``standard'' format needs to be used since otherwise it does not represent a valid data structure.

VALID:

```
httpd:
  pkg.installed: []
  service.running: []
```

Occurs as the only index in the *State declaration* list.

Function arg declaration

A single key dictionary referencing a Python type which is to be passed to the named *Function declaration* as a parameter. The type must be the data type expected by the function.

Occurs under a *Function declaration*.

For example in the following state declaration `user`, `group`, and `mode` are passed as arguments to the `managed` function in the `file` state module:

```
/etc/http/conf/http.conf:
  file.managed:
    - user: root
    - group: root
    - mode: 644
```

Name declaration

Overrides the `name` argument of a *State declaration*. If `name` is not specified the *ID declaration* satisfies the `name` argument.

The name is always a single key dictionary referencing a string.

Overriding `name` is useful for a variety of scenarios.

For example, avoiding clashing ID declarations. The following two state declarations cannot both have `/etc/motd` as the ID declaration:

```
motd_perms:
  file.managed:
    - name: /etc/motd
    - mode: 644

motd_quote:
  file.append:
    - name: /etc/motd
    - text: "Of all smells, bread; of all tastes, salt."
```

Another common reason to override `name` is if the ID declaration is long and needs to be referenced in multiple places. In the example below it is much easier to specify `mywebsite` than to specify `/etc/apache2/sites-available/mywebsite.com` multiple times:

```
mywebsite:
  file.managed:
    - name: /etc/apache2/sites-available/mywebsite.com
    - source: salt://mywebsite.com

a2ensite mywebsite.com:
  cmd.wait:
    - unless: test -L /etc/apache2/sites-enabled/mywebsite.com
    - watch:
      - file: mywebsite

apache2:
  service.running:
    - watch:
      - file: mywebsite
```

Names declaration

Expands the contents of the containing *State declaration* into multiple state declarations, each with its own name.

For example, given the following state declaration:

```
python-pkgs:
  pkg.installed:
    - names:
      - python-django
      - python-crypto
      - python-yaml
```

Once converted into the lowstate data structure the above state declaration will be expanded into the following three state declarations:

```
python-django:
  pkg.installed

python-crypto:
  pkg.installed

python-yaml:
  pkg.installed
```

Other values can be overridden during the expansion by providing an additional dictionary level.

New in version 2014.7.0.

```
ius:
  pkgrepo.managed:
    - humanname: IUS Community Packages for Enterprise Linux 6 - $basearch
    - gpgcheck: 1
    - baseurl: http://mirror.rackspace.com/ius/stable/CentOS/6/$basearch
    - gpgkey: http://dl.iuscommunity.org/pub/ius/IUS-COMMUNITY-GPG-KEY
    - names:
        - ius
        - ius-devel:
            - baseurl: http://mirror.rackspace.com/ius/development/CentOS/6/$basearch
```

Large example

Here is the layout in yaml using the names of the highdata structure components.

```
<Include Declaration>:
  - <Module Reference>
  - <Module Reference>

<Extend Declaration>:
  <ID Declaration>:
    [<overrides>]


# standard declaration

<ID Declaration>:
  <State Module>:
    - <Function>
    - <Function Arg>
    - <Function Arg>
```

```
    - <Function Arg>
    - <Name>: <name>
    - <Requisite Declaration>:
      - <Requisite Reference>
      - <Requisite Reference>


# inline function and names

<ID Declaration>:
  <State Module>.<Function>:
    - <Function Arg>
    - <Function Arg>
    - <Function Arg>
    - <Names>:
      - <name>
      - <name>
      - <name>
    - <Requisite Declaration>:
      - <Requisite Reference>
      - <Requisite Reference>


# multiple states for single id

<ID Declaration>:
  <State Module>:
    - <Function>
    - <Function Arg>
    - <Name>: <name>
    - <Requisite Declaration>:
      - <Requisite Reference>
  <State Module>:
    - <Function>
    - <Function Arg>
    - <Names>:
      - <name>
      - <name>
    - <Requisite Declaration>:
      - <Requisite Reference>
```

22.26.9 Include and Exclude

Salt sls files can include other sls files and exclude sls files that have been otherwise included. This allows for an sls file to easily extend or manipulate other sls files.

Include

When other sls files are included, everything defined in the included sls file will be added to the state run. When including define a list of sls formulas to include:

```
include:
  - http
  - libvirt
```

The include statement will include sls formulas from the same environment that the including sls formula is in. But the environment can be explicitly defined in the configuration to override the running environment, therefore if an sls formula needs to be included from an external environment named ``dev'' the following syntax is used:

```
include:
  - dev: http
```

Relative Include

In Salt 0.16.0 the capability to include sls formulas which are relative to the running sls formula was added, simply precede the formula name with a .:

```
include:
  - .virt
  - .virt.hyper
```

Exclude

The exclude statement, added in Salt 0.10.3 allows an sls to hard exclude another sls file or a specific id. The component is excluded after the high data has been compiled, so nothing should be able to override an exclude.

Since the exclude can remove an id or an sls the type of component to exclude needs to be defined. an exclude statement that verifies that the running highstate does not contain the *http* sls and the */etc/vimrc* id would look like this:

```
exclude:
  - sls: http
  - id: /etc/vimrc
```

22.26.10 State System Layers

The Salt state system is comprised of multiple layers. While using Salt does not require an understanding of the state layers, a deeper understanding of how Salt compiles and manages states can be very beneficial.

Function Call

The lowest layer of functionality in the state system is the direct state function call. State executions are executions of single state functions at the core. These individual functions are defined in state modules and can be called directly via the `state.single` command.

```
salt '*' state.single pkg.installed name='vim'
```

Low Chunk

The low chunk is the bottom of the Salt state compiler. This is a data representation of a single function call. The low chunk is sent to the state caller and used to execute a single state function.

A single low chunk can be executed manually via the `state.low` command.

```
salt '*' state.low '{name: vim, state: pkg, fun: installed}'
```

The passed data reflects what the state execution system gets after compiling the data down from sls formulas.

Low State

The *Low State* layer is the list of low chunks ``evaluated'' in order. To see what the low state looks like for a highstate, run:

```
salt '*' state.show_lowstate
```

This will display the raw lowstate in the order which each low chunk will be evaluated. The order of evaluation is not necessarily the order of execution, since requisites are evaluated at runtime. Requisite execution and evaluation is finite; this means that the order of execution can be ascertained with 100% certainty based on the order of the low state.

High Data

High data is the data structure represented in YAML via SLS files. The High data structure is created by merging the data components rendered inside sls files (or other render systems). The High data can be easily viewed by executing the `state.show_highstate` or `state.show_sls` functions. Since this data is a somewhat complex data structure, it may be easier to read using the json, yaml, or pprint outputters:

```
salt '*' state.show_highstate --out yaml
salt '*' state.show_sls edit.vim --out pprint
```

SLS

Above ``High Data'', the logical layers are no longer technically required to be executed, or to be executed in a hierarchy. This means that how the High data is generated is optional and very flexible. The SLS layer allows for many mechanisms to be used to render sls data from files or to use the fileserver backend to generate sls and file data from external systems.

The SLS layer can be called directly to execute individual sls formulas.

Note: SLS Formulas have historically been called ``SLS files''. This is because a single SLS was only constituted in a single file. Now the term ``SLS Formula'' better expresses how a compartmentalized SLS can be expressed in a much more dynamic way by combining pillar and other sources, and the SLS can be dynamically generated.

To call a single SLS formula named `edit.vim`, execute `state.sls`:

```
salt '*' state.sls edit.vim
```

HighState

Calling SLS directly logically assigns what states should be executed from the context of the calling minion. The Highstate layer is used to allow for full contextual assignment of what is executed where to be tied to groups of, or individual, minions entirely from the master. This means that the environment of a minion, and all associated execution data pertinent to said minion, can be assigned from the master without needing to execute or configure

anything on the target minion. This also means that the minion can independently retrieve information about its complete configuration from the master.

To execute the High State call `state.highstate`:

```
salt '*' state.highstate
```

OverState

The overstate layer expresses the highest functional layer of Salt's automated logic systems. The Overstate allows for stateful and functional orchestration of routines from the master. The overstate defines in data execution stages which minions should execute states, or functions, and in what order using requisite logic.

22.26.11 The Orchestrate Runner

Note: This documentation has been moved *here*.

22.26.12 Ordering States

The way in which configuration management systems are executed is a hotly debated topic in the configuration management world. Two major philosophies exist on the subject, to either execute in an imperative fashion where things are executed in the order in which they are defined, or in a declarative fashion where dependencies need to be mapped between objects.

Imperative ordering is finite and generally considered easier to write, but declarative ordering is much more powerful and flexible but generally considered more difficult to create.

Salt has been created to get the best of both worlds. States are evaluated in a finite order, which guarantees that states are always executed in the same order, and the states runtime is declarative, making Salt fully aware of dependencies via the *requisite* system.

State Auto Ordering

Salt always executes states in a finite manner, meaning that they will always execute in the same order regardless of the system that is executing them. But in Salt 0.17.0, the `state_auto_order` option was added. This option makes states get evaluated in the order in which they are defined in sls files.

The evaluation order makes it easy to know what order the states will be executed in, but it is important to note that the requisite system will override the ordering defined in the files, and the `order` option described below will also override the order in which states are defined in sls files.

If the classic ordering is preferred (lexicographic), then set `state_auto_order` to `False` in the master configuration file.

Requisite Statements

Note: This document represents behavior exhibited by Salt requisites as of version 0.9.7 of Salt.

Often when setting up states any single action will require or depend on another action. Salt allows for the building of relationships between states with requisite statements. A requisite statement ensures that the named state is

evaluated before the state requiring it. There are three types of requisite statements in Salt, **require**, **watch**, and **prereq**.

These requisite statements are applied to a specific state declaration:

```
httpd:
  pkg.installed: []
  file.managed:
    - name: /etc/httpd/conf/httpd.conf
    - source: salt://httpd/httpd.conf
    - require:
      - pkg: httpd
```

In this example, the **require** requisite is used to declare that the file /etc/httpd/conf/httpd.conf should only be set up if the pkg state executes successfully.

The requisite system works by finding the states that are required and executing them before the state that requires them. Then the required states can be evaluated to see if they have executed correctly.

Require statements can refer to any state defined in Salt. The basic examples are *pkg*, *service*, and *file*, but any used state can be referenced.

In addition to state declarations such as pkg, file, etc., **sls** type requisites are also recognized, and essentially allow `chaining' of states. This provides a mechanism to ensure the proper sequence for complex state formulas, especially when the discrete states are split or groups into separate sls files:

```
include:
  - network

httpd:
  pkg.installed: []
  service.running:
    - require:
      - pkg: httpd
      - sls: network
```

In this example, the httpd service running state will not be applied (i.e., the httpd service will not be started) unless both the httpd package is installed AND the network state is satisfied.

Note: Requisite matching

Requisites match on both the ID Declaration and the `name` parameter. Therefore, if using the `pkgs` or `sources` argument to install a list of packages in a pkg state, it's important to note that it is impossible to match an individual package in the list, since all packages are installed as a single state.

Multiple Requisites

The requisite statement is passed as a list, allowing for the easy addition of more requisites. Both requisite types can also be separately declared:

```
httpd:
  pkg.installed: []
  service.running:
    - enable: True
    - watch:
      - file: /etc/httpd/conf/httpd.conf
```

```
    - require:
      - pkg: httpd
      - user: httpd
      - group: httpd
  file.managed:
    - name: /etc/httpd/conf/httpd.conf
    - source: salt://httpd/httpd.conf
    - require:
      - pkg: httpd
  user.present: []
  group.present: []
```

In this example, the httpd service is only going to be started if the package, user, group, and file are executed successfully.

Requisite Documentation

For detailed information on each of the individual requisites, *please look here.*

The Order Option

Before using the *order* option, remember that the majority of state ordering should be done with a *Requisite declaration*, and that a requisite declaration will override an *order* option, so a state with order option should not require or required by other states.

The order option is used by adding an order number to a state declaration with the option *order*:

```
vim:
  pkg.installed:
    - order: 1
```

By adding the order option to *1* this ensures that the vim package will be installed in tandem with any other state declaration set to the order *1*.

Any state declared without an order option will be executed after all states with order options are executed.

But this construct can only handle ordering states from the beginning. Certain circumstances will present a situation where it is desirable to send a state to the end of the line. To do this, set the order to `last`:

```
vim:
  pkg.installed:
    - order: last
```

22.26.13 OverState System

Note: This documentation has been moved *here*.

22.26.14 State Providers

New in version 0.9.8.

Salt predetermines what modules should be mapped to what uses based on the properties of a system. These determinations are generally made for modules that provide things like package and service management.

Sometimes in states, it may be necessary to use an alternative module to provide the needed functionality. For instance, an older Arch Linux system may not be running systemd, so instead of using the systemd service module, you can revert to the default service module:

```
httpd:
  service.running:
    - enable: True
    - provider: service
```

In this instance, the basic `service` module (which manages **`sysvinit`**-based services) will replace the `systemd` module which is used by default on Arch Linux.

However, if it is necessary to make this override for most or every service, it is better to just override the provider in the minion config file, as described in the section below.

Setting a Provider in the Minion Config File

Sometimes, when running Salt on custom Linux spins, or distribution that are derived from other distributions, Salt does not successfully detect providers. The providers which are most likely to be affected by this are:

- pkg
- service
- user
- group

When something like this happens, rather than specifying the provider manually in each state, it easier to use the `providers` parameter in the minion config file to set the provider.

If you end up needing to override a provider because it was not detected, please let us know! File an issue on the issue tracker, and provide the output from the `grains.items` function, taking care to sanitize any sensitive information.

Below are tables that should help with deciding which provider to use if one needs to be overridden.

Provider: `pkg`

Execution Module	Used for
apt	Debian/Ubuntu-based distros which use `apt-get(8)` for package management
brew	Mac OS software management using Homebrew
ebuild	Gentoo-based systems (utilizes the `portage` python module as well as `emerge(1)`)
freebsdpkg	FreeBSD-based OSes using `pkg_add(1)`
openbsdpkg	OpenBSD-based OSes using `pkg_add(1)`
pacman	Arch Linux-based distros using `pacman(8)`
pkgin	NetBSD-based OSes using `pkgin(1)`
pkgng	FreeBSD-based OSes using `pkg(8)`
pkgutil	Solaris-based OSes using OpenCSW`s `pkgutil(1)`
solarispkg	Solaris-based OSes using `pkgadd(1M)`
solarisips	Solaris-based OSes using IPS `pkg(1)`
win_pkg	Windows
yumpkg	RedHat-based distros and derivatives (wraps `yum(8)`)
zypper	SUSE-based distros using `zypper(8)`

Provider: `service`

Execution Module	Used for
debian_service	Debian (non-systemd)
freebsdservice	FreeBSD-based OSes using `service(8)`
gentoo_service	Gentoo Linux using **`sysvinit`** and `rc-update(8)`
launchctl	Mac OS hosts using `launchctl(1)`
netbsdservice	NetBSD-based OSes
openbsdservice	OpenBSD-based OSes
rh_service	RedHat-based distros and derivatives using `service(8)` and `chkconfig(8)`. Supports both pure sysvinit and mixed sysvinit/upstart systems.
service	Fallback which simply wraps sysvinit scripts
smf	Solaris-based OSes which use SMF
systemd	Linux distros which use systemd
upstart	Ubuntu-based distros using upstart
win_service	Windows

Provider: `user`

Execution Module	Used for
useradd	Linux, NetBSD, and OpenBSD systems using `useradd(8)`, `userdel(8)`, and `usermod(8)`
pw_user	FreeBSD-based OSes using `pw(8)`
solaris_user	Solaris-based OSes using `useradd(1M)`, `userdel(1M)`, and `usermod(1M)`
win_useradd	Windows

Provider: group

Execution Module	Used for
groupadd	Linux, NetBSD, and OpenBSD systems using `groupadd(8)`, `groupdel(8)`, and `groupmod(8)`
pw_group	FreeBSD-based OSes using `pw(8)`
solaris_group	Solaris-based OSes using `groupadd(1M)`, `groupdel(1M)`, and `groupmod(1M)`
win_groupadd	Windows

Arbitrary Module Redirects

The provider statement can also be used for more powerful means, instead of overwriting or extending the module used for the named service an arbitrary module can be used to provide certain functionality.

```
emacs:
  pkg.installed:
    - provider:
      - cmd: customcmd
```

In this example, the state is being instructed to use a custom module to invoke commands.

Arbitrary module redirects can be used to dramatically change the behavior of a given state.

22.26.15 Requisites and Other Global State Arguments

Requisites

The Salt requisite system is used to create relationships between states. The core idea being that, when one state is dependent somehow on another, that inter-dependency can be easily defined.

Requisites come in two types: Direct requisites (such as `require`), and requisite_ins (such as `require_in`). The relationships are directional: a direct requisite requires something from another state. However, a requisite_in inserts a requisite into the targeted state pointing to the targeting state. The following example demonstrates a direct requisite:

```
vim:
  pkg.installed: []

/etc/vimrc:
  file.managed:
    - source: salt://edit/vimrc
    - require:
      - pkg: vim
```

In the example above, the file `/etc/vimrc` depends on the vim package.

Requisite_in statements are the opposite. Instead of saying ``I depend on something'', requisite_ins say ``Someone depends on me'':

```
vim:
  pkg.installed:
    - require_in:
```

```
      - file: /etc/vimrc

/etc/vimrc:
  file.managed:
    - source: salt://edit/vimrc
```

So here, with a requisite_in, the same thing is accomplished as in the first example, but the other way around. The vim package is saying ``/etc/vimrc depends on me''. This will result in a `require` being inserted into the `/etc/vimrc` state which targets the `vim` state.

In the end, a single dependency map is created and everything is executed in a finite and predictable order.

Note: Requisite matching

Requisites match on both the ID Declaration and the `name` parameter. This means that, in the example above, the `require_in` requisite would also have been matched if the `/etc/vimrc` state was written as follows:

```
vimrc:
  file.managed:
    - name: /etc/vimrc
    - source: salt://edit/vimrc
```

Direct Requisite and Requisite_in types

There are six direct requisite statements that can be used in Salt: `require`, `watch`, `prereq`, `use`, `onchanges`, and `onfail`. Each direct requisite also has a corresponding requisite_in: `require_in`, `watch_in`, `prereq_in`, `use_in`, `onchanges_in`, and `onfail_in`. All of the requisites define specific relationships and always work with the dependency logic defined above.

require The use of `require` demands that the dependent state executes before the depending state. The state containing the `require` requisite is defined as the depending state. The state specified in the `require` statement is defined as the dependent state. If the dependent state's execution succeeds, the depending state will then execute. If the dependent state's execution fails, the depending state will not execute. In the first example above, the file `/etc/vimrc` will only execute after the vim package is installed successfully.

Require an entire sls file As of Salt 0.16.0, it is possible to require an entire sls file. Do this first by including the sls file and then setting a state to `require` the included sls file:

```
include:
  - foo

bar:
  pkg.installed:
    - require:
      - sls: foo
```

watch `watch` statements are used to add additional behavior when there are changes in other states.

Note: If a state should only execute when another state has changes, and otherwise do nothing, the new `onchanges` requisite should be used instead of `watch`. `watch` is designed to add *additional* behavior when there are changes, but otherwise execute normally.

The state containing the `watch` requisite is defined as the watching state. The state specified in the `watch` statement is defined as the watched state. When the watched state executes, it will return a dictionary containing a key named ``changes''. Here are two examples of state return dictionaries, shown in json for clarity:

```
"local": {
    "file_|-/tmp/foo_|-/tmp/foo_|-directory": {
        "comment": "Directory /tmp/foo updated",
        "__run_num__": 0,
        "changes": {
            "user": "bar"
        },
        "name": "/tmp/foo",
        "result": true
    }
}

"local": {
    "pkgrepo_|-salt-minion_|-salt-minion_|-managed": {
        "comment": "Package repo 'salt-minion' already configured",
        "__run_num__": 0,
        "changes": {},
        "name": "salt-minion",
        "result": true
    }
}
```

If the ``result'' of the watched state is `True`, the watching state *will execute normally*. This part of `watch` mirrors the functionality of the `require` requisite. If the ``result'' of the watched state is `False`, the watching state will never run, nor will the watching state's `mod_watch` function execute.

However, if the ``result'' of the watched state is `True`, and the ``changes'' key contains a populated dictionary (changes occurred in the watched state), then the `watch` requisite can add additional behavior. This additional behavior is defined by the `mod_watch` function within the watching state module. If the `mod_watch` function exists in the watching state module, it will be called *in addition to* the normal watching state. The return data from the `mod_watch` function is what will be returned to the master in this case; the return data from the main watching function is discarded.

If the ``changes'' key contains an empty dictionary, the `watch` requisite acts exactly like the `require` requisite (the watching state will execute if ``result'' is `True`, and fail if ``result'' is `False` in the watched state).

Note: Not all state modules contain `mod_watch`. If `mod_watch` is absent from the watching state module, the `watch` requisite behaves exactly like a `require` requisite.

A good example of using `watch` is with a `service.running` state. When a service watches a state, then the service is reloaded/restarted when the watched state changes, in addition to Salt ensuring that the service is running.

```
ntpd:
  service.running:
    - watch:
      - file: /etc/ntp.conf
  file.managed:
    - name: /etc/ntp.conf
    - source: salt://ntp/files/ntp.conf
```

prereq New in version 0.16.0.

`prereq` allows for actions to be taken based on the expected results of a state that has not yet been executed. The state containing the `prereq` requisite is defined as the pre-requiring state. The state specified in the `prereq` statement is defined as the pre-required state.

When a `prereq` requisite is evaluated, the pre-required state reports if it expects to have any changes. It does this by running the pre-required single state as a test-run by enabling `test=True`. This test-run will return a dictionary containing a key named ``changes''. (See the `watch` section above for examples of ``changes'' dictionaries.)

If the ``changes'' key contains a populated dictionary, it means that the pre-required state expects changes to occur when the state is actually executed, as opposed to the test-run. The pre-requiring state will now actually run. If the pre-requiring state executes successfully, the pre-required state will then execute. If the pre-requiring state fails, the pre-required state will not execute.

If the ``changes'' key contains an empty dictionary, this means that changes are not expected by the pre-required state. Neither the pre-required state nor the pre-requiring state will run.

The best way to define how `prereq` operates is displayed in the following practical example: When a service should be shut down because underlying code is going to change, the service should be off-line while the update occurs. In this example, `graceful-down` is the pre-requiring state and `site-code` is the pre-required state.

```
graceful-down:
  cmd.run:
    - name: service apache graceful
    - prereq:
      - file: site-code

site-code:
  file.recurse:
    - name: /opt/site_code
    - source: salt://site/code
```

In this case the apache server will only be shutdown if the site-code state expects to deploy fresh code via the file.recurse call. The site-code deployment will only be executed if the graceful-down run completes successfully.

onfail New in version 2014.7.0.

The `onfail` requisite allows for reactions to happen strictly as a response to the failure of another state. This can be used in a number of ways, such as executing a second attempt to set up a service or begin to execute a separate thread of states because of a failure.

The `onfail` requisite is applied in the same way as `require` as `watch`:

```
primary_mount:
  mount.mounted:
    - name: /mnt/share
    - device: 10.0.0.45:/share
    - fstype: nfs

backup_mount:
  mount.mounted:
    - name: /mnt/share
    - device: 192.168.40.34:/share
    - fstype: nfs
    - onfail:
      - mount: primary_mount
```

onchanges New in version 2014.7.0.

The `onchanges` requisite makes a state only apply if the required states generate changes, and if the watched state's ``result'' is `True`. This can be a useful way to execute a post hook after changing aspects of a system.

use The `use` requisite is used to inherit the arguments passed in another id declaration. This is useful when many files need to have the same defaults.

```
/etc/foo.conf:
  file.managed:
    - source: salt://foo.conf
    - template: jinja
    - mkdirs: True
    - user: apache
    - group: apache
    - mode: 755

/etc/bar.conf
  file.managed:
    - source: salt://bar.conf
    - use:
      - file: /etc/foo.conf
```

The `use` statement was developed primarily for the networking states but can be used on any states in Salt. This makes sense for the networking state because it can define a long list of options that need to be applied to multiple network interfaces.

The `use` statement does not inherit the requisites arguments of the targeted state. This means also a chain of `use` requisites would not inherit inherited options.

The _in versions of requisites All of the requisites also have corresponding requisite_in versions, which do the reverse of their normal counterparts. The examples below all use `require_in` as the example, but note that all of the `_in` requisites work the same way: They result in a normal requisite in the targeted state, which targets the state which has defines the requisite_in. Thus, a `require_in` causes the target state to `require` the targeting state. Similarly, a `watch_in` causes the target state to `watch` the targeting state. This pattern continues for the rest of the requisites.

If a state declaration needs to be required by another state declaration then `require_in` can accommodate it. Therefore, these two sls files would be the same in the end:

Using `require`

```
httpd:
  pkg.installed: []
  service.running:
    - require:
      - pkg: httpd
```

Using `require_in`

```
httpd:
  pkg.installed:
    - require_in:
      - service: httpd
  service.running: []
```

The `require_in` statement is particularly useful when assigning a require in a separate sls file. For instance it may be common for httpd to require components used to set up PHP or mod_python, but the HTTP state does not need to be aware of the additional components that require it when it is set up:

http.sls

```
httpd:
  pkg.installed: []
  service.running:
    - require:
      - pkg: httpd
```

php.sls

```
include:
  - http

php:
  pkg.installed:
    - require_in:
      - service: httpd
```

mod_python.sls

```
include:
  - http

mod_python:
  pkg.installed:
    - require_in:
      - service: httpd
```

Now the httpd server will only start if php or mod_python are first verified to be installed. Thus allowing for a requisite to be defined ``after the fact''.

Altering States

The state altering system is used to make sure that states are evaluated exactly as the user expects. It can be used to double check that a state preformed exactly how it was expected to, or to make 100% sure that a state only runs under certain conditions. The use of unless or onlyif options help make states even more stateful. The check_cmds option helps ensure that the result of a state is evaluated correctly.

Unless

New in version 2014.7.0.

The `unless` requisite specifies that a state should only run when any of the specified commands return `False`. The `unless` requisite operates as NOR and is useful in giving more granular control over when a state should execute.

NOTE: Under the hood `unless` calls `cmd.retcode` with `python_shell=True`. This means the commands referenced by unless will be parsed by a shell, so beware of side-effects as this shell will be run with the same privileges as the salt-minion.

```
vim:
  pkg.installed:
    - unless:
      - rpm -q vim-enhanced
      - ls /usr/bin/vim
```

In the example above, the state will only run if either the vim-enhanced package is not installed (returns `False`) or if /usr/bin/vim does not exist (returns `False`). The state will run if both commands return `False`.

However, the state will not run if both commands return `True`.

Unless checks are resolved for each name to which they are associated.

For example:

```
deploy_app:
  cmd.run:
    - names:
      - first_deploy_cmd
      - second_deploy_cmd
  - unless: ls /usr/bin/vim
```

In the above case, `some_check` will be run prior to _each_ name -- once for `first_deploy_cmd` and a second time for `second_deploy_cmd`.

Onlyif

New in version 2014.7.0.

`onlyif` is the opposite of `unless`. If all of the commands in `onlyif` return `True`, then the state is run. If any of the specified commands return `False`, the state will not run.

NOTE: Under the hood `onlyif` calls `cmd.retcode` with `python_shell=True`. This means the commands referenced by unless will be parsed by a shell, so beware of side-effects as this shell will be run with the same privileges as the salt-minion.

```
stop-volume:
  module.run:
    - name: glusterfs.stop_volume
    - m_name: work
    - onlyif:
      - gluster volume status work
    - order: 1

remove-volume:
  module.run:
    - name: glusterfs.delete
    - m_name: work
    - onlyif:
      - gluster volume info work
    - watch:
      - cmd: stop-volume
```

The above example ensures that the stop_volume and delete modules only run if the gluster commands return a 0 ret value.

Listen/Listen_in

New in version 2014.7.0.

listen and its counterpart listen_in trigger mod_wait functions for states, when those states succeed and result in changes, similar to how watch its counterpart watch_in. Unlike watch and watch_in, listen, and listen_in will not modify the order of states and can be used to ensure your states are executed in the order they are defined. All listen/listen_in actions will occur at the end of a state run, after all states have completed.

```
restart-apache2:
  service.running:
    - name: apache2
    - listen:
      - file: /etc/apache2/apache2.conf

configure-apache2:
  file.managed:
    - path: /etc/apache2/apache2.conf
    - source: salt://apache2/apache2.conf
```

This example will cause apache2 to be restarted when the apache2.conf file is changed, but the apache2 restart will happen at the end of the state run.

```
restart-apache2:
  service.running:
    - name: apache2

configure-apache2:
  file.managed:
    - path: /etc/apache2/apache2.conf
    - source: salt://apache2/apache2.conf
    - listen_in:
      - service: apache2
```

This example does the same as the above example, but puts the state argument on the file resource, rather than the service resource.

check_cmd

New in version 2014.7.0.

Check Command is used for determining that a state did or did not run as expected.

NOTE: Under the hood `check_cmd` calls `cmd.retcode` with `python_shell=True`. This means the commands referenced by unless will be parsed by a shell, so beware of side-effects as this shell will be run with the same privileges as the salt-minion.

```
comment-repo:
  file.replace:
    - path: /etc/yum.repos.d/fedora.repo
    - pattern: ^enabled=0
    - repl: enabled=1
    - check_cmd:
      - grep 'enabled=0' /etc/yum.repos.d/fedora.repo && return 1 || return 0
```

This will attempt to do a replace on all enabled=0 in the .repo file, and replace them with enabled=1. The check_cmd is just a bash command. It will do a grep for enabled=0 in the file, and if it finds any, it will return a 0, which will prompt the && portion of the command to return a 1, causing check_cmd to set the state as failed. If it returns a 1, meaning it didn't find any `enabled=0' it will hit the || portion of the command, returning a 0, and declaring the function succeeded.

Overriding Checks

There are two commands used for the above checks.

`mod_run_check` is used to check for `onlyif` and `unless`. If the goal is to override the global check for these to variables, include a `mod_run_check` in the salt/states/ file.

`mod_run_check_cmd` is used to check for the check_cmd options. To override this one, include a `mod_run_check_cmd` in the states file for the state.

22.26.16 Startup States

Sometimes it may be desired that the salt minion execute a state run when it is started. This alleviates the need for the master to initiate a state run on a new minion and can make provisioning much easier.

As of Salt 0.10.3 the minion config reads options that allow for states to be executed at startup. The options are *startup_states*, *sls_list*, and *top_file*.

The *startup_states* option can be passed one of a number of arguments to define how to execute states. The available options are:

highstate Execute `state.highstate`

sls Read in the `sls_list` option and execute the named sls files

top Read in the `top_file` option and execute states based on that top file on the Salt Master

Examples:

Execute `state.highstate` when starting the minion:

```
startup_states: highstate
```

Execute the sls files *edit.vim* and *hyper*:

```
startup_states: sls

sls_list:
  - edit.vim
  - hyper
```

22.26.17 State Testing

Executing a Salt state run can potentially change many aspects of a system and it may be desirable to first see what a state run is going to change before applying the run.

Salt has a test interface to report on exactly what will be changed, this interface can be invoked on any of the major state run functions:

```
salt '*' state.highstate test=True
salt '*' state.sls test=True
salt '*' state.single test=True
```

The test run is mandated by adding the `test=True` option to the states. The return information will show states that will be applied in yellow and the result is reported as `None`.

Default Test

If the value `test` is set to `True` in the minion configuration file then states will default to being executed in test mode. If this value is set then states can still be run by calling test=False:

```
salt '*' state.highstate test=False
salt '*' state.sls test=False
salt '*' state.single test=False
```

22.26.18 The Top File

The top file is used to map what SLS modules get loaded onto what minions via the state system. The top file creates a few general abstractions. First it maps what nodes should pull from which environments, next it defines which matches systems should draw from.

Environments

Environments allow conceptually organizing state tree directories. Environments can be made to be self-contained or state trees can be made to bleed through environments.

Note: Environments in Salt are very flexible. This section defines how the top file can be used to define what states from what environments are to be used for specific minions.

If the intent is to bind minions to specific environments, then the *environment* option can be set in the minion configuration file.

The environments in the top file corresponds with the environments defined in the `file_roots` variable. In a simple, single environment setup you only have the `base` environment, and therefore only one state tree. Here is a simple example of `file_roots` in the master configuration:

```
file_roots:
  base:
    - /srv/salt
```

This means that the top file will only have one environment to pull from, here is a simple, single environment top file:

```
base:
  '*':
    - core
    - edit
```

This also means that `/srv/salt` has a state tree. But if you want to use multiple environments, or partition the file server to serve more than just the state tree, then the `file_roots` option can be expanded:

```
file_roots:
  base:
    - /srv/salt/base
  dev:
    - /srv/salt/dev
  qa:
    - /srv/salt/qa
  prod:
    - /srv/salt/prod
```

Then our top file could reference the environments:

```
dev:
  'webserver*dev*':
    - webserver
  'db*dev*':
    - db
qa:
  'webserver*qa*':
    - webserver
  'db*qa*':
    - db
prod:
  'webserver*prod*':
    - webserver
  'db*prod*':
    - db
```

In this setup we have state trees in three of the four environments, and no state tree in the `base` environment. Notice that the targets for the minions specify environment data. In Salt the master determines who is in what environment, and many environments can be crossed together. For instance, a separate global state tree could be added to the `base` environment if it suits your deployment:

```
base:
  '*':
    - global
dev:
  'webserver*dev*':
    - webserver
  'db*dev*':
    - db
qa:
  'webserver*qa*':
    - webserver
  'db*qa*':
    - db
prod:
  'webserver*prod*':
    - webserver
  'db*prod*':
    - db
```

In this setup all systems will pull the global SLS from the base environment, as well as pull from their respective environments. If you assign only one SLS to a system, as in this example, a shorthand is also available:

```
base:
  '*': global
dev:
  'webserver*dev*': webserver
  'db*dev*':        db
qa:
  'webserver*qa*': webserver
  'db*qa*':        db
prod:
  'webserver*prod*': webserver
  'db*prod*':        db
```

Note: The top files from all defined environments will be compiled into a single top file for all states. Top files are environment agnostic.

Remember, that since everything is a file in Salt, the environments are primarily file server environments, this means that environments that have nothing to do with states can be defined and used to distribute other files. A clean and recommended setup for multiple environments would look like this:

```
# Master file_roots configuration:
file_roots:
  base:
    - /srv/salt/base
  dev:
    - /srv/salt/dev
  qa:
    - /srv/salt/qa
  prod:
    - /srv/salt/prod
```

Then only place state trees in the dev, qa, and prod environments, leaving the base environment open for generic file transfers. Then the top.sls file would look something like this:

```
dev:
  'webserver*dev*':
    - webserver
  'db*dev*':
    - db
qa:
  'webserver*qa*':
    - webserver
  'db*qa*':
    - db
prod:
  'webserver*prod*':
    - webserver
  'db*prod*':
    - db
```

Other Ways of Targeting Minions

In addition to globs, minions can be specified in top files a few other ways. Some common ones are *compound matches* and *node groups*.

Here is a slightly more complex top file example, showing the different types of matches you can perform:

```
base:
    '*':
        - ldap-client
        - networking
        - salt.minion

    'salt-master*':
        - salt.master

    '^(memcache|web).(qa|prod).loc$':
        - match: pcre
        - nagios.mon.web
        - apache.server

    'os:Ubuntu':
        - match: grain
        - repos.ubuntu

    'os:(RedHat|CentOS)':
        - match: grain_pcre
        - repos.epel

    'foo,bar,baz':
        - match: list
        - database

    'somekey:abc':
        - match: pillar
        - xyz

    'nag1* or G@role:monitoring':
        - match: compound
        - nagios.server
```

In this example `top.sls`, all minions get the ldap-client, networking, and salt.minion states. Any minion with an id matching the `salt-master*` glob will get the salt.master state. Any minion with ids matching the regular expression `^(memcache|web).(qa|prod).loc$` will get the nagios.mon.web and apache.server states. All Ubuntu minions will receive the repos.ubuntu state, while all RHEL and CentOS minions will receive the repos.epel state. The minions `foo`, `bar`, and `baz` will receive the database state. Any minion with a pillar named `somekey`, having a value of `abc` will receive the xyz state. Finally, minions with ids matching the nag1* glob or with a grain named `role` equal to `monitoring` will receive the nagios.server state.

How Top Files Are Compiled

> **Warning**: There is currently a known issue with the topfile compilation. The below may not be completely valid until https://github.com/saltstack/salt/issues/12483#issuecomment-64181598 is closed.

As mentioned earlier, the top files in the different environments are compiled into a single set of data. The way in which this is done follows a few rules, which are important to understand when arranging top files in different environments. The examples below all assume that the `file_roots` are set as in the *above multi-environment example*.

1. The `base` environment's top file is processed first. Any environment which is defined in the `base` top.sls as well as another environment's top file, will use the instance of the environment configured in `base` and ignore all other instances. In other words, the `base` top file is authoritative when defining environments. Therefore, in the example below, the `dev` section in `/srv/salt/dev/top.sls` would be completely ignored.

`/srv/salt/base/top.sls:`

```
base:
  '*':
    - common
dev:
  'webserver*dev*':
    - webserver
  'db*dev*':
    - db
```

`/srv/salt/dev/top.sls:`

```
dev:
  '10.10.100.0/24':
    - match: ipcidr
    - deployments.dev.site1
  '10.10.101.0/24':
    - match: ipcidr
    - deployments.dev.site2
```

Note: The rules below assume that the environments being discussed were not defined in the `base` top file.

2. If, for some reason, the `base` environment is not configured in the `base` environment's top file, then the other environments will be checked in alphabetical order. The first top file found to contain a section for the `base` environment wins, and the other top files' `base` sections are ignored. So, provided there is no `base` section in the `base` top file, with the below two top files the `dev` environment would win out, and the `common.centos` SLS would not be applied to CentOS hosts.

`/srv/salt/dev/top.sls:`

```
base:
  'os:Ubuntu':
    - common.ubuntu
dev:
  'webserver*dev*':
    - webserver
  'db*dev*':
    - db
```

`/srv/salt/qa/top.sls:`

```
base:
  'os:Ubuntu':
    - common.ubuntu
  'os:CentOS':
    - common.centos
qa:
  'webserver*qa*':
    - webserver
```

```
  'db*qa*':
    - db
```

3. For environments other than `base`, the top file in a given environment will be checked for a section matching the environment's name. If one is found, then it is used. Otherwise, the remaining (non-`base`) environments will be checked in alphabetical order. In the below example, the `qa` section in `/srv/salt/dev/top.sls` will be ignored, but if `/srv/salt/qa/top.sls` were cleared or removed, then the states configured for the `qa` environment in `/srv/salt/dev/top.sls` will be applied.

`/srv/salt/dev/top.sls:`

```
dev:
  'webserver*dev*':
    - webserver
  'db*dev*':
    - db
qa:
  '10.10.200.0/24':
    - match: ipcidr
    - deployments.qa.site1
  '10.10.201.0/24':
    - match: ipcidr
    - deployments.qa.site2
```

`/srv/salt/qa/top.sls:`

```
qa:
  'webserver*qa*':
    - webserver
  'db*qa*':
    - db
```

Note: When in doubt, the simplest way to configure your states is with a single top.sls in the `base` environment.

22.26.19 SLS Template Variable Reference

The template engines available to sls files and file templates come loaded with a number of context variables. These variables contain information and functions to assist in the generation of templates. See each variable below for its availability -- not all variables are available in all templating contexts.

Salt

The *salt* variable is available to abstract the salt library functions. This variable is a python dictionary containing all of the functions available to the running salt minion. It is available in all salt templates.

```
{% for file in salt['cmd.run']('ls -1 /opt/to_remove').splitlines() %}
/opt/to_remove/{{ file }}:
  file.absent
{% endfor %}
```

Opts

The *opts* variable abstracts the contents of the minion's configuration file directly to the template. The *opts* variable is a dictionary. It is available in all templates.

```
{{ opts['cachedir'] }}
```

The `config.get` function also searches for values in the *opts* dictionary.

Pillar

The *pillar* dictionary can be referenced directly, and is available in all templates:

```
{{ pillar['key'] }}
```

Using the `pillar.get` function via the *salt* variable is generally recommended since a default can be safely set in the event that the value is not available in pillar and dictionaries can be traversed directly:

```
{{ salt['pillar.get']('key', 'failover_value') }}
{{ salt['pillar.get']('stuff:more:deeper') }}
```

Grains

The *grains* dictionary makes the minion's grains directly available, and is available in all templates:

```
{{ grains['os'] }}
```

The `grains.get` function can be used to traverse deeper grains and set defaults:

```
{{ salt['grains.get']('os') }}
```

env

The *env* variable is available in only in sls files when gathering the sls from an environment.

```
{{ env }}
```

sls

The *sls* variable contains the sls reference value, and is only available in the actual SLS file (not in any files referenced in that SLS). The sls reference value is the value used to include the sls in top files or via the include option.

```
{{ sls }}
```

22.26.20 State Modules

State Modules are the components that map to actual enforcement and management of Salt states.

States are Easy to Write!

State Modules should be easy to write and straightforward. The information passed to the SLS data structures will map directly to the states modules.

Mapping the information from the SLS data is simple, this example should illustrate:

```
/etc/salt/master: # maps to "name"
  file: # maps to State module filename e.g. https://github.com/saltstack/salt/tree/develop/salt/stat
    - managed # maps to the managed function in the file State module
    - user: root # one of many options passed to the manage function
    - group: root
    - mode: 644
    - source: salt://salt/master
```

Therefore this SLS data can be directly linked to a module, function, and arguments passed to that function.

This does issue the burden, that function names, state names and function arguments should be very human readable inside state modules, since they directly define the user interface.

Keyword Arguments

Salt passes a number of keyword arguments to states when rendering them, including the environment, a unique identifier for the state, and more. Additionally, keep in mind that the requisites for a state are part of the keyword arguments. Therefore, if you need to iterate through the keyword arguments in a state, these must be considered and handled appropriately. One such example is in the `pkgrepo.managed` state, which needs to be able to handle arbitrary keyword arguments and pass them to module execution functions. An example of how these keyword arguments can be handled can be found here.

Using Custom State Modules

Place your custom state modules inside a `_states` directory within the `file_roots` specified by the master config file. These custom state modules can then be distributed in a number of ways. Custom state modules are distributed when `state.highstate` is run, or by executing the `saltutil.sync_states` or `saltutil.sync_all` functions.

Any custom states which have been synced to a minion, that are named the same as one of Salt's default set of states, will take the place of the default state with the same name. Note that a state's default name is its filename (i.e. `foo.py` becomes state `foo`), but that its name can be overridden by using a *__virtual__ function*.

Cross Calling Modules

As with Execution Modules, State Modules can also make use of the `__salt__` and `__grains__` data.

It is important to note that the real work of state management should not be done in the state module unless it is needed. A good example is the pkg state module. This module does not do any package management work, it just calls the pkg execution module. This makes the pkg state module completely generic, which is why there is only one pkg state module and many backend pkg execution modules.

On the other hand some modules will require that the logic be placed in the state module, a good example of this is the file module. But in the vast majority of cases this is not the best approach, and writing specific execution modules to do the backend work will be the optimal solution.

Return Data

A State Module must return a dict containing the following keys/values:

- **name**: The same value passed to the state as ``name''.
- **changes**: A dict describing the changes made. Each thing changed should be a key, with its value being another dict with keys called ``old'' and ``new'' containing the old/new values. For example, the pkg state's **changes** dict has one key for each package changed, with the ``old'' and ``new'' keys in its sub-dict containing the old and new versions of the package.
- **result**: A boolean value. *True* if the action was successful, otherwise *False*.
- **comment**: A string containing a summary of the result.

Test State

All states should check for and support `test` being passed in the options. This will return data about what changes would occur if the state were actually run. An example of such a check could look like this:

```
# Return comment of changes if test.
if __opts__['test']:
    ret['result'] = None
    ret['comment'] = 'State Foo will execute with param {0}'.format(bar)
    return ret
```

Make sure to test and return before performing any real actions on the minion.

Watcher Function

If the state being written should support the watch requisite then a watcher function needs to be declared. The watcher function is called whenever the watch requisite is invoked and should be generic to the behavior of the state itself.

The watcher function should accept all of the options that the normal state functions accept (as they will be passed into the watcher function).

A watcher function typically is used to execute state specific reactive behavior, for instance, the watcher for the service module restarts the named service and makes it useful for the watcher to make the service react to changes in the environment.

The watcher function also needs to return the same data that a normal state function returns.

Mod_init Interface

Some states need to execute something only once to ensure that an environment has been set up, or certain conditions global to the state behavior can be predefined. This is the realm of the mod_init interface.

A state module can have a function called **mod_init** which executes when the first state of this type is called. This interface was created primarily to improve the pkg state. When packages are installed the package metadata needs to be refreshed, but refreshing the package metadata every time a package is installed is wasteful. The mod_init

function for the pkg state sets a flag down so that the first, and only the first, package installation attempt will refresh the package database (the package database can of course be manually called to refresh via the `refresh` option in the pkg state).

The mod_init function must accept the **Low State Data** for the given executing state as an argument. The low state data is a dict and can be seen by executing the state.show_lowstate function. Then the mod_init function must return a bool. If the return value is True, then the mod_init function will not be executed again, meaning that the needed behavior has been set up. Otherwise, if the mod_init function returns False, then the function will be called the next time.

A good example of the mod_init function is found in the pkg state module:

```
def mod_init(low):
    '''
    Refresh the package database here so that it only needs to happen once
    '''
    if low['fun'] == 'installed' or low['fun'] == 'latest':
        rtag = __gen_rtag()
        if not os.path.exists(rtag):
            open(rtag, 'w+').write('')
        return True
    else:
        return False
```

The mod_init function in the pkg state accepts the low state data as `low` and then checks to see if the function being called is going to install packages, if the function is not going to install packages then there is no need to refresh the package database. Therefore if the package database is prepared to refresh, then return True and the mod_init will not be called the next time a pkg state is evaluated, otherwise return False and the mod_init will be called next time a pkg state is evaluated.

Full State Module Example

The following is a simplistic example of a full state module and function. Remember to call out to execution modules to perform all the real work. The state module should only perform ``before'' and ``after'' checks.

1. Make a custom state module by putting the code into a file at the following path: **/srv/salt/_states/my_custom_state.py**.
2. Distribute the custom state module to the minions:

```
salt '*' saltutil.sync_states
```

3. Write a new state to use the custom state by making a new state file, for instance **/srv/salt/my_custom_state.sls**.
4. Add the following SLS configuration to the file created in Step 3:

```
human_friendly_state_id:        # An arbitrary state ID declaration.
  my_custom_state:              # The custom state module name.
    - enforce_custom_thing      # The function in the custom state module.
    - name: a_value             # Maps to the ``name`` parameter in the custom function.
    - foo: Foo                  # Specify the required ``foo`` parameter.
    - bar: False                # Override the default value for the ``bar`` parameter.
```

Example state module

```
import salt.exceptions

def enforce_custom_thing(name, foo, bar=True):
    '''
    Enforce the state of a custom thing

    This state module does a custom thing. It calls out to the execution module
    ``my_custom_module`` in order to check the current system and perform any
    needed changes.

    name
        The thing to do something to
    foo
        A required argument
    bar : True
        An argument with a default value
    '''
    ret = {'name': name, 'changes': {}, 'result': False, 'comment': ''}

    # Start with basic error-checking. Do all the passed parameters make sense
    # and agree with each-other?
    if bar == True and foo.startswith('Foo'):
        raise salt.exceptions.SaltInvocationError(
            'Argument "foo" cannot start with "Foo" if argument "bar" is True.')

    # Check the current state of the system. Does anything need to change?
    current_state = __salt__['my_custom_module.current_state'](name)

    if current_state == foo:
        ret['result'] = True
        ret['comment'] = 'System already in the correct state'
        return ret

    # The state of the system does need to be changed. Check if we're running
    # in ``test=true`` mode.
    if __opts__['test'] == True:
        ret['comment'] = 'The state of "{0}" will be changed.'.format(name)
        ret['changes'] = {
            'old': current_state,
            'new': 'Description, diff, whatever of the new state',
        }

        # Return ``None`` when running with ``test=true``.
        ret['result'] = None

        return ret

    # Finally, make the actual change and return the result.
    new_state = __salt__['my_custom_module.change_state'](name, foo)

    ret['comment'] = 'The state of "{0}" was changed!'.format(name)

    ret['changes'] = {
        'old': current_state,
        'new': new_state,
```

```
    }

    ret['result'] = True

    return ret
```

22.26.21 State Management

State management, also frequently called Software Configuration Management (SCM), is a program that puts and keeps a system into a predetermined state. It installs software packages, starts or restarts services or puts configuration files in place and watches them for changes.

Having a state management system in place allows one to easily and reliably configure and manage a few servers or a few thousand servers. It allows configurations to be kept under version control.

Salt States is an extension of the Salt Modules that we discussed in the previous *remote execution* tutorial. Instead of calling one-off executions the state of a system can be easily defined and then enforced.

22.26.22 Understanding the Salt State System Components

The Salt state system is comprised of a number of components. As a user, an understanding of the SLS and renderer systems are needed. But as a developer, an understanding of Salt states and how to write the states is needed as well.

Note: States are compiled and executed only on minions that have been targeted. To execute functions directly on masters, see *runners*.

Salt SLS System

The primary system used by the Salt state system is the SLS system. SLS stands for **S**a**L**t **S**tate.

The Salt States are files which contain the information about how to configure Salt minions. The states are laid out in a directory tree and can be written in many different formats.

The contents of the files and they way they are laid out is intended to be as simple as possible while allowing for maximum flexibility. The files are laid out in states and contains information about how the minion needs to be configured.

SLS File Layout

SLS files are laid out in the Salt file server.

A simple layout can look like this:

```
top.sls
ssh.sls
sshd_config
users/init.sls
users/admin.sls
salt/master.sls
web/init.sls
```

The `top.sls` file is a key component. The `top.sls` files is used to determine which SLS files should be applied to which minions.

The rest of the files with the `.sls` extension in the above example are state files.

Files without a `.sls` extensions are seen by the Salt master as files that can be downloaded to a Salt minion.

States are translated into dot notation. For example, the `ssh.sls` file is seen as the ssh state and the `users/admin.sls` file is seen as the users.admin state.

Files named `init.sls` are translated to be the state name of the parent directory, so the `web/init.sls` file translates to the `web` state.

In Salt, everything is a file; there is no ``magic translation'' of files and file types. This means that a state file can be distributed to minions just like a plain text or binary file.

SLS Files

The Salt state files are simple sets of data. Since SLS files are just data they can be represented in a number of different ways.

The default format is YAML generated from a Jinja template. This allows for the states files to have all the language constructs of Python and the simplicity of YAML.

State files can then be complicated Jinja templates that translate down to YAML, or just plain and simple YAML files.

The State files are simply common data structures such as dictionaries and lists, constructed using a templating language such as YAML.

Here is an example of a Salt State:

```
vim:
  pkg.installed: []

salt:
  pkg.latest:
    - name: salt
  service.running:
    - names:
      - salt-master
      - salt-minion
    - require:
      - pkg: salt
    - watch:
      - file: /etc/salt/minion

/etc/salt/minion:
  file.managed:
    - source: salt://salt/minion
    - user: root
    - group: root
    - mode: 644
    - require:
      - pkg: salt
```

This short stanza will ensure that vim is installed, Salt is installed and up to date, the salt-master and salt-minion daemons are running and the Salt minion configuration file is in place. It will also ensure everything is deployed in the right order and that the Salt services are restarted when the watched file updated.

The Top File

The top file controls the mapping between minions and the states which should be applied to them.

The top file specifies which minions should have which SLS files applied and which environments they should draw those SLS files from.

The top file works by specifying environments on the top-level.

Each environment contains globs to match minions. Finally, each glob contains a list of lists of Salt states to apply to matching minions:

```
base:
  '*':
    - salt
    - users
    - users.admin
  'saltmaster.*':
    - match: pcre
    - salt.master
```

This above example uses the base environment which is built into the default Salt setup.

The base environment has two globs. First, the `*' glob contains a list of SLS files to apply to all minions.

The second glob contains a regular expression that will match all minions with an ID matching saltmaster.* and specifies that for those minions, the salt.master state should be applied.

Reloading Modules

Some Salt states require that specific packages be installed in order for the module to load. As an example the `pip` state module requires the pip package for proper name and version parsing.

In most of the common cases, Salt is clever enough to transparently reload the modules. For example, if you install a package, Salt reloads modules because some other module or state might require just that package which was installed.

On some edge-cases salt might need to be told to reload the modules. Consider the following state file which we'll call `pep8.sls`:

```
python-pip:
  cmd.run:
    - name: |
        easy_install --script-dir=/usr/bin -U pip
    - cwd: /

pep8:
  pip.installed:
    - require:
      - cmd: python-pip
```

The above example installs pip using `easy_install` from setuptools and installs pep8 using `pip`, which, as told earlier, requires pip to be installed system-wide. Let's execute this state:

```
salt-call state.sls pep8
```

The execution output would be something like:

```
----------
    State: - pip
    Name:      pep8
    Function:  installed
        Result:    False
        Comment:   State pip.installed found in sls pep8 is unavailable

        Changes:

Summary
------------
Succeeded: 1
Failed:    1
------------
Total:     2
```

If we executed the state again the output would be:

```
----------
    State: - pip
    Name:      pep8
    Function:  installed
        Result:    True
        Comment:   Package was successfully installed
        Changes:   pep8==1.4.6: Installed

Summary
------------
Succeeded: 2
Failed:    0
------------
Total:     2
```

Since we installed pip using `cmd`, Salt has no way to know that a system-wide package was installed.

On the second execution, since the required pip package was installed, the state executed correctly.

Note: Salt does not reload modules on every state run because doing so would greatly slow down state execution.

So how do we solve this *edge-case*? `reload_modules`!

`reload_modules` is a boolean option recognized by salt on **all** available states which forces salt to reload its modules once a given state finishes.

The modified state file would now be:

```
python-pip:
  cmd.run:
    - name: |
        easy_install --script-dir=/usr/bin -U pip
    - cwd: /
    - reload_modules: true

pep8:
  pip.installed:
    - require:
      - cmd: python-pip
```

Let's run it, once:

```
salt-call state.sls pep8
```

The output is:

```
----------
    State: - pip
    Name:      pep8
    Function:  installed
        Result:    True
        Comment:   Package was successfully installed
        Changes:   pep8==1.4.6: Installed

Summary
------------
Succeeded: 2
Failed:    0
------------
Total:     2
```

22.27 Full list of builtin state modules

alias	Configuration of email aliases
alternatives	Configuration of the alternatives system
apache	Apache state
apache_module	Manage Apache Modules
apt	Package management operations specific to APT- and DEB-based systems
archive	Extract an archive
artifactory	This state downloads artifacts from artifactory.
at	Configuration disposable regularly scheduled tasks for at.
augeas	Configuration management using Augeas
aws_sqs	Manage SQS Queues
blockdev	Management of Block Devices
boto_asg	Manage Autoscale Groups
boto_cloudwatch_alarm	Manage Cloudwatch alarms
boto_dynamodb	Manage DynamoDB Tables
boto_elasticache	Manage Elasticache
boto_elb	Manage ELBs
boto_iam_role	Manage IAM roles
boto_lc	Manage Launch Configurations
boto_route53	Manage Route53 records
boto_secgroup	Manage Security Groups
boto_sns	Manage SNS Topics
boto_sqs	Manage SQS Queues
chef	Execute Chef client runs
cloud	Using states instead of maps to deploy clouds
cmd	Execution of arbitrary commands
composer	Installation of Composer Packages
cron	Management of cron, the Unix command scheduler

Continued on next page

Table 22.13 -- continued from previous page

cyg	Installation of Cygwin packages.
ddns	Dynamic DNS updates
debconfmod	Management of debconf selections
disk	Disk monitoring state
dockerio	Manage Docker containers
drac	Management of Dell DRAC
environ	Support for getting and setting the environment variables of the current salt process.
eselect	Management of Gentoo configuration using eselect
event	Send events through Salt's event system during state runs
file	Operations on regular files, special files, directories, and symlinks
gem	Installation of Ruby modules packaged as gems
git	Interaction with Git repositories
glusterfs	Manage glusterfs pool.
gnomedesktop	Configuration of the GNOME desktop
grafana	Manage Grafana Dashboards
grains	Manage grains on the minion
group	Management of user groups
hg	Interaction with Mercurial repositories
hipchat	Send a message to Hipchat
host	Management of addresses and names in hosts file
htpasswd	Support for htpasswd module
http	HTTP monitoring states
incron	Management of incron, the inotify cron
influxdb_database	Management of InfluxDB databases
influxdb_user	Management of InfluxDB users
ini_manage	Manage ini files
ipset	Management of ipsets
iptables	Management of iptables
jboss7	Manage JBoss 7 Application Server via CLI interface
keyboard	Management of keyboard layouts
keystone	Management of Keystone users
kmod	Loading and unloading of kernel modules
layman	Management of Gentoo Overlays using layman
libvirt	Manage libvirt certificates
linux_acl	Linux File Access Control Lists
locale	Management of languages/locales
lvm	Management of Linux logical volumes
lvs_server	Management of LVS (Linux Virtual Server) Real Server
lvs_service	Management of LVS (Linux Virtual Server) Service
lxc	Manage Linux Containers
makeconf	Management of Gentoo make.conf
mdadm	Managing software RAID with mdadm
memcached	States for Management of Memcached Keys
modjk	State to control Apache modjk
modjk_worker	Manage modjk workers
module	Execution of Salt modules from within states
mongodb_database	Management of Mongodb databases
mongodb_user	Management of Mongodb users
mount	Mounting of filesystems
mysql_database	Management of MySQL databases (schemas)

Continued on next page

Table 22.13 -- continued from previous page

mysql_grants	Management of MySQL grants (user permissions)
mysql_query	Execution of MySQL queries
mysql_user	Management of MySQL users
network	Configuration of network interfaces
nftables	Management of nftables
npm	Installation of NPM Packages
ntp	Management of NTP servers
openstack_config	Manage OpenStack configuration file settings.
pagerduty	Create an Event in PagerDuty
pecl	Installation of PHP Extensions Using pecl
pip_state	Installation of Python Packages Using pip
pkg	Installation of packages using OS package managers such as yum or apt-get
pkgng	Manage package remote repo using FreeBSD pkgng
pkgrepo	Management of APT/YUM package repos
portage_config	Management of Portage package configuration on Gentoo
ports	Manage software from FreeBSD ports
postgres_database	Management of PostgreSQL databases
postgres_extension	Management of PostgreSQL extensions (e.g.: postgis)
postgres_group	Management of PostgreSQL groups (roles)
postgres_schema	Management of PostgreSQL schemas
postgres_user	Management of PostgreSQL users (roles)
powerpath	Powerpath configuration support
process	Process Management
pyenv	Managing python installations with pyenv
pyrax_queues	Manage Rackspace Queues
quota	Management of POSIX Quotas
rabbitmq_cluster	Manage RabbitMQ Clusters
rabbitmq_plugin	Manage RabbitMQ Plugins
rabbitmq_policy	Manage RabbitMQ Policies
rabbitmq_user	Manage RabbitMQ Users
rabbitmq_vhost	Manage RabbitMQ Virtual Hosts
rbenv	Managing Ruby installations with rbenv
rdp	Manage RDP Service on Windows servers
redismod	Management of Redis server
reg	Manage the registry on Windows
rvm	Managing Ruby installations and gemsets with Ruby Version Manager (RVM)
saltmod	Control the Salt command interface
schedule	Management of the Salt scheduler
selinux	Management of SELinux rules
serverdensity_device	Monitor Server with Server Density
service	Starting or restarting of services and daemons
slack	Send a message to Slack
smtp	Sending Messages via SMTP
splunk_search	Splunk Search State Module
ssh_auth	Control of entries in SSH authorized_key files
ssh_known_hosts	Control of SSH known_hosts entries
stateconf	Stateconf System
status	Minion status monitoring
supervisord	Interaction with the Supervisor daemon
svn	Manage SVN repositories

Continued on next page

Table 22.13 -- continued from previous page

sysctl	Configuration of the Linux kernel using sysctl
syslog_ng	State module for syslog_ng
sysrc	
test	Test States
timezone	Management of timezones
tomcat	This state uses the manager webapp to manage Apache tomcat webapps
user	Management of user accounts
vbox_guest	VirtualBox Guest Additions installer state
virtualenv_mod	Setup of Python virtualenv sandboxes
win_dns_client	Module for configuring DNS Client on Windows systems
win_firewall	State for configuring Windows Firewall
win_network	Configuration of network interfaces on Windows hosts
win_path	Manage the Windows System PATH
win_servermanager	Manage Windows features via the ServerManager powershell module
win_system	Management of Windows system information
win_update	Management of the windows update agent
winrepo	Manage Windows Package Repository
xmpp	Sending Messages over XMPP
zcbuildout	Management of zc.buildout
zk_concurrency	Control concurrency of steps within state execution using zookeeper

22.27.1 salt.states.alias

Configuration of email aliases

The mail aliases file can be managed to contain definitions for specific email aliases:

```
username:
  alias.present:
    - target: user@example.com
```

```
thomas:
  alias.present:
    - target: thomas@example.com
```

salt.states.alias.**absent**(*name*)

Ensure that the named alias is absent

name The alias to remove

salt.states.alias.**present**(*name*, *target*)

Ensures that the named alias is present with the given target or list of targets. If the alias exists but the target differs from the previous entry, the target(s) will be overwritten. If the alias does not exist, the alias will be created.

name The local user/address to assign an alias to

target The forwarding address

22.27.2 salt.states.alternatives

Configuration of the alternatives system

Control the alternatives system

```
{% set my_hadoop_conf = '/opt/hadoop/conf' %}

{{ my_hadoop_conf }}:
  file.directory

hadoop-0.20-conf:
  alternatives.install:
    - name: hadoop-0.20-conf
    - link: /etc/hadoop-0.20/conf
    - path: {{ my_hadoop_conf }}
    - priority: 30
    - require:
      - file: {{ my_hadoop_conf }}

hadoop-0.20-conf:
  alternatives.remove:
      - name: hadoop-0.20-conf
      - path: {{ my_hadoop_conf }}
```

salt.states.alternatives.**auto**(*name*)

New in version 0.17.0.

Instruct alternatives to use the highest priority path for <name>

name is the master name for this link group (e.g. pager)

salt.states.alternatives.**install**(*name*, *link*, *path*, *priority*)

Install new alternative for defined <name>

name is the master name for this link group (e.g. pager)

link is the symlink pointing to /etc/alternatives/<name>. (e.g. /usr/bin/pager)

path is the location of the new alternative target. NB: This file / directory must already exist. (e.g. /usr/bin/less)

priority is an integer; options with higher numbers have higher priority in automatic mode.

salt.states.alternatives.**remove**(*name*, *path*)

Removes installed alternative for defined <name> and <path> or fallback to default alternative, if some defined before.

name is the master name for this link group (e.g. pager)

path is the location of one of the alternative target files. (e.g. /usr/bin/less)

salt.states.alternatives.**set_**(*name*, *path*)

New in version 0.17.0.

Sets alternative for <name> to <path>, if <path> is defined as an alternative for <name>.

name is the master name for this link group (e.g. pager)

path is the location of one of the alternative target files. (e.g. /usr/bin/less)

22.27.3 salt.states.apache

Apache state

New in version 2014.7.0.

Allows for inputting a yaml dictionary into a file for apache configuration files.

The variable `this` is special and signifies what should be included with the above word between angle brackets (<>).

```
/etc/httpd/conf.d/website.com.conf:
  apache.configfile:
    - config:
      - VirtualHost:
          this: '*:80'
          ServerName:
            - website.com
          ServerAlias:
            - www.website.com
            - dev.website.com
          ErrorLog: logs/website.com-error_log
          CustomLog: logs/website.com-access_log combined
          DocumentRoot: /var/www/vhosts/website.com
          Directory:
            this: /var/www/vhosts/website.com
            Order: Deny,Allow
            Deny from: all
            Allow from:
              - 127.0.0.1
              - 192.168.100.0/24
            Options:
              - +Indexes
              - FollowSymlinks
            AllowOverride: All
```

salt.states.apache.**configfile**(*name*, *config*)

22.27.4 salt.states.apache_module

Manage Apache Modules

New in version 2014.7.0.

Enable and disable apache modules.

```
Enable cgi module:
    apache_module.enable:
        - name: cgi

Disable cgi module:
    apache_module.disable:
        - name: cgi
```

salt.states.apache_module.**disable**(*name*)

Ensure an Apache module is disabled.

name Name of the Apache module

salt.states.apache_module.**enable**(*name*)

Ensure an Apache module is enabled.

name Name of the Apache module

22.27.5 salt.states.apt

Package management operations specific to APT- and DEB-based systems

salt.states.apt.**held**(*name*)

Set package in `hold' state, meaning it will not be upgraded.

name The name of the package, e.g., `tmux'

22.27.6 salt.states.archive

Extract an archive

New in version 2014.1.0.

salt.states.archive.**extracted**(*name*, *source*, *archive_format*, *archive_user=None*, *tar_options=None*, *source_hash=None*, *if_missing=None*, *keep=False*)

New in version 2014.1.0.

State that make sure an archive is extracted in a directory. The downloaded archive is erased if successfully extracted. The archive is downloaded only if necessary.

Note: If `if_missing` is not defined, this state will check for `name` instead. If `name` exists, it will assume the archive was previously extracted successfully and will not extract it again.

```
graylog2-server:
  archive.extracted:
    - name: /opt/
    - source: https://github.com/downloads/Graylog2/graylog2-server/graylog2-server-0.9.6p1.tar.
    - source_hash: md5=499ae16dcae71eeb7c3a30c75ea7a1a6
    - tar_options: J
    - archive_format: tar
    - if_missing: /opt/graylog2-server-0.9.6p1/
```

```
graylog2-server:
  archive.extracted:
    - name: /opt/
    - source: https://github.com/downloads/Graylog2/graylog2-server/graylog2-server-0.9.6p1.tar.
    - source_hash: md5=499ae16dcae71eeb7c3a30c75ea7a1a6
    - archive_format: tar
    - tar_options: v
    - if_missing: /opt/graylog2-server-0.9.6p1/
```

name Directory name where to extract the archive

source Archive source, same syntax as file.managed source argument.

source_hash Hash of source file, or file with list of hash-to-file mappings. It uses the same syntax as the file.managed source_hash argument.

archive_format tar, zip or rar

archive_user: user to extract files as

if_missing Some archives, such as tar, extract themselves in a subfolder. This directive can be used to validate if the archive had been previously extracted.

tar_options Required if used with `archive_format: tar`, otherwise optional. It needs to be the tar argument specific to the archive being extracted, such as `J' for LZMA or `v' to verbosely list files processed. Using this option means that the tar executable on the target will be used, which is less platform independent. Main operators like -x, --extract, --get, -c and -f/--file **should not be used** here. If `archive_format` is `zip` or `rar` and this option is not set, then the Python tarfile module is used. The tarfile module supports gzip and bz2 in Python 2.

keep Keep the archive in the minion's cache

22.27.7 salt.states.jboss7

This state downloads artifacts from artifactory.

`salt.states.artifactory.`**`downloaded`**(*name*, *artifact*, *target_dir='/tmp'*, *target_file=None*)

Ensures that the artifact from artifactory exists at given location. If it doesn't exist, then it will be downloaded. It it already exists then the checksum of existing file is checked against checksum in artifactory. If it is different then the step will fail.

artifact:

Details of the artifact to be downloaded from artifactory.

- artifactory_url: URL of the artifactory instance
- repository: Repository in artifactory
- artifact_id: Artifact ID
- group_id: Group ID
- packaging: Packaging
- version: Version

target_dir: Directory where the artifact should be downloaded. By default it is downloaded to /tmp directory.

target_file: Target file to download artifact to. By default file name is resolved by artifactory.

Example:: Download artifact to a specific file:

```
jboss_module_downloaded:
  artifactory.downloaded:
   - artifact:
        artifactory_url: http://artifactory.intranet.company.com/artifactory
        repository: 'libs-release-local'
        artifact_id: 'module'
        group_id: 'com.company.module'
        packaging: 'jar'
        version: '1.0'
   - target_file: /opt/jboss7/modules/com/company/lib/module.jar
```

Download artifact to the folder (automatically resolves file name):

```
jboss_module_downloaded:
  artifactory.downloaded:
   - artifact:
        artifactory_url: http://artifactory.intranet.company.com/artifactory
        repository: 'libs-release-local'
        artifact_id: 'module'
        group_id: 'com.company.module'
        packaging: 'jar'
```

```
            version: '1.0'
        - target_dir: /opt/jboss7/modules/com/company/lib
```

22.27.8 salt.states.at

Configuration disposable regularly scheduled tasks for at.

The at state can be add disposable regularly scheduled tasks for your system.

`salt.states.at.`**`absent`**(*name*, *jobid=None*, ***kwargs*)

Remove a job from queue The `kwargs' can include hour. minute. day. month. year

limit Target range

tag Job's tag

runas Runs user-specified jobs

```
    example1:
      at.absent:
        - limit: all
```

```
    example2:
      at.absent:
        - limit: all
        - year: 13
```

```
    example3:
      at.absent:
        - limit: all
        - tag: rose
        - runas: jim
```

```
    example4:
      at.absent:
        - limit: all
        - tag: rose
        - day: 13
        - hour: 16
```

`salt.states.at.`**`present`**(*name*, *timespec*, *tag=None*, *user=None*, *job=None*)

Add a job to queue.

job Command to run.

timespec The `timespec' follows the format documented in the at(1) manpage.

tag Make a tag for the job.

user The user to run the at job

New in version 2014.1.4.

```
    rose:
      at.present:
        - job: 'echo "I love saltstack" > love'
        - timespec: '9:09 11/09/13'
        - tag: love
        - user: jam
```

22.27.9 salt.states.augeas

Configuration management using Augeas

New in version 0.17.0.

This state requires the `augeas` Python module.

Augeas can be used to manage configuration files.

Warning: Minimal installations of Debian and Ubuntu have been seen to have packaging bugs with python-augeas, causing the augeas module to fail to import. If the minion has the augeas module installed, and the state fails with a comment saying that the state is unavailable, first restart the salt-minion service. If the problem persists past that, the following command can be run from the master to determine what is causing the import to fail:

```
salt minion-id cmd.run 'python -c "from augeas import Augeas"'
```

For affected Debian/Ubuntu hosts, installing `libpython2.7` has been known to resolve the issue.

`salt.states.augeas.`**`change`**(*name*, *context=None*, *changes=None*, *lens=None*, ***kwargs*)

New in version 2014.7.0.

This state replaces `setvalue()`.

Issue changes to Augeas, optionally for a specific context, with a specific lens.

name State name

context The context to use. Set this to a file path, prefixed by `/files`, to avoid redundancy, e.g.:

```
redis-conf:
  augeas.change:
    - context: /files/etc/redis/redis.conf
    - changes:
      - set bind 0.0.0.0
      - set maxmemory 1G
```

changes List of changes that are issued to Augeas. Available commands are `set`, `mv`/`move`, `ins`/`insert`, and `rm`/`remove`.

lens The lens to use, needs to be suffixed with *.lns*, e.g.: *Nginx.lns*. See the list of stock lenses shipped with Augeas.

Usage examples:

Set the `bind` parameter in `/etc/redis/redis.conf`:

```
redis-conf:
  augeas.change:
    - changes:
      - set /files/etc/redis/redis.conf/bind 0.0.0.0
```

Note: Use the `context` parameter to specify the file you want to manipulate. This way you don't have to include this in the changes every time:

```
redis-conf:
  augeas.change:
    - context: /files/etc/redis/redis.conf
    - changes:
      - set bind 0.0.0.0
```

```
        - set databases 4
        - set maxmemory 1G
```

Augeas is aware of a lot of common configuration files and their syntax. It knows the difference between for example ini and yaml files, but also files with very specific syntax, like the hosts file. This is done with *lenses*, which provide mappings between the Augeas tree and the file.

There are many preconfigured lenses that come with Augeas by default, and they specify the common locations for configuration files. So most of the time Augeas will know how to manipulate a file. In the event that you need to manipulate a file that Augeas doesn't know about, you can specify the lens to use like this:

```
redis-conf:
  augeas.change:
    - lens: redis
    - context: /files/etc/redis/redis.conf
    - changes:
      - set bind 0.0.0.0
```

Note: Even though Augeas knows that `/etc/redis/redis.conf` is a Redis configuration file and knows how to parse it, it is recommended to specify the lens anyway. This is because by default, Augeas loads all known lenses and their associated file paths. All these files are parsed when Augeas is loaded, which can take some time. When specifying a lens, Augeas is loaded with only that lens, which speeds things up quite a bit.

A more complex example, this adds an entry to the services file for Zabbix, and removes an obsolete service:

```
zabbix-service:
  augeas.change:
    - lens: services
    - context: /files/etc/services
    - changes:
      - ins service-name after service-name[last()]
      - set service-name[last()] zabbix-agent
      - set service-name[. = 'zabbix-agent']/#comment "Zabbix Agent service"
      - set service-name[. = 'zabbix-agent']/port 10050
      - set service-name[. = 'zabbix-agent']/protocol tcp
      - rm service-name[. = 'im-obsolete']
    - unless: grep "zabbix-agent" /etc/services
```

Warning: Don't forget the `unless` here, otherwise a new entry will be added every time this state is run.

22.27.10 salt.states.aws_sqs

Manage SQS Queues

Create and destroy SQS queues. Be aware that this interacts with Amazon's services, and so may incur charges.

This module uses the awscli tool provided by Amazon. This can be downloaded from pip. Also check the documentation for awscli for configuration information.

```
myqueue:
    aws_sqs.exists:
        - region: eu-west-1
```

salt.states.aws_sqs.**absent**(*name*, *region*, *user=None*, *opts=False*)
Remove the named SQS queue if it exists.

name Name of the SQS queue.

region Region to remove the queue from

user Name of the user performing the SQS operations

opts Include additional arguments and options to the aws command line

salt.states.aws_sqs.**exists**(*name*, *region*, *user=None*, *opts=False*)
Ensure the SQS queue exists.

name Name of the SQS queue.

region Region to create the queue

user Name of the user performing the SQS operations

opts Include additional arguments and options to the aws command line

22.27.11 salt.states.blockdev

Management of Block Devices

A state module to manage blockdevices

```
/dev/sda:
  blockdev.tuned:
    - read-only: True

master-data:
  blockdev.tuned::
    - name : /dev/vg/master-data
    - read-only: True
    - read-ahead: 1024
```

New in version 2014.7.0.

salt.states.blockdev.**formatted**(*name*, *fs_type='ext4'*, ***kwargs*)
Manage filesystems of partitions.

name The name of the block device

fs_type The filesystem it should be formatted as

salt.states.blockdev.**tuned**(*name*, ***kwargs*)
Manage options of block device

name The name of the block device

opts:

- **read-ahead** Read-ahead buffer size
- **filesystem-read-ahead** Filesystem Read-ahead buffer size
- **read-only** Set Read-Only
- **read-write** Set Read-Write

22.27.12 salt.states.boto_asg

Manage Autoscale Groups

New in version 2014.7.0.

Create and destroy autoscale groups. Be aware that this interacts with Amazon's services, and so may incur charges.

This module uses boto, which can be installed via package, or pip.

This module accepts explicit autoscale credentials but can also utilize IAM roles assigned to the instance trough Instance Profiles. Dynamic credentials are then automatically obtained from AWS API and no further configuration is necessary. More Information available at:

```
http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/iam-roles-for-amazon-ec2.html
```

If IAM roles are not used you need to specify them either in a pillar or in the minion's config file:

```
asg.keyid: GKTADJGHEIQSXMKKRBJ08H
asg.key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs
```

It's also possible to specify key, keyid and region via a profile, either as a passed in dict, or as a string to pull from pillars or minion config:

```
myprofile:
    keyid: GKTADJGHEIQSXMKKRBJ08H
    key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs
    region: us-east-1
```

```
Ensure myasg exists:
  boto_asg.present:
    - name: myasg
    - launch_config_name: mylc
    - availability_zones:
      - us-east-1a
      - us-east-1b
    - min_size: 1
    - max_size: 1
    - desired_capacity: 1
    - load_balancers:
      - myelb
    - suspended_processes:
        - AddToLoadBalancer
        - AlarmNotification
    - scaling_policies
        ----------
        - adjustment_type: ChangeInCapacity
        - as_name: api-production-iad
        - cooldown: 1800
        - min_adjustment_step: None
        - name: ScaleDown
        - scaling_adjustment: -1
    - region: us-east-1
    - keyid: GKTADJGHEIQSXMKKRBJ08H
    - key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs
```

```
# Using a profile from pillars.
Ensure myasg exists:
  boto_asg.present:
    - name: myasg
    - launch_config_name: mylc
    - availability_zones:
      - us-east-1a
      - us-east-1b
    - min_size: 1
    - max_size: 1
    - desired_capacity: 1
    - load_balancers:
      - myelb
    - profile: myprofile

# Passing in a profile.
Ensure myasg exists:
  boto_asg.present:
    - name: myasg
    - launch_config_name: mylc
    - availability_zones:
      - us-east-1a
      - us-east-1b
    - min_size: 1
    - max_size: 1
    - desired_capacity: 1
    - load_balancers:
      - myelb
    - profile:
        keyid: GKTADJGHEIQSXMKKRBJ08H
        key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs
        region: us-east-1

# Deleting an autoscale group with running instances.
Ensure myasg is deleted:
  boto_asg.absent:
    - name: myasg
    # If instances exist, we must force the deletion of the asg.
    - force: True
```

It's possible to specify cloudwatch alarms that will be setup along with the ASG. Note the alarm name will be the name attribute defined, plus the ASG resource name.

```
Ensure myasg exists:
  boto_asg.present:
    - name: myasg
    - launch_config_name: mylc
    - availability_zones:
      - us-east-1a
      - us-east-1b
    - min_size: 1
    - max_size: 1
    - desired_capacity: 1
    - load_balancers:
      - myelb
    - profile: myprofile
    - alarms:
```

```
        CPU:
          name: 'ASG CPU **MANAGED BY SALT**'
          attributes:
            metric: CPUUtilization
            namespace: AWS/EC2
            statistic: Average
            comparison: '>='
            threshold: 65.0
            period: 60
            evaluation_periods: 30
            unit: null
            description: 'ASG CPU'
            alarm_actions: [ 'arn:aws:sns:us-east-1:12345:myalarm' ]
            insufficient_data_actions: []
            ok_actions: [ 'arn:aws:sns:us-east-1:12345:myalarm' ]
```

You can also use alarms from pillars, and override values from the pillar alarms by setting overrides on the resource. Note that `boto_asg_alarms' will be used as a default value for all resources, if defined and can be used to ensure alarms are always set for an ASG resource.

Setting the alarms in a pillar:

```
my_asg_alarm:
  CPU:
    name: 'ASG CPU **MANAGED BY SALT**'
    attributes:
      metric: CPUUtilization
      namespace: AWS/EC2
      statistic: Average
      comparison: '>='
      threshold: 65.0
      period: 60
      evaluation_periods: 30
      unit: null
      description: 'ASG CPU'
      alarm_actions: [ 'arn:aws:sns:us-east-1:12345:myalarm' ]
      insufficient_data_actions: []
      ok_actions: [ 'arn:aws:sns:us-east-1:12345:myalarm' ]
```

Overriding the alarm values on the resource:

```
Ensure myasg exists:
  boto_asg.present:
    - name: myasg
    - launch_config_name: mylc
    - availability_zones:
      - us-east-1a
      - us-east-1b
    - min_size: 1
    - max_size: 1
    - desired_capacity: 1
    - load_balancers:
      - myelb
    - profile: myprofile
    - alarms_from_pillar: my_asg_alarm
    # override CPU:attributes:threshold
    - alarms:
```

```
        CPU:
          attributes:
            threshold: 50.0
```

salt.states.boto_asg.**absent**(*name*, *force=False*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Ensure the named autoscale group is deleted.

name Name of the autoscale group.

force Force deletion of autoscale group.

region The region to connect to.

key Secret key to be used.

keyid Access key to be used.

profile A dict with region, key and keyid, or a pillar key (string) that contains a dict with region, key and keyid.

salt.states.boto_asg.**present**(*name*, *launch_config_name*, *availability_zones*, *min_size*, *max_size*, *launch_config=None*, *desired_capacity=None*, *load_balancers=None*, *default_cooldown=None*, *health_check_type=None*, *health_check_period=None*, *placement_group=None*, *vpc_zone_identifier=None*, *tags=None*, *termination_policies=None*, *suspended_processes=None*, *scaling_policies=None*, *alarms=None*, *alarms_from_pillar='boto_asg_alarms'*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Ensure the autoscale group exists.

name Name of the autoscale group.

launch_config_name Name of the launch config to use for the group. Or, if launch_config is specified, this will be the launch config name's prefix. (see below)

launch_config A dictionary of launch config attributes. If specified, a launch config will be used or created, matching this set of attributes, and the autoscale group will be set to use that launch config. The launch config name will be the launch_config_name followed by a hyphen followed by a hash of the launch_config dict contents.

availability_zones List of availability zones for the group.

min_size Minimum size of the group.

max_size Maximum size of the group.

desired_capacity The desired capacity of the group.

load_balancers List of load balancers for the group. Once set this can not be updated (Amazon restriction).

default_cooldown Number of seconds after a Scaling Activity completes before any further scaling activities can start.

health_check_type The service you want the health status from, Amazon EC2 or Elastic Load Balancer (EC2 or ELB).

health_check_period Length of time in seconds after a new EC2 instance comes into service that Auto Scaling starts checking its health.

placement_group Physical location of your cluster placement group created in Amazon EC2. Once set this can not be updated (Amazon restriction).

vpc_zone_identifier A list of the subnet identifiers of the Virtual Private Cloud.

tags

A list of tags. Example:

- key: `key' value: `value' propagate_at_launch: true

termination_policies A list of termination policies. Valid values are: "OldestInstance", "NewestInstance", "OldestLaunchConfiguration", "ClosestToNextInstanceHour", "Default". If no value is specified, the "Default" value is used.

suspended_processes List of processes to be suspended. see http://docs.aws.amazon.com/AutoScaling/latest/DeveloperGuide/U

scaling_policies List of scaling policies. Each policy is a dict of key-values described by http://boto.readthedocs.org/en/latest/ref/autoscale.html#boto.ec2.autoscale.policy.ScalingPolicy

alarms: a dictionary of name->boto_cloudwatch_alarm sections to be associated with this ASG. All attributes should be specified except for dimension which will be automatically set to this ASG. See the boto_cloudwatch_alarm state for information about these attributes.

alarms_from_pillar: name of pillar dict that contains alarm settings. Alarms defined for this specific state will override those from pillar.

region The region to connect to.

key Secret key to be used.

keyid Access key to be used.

profile A dict with region, key and keyid, or a pillar key (string) that contains a dict with region, key and keyid.

22.27.13 salt.states.boto_cloudwatch_alarm

Manage Cloudwatch alarms

New in version 2014.7.0.

Create and destroy cloudwatch alarms. Be aware that this interacts with Amazon's services, and so may incur charges.

This module uses boto, which can be installed via package, or pip.

This module accepts explicit credentials but can also utilize IAM roles assigned to the instance trough Instance Profiles. Dynamic credentials are then automatically obtained from AWS API and no further configuration is necessary. More Information available at:

http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/iam-roles-for-amazon-ec2.html

If IAM roles are not used you need to specify them either in a pillar or in the minion's config file:

```
cloudwatch.keyid: GKTADJGHEIQSXMKKRBJ08H
cloudwatch.key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs
```

It's also possible to specify key, keyid and region via a profile, either as a passed in dict, or as a string to pull from pillars or minion config:

```
myprofile:
  keyid: GKTADJGHEIQSXMKKRBJ08H
  key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs
  region: us-east-1
```

```
my test alarm:
  boto_cloudwatch_alarm.present:
    - name: my test alarm
    - attributes:
        metric: ApproximateNumberOfMessagesVisible
        namespace: AWS/SQS
        statistic: Average
        comparison: ">="
        threshold: 20000.0
        period: 60
        evaluation_periods: 1
        description: test alarm via salt
        dimensions:
          QueueName:
            - the-sqs-queue-name
        alarm_actions:
          - arn:aws:sns:us-east-1:1111111:myalerting-action
```

salt.states.boto_cloudwatch_alarm.**absent**(*name*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Ensure the named cloudwatch alarm is deleted.

name Name of the alarm.

region Region to connect to.

key Secret key to be used.

keyid Access key to be used.

profile A dict with region, key and keyid, or a pillar key (string) that contains a dict with region, key and keyid.

salt.states.boto_cloudwatch_alarm.**present**(*name*, *attributes*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Ensure the cloudwatch alarm exists.

name Name of the alarm

attributes A dict of key/value cloudwatch alarm attributes.

region Region to connect to.

key Secret key to be used.

keyid Access key to be used.

profile A dict with region, key and keyid, or a pillar key (string) that contains a dict with region, key and keyid.

22.27.14 salt.states.boto_dynamodb

Manage DynamoDB Tables

New in version 2015.5.0.

Create and destroy DynamoDB tables. Be aware that this interacts with Amazon's services, and so may incur charges.

This module uses `boto`, which can be installed via package, or pip.

This module accepts explicit DynamoDB credentials but can also utilize IAM roles assigned to the instance through Instance Profiles. Dynamic credentials are then automatically obtained from AWS API and no further configuration is necessary. More information available here.

If IAM roles are not used you need to specify them either in a pillar file or in the minion's config file:

```
keyid: GKTADJGHEIQSXMKKRBJ08H
key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs
region: us-east-1
```

It's also possible to specify `key`, `keyid` and `region` via a profile, either passed in as a dict, or as a string to pull from pillars or minion config:

```
myprofile:
    keyid: GKTADJGHEIQSXMKKRBJ08H
    key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs
        region: us-east-1
```

```
```

Ensure DynamoDB table does not exist:

boto_dynamodb.absent:

- table_name: new_table
- keyid: GKTADJGHEIQSXMKKRBJ08H
- key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs
- region: us-east-1

Ensure DynamoDB table exists:

boto_dynamodb.present:

- table_name: new_table
- read_capacity_units: 1
- write_capacity_units: 2
- hash_key: primary_id
- hash_key_data_type: N
- range_key: start_timestamp
- range_key_data_type: N
- keyid: GKTADJGHEIQSXMKKRBJ08H
- key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs
- region: us-east-1
- **local_indexes**:
  - **index**:
    - name: ``primary_id_end_timestamp_index''
    - hash_key: primary_id
    - hash_key_data_type: N

        * range_key: end_timestamp
        * range_key_data_type: N
    - **global_indexes**:
        - **index**:
            * name: ``name_end_timestamp_index''
            * hash_key: name
            * hash_key_data_type: S
            * range_key: end_timestamp
            * range_key_data_type: N
            * read_capacity_units: 3
            * write_capacity_units: 4

salt.states.boto_dynamodb.**absent**(*table_name*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Ensure the DynamoDB table does not exist.

table_name Name of the DynamoDB table.

region Region to connect to.

key Secret key to be used.

keyid Access key to be used.

profile A dict with region, key and keyid, or a pillar key (string) that contains a dict with region, key and keyid.

salt.states.boto_dynamodb.**present**(*table_name*, *region=None*, *key=None*, *keyid=None*, *profile=None*, *read_capacity_units=None*, *write_capacity_units=None*, *hash_key=None*, *hash_key_data_type=None*, *range_key=None*, *range_key_data_type=None*, *local_indexes=None*, *global_indexes=None*)

Ensure the DynamoDB table exists. Note: all properties of the table can only be set during table creation. Adding or changing indexes or key schema cannot be done after table creation

table_name Name of the DynamoDB table

region Region to connect to.

key Secret key to be used.

keyid Access key to be used.

profile A dict with region, key and keyid, or a pillar key (string) that contains a dict with region, key and keyid.

read_capacity_units The read throughput for this table

write_capacity_units The write throughput for this table

hash_key The name of the attribute that will be used as the hash key for this table

hash_key_data_type The DynamoDB datatype of the hash key

range_key The name of the attribute that will be used as the range key for this table

range_key_data_type The DynamoDB datatype of the range key

local_indexes The local indexes you would like to create

global_indexes The local indexes you would like to create

22.27.15 salt.states.boto_elasticache

Manage Elasticache

New in version 2014.7.0.

Create, destroy and update Elasticache clusters. Be aware that this interacts with Amazon's services, and so may incur charges.

Note: This module currently only supports creation and deletion of elasticache resources and will not modify clusters when their configuration changes in your state files.

This module uses `boto`, which can be installed via package, or pip.

This module accepts explicit elasticache credentials but can also utilize IAM roles assigned to the instance through Instance Profiles. Dynamic credentials are then automatically obtained from AWS API and no further configuration is necessary. More information available here.

If IAM roles are not used you need to specify them either in a pillar file or in the minion's config file:

```
elasticache.keyid: GKTADJGHEIQSXMKKRBJ08H
elasticache.key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs
```

It's also possible to specify `key`, `keyid` and `region` via a profile, either passed in as a dict, or as a string to pull from pillars or minion config:

```
myprofile:
  keyid: GKTADJGHEIQSXMKKRBJ08H
  key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs
    region: us-east-1
```

```
Ensure myelasticache exists:
  boto_elasticache.present:
    - name: myelasticache
    - engine: redis
    - cache_node_type: cache.t1.micro
    - num_cache_nodes: 1
    - notification_topic_arn: arn:aws:sns:us-east-1:879879:my-sns-topic
    - region: us-east-1
    - keyid: GKTADJGHEIQSXMKKRBJ08H
    - key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs

# Using a profile from pillars
Ensure myelasticache exists:
  boto_elasticache.present:
    - name: myelasticache
    - engine: redis
    - cache_node_type: cache.t1.micro
    - num_cache_nodes: 1
    - notification_topic_arn: arn:aws:sns:us-east-1:879879:my-sns-topic
    - region: us-east-1
    - profile: myprofile
```

```
# Passing in a profile
Ensure myelasticache exists:
  boto_elasticache.present:
    - name: myelasticache
    - engine: redis
    - cache_node_type: cache.t1.micro
    - num_cache_nodes: 1
    - notification_topic_arn: arn:aws:sns:us-east-1:879879:my-sns-topic
    - region: us-east-1
    - profile:
        keyid: GKTADJGHEIQSXMKKRBJ08H
        key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs
```

salt.states.boto_elasticache.**absent**(*name*, *wait=True*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Ensure the named elasticache cluster is deleted.

name Name of the cache cluster.

wait Boolean. Wait for confirmation from boto that the cluster is in the deleting state.

region Region to connect to.

key Secret key to be used.

keyid Access key to be used.

profile A dict with region, key and keyid, or a pillar key (string) that contains a dict with region, key and keyid.

salt.states.boto_elasticache.**present**(*name*, *engine*, *cache_node_type*, *num_cache_nodes=1*, *preferred_availability_zone=None*, *port=None*, *cache_parameter_group_name=None*, *cache_security_group_names=None*, *replication_group_id=None*, *auto_minor_version_upgrade=True*, *security_group_ids=None*, *cache_subnet_group_name=None*, *engine_version=None*, *notification_topic_arn=None*, *preferred_maintenance_window=None*, *wait=True*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Ensure the cache cluster exists.

name Name of the cache cluster (cache cluster id).

engine The name of the cache engine to be used for this cache cluster. Valid values are memcached or redis.

cache_node_type The compute and memory capacity of the nodes in the cache cluster. cache.t1.micro, cache.m1.small, etc. See: http://boto.readthedocs.org/en/latest/ref/elasticache.html#boto.elasticache.layer1.ElastiCacheConnection.create_cache_cl

num_cache_nodes The number of cache nodes that the cache cluster will have.

preferred_availability_zone The EC2 Availability Zone in which the cache cluster will be created. All cache nodes belonging to a cache cluster are placed in the preferred availability zone.

port The port number on which each of the cache nodes will accept connections.

cache_parameter_group_name The name of the cache parameter group to associate with this cache cluster. If this argument is omitted, the default cache parameter group for the specified engine will be used.

cache_security_group_names A list of cache security group names to associate with this cache cluster. Use this parameter only when you are creating a cluster outside of a VPC.

replication_group_id The replication group to which this cache cluster should belong. If this parameter is specified, the cache cluster will be added to the specified replication group as a read replica; otherwise, the cache cluster will be a standalone primary that is not part of any replication group.

auto_minor_version_upgrade Determines whether minor engine upgrades will be applied automatically to the cache cluster during the maintenance window. A value of True allows these upgrades to occur; False disables automatic upgrades.

security_group_ids One or more VPC security groups associated with the cache cluster. Use this parameter only when you are creating a cluster in a VPC.

cache_subnet_group_name The name of the cache subnet group to be used for the cache cluster. Use this parameter only when you are creating a cluster in a VPC.

engine_version The version number of the cache engine to be used for this cluster.

notification_topic_arn The Amazon Resource Name (ARN) of the Amazon Simple Notification Service (SNS) topic to which notifications will be sent. The Amazon SNS topic owner must be the same as the cache cluster owner.

preferred_maintenance_window The weekly time range (in UTC) during which system maintenance can occur. Example: sun:05:00-sun:09:00

wait Boolean. Wait for confirmation from boto that the cluster is in the creating state.

region Region to connect to.

key Secret key to be used.

keyid Access key to be used.

profile A dict with region, key and keyid, or a pillar key (string) that contains a dict with region, key and keyid.

22.27.16 salt.states.boto_elb

Manage ELBs

New in version 2014.7.0.

Create and destroy ELBs. Be aware that this interacts with Amazon's services, and so may incur charges.

This module uses `boto`, which can be installed via package, or pip.

This module accepts explicit elb credentials but can also utilize IAM roles assigned to the instance through Instance Profiles. Dynamic credentials are then automatically obtained from AWS API and no further configuration is necessary. More information available here.

If IAM roles are not used you need to specify them either in a pillar file or in the minion's config file:

```
elb.keyid: GKTADJGHEIQSXMKKRBJ08H
elb.key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs
```

It's also possible to specify `key`, `keyid` and `region` via a profile, either passed in as a dict, or as a string to pull from pillars or minion config:

```
myprofile:
    keyid: GKTADJGHEIQSXMKKRBJ08H
    key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs
    region: us-east-1
```

```
Ensure myelb ELB exists:
    boto_elb.present:
        - name: myelb
        - region: us-east-1
        - availability_zones:
            - us-east-1a
            - us-east-1c
            - us-east-1d
        - keyid: GKTADJGHEIQSXMKKRBJ08H
        - key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs
        - listeners:
            - elb_port: 443
              instance_port: 80
              elb_protocol: HTTPS
              instance_protocol: HTTP
              certificate: 'arn:aws:iam::1111111:server-certificate/mycert'
            - elb_port: 8210
              instance_port: 8210
              elb_protocol: TCP
        - health_check:
            target: 'HTTP:80/'
        - attributes:
            cross_zone_load_balancing:
              enabled: true
            access_log:
              enabled: true
              s3_bucket_name: 'mybucket'
              s3_bucket_prefix: 'my-logs'
              emit_interval: 5
        - cnames:
            - name: mycname.example.com.
              zone: example.com.
              ttl: 60
            - name: myothercname.example.com.
              zone: example.com.

# Using a profile from pillars
Ensure myelb ELB exists:
    boto_elb.present:
        - name: myelb
        - region: us-east-1
        - profile: myelbprofile

# Passing in a profile
Ensure myelb ELB exists:
    boto_elb.present:
        - name: myelb
        - region: us-east-1
        - profile:
            keyid: GKTADJGHEIQSXMKKRBJ08H
            key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs
```

It's possible to specify attributes from pillars by specifying a pillar. You can override the values defined in the pillard by setting the attributes on the resource. The module will use the default pillar key `boto_elb_attributes', which allows you to set default attributes for all ELB resources.

Setting the attributes pillar:

```
my_elb_attributes:
  cross_zone_load_balancing:
    enabled: true
  connection_draining:
    enabled: true
    timeout: 20
  access_log:
    enabled: true
    s3_bucket_name: 'mybucket'
    s3_bucket_prefix: 'my-logs'
    emit_interval: 5
```

Overriding the attribute values on the resource:

```
Ensure myelb ELB exists:
    boto_elb.present:
        - name: myelb
        - region: us-east-1
        - attributes_from_pillar: my_elb_attributes
        # override cross_zone_load_balancing:enabled
        - attributes:
            cross_zone_load_balancing:
              enabled: false
        - profile: myelbprofile
```

It's possible to specify cloudwatch alarms that will be setup along with the ELB. Note the alarm name will be defined by the name attribute provided, plus the ELB resource name.

```
Ensure myelb ELB exists:
    boto_elb.present:
        - name: myelb
        - region: us-east-1
        - profile: myelbprofile
        - alarms:
            UnHealthyHostCount:
              name: 'ELB UnHealthyHostCount **MANAGED BY SALT**'
              attributes:
                metric: UnHealthyHostCount
                namespace: AWS/ELB
                statistic: Average
                comparison: '>='
                threshold: 1.0
                period: 600
                evaluation_periods: 6
                unit: null
                description: ELB UnHealthyHostCount
                alarm_actions: ['arn:aws:sns:us-east-1:12345:myalarm']
                insufficient_data_actions: []
                ok_actions: ['arn:aws:sns:us-east-1:12345:myalarm']
```

You can also use alarms from pillars, and override values from the pillar alarms by setting overrides on the resource. Note that `boto_elb_alarms' will be used as a default value for all resources, if defined and can be used to ensure alarms are always set for a resource.

Setting the alarms in a pillar:

```
my_elb_alarm:
  UnHealthyHostCount:
    name: 'ELB UnHealthyHostCount **MANAGED BY SALT**'
    attributes:
      metric: UnHealthyHostCount
      namespace: AWS/ELB
      statistic: Average
      comparison: '>='
      threshold: 1.0
      period: 600
      evaluation_periods: 6
      unit: null
      description: ELB UnHealthyHostCount
      alarm_actions: ['arn:aws:sns:us-east-1:12345:myalarm']
      insufficient_data_actions: []
      ok_actions: ['arn:aws:sns:us-east-1:12345:myalarm']
```

Overriding the alarm values on the resource:

```
Ensure myelb ELB exists:
    boto_elb.present:
        - name: myelb
        - region: us-east-1
        - profile: myelbprofile
        - alarms_from_pillar: my_elb_alarm
        # override UnHealthyHostCount:attributes:threshold
        - alarms:
            UnHealthyHostCount:
              attributes:
                threshold: 2.0
```

salt.states.boto_elb.**absent**(*name*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

salt.states.boto_elb.**present**(*name*, *listeners*, *availability_zones=None*, *subnets=None*, *security_groups=None*, *scheme='internet-facing'*, *health_check=None*, *attributes=None*, *attributes_from_pillar='boto_elb_attributes'*, *cnames=None*, *alarms=None*, *alarms_from_pillar='boto_elb_alarms'*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Ensure the IAM role exists.

name Name of the IAM role.

availability_zones A list of availability zones for this ELB.

listeners

A list of listener lists; example:

[[`443', `HTTPS', `arn:aws:iam::1111111:server-certificate/mycert'], [`8443', `80', `HTTPS', `HTTP', `arn:aws:iam::1111111:server-certificate/mycert']

]

subnets A list of subnet IDs in your VPC to attach to your LoadBalancer.

security_groups The security groups assigned to your LoadBalancer within your VPC.

scheme The type of a LoadBalancer. internet-facing or internal. Once set, can not be modified.

health_check A dict defining the health check for this ELB.

attributes A dict defining the attributes to set on this ELB.

attributes_from_pillar name of pillar dict that contains attributes. Attributes defined for this specific state will override those from pillar.

cnames A list of cname dicts with attributes: name, zone, ttl, and identifier. See the boto_route53 state for information about these attributes.

alarms: a dictionary of name->boto_cloudwatch_alarm sections to be associated with this ELB. All attributes should be specified except for dimension which will be automatically set to this ELB. See the boto_cloudwatch_alarm state for information about these attributes.

alarms_from_pillar: name of pillar dict that contains alarm settings. Alarms defined for this specific state will override those from pillar.

region Region to connect to.

key Secret key to be used.

keyid Access key to be used.

profile A dict with region, key and keyid, or a pillar key (string) that contains a dict with region, key and keyid.

22.27.17 salt.states.boto_iam_role

Manage IAM roles

New in version 2014.7.0.

This module uses `boto`, which can be installed via package, or pip.

This module accepts explicit IAM credentials but can also utilize IAM roles assigned to the instance through Instance Profiles. Dynamic credentials are then automatically obtained from AWS API and no further configuration is necessary. More information available here.

If IAM roles are not used you need to specify them either in a pillar file or in the minion's config file:

```
iam.keyid: GKTADJGHEIQSXMKKRBJ08H
iam.key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs
```

It's also possible to specify `key`, `keyid` and `region` via a profile, either passed in as a dict, or as a string to pull from pillars or minion config:

```
myprofile:
    keyid: GKTADJGHEIQSXMKKRBJ08H
    key: askjghsdfjkghWupUjasdflkdfklgjsdfjajkghs
    region: us-east-1
```

Creating a role will automatically create an instance profile and associate it with the role. This is the default behavior of the AWS console.

```
myrole:
    boto_iam_role.present:
        - region: us-east-1
        - key: GKTADJGHEIQSXMKKRBJ08H
        - keyid: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs
        - policies_from_pillars:
            - shared_iam_bootstrap_policy
```

```
        - policies:
            MySQSPolicy:
                Statement:
                  - Action:
                        - sqs:*
                    Effect: Allow
                    Resource:
                        - arn:aws:sqs:*:*:*
                    Sid: MyPolicySQS1
            MyS3Policy:
                Statement:
                  - Action:
                        - s3:GetObject
                    Effect: Allow
                    Resource:
                        - arn:aws:s3:*:*:mybucket/*

# Using a credentials profile from pillars
myrole:
    boto_iam_role.present:
        - region: us-east-1
        - profile: myiamprofile

# Passing in a credentials profile
myrole:
    boto_iam_role.present:
        - region: us-east-1
        - profile:
            key: GKTADJGHEIQSXMKKRBJ08H
            keyid: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs
```

salt.states.boto_iam_role.**absent**(*name*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Ensure the IAM role is deleted.

name Name of the IAM role.

region Region to connect to.

key Secret key to be used.

keyid Access key to be used.

profile A dict with region, key and keyid, or a pillar key (string) that contains a dict with region, key and keyid.

salt.states.boto_iam_role.**present**(*name*, *policy_document=None*, *path=None*, *policies=None*, *policies_from_pillars=None*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Ensure the IAM role exists.

name Name of the IAM role.

policy_document The policy that grants an entity permission to assume the role. (See http://boto.readthedocs.org/en/latest/ref/iam.html#boto.iam.connection.IAMConnection.create_role)

path The path to the instance profile. (See http://boto.readthedocs.org/en/latest/ref/iam.html#boto.iam.connection.IAMConnect

policies A dict of IAM role policies.

policies_from_pillars A list of pillars that contain role policy dicts. Policies in the pillars will be merged in the order defined in the list and key conflicts will be handled by later defined keys overriding earlier defined

keys. The policies defined here will be merged with the policies defined in the policies argument. If keys conflict, the keys in the policies argument will override the keys defined in policies_from_pillars.

region Region to connect to.

key Secret key to be used.

keyid Access key to be used.

profile A dict with region, key and keyid, or a pillar key (string) that contains a dict with region, key and keyid.

22.27.18 salt.states.boto_lc

Manage Launch Configurations

New in version 2014.7.0.

Create and destroy Launch Configurations. Be aware that this interacts with Amazon's services, and so may incur charges.

A limitation of this module is that you can not modify launch configurations once they have been created. If a launch configuration with the specified name exists, this module will always report success, even if the specified configuration doesn't match. This is due to a limitation in Amazon's launch configuration API, as it only allows launch configurations to be created and deleted.

Also note that a launch configuration that's in use by an autoscale group can not be deleted until the autoscale group is no longer using it. This may affect the way in which you want to order your states.

This module uses `boto`, which can be installed via package, or pip.

This module accepts explicit autoscale credentials but can also utilize IAM roles assigned to the instance through Instance Profiles. Dynamic credentials are then automatically obtained from AWS API and no further configuration is necessary. More information available here.

If IAM roles are not used you need to specify them either in a pillar file or in the minion's config file:

```
asg.keyid: GKTADJGHEIQSXMKKRBJ08H
asg.key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs
```

It's also possible to specify `key`, `keyid` and `region` via a profile, either passed in as a dict, or as a string to pull from pillars or minion config:

```
myprofile:
    keyid: GKTADJGHEIQSXMKKRBJ08H
    key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs
    region: us-east-1
```

Credential information is shared with autoscale groups as launch configurations and autoscale groups are completely dependent on each other.

```
Ensure mylc exists:
  boto_lc.present:
    - name: mylc
    - image_id: ami-0b9c9f62
    - key_name: mykey
    - security_groups:
        - mygroup
```

```
    - instance_type: m1.small
    - instance_monitoring: true
    - block_device_mappings:
        - '/dev/sda1':
            size: 20
    - cloud_init:
        scripts:
          'run_salt.sh': |
            #!/bin/bash

            add-apt-repository -y ppa:saltstack/salt
            apt-get update
            apt-get install -y salt-minion
            salt-call state.highstate
    - region: us-east-1
    - keyid: GKTADJGHEIQSXMKKRBJ08H
    - key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs

# Using a profile from pillars.
Ensure mylc exists:
  boto_lc.present:
    - name: mylc
    - image_id: ami-0b9c9f62
    - profile: myprofile

# Passing in a profile.
Ensure mylc exists:
  boto_lc.present:
    - name: mylc
    - image_id: ami-0b9c9f62
    - profile:
        keyid: GKTADJGHEIQSXMKKRBJ08H
        key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs
        region: us-east-1
```

salt.states.boto_lc.**absent**(*name*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Ensure the named launch configuration is deleted.

name Name of the launch configuration.

region The region to connect to.

key Secret key to be used.

keyid Access key to be used.

profile A dict with region, key and keyid, or a pillar key (string) that contains a dict with region, key and keyid.

salt.states.boto_lc.**present**(*name*, *image_id*, *key_name=None*, *security_groups=None*, *user_data=None*, *cloud_init=None*, *instance_type='m1.small'*, *kernel_id=None*, *ramdisk_id=None*, *block_device_mappings=None*, *instance_monitoring=False*, *spot_price=None*, *instance_profile_name=None*, *ebs_optimized=False*, *associate_public_ip_address=None*, *volume_type=None*, *delete_on_termination=True*, *iops=None*, *use_block_device_types=False*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Ensure the launch configuration exists.

name Name of the launch configuration.

image_id AMI to use for instances. AMI must exist or creation of the launch configuration will fail.

key_name Name of the EC2 key pair to use for instances. Key must exist or creation of the launch configuration will fail.

security_groups List of Names or security group id's of the security groups with which to associate the EC2 instances or VPC instances, respectively. Security groups must exist, or creation of the launch configuration will fail.

user_data The user data available to launched EC2 instances.

cloud_init A dict of cloud_init configuration. Currently supported values: scripts, cloud-config. Mutually exclusive with user_data.

instance_type The instance type. ex: m1.small.

kernel_id The kernel id for the instance.

ramdisk_id The RAM disk ID for the instance.

block_device_mappings A dict of block device mappings.

instance_monitoring Whether instances in group are launched with detailed monitoring.

spot_price The spot price you are bidding. Only applies if you are building an autoscaling group with spot instances.

instance_profile_name The name or the Amazon Resource Name (ARN) of the instance profile associated with the IAM role for the instance. Instance profile must exist or the creation of the launch configuration will fail.

ebs_optimized Specifies whether the instance is optimized for EBS I/O (true) or not (false).

associate_public_ip_address Used for Auto Scaling groups that launch instances into an Amazon Virtual Private Cloud. Specifies whether to assign a public IP address to each instance launched in a Amazon VPC.

volume_type Undocumented in boto.

delete_on_termination Undocumented in boto.

iops Undocumented in boto.

use_block_device_types Undocumented in boto.

region The region to connect to.

key Secret key to be used.

keyid Access key to be used.

profile A dict with region, key and keyid, or a pillar key (string) that contains a dict with region, key and keyid.

22.27.19 salt.states.boto_route53

Manage Route53 records

New in version 2014.7.0.

Create and delete Route53 records. Be aware that this interacts with Amazon's services, and so may incur charges.

This module uses `boto`, which can be installed via package, or pip.

This module accepts explicit route53 credentials but can also utilize IAM roles assigned to the instance through Instance Profiles. Dynamic credentials are then automatically obtained from AWS API and no further configuration is necessary. More information available here.

If IAM roles are not used you need to specify them either in a pillar file or in the minion's config file:

```
route53.keyid: GKTADJGHEIQSXMKKRBJ08H
route53.key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs
```

It's also possible to specify `key`, `keyid` and `region` via a profile, either passed in as a dict, or as a string to pull from pillars or minion config:

```
myprofile:
    keyid: GKTADJGHEIQSXMKKRBJ08H
    key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs
    region: us-east-1
```

```
mycnamerecord:
    boto_route53.present:
        - name: test.example.com.
        - value: my-elb.us-east-1.elb.amazonaws.com.
        - zone: example.com.
        - ttl: 60
        - record_type: CNAME
        - region: us-east-1
        - keyid: GKTADJGHEIQSXMKKRBJ08H
        - key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs

# Using a profile from pillars
myarecord:
    boto_route53.present:
        - name: test.example.com.
        - value: 1.1.1.1
        - zone: example.com.
        - ttl: 60
        - record_type: A
        - region: us-east-1
        - profile: myprofile

# Passing in a profile
myarecord:
    boto_route53.present:
        - name: test.example.com.
        - value: 1.1.1.1
        - zone: example.com.
        - ttl: 60
        - record_type: A
        - region: us-east-1
        - profile:
            keyid: GKTADJGHEIQSXMKKRBJ08H
            key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs
```

`salt.states.boto_route53.`**`absent`**(*name*, *zone*, *record_type*, *identifier=None*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Ensure the Route53 record is deleted.

name Name of the record.

zone The zone to delete the record from.

identifier An identifier to match for deletion.

region The region to connect to.

key Secret key to be used.

keyid Access key to be used.

profile A dict with region, key and keyid, or a pillar key (string) that contains a dict with region, key and keyid.

`salt.states.boto_route53.`**`present`**(*name*, *value*, *zone*, *record_type*, *ttl=None*, *identifier=None*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Ensure the Route53 record is present.

name Name of the record.

value Value of the record.

zone The zone to create the record in.

record_type The record type. Currently supported values: A, CNAME, MX

ttl The time to live for the record.

identifier The unique identifier to use for this record.

region The region to connect to.

key Secret key to be used.

keyid Access key to be used.

profile A dict with region, key and keyid, or a pillar key (string) that contains a dict with region, key and keyid.

22.27.20 salt.states.boto_secgroup

Manage Security Groups

New in version 2014.7.0.

Create and destroy Security Groups. Be aware that this interacts with Amazon's services, and so may incur charges.

This module uses `boto`, which can be installed via package, or pip.

This module accepts explicit EC2 credentials but can also utilize IAM roles assigned to the instance through Instance Profiles. Dynamic credentials are then automatically obtained from AWS API and no further configuration is necessary. More information available here.

If IAM roles are not used you need to specify them either in a pillar file or in the minion's config file:

```
secgroup.keyid: GKTADJGHEIQSXMKKRBJ08H
secgroup.key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs
```

It's also possible to specify `key`, `keyid` and `region` via a profile, either passed in as a dict, or as a string to pull from pillars or minion config:

```
myprofile:
    keyid: GKTADJGHEIQSXMKKRBJ08H
    key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs
    region: us-east-1
```

```
Ensure mysecgroup exists:
    boto_secgroup.present:
        - name: mysecgroup
        - description: My security group
        - rules:
            - ip_protocol: tcp
              from_port: 80
              to_port: 80
              cidr_ip:
                - 10.0.0.0/0
                - 192.168.0.0/0
        - region: us-east-1
        - keyid: GKTADJGHEIQSXMKKRBJ08H
        - key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs

# Using a profile from pillars
Ensure mysecgroup exists:
    boto_secgroup.present:
        - name: mysecgroup
        - description: My security group
        - region: us-east-1
        - profile: myprofile

# Passing in a profile
Ensure mysecgroup exists:
    boto_secgroup.present:
        - name: mysecgroup
        - description: My security group
        - region: us-east-1
        - profile:
            keyid: GKTADJGHEIQSXMKKRBJ08H
            key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs
```

salt.states.boto_secgroup.**absent**(*name*, *vpc_id=None*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Ensure a security group with the specified name does not exist.

name Name of the security group.

vpc_id The ID of the VPC to create the security group in, if any.

region Region to connect to.

key Secret key to be used.

keyid Access key to be used.

profile A dict with region, key and keyid, or a pillar key (string) that contains a dict with region, key and keyid.

salt.states.boto_secgroup.**present**(*name*, *description*, *vpc_id=None*, *rules=None*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Ensure the security group exists with the specified rules.

name Name of the security group.

description A description of this security group.

vpc_id The ID of the VPC to create the security group in, if any.

rules A list of ingress rule dicts.

region Region to connect to.

key Secret key to be used.

keyid Access key to be used.

profile A dict with region, key and keyid, or a pillar key (string) that contains a dict with region, key and keyid.

22.27.21 salt.states.boto_sns

Manage SNS Topics

Create and destroy SNS topics. Be aware that this interacts with Amazon's services, and so may incur charges.

This module uses `boto`, which can be installed via package, or pip.

This module accepts explicit AWS credentials but can also utilize IAM roles assigned to the instance through Instance Profiles. Dynamic credentials are then automatically obtained from AWS API and no further configuration is necessary. More information available here.

If IAM roles are not used you need to specify them either in a pillar file or in the minion's config file:

```
sns.keyid: GKTADJGHEIQSXMKKRBJ08H
sns.key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs
```

It's also possible to specify `key`, `keyid` and `region` via a profile, either passed in as a dict, or as a string to pull from pillars or minion config:

```
myprofile:
    keyid: GKTADJGHEIQSXMKKRBJ08H
    key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs
    region: us-east-1
```

```
mytopic:
    boto_sns.present:
        - region: us-east-1
        - keyid: GKTADJGHEIQSXMKKRBJ08H
        - key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs

# Using a profile from pillars
mytopic:
    boto_sns.present:
        - region: us-east-1
        - profile: mysnsprofile

# Passing in a profile
mytopic:
    boto_sns.present:
        - region: us-east-1
        - profile:
```

```
            keyid: GKTADJGHEIQSXMKKRBJ08H
            key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs
```

salt.states.boto_sns.**absent**(*name*, *region=None*, *key=None*, *keyid=None*, *profile=None*)
Ensure the named sns topic is deleted.

name Name of the SNS topic.

region Region to connect to.

key Secret key to be used.

keyid Access key to be used.

profile A dict with region, key and keyid, or a pillar key (string) that contains a dict with region, key and keyid.

salt.states.boto_sns.**present**(*name*, *region=None*, *key=None*, *keyid=None*, *profile=None*)
Ensure the SNS topic exists.

name Name of the SNS topic.

region Region to connect to.

key Secret key to be used.

keyid Access key to be used.

profile A dict with region, key and keyid, or a pillar key (string) that contains a dict with region, key and keyid.

22.27.22 salt.states.boto_sqs

Manage SQS Queues

New in version 2014.7.0.

Create and destroy SQS queues. Be aware that this interacts with Amazon's services, and so may incur charges.

This module uses `boto`, which can be installed via package, or pip.

This module accepts explicit SQS credentials but can also utilize IAM roles assigned to the instance through Instance Profiles. Dynamic credentials are then automatically obtained from AWS API and no further configuration is necessary. More information available here.

If IAM roles are not used you need to specify them either in a pillar file or in the minion's config file:

```
sqs.keyid: GKTADJGHEIQSXMKKRBJ08H
sqs.key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs
```

It's also possible to specify `key`, `keyid` and `region` via a profile, either passed in as a dict, or as a string to pull from pillars or minion config:

```
myprofile:
    keyid: GKTADJGHEIQSXMKKRBJ08H
    key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs
    region: us-east-1
```

```
myqueue:
    boto_sqs.present:
        - region: us-east-1
        - keyid: GKTADJGHEIQSXMKKRBJ08H
        - key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs
        - attributes:
            ReceiveMessageWaitTimeSeconds: 20

# Using a profile from pillars
myqueue:
    boto_sqs.present:
        - region: us-east-1
        - profile: mysqsprofile

# Passing in a profile
myqueue:
    boto_sqs.present:
        - region: us-east-1
        - profile:
            keyid: GKTADJGHEIQSXMKKRBJ08H
            key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs
```

salt.states.boto_sqs.**absent**(*name*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Ensure the named sqs queue is deleted.

name Name of the SQS queue.

region Region to connect to.

key Secret key to be used.

keyid Access key to be used.

profile A dict with region, key and keyid, or a pillar key (string) that contains a dict with region, key and keyid.

salt.states.boto_sqs.**present**(*name*, *attributes=None*, *region=None*, *key=None*, *keyid=None*, *profile=None*)

Ensure the SQS queue exists.

name Name of the SQS queue.

attributes A dict of key/value SQS attributes.

region Region to connect to.

key Secret key to be used.

keyid Access key to be used.

profile A dict with region, key and keyid, or a pillar key (string) that contains a dict with region, key and keyid.

22.27.23 salt.states.chef

Execute Chef client runs

Run chef-client or chef-solo

```
my-chef-run:
  chef.client:
    - override-runlist: 'demo1,demo2'
    - server: 'https://chef.domain.com'

default-chef-run:
  chef.client: []

my-solo-run:
  chef.solo:
    - environment: dev
```

salt.states.chef.**client**(*name*, ***kwargs*)

name Unique identifier for the state. Does not affect the Chef run.

server The chef server URL

client_key Set the client key file location

config The configuration file to use

config-file-jail Directory under which config files are allowed to be loaded (no client.rb or knife.rb outside this path will be loaded).

environment Set the Chef Environment on the node

group Group to set privilege to

json-attributes Load attributes from a JSON file or URL

localmode Point chef-client at local repository if True

log_level Set the log level (debug, info, warn, error, fatal)

logfile Set the log file location

node-name The node name for this client

override-runlist Replace current run list with specified items for a single run

pid Set the PID file location, defaults to /tmp/chef-client.pid

run-lock-timeout Set maximum duration to wait for another client run to finish, default is indefinitely.

runlist Permanently replace current run list with specified items

user User to set privilege to

validation_key Set the validation key file location, used for registering new clients

salt.states.chef.**solo**(*name*, ***kwargs*)

name Unique identifier for the state. Does not affect the Chef run.

config The configuration file to use

environment Set the Chef Environment on the node

group Group to set privilege to

json-attributes Load attributes from a JSON file or URL

log_level Set the log level (debug, info, warn, error, fatal)

logfile Set the log file location

node-name The node name for this client

override-runlist Replace current run list with specified items for a single run

recipe-url Pull down a remote gzipped tarball of recipes and untar it to the cookbook cache

run-lock-timeout Set maximum duration to wait for another client run to finish, default is indefinitely.

user User to set privilege to

22.27.24 salt.states.cloud

Using states instead of maps to deploy clouds

New in version 2014.1.0.

Use this minion to spin up a cloud instance:

```
my-ec2-instance:
  cloud.profile:
    my-ec2-config
```

`salt.states.cloud.`**`absent`**(*name*, *onlyif=None*, *unless=None*)

Ensure that no instances with the specified names exist.

CAUTION: This is a destructive state, which will search all configured cloud providers for the named instance, and destroy it.

name The name of the instance to destroy

onlyif Do run the state only if is unless succeed

unless Do not run the state at least unless succeed

`salt.states.cloud.`**`present`**(*name*, *cloud_provider*, *onlyif=None*, *unless=None*, ***kwargs*)

Spin up a single instance on a cloud provider, using salt-cloud. This state does not take a profile argument; rather, it takes the arguments that would normally be configured as part of the state.

Note that while this function does take any configuration argument that would normally be used to create an instance, it will not verify the state of any of those arguments on an existing instance. Stateful properties of an instance should be configured using their own individual state (i.e., cloud.tagged, cloud.untagged, etc).

name The name of the instance to create

cloud_provider The name of the cloud provider to use

onlyif Do run the state only if is unless succeed

unless Do not run the state at least unless succeed

`salt.states.cloud.`**`profile`**(*name*, *profile*, *onlyif=None*, *unless=None*, ***kwargs*)

Create a single instance on a cloud provider, using a salt-cloud profile.

Note that while profiles used this function do take any configuration argument that would normally be used to create an instance using a profile, this state will not verify the state of any of those arguments on an existing instance. Stateful properties of an instance should be configured using their own individual state (i.e., cloud.tagged, cloud.untagged, etc).

name The name of the instance to create

profile The name of the cloud profile to use

onlyif Do run the state only if is unless succeed

unless Do not run the state at least unless succeed

kwargs Any profile override or addition

`salt.states.cloud.`**`volume_absent`**(*name*, *provider=None*, ***kwargs*)
Check that a block volume exists.

`salt.states.cloud.`**`volume_attached`**(*name*, *server_name*, *provider=None*, ***kwargs*)
Check if a block volume is attached.

`salt.states.cloud.`**`volume_detached`**(*name*, *server_name=None*, *provider=None*, ***kwargs*)
Check if a block volume is attached.

Returns True if server or Volume do not exist.

`salt.states.cloud.`**`volume_present`**(*name*, *provider=None*, ***kwargs*)
Check that a block volume exists.

22.27.25 salt.states.cmd

Execution of arbitrary commands

The cmd state module manages the enforcement of executed commands, this state can tell a command to run under certain circumstances.

A simple example to execute a command:

```
date > /tmp/salt-run:
  cmd.run
```

Only run if another execution failed, in this case truncate syslog if there is no disk space:

```
> /var/log/messages:
  cmd.run:
    - unless: echo 'foo' > /tmp/.test && rm -f /tmp/.test
```

Only run if the file specified by `creates` does not exist, in this case touch /tmp/foo if it does not exist.

```
touch /tmp/foo:
  cmd.run:
    - creates: /tmp/foo
```

Note: The `creates` option was added to version 2014.7.0

Salt determines whether the `cmd` state is successfully enforced based on the exit code returned by the command. If the command returns a zero exit code, then salt determines that the state was successfully enforced. If the script returns a non-zero exit code, then salt determines that it failed to successfully enforce the state. If a command returns a non-zero exit code but you wish to treat this as a success, then you must place the command in a script and explicitly set the exit code of the script to zero.

Please note that the success or failure of the state is not affected by whether a state change occurred nor the stateful argument.

When executing a command or script, the state (i.e., changed or not) of the command is unknown to Salt's state system. Therefore, by default, the `cmd` state assumes that any command execution results in a changed state.

This means that if a `cmd` state is watched by another state then the state that's watching will always be executed due to the *changed* state in the `cmd` state.

Many state functions in this module now also accept a `stateful` argument. If `stateful` is specified to be true then it is assumed that the command or script will determine its own state and communicate it back by following a simple protocol described below:

1. **If there's nothing in the stdout of the command, then assume no changes.** Otherwise, the stdout must be either in JSON or its *last* non-empty line must be a string of key=value pairs delimited by spaces (no spaces on either side of `=`).
2. **If it's JSON then it must be a JSON object (e.g., {}).** If it's key=value pairs then quoting may be used to include spaces. (Python's shlex module is used to parse the key=value string)

   Two special keys or attributes are recognized in the output:

```
changed: bool (i.e., 'yes', 'no', 'true', 'false', case-insensitive)
comment: str  (i.e., any string)
```

   So, only if `changed` is `True` then assume the command execution has changed the state, and any other key values or attributes in the output will be set as part of the changes.
3. **If there's a comment then it will be used as the comment of the state.**

   Here's an example of how one might write a shell script for use with a stateful command:

```
#!/bin/bash
#
echo "Working hard..."

# writing the state line
echo  # an empty line here so the next line will be the last.
echo "changed=yes comment='something has changed' whatever=123"
```

   And an example SLS file using this module:

```
Run myscript:
  cmd.run:
    - name: /path/to/myscript
    - cwd: /
    - stateful: True

Run only if myscript changed something:
  cmd.wait:
    - name: echo hello
    - cwd: /
    - watch:
        - cmd: Run myscript
```

   Note that if the `cmd.wait` state also specifies `stateful: True` it can then be watched by some other states as well.

`cmd.wait` is not restricted to watching only cmd states. For example it can also watch a git state for changes

```
# Watch for changes to a git repo and rebuild the project on updates
my-project:
  git.latest:
    - name: git@github.com/repo/foo
    - target: /opt/foo
    - rev: master
  cmd.wait:
```

```
    - name: make install
    - cwd: /opt/foo
    - watch:
      - git: my-project
```

Should I use `cmd.run` or `cmd.wait`? ----------

These two states are often confused. The important thing to remember about them is that `cmd.run` states are run each time the SLS file that contains them is applied. If it is more desirable to have a command that only runs after some other state changes, then `cmd.wait` does just that. `cmd.wait` is designed to *watch* other states, and is executed when the state it is watching changes. Example:

```
/usr/local/bin/postinstall.sh:
  cmd.wait:
    - watch:
      - pkg: mycustompkg
  file.managed:
    - source: salt://utils/scripts/postinstall.sh

mycustompkg:
  pkg.installed:
    - require:
      - file: /usr/local/bin/postinstall.sh
```

How do I create an environment from a pillar map?

The map that comes from a pillar cannot be directly consumed by the env option. To use it one must convert it to a list. Example:

```
printenv:
  cmd.run:
    - env:
      {% for key, value in pillar['keys'].iteritems() %}
      - '{{ key }}': '{{ value }}'
      {% endfor %}
```

`salt.states.cmd.`**`call`**(*name*, *func*, *args=()*, *kws=None*, *onlyif=None*, *unless=None*, *creates=None*, *output_loglevel='debug'*, *use_vt=False*, ***kwargs*)

Invoke a pre-defined Python function with arguments specified in the state declaration. This function is mainly used by the `salt.renderers.pydsl` renderer.

The interpretation of `onlyif` and `unless` arguments are identical to those of `cmd.run`, and all other arguments(`cwd`, `runas`, ...) allowed by `cmd.run` are allowed here, except that their effects apply only to the commands specified in *onlyif* and *unless* rather than to the function to be invoked.

In addition, the `stateful` argument has no effects here.

The return value of the invoked function will be interpreted as follows.

If it's a dictionary then it will be passed through to the state system, which expects it to have the usual structure returned by any salt state function.

Otherwise, the return value (denoted as `result` in the code below) is expected to be a JSON serializable object, and this dictionary is returned:

```
    {
        'name': name
        'changes': {'retval': result},
        'result': True if result is None else bool(result),
        'comment': result if isinstance(result, string_types) else ''
    }
```

salt.states.cmd.**mod_run_check**(*cmd_kwargs*, *onlyif*, *unless*, *group*, *creates*)
Execute the onlyif and unless logic. Return a result dict if: * group is not available * onlyif failed (onlyif != 0) * unless succeeded (unless == 0) else return True

salt.states.cmd.**mod_watch**(*name*, ***kwargs*)
Execute a cmd function based on a watch call

salt.states.cmd.**run**(*name*, *onlyif=None*, *unless=None*, *creates=None*, *cwd=None*, *user=None*, *group=None*, *shell=None*, *env=None*, *stateful=False*, *umask=None*, *output_loglevel='debug'*, *quiet=False*, *timeout=None*, *use_vt=False*, ***kwargs*)
Run a command if certain circumstances are met. Use `cmd.wait` if you want to use the `watch` requisite.

name The command to execute, remember that the command will execute with the path and permissions of the salt-minion.

onlyif A command to run as a check, run the named command only if the command passed to the `onlyif` option returns true

unless A command to run as a check, only run the named command if the command passed to the `unless` option returns false

cwd The current working directory to execute the command in, defaults to /root

user The user name to run the command as

group The group context to run the command as

shell The shell to use for execution, defaults to the shell grain

env A list of environment variables to be set prior to execution. Example:

```
        salt://scripts/foo.sh:
          cmd.script:
            - env:
              - BATCH: 'yes'
```

> **Warning**: The above illustrates a common PyYAML pitfall, that **yes**, **no**, **on**, **off**, **true**, and **false** are all loaded as boolean `True` and `False` values, and must be enclosed in quotes to be used as strings. More info on this (and other) PyYAML idiosyncrasies can be found *here*.

stateful The command being executed is expected to return data about executing a state

umask The umask (in octal) to use when running the command.

output_loglevel Control the loglevel at which the output from the command is logged. Note that the command being run will still be logged (loglevel: DEBUG) regardless, unless `quiet` is used for this value.

quiet The command will be executed quietly, meaning no log entries of the actual command or its return data. This is deprecated as of the **2014.1.0** release, and is being replaced with `output_loglevel: quiet`.

timeout If the command has not terminated after timeout seconds, send the subprocess sigterm, and if sigterm is ignored, follow up with sigkill

creates Only run if the file specified by `creates` does not exist.

New in version 2014.7.0.

use_vt Use VT utils (saltstack) to stream the command output more interactively to the console and the logs. This is experimental.

Note: cmd.run supports the usage of `reload_modules`. This functionality allows you to force Salt to reload all modules. You should only use `reload_modules` if your cmd.run does some sort of installation (such as `pip`), if you do not reload the modules future items in your state which rely on the software being installed will fail.

```
getpip:
  cmd.run:
    - name: /usr/bin/python /usr/local/sbin/get-pip.py
    - unless: which pip
    - require:
      - pkg: python
      - file: /usr/local/sbin/get-pip.py
    - reload_modules: True
```

`salt.states.cmd.`**`script`**(*name*, *source=None*, *template=None*, *onlyif=None*, *unless=None*, *creates=None*, *cwd=None*, *user=None*, *group=None*, *shell=None*, *env=None*, *stateful=False*, *umask=None*, *timeout=None*, *use_vt=False*, *output_loglevel='debug'*, ***kwargs*)

Download a script and execute it with specified arguments.

source The location of the script to download. If the file is located on the master in the directory named spam, and is called eggs, the source string is salt://spam/eggs

template If this setting is applied then the named templating engine will be used to render the downloaded file. Currently jinja, mako, and wempy are supported

name Either ``cmd arg1 arg2 arg3...'' (cmd is not used) or a source ``salt://...''.

onlyif Run the named command only if the command passed to the `onlyif` option returns true

unless Run the named command only if the command passed to the `unless` option returns false

cwd The current working directory to execute the command in, defaults to /root

user The name of the user to run the command as

group The group context to run the command as

shell The shell to use for execution. The default is set in grains['shell']

env A list of environment variables to be set prior to execution. Example:

```
salt://scripts/foo.sh:
  cmd.script:
    - env:
      - BATCH: 'yes'
```

> **Warning**: The above illustrates a common PyYAML pitfall, that **yes**, **no**, **on**, **off**, **true**, and **false** are all loaded as boolean `True` and `False` values, and must be enclosed in quotes to be used as strings. More info on this (and other) PyYAML idiosyncrasies can be found *here*.

umask The umask (in octal) to use when running the command.

stateful The command being executed is expected to return data about executing a state

timeout If the command has not terminated after timeout seconds, send the subprocess sigterm, and if sigterm is ignored, follow up with sigkill

args String of command line args to pass to the script. Only used if no args are specified as part of the *name* argument. To pass a string containing spaces in YAML, you will need to doubly-quote it: ``arg1 `arg two' arg3''

creates Only run if the file specified by `creates` does not exist.

New in version 2014.7.0.

use_vt Use VT utils (saltstack) to stream the command output more interactively to the console and the logs. This is experimental.

output_loglevel Control the loglevel at which the output from the command is logged. Note that the command being run will still be logged (loglevel: DEBUG) regardless, unless `quiet` is used for this value.

`salt.states.cmd.`**`wait`**(*name*, *onlyif=None*, *unless=None*, *creates=None*, *cwd=None*, *user=None*, *group=None*, *shell=None*, *env=()*, *stateful=False*, *umask=None*, *output_loglevel='debug'*, *use_vt=False*, ***kwargs*)

Run the given command only if the watch statement calls it

name The command to execute, remember that the command will execute with the path and permissions of the salt-minion.

onlyif A command to run as a check, run the named command only if the command passed to the `onlyif` option returns true

unless A command to run as a check, only run the named command if the command passed to the `unless` option returns false

cwd The current working directory to execute the command in, defaults to /root

user The user name to run the command as

group The group context to run the command as

shell The shell to use for execution, defaults to /bin/sh

env A list of environment variables to be set prior to execution. Example:

```
salt://scripts/foo.sh:
  cmd.script:
    - env:
      - BATCH: 'yes'
```

> **Warning:** The above illustrates a common PyYAML pitfall, that **yes**, **no**, **on**, **off**, **true**, and **false** are all loaded as boolean `True` and `False` values, and must be enclosed in quotes to be used as strings. More info on this (and other) PyYAML idiosyncrasies can be found *here*.

umask The umask (in octal) to use when running the command.

stateful The command being executed is expected to return data about executing a state

creates Only run if the file specified by `creates` does not exist.

New in version 2014.7.0.

output_loglevel Control the loglevel at which the output from the command is logged. Note that the command being run will still be logged (loglevel: DEBUG) regardless, unless `quiet` is used for this value.

use_vt Use VT utils (saltstack) to stream the command output more interactively to the console and the logs. This is experimental.

salt.states.cmd.**wait_call**(*name*, *func*, *args=()*, *kws=None*, *onlyif=None*, *unless=None*, *creates=None*, *stateful=False*, *use_vt=False*, *output_loglevel='debug'*, ***kwargs*)

salt.states.cmd.**wait_script**(*name*, *source=None*, *template=None*, *onlyif=None*, *unless=None*, *cwd=None*, *user=None*, *group=None*, *shell=None*, *env=None*, *stateful=False*, *umask=None*, *use_vt=False*, *output_loglevel='debug'*, ***kwargs*)

Download a script from a remote source and execute it only if a watch statement calls it.

source The source script being downloaded to the minion, this source script is hosted on the salt master server. If the file is located on the master in the directory named spam, and is called eggs, the source string is salt://spam/eggs

template If this setting is applied then the named templating engine will be used to render the downloaded file, currently jinja, mako, and wempy are supported

name The command to execute, remember that the command will execute with the path and permissions of the salt-minion.

onlyif A command to run as a check, run the named command only if the command passed to the `onlyif` option returns true

unless A command to run as a check, only run the named command if the command passed to the `unless` option returns false

cwd The current working directory to execute the command in, defaults to /root

user The user name to run the command as

group The group context to run the command as

shell The shell to use for execution, defaults to the shell grain

env A list of environment variables to be set prior to execution. Example:

```
salt://scripts/foo.sh:
  cmd.script:
    - env:
      - BATCH: 'yes'
```

> **Warning**: The above illustrates a common PyYAML pitfall, that **yes**, **no**, **on**, **off**, **true**, and **false** are all loaded as boolean `True` and `False` values, and must be enclosed in quotes to be used as strings. More info on this (and other) PyYAML idiosyncrasies can be found *here*.

umask The umask (in octal) to use when running the command.

stateful The command being executed is expected to return data about executing a state

use_vt

Use VT utils (saltstack) to stream the command output more interactively to the console and the logs. This is experimental.

output_loglevel Control the loglevel at which the output from the command is logged. Note that the command being run will still be logged (loglevel: DEBUG) regardless, unless `quiet` is used for this value.

22.27.26 salt.states.composer

Installation of Composer Packages

These states manage the installed packages for composer for PHP. Note that either composer is installed and accessible via a bin directory or you can pass the location of composer in the state.

```
get-composer:
  cmd.run:
    - name: 'CURL=`which curl`; $CURL -sS https://getcomposer.org/installer | php'
    - unless: test -f /usr/local/bin/composer
    - cwd: /root/

install-composer:
  cmd.wait:
    - name: mv /root/composer.phar /usr/local/bin/composer
    - cwd: /root/
    - watch:
      - cmd: get-composer

/path/to/project:
  composer.installed:
    - no_dev: true
    - require:
      - cmd: install-composer


# Without composer installed in your PATH
# Note: composer.phar must be executable for state to work properly
/path/to/project:
  composer.installed:
    - composer: /path/to/composer.phar
    - php: /usr/local/bin/php
    - no_dev: true
```

salt.states.composer.**installed**(*name*, *composer=None*, *php=None*, *user=None*, *prefer_source=None*, *prefer_dist=None*, *no_scripts=None*, *no_plugins=None*, *optimize=None*, *no_dev=None*, *quiet=False*, *composer_home='/root'*, *always_check=True*)

Verify that the correct versions of composer dependencies are present.

dir Directory location of the composer.json file.

composer Location of the composer.phar file. If not set composer will just execute ``composer'' as if it is installed globally. (i.e. /path/to/composer.phar)

php Location of the php executable to use with composer. (i.e. /usr/bin/php)

user Which system user to run composer as.

New in version 2014.1.4.

prefer_source --prefer-source option of composer.

prefer_dist --prefer-dist option of composer.

no_scripts --no-scripts option of composer.

no_plugins --no-plugins option of composer.

optimize --optimize-autoloader option of composer. Recommended for production.

no_dev --no-dev option for composer. Recommended for production.

quiet --quiet option for composer. Whether or not to return output from composer.

composer_home $COMPOSER_HOME environment variable

always_check If True, _always_ run *composer install* in the directory. This is the default behavior. If False, only run *composer install* if there is no vendor directory present.

`salt.states.composer.`**`update`**(*name*, *composer=None*, *php=None*, *user=None*, *prefer_source=None*, *prefer_dist=None*, *no_scripts=None*, *no_plugins=None*, *optimize=None*, *no_dev=None*, *quiet=False*, *composer_home='/root'*)

Composer update the directory to ensure we have the latest versions of all project dependencies.

dir Directory location of the composer.json file.

composer Location of the composer.phar file. If not set composer will just execute ``composer'' as if it is installed globally. (i.e. /path/to/composer.phar)

php Location of the php executable to use with composer. (i.e. /usr/bin/php)

user Which system user to run composer as.

New in version 2014.1.4.

prefer_source --prefer-source option of composer.

prefer_dist --prefer-dist option of composer.

no_scripts --no-scripts option of composer.

no_plugins --no-plugins option of composer.

optimize --optimize-autoloader option of composer. Recommended for production.

no_dev --no-dev option for composer. Recommended for production.

quiet --quiet option for composer. Whether or not to return output from composer.

composer_home $COMPOSER_HOME environment variable

22.27.27 salt.states.cron

Management of cron, the Unix command scheduler

Cron declarations require a number of parameters. The following are the parameters used by Salt to define the various timing values for a cron job:

- `minute`
- `hour`
- `daymonth`
- `month`
- `dayweek` (0 to 6 are Sunday through Saturday, 7 can also be used for Sunday)

Warning: Any timing arguments not specified take a value of `*`. This means that setting `hour` to `5`, while not defining the `minute` param, will result in Salt adding a job that will execute every minute between 5 and 6 A.M.! Additionally, the default user for these states is `root`. Therefore, if the cron job is for another user, it is necessary to specify that user with the `user` parameter.

A long time ago (before 2014.2), when making changes to an existing cron job, the name declaration is the parameter used to uniquely identify the job, so if an existing cron that looks like this:

```
date > /tmp/crontest:
  cron.present:
    - user: root
    - minute: 5
```

Is changed to this:

```
date > /tmp/crontest:
  cron.present:
    - user: root
    - minute: 7
    - hour: 2
```

Then the existing cron will be updated, but if the cron command is changed, then a new cron job will be added to the user's crontab.

The current behavior is still relying on that mechanism, but you can also specify an identifier to identify your crontabs:

```
date > /tmp/crontest:
  cron.present:
    - identifier: SUPERCRON
    - user: root
    - minute: 7
    - hour: 2
```

New in version 2014.1.2.

And, some months later, you modify it:

```
superscript > /tmp/crontest:
  cron.present:
    - identifier: SUPERCRON
    - user: root
    - minute: 3
    - hour: 4
```

New in version 2014.1.2.

The old **date > /tmp/crontest** will be replaced by **superscript > /tmp/crontest**.

Additionally, Salt also supports running a cron every `x minutes` very similarly to the Unix convention of using `*/5` to have a job run every five minutes. In Salt, this looks like:

```
date > /tmp/crontest:
  cron.present:
    - user: root
    - minute: '*/5'
```

The job will now run every 5 minutes.

Additionally, the temporal parameters (minute, hour, etc.) can be randomized by using `random` instead of using a specific value. For example, by using the `random` keyword in the `minute` parameter of a cron state, the same cron job can be pushed to hundreds or thousands of hosts, and they would each use a randomly-generated minute.

This can be helpful when the cron job accesses a network resource, and it is not desirable for all hosts to run the job concurrently.

```
/path/to/cron/script:
  cron.present:
    - user: root
    - minute: random
    - hour: 2
```

New in version 0.16.0.

Since Salt assumes a value of `*` for unspecified temporal parameters, adding a parameter to the state and setting it to `random` will change that value from `*` to a randomized numeric value. However, if that field in the cron entry on the minion already contains a numeric value, then using the `random` keyword will not modify it.

`salt.states.cron.`**`absent`**(*name*, *user='root'*, *identifier=None*, ***kwargs*)

Verifies that the specified cron job is absent for the specified user; only the name is matched when removing a cron job.

name The command that should be absent in the user crontab.

user The name of the user whose crontab needs to be modified, defaults to the root user

identifier Custom-defined identifier for tracking the cron line for future crontab edits. This defaults to the state id

`salt.states.cron.`**`env_absent`**(*name*, *user='root'*)

Verifies that the specified environment variable is absent from the crontab for the specified user

name The name of the environment variable to remove from the user crontab

user The name of the user whose crontab needs to be modified, defaults to the root user

`salt.states.cron.`**`env_present`**(*name*, *value=None*, *user='root'*)

Verifies that the specified environment variable is present in the crontab for the specified user.

name The name of the environment variable to set in the user crontab

user The name of the user whose crontab needs to be modified, defaults to the root user

value The value to set for the given environment variable

`salt.states.cron.`**`file`**(*name*, *source_hash=''*, *user='root'*, *template=None*, *context=None*, *replace=True*, *defaults=None*, *env=None*, *backup=''*, ***kwargs*)

Provides file.managed-like functionality (templating, etc.) for a pre-made crontab file, to be assigned to a given user.

name The source file to be used as the crontab. This source file can be hosted on either the salt master server, or on an HTTP or FTP server. For files hosted on the salt file server, if the file is located on the master in the directory named spam, and is called eggs, the source string is `salt://spam/eggs`

If the file is hosted on a HTTP or FTP server then the source_hash argument is also required

source_hash This can be either a file which contains a source hash string for the source, or a source hash string. The source hash string is the hash algorithm followed by the hash of the file: `md5=e138491e9d5b97023cea823fe17bac22`

user The user to whom the crontab should be assigned. This defaults to root.

template If this setting is applied then the named templating engine will be used to render the downloaded file. Currently, jinja and mako are supported.

context Overrides default context variables passed to the template.

replace If the crontab should be replaced, if False then this command will be ignored if a crontab exists for the specified user. Default is True.

defaults Default context passed to the template.

backup Overrides the default backup mode for the user's crontab.

`salt.states.cron.`**`present`**(*name*, *user='root'*, *minute='*'*, *hour='*'*, *daymonth='*'*, *month='*'*, *dayweek='*'*, *comment=None*, *identifier=None*)

Verifies that the specified cron job is present for the specified user. For more advanced information about what exactly can be set in the cron timing parameters, check your cron system's documentation. Most Unix-like systems' cron documentation can be found via the crontab man page: `man 5 crontab`.

name The command that should be executed by the cron job.

user The name of the user whose crontab needs to be modified, defaults to the root user

minute The information to be set into the minute section, this can be any string supported by your cron system's the minute field. Default is `*`

hour The information to be set in the hour section. Default is `*`

daymonth The information to be set in the day of month section. Default is `*`

month The information to be set in the month section. Default is `*`

dayweek The information to be set in the day of week section. Default is `*`

comment User comment to be added on line previous the cron job

identifier Custom-defined identifier for tracking the cron line for future crontab edits. This defaults to the state id

22.27.28 salt.states.cyg

Installation of Cygwin packages.

A state module to manage cygwin packages. Packages can be installed or removed.

```
dos2unix:
  cyg.installed
```

class `salt.states.cyg.`**`DictDiffer`**(*current_dict*, *past_dict*)

Calculate the difference between two dictionaries.

1.items added

2.items removed

3.keys same in both but changed values

4.keys same in both and unchanged values

`added`()

Return a set of additions to past_dict.

`changed`()

Return a set of the keys with changed values.

`removed`()

Return a set of things removed from past_dict.

same()
True if the two dicts are the same.

unchanged()
Return a set of the keys with unchanged values.

salt.states.cyg.**installed**(*name*, *cyg_arch='x86_64'*, *mirrors=None*)
Make sure that a package is installed.

name The name of the package to install

cyg_arch [x86_64] The cygwin architecture to install the package into. Current options are x86 and x86_64

mirrors [None] List of mirrors to check. None will use a default mirror (kernel.org)

CLI Example:

```
rsync:
  cyg.installed:
    - mirrors:
      - http://mirror/without/public/key: ""
      - http://mirror/with/public/key: http://url/of/public/key
```

salt.states.cyg.**removed**(*name*, *cyg_arch='x86_64'*, *mirrors=None*)
Make sure that a package is not installed.

name The name of the package to uninstall

cyg_arch [x86_64] The cygwin architecture to remove the package from. Current options are x86 and x86_64

mirrors [None] List of mirrors to check. None will use a default mirror (kernel.org)

CLI Example:

```
rsync:
  cyg.removed:
    - mirrors:
      - http://mirror/without/public/key: ""
      - http://mirror/with/public/key: http://url/of/public/key
```

salt.states.cyg.**updated**(*name=None*, *cyg_arch='x86_64'*, *mirrors=None*)
Make sure all packages are up to date.

name [None] No affect, salt fails poorly without the arg available

cyg_arch [x86_64] The cygwin architecture to update. Current options are x86 and x86_64

mirrors [None] List of mirrors to check. None will use a default mirror (kernel.org)

CLI Example:

```
rsync:
  cyg.updated:
    - mirrors:
      - http://mirror/without/public/key: ""
      - http://mirror/with/public/key: http://url/of/public/key
```

22.27.29 salt.states.ddns

Dynamic DNS updates

Ensure a DNS record is present or absent utilizing RFC 2136 type dynamic updates. Requires dnspython module.

```
webserver:
  ddns.present:
    - zone: example.com
    - ttl: 60
    - data: 111.222.333.444
    - nameserver: 123.234.345.456
    - keyfile: /srv/salt/tsig_key.txt
```

salt.states.ddns.**absent**(*name*, *zone*, *data=None*, *rdtype=None*, ***kwargs*)

Ensures that the named DNS record is absent.

name The host portion of the DNS record, e.g., `webserver'

zone The zone to check

data Data for the DNS record. E.g., the IP address for an A record. If omitted, all records matching name (and rdtype, if provided) will be purged.

rdtype DNS resource type. If omitted, all types will be purged.

****kwargs** Additional arguments the ddns.delete function may need (e.g. nameserver, keyfile, keyname).

salt.states.ddns.**present**(*name*, *zone*, *ttl*, *data*, *rdtype='A'*, ***kwargs*)

Ensures that the named DNS record is present with the given ttl.

name The host portion of the DNS record, e.g., `webserver'

zone The zone to check/update

ttl TTL for the record

data Data for the DNS record. E.g., the IP address for an A record.

rdtype DNS resource type. Default `A'.

****kwargs** Additional arguments the ddns.update function may need (e.g. nameserver, keyfile, keyname).

22.27.30 salt.states.debconfmod

Management of debconf selections

depends

- debconf-utils package

The debconfmod state module manages the enforcement of debconf selections, this state can set those selections prior to package installation.

Available Functions

The debconfmod state has two functions, the `set` and `set_file` functions

set Set debconf selections from the state itself

set_file Set debconf selections from a file

```
nullmailer-debconf:
  debconf.set:
    - name: nullmailer
```

```
    - data:
        'shared/mailname': {'type': 'string', 'value': 'server.domain.tld'}
        'nullmailer/relayhost': {'type': 'string', 'value': 'mail.domain.tld'}
ferm-debconf:
  debconf.set:
    - name: ferm
    - data:
        'ferm/enable': {'type': 'boolean', 'value': True}
```

Note: Due to how PyYAML imports nested dicts (see *here*), the values in the `data` dict must be indented four spaces instead of two.

`salt.states.debconfmod.`**`set`**(*name*, *data*)

Set debconf selections

```
    <state_id>:
      debconf.set:
        - name: <name>
        - data:
            <question>: {'type': <type>, 'value': <value>}
            <question>: {'type': <type>, 'value': <value>}

    <state_id>:
      debconf.set:
        - name: <name>
        - data:
            <question>: {'type': <type>, 'value': <value>}
            <question>: {'type': <type>, 'value': <value>}
```

name: The package name to set answers for.

data: A set of questions/answers for debconf. Note that everything under this must be indented twice.

question: The question the is being pre-answered

type: The type of question that is being asked (string, boolean, select, etc.)

value: The answer to the question

`salt.states.debconfmod.`**`set_file`**(*name*, *source*, *template=None*, *context=None*, *defaults=None*, ***kwargs*)

Set debconf selections from a file or a template

```
    <state_id>:
      debconf.set_file:
        - source: salt://pathto/pkg.selections

    <state_id>:
      debconf.set_file:
        - source: salt://pathto/pkg.selections?saltenv=myenvironment

    <state_id>:
      debconf.set_file:
        - source: salt://pathto/pkg.selections.jinja2
        - template: jinja2
        - context:
            some_value: "false"
```

source: The location of the file containing the package selections

template If this setting is applied then the named templating engine will be used to render the package selections file, currently jinja, mako, and wempy are supported

context Overrides default context variables passed to the template.

defaults Default context passed to the template.

22.27.31 salt.states.disk

Disk monitoring state

Monitor the state of disk resources

`salt.states.disk.status`(*name*, *maximum=None*, *minimum=None*)
Return the current disk usage stats for the named mount point

22.27.32 salt.states.dockerio

Manage Docker containers

Docker is a lightweight, portable, self-sufficient software container wrapper. The base supported wrapper type is LXC, cgroups, and the Linux Kernel.

Warning: This state module is beta. The API is subject to change. No promise as to performance or functionality is yet present.

Note: This state module requires docker-py which supports Docker Remote API version 1.6.

Available Functions

- built

```
corp/mysuperdocker_img:
  docker.built:
    - path: /path/to/dir/container
```

- pulled

```
ubuntu:
  docker.pulled:
    - tag: latest
```

- pushed

```
corp/mysuperdocker_img:
  docker.pushed
```

- installed

```
mysuperdocker-container:
  docker.installed:
    - name: mysuperdocker
```

```
        - hostname: superdocker
        - image: corp/mysuperdocker_img
```

- loaded

```
    mysuperdocker-file:
      docker.loaded:
        - name: mysuperdocker
        - source: salt://_files/tmp/docker_image.tar
```

- running

```
    my_service:
      docker.running:
        - container: mysuperdocker
        - port_bindings:
            "5000/tcp":
                HostIp: ""
                HostPort: "5000"
```

Note: The `port_bindings` argument above is a dictionary. The double indentation is required for PyYAML to load the data structure properly as a python dictionary. More information can be found *here*

- absent

```
    mys_old_uperdocker:
      docker.absent
```

- run

```
    /finish-install.sh:
      docker.run:
        - cid: mysuperdocker
        - unless: grep -q something /var/log/foo
        - docker_unless: grep -q done /install_log
```

Note: The docker modules are named `dockerio` because the name `docker' would conflict with the underlying docker-py library.

`salt.states.dockerio.`**`absent`**(*name*)

Ensure that the container is absent; if not, it will will be killed and destroyed. (*docker inspect*)

name: Either the container name or id

`salt.states.dockerio.`**`built`**(*name*, *path=None*, *quiet=False*, *nocache=False*, *rm=True*, *force=False*, *timeout=None*, **args*, ***kwargs*)

Build a docker image from a path or URL to a dockerfile. (*docker build*)

name Name of the image

path URL (e.g. *url/branch/docker_dir/dockerfile*) or filesystem path to the dockerfile

`salt.states.dockerio.`**`installed`**(*name*, *image*, *command=None*, *hostname=None*, *user=None*, *detach=True*, *stdin_open=False*, *tty=False*, *mem_limit=0*, *ports=None*, *environment=None*, *dns=None*, *volumes=None*, *volumes_from=None*, *cpu_shares=None*, *cpuset=None*, **args*, ***kwargs*)

Ensure that a container with the given name exists; if not, build a new container from the specified image. (*docker run*)

name Name for the container

image Image from which to build this container

environment

Environment variables for the container, either

- a mapping of key, values
- a list of mappings of key, values

ports

List of ports definitions, either:

- a port to map
- a mapping of mapping portInHost : PortInContainer

volumes List of volumes

For other parameters, see absolutely first the salt.modules.dockerio execution module and the docker-py python bindings for docker documentation <https://github.com/dotcloud/docker-py#api>`_ for *docker.create_container*.

Note: This command does not verify that the named container is running the specified image.

`salt.states.dockerio.`**`loaded`**`(`*name*, *source=None*, *source_hash=''*, *force=False*`)`
Load an image into the local docker registry (*docker load*)

name Name of the docker image

source The source .tar file to download to the minion, created by docker save this source file can be hosted on either the salt master server, or on an HTTP or FTP server.

If the file is hosted on a HTTP or FTP server then the source_hash argument is also required

---

Note: See first the documentation for salt file.managed <http://docs.saltstack.com/en/latest/ref/states/all/_ salt.states.file.html#salt.states.file.managed>

---

source_hash

This can be one of the following:

1. a source hash string
2. the URI of a file that contains source hash strings

Note: See first the documentation for salt file.managed <http://docs.saltstack.com/en/latest/ref/states/all/_ salt.states.file.html#salt.states.file.managed>

---

force Load even if the image exists

`salt.states.dockerio.`**`mod_watch`**`(`*name*, *sfun=None*, **args*, ***kw*`)`

`salt.states.dockerio.`**`present`**`(`*name*, *image=None*, *is_latest=False*`)`
If a container with the given name is not present, this state will fail. Supports optionally checking for specific image/version (*docker inspect*)

name: container id

image: image the container should be running (defaults to any)

is_latest: also check if the container runs the latest version of the image (latest defined as the latest pulled onto the local machine)

salt.states.dockerio.**pulled**(*name*, *tag='latest'*, *force=False*, *insecure_registry=False*, **args*, ***kwargs*)

Pull an image from a docker registry. (*docker pull*)

Note: See first the documentation for *docker login*, *docker pull*, *docker push*, and docker.import_image (docker import). NOTE that we added in SaltStack a way to authenticate yourself with the Docker Hub Registry by supplying your credentials (username, email & password) using pillars. For more information, see salt.modules.dockerio execution module.

---

name Name of the image

tag Tag of the image

force Pull even if the image is already pulled

insecure_registry Set to `True` to allow connections to non-HTTPS registries. Default `False`.

salt.states.dockerio.**pushed**(*name*, *tag='latest'*, *insecure_registry=False*)

Push an image from a docker registry. (*docker push*)

Note: See first the documentation for *docker login*, *docker pull*, *docker push*, and docker.import_image (docker import). NOTE that we added in SaltStack a way to authenticate yourself with the Docker Hub Registry by supplying your credentials (username, email & password) using pillars. For more information, see salt.modules.dockerio execution module.

---

name Name of the image

tag Tag of the image [Optional]

insecure_registry Set to `True` to allow connections to non-HTTPS registries. Default `False`.

salt.states.dockerio.**run**(*name*, *cid=None*, *hostname=None*, *onlyif=None*, *unless=None*, *docked_onlyif=None*, *docked_unless=None*, **args*, ***kwargs*)

Run a command in a specific container

You can match by either name or hostname

name command to run in the container

cid Container id or name

state_id state_id

onlyif Only execute cmd if statement on the host returns 0

unless Do not execute cmd if statement on the host returns 0

docked_onlyif Only execute cmd if statement in the container returns 0

docked_unless Do not execute cmd if statement in the container returns 0

salt.states.dockerio.**running**(*name*, *image*, *container=None*, *command=None*, *hostname=None*, *user=None*, *detach=True*, *stdin_open=False*, *tty=False*, *mem_limit=0*, *ports=None*, *environment=None*, *dns=None*, *volumes=None*, *volumes_from=None*, *start=True*, *cap_add=None*, *cap_drop=None*, *privileged=None*, *lxc_conf=None*, *network_mode=None*, *check_is_running=True*, *publish_all_ports=False*, *links=None*, *restart_policy=None*, *cpu_shares=None*, *cpuset=None*, **args*, ***kwargs*)

Ensure that a container is running. If the container does not exist, it will be created from the specified image. (*docker run*)

name / container Name for the container

image Image from which to build this container

environment

Environment variables for the container, either

- a mapping of key, values
- a list of mappings of key, values

ports

List of ports definitions, either:

- a port to map
- a mapping of mapping portInHost : PortInContainer

```
- ports:
  - "5000/tcp":
        HostIp: ""
        HostPort: "5000"
```

publish_all_ports Publish all ports from the port list (default is false, only meaningful if port does not contain portinhost:portincontainer mapping)

volumes List of volumes to mount or create in the container (like `-v` of `docker run` command), mapping host directory to container directory. To create a volume in the container:

```
- volumes:
  - "/var/log/service"
```

For read-write mounting, use the short form (note that the notion of source:target from docker is preserved):

```
- volumes:
  - /var/log/service: /var/log/service
```

Or, to specify read-only mounting, use the extended form:

```
- volumes:
  - /home/user1:
    bind: /mnt/vol2
    ro: true
  - /var/www:
    bind: /mnt/vol1
    ro: false
```

Or (mostly for backwards compatibility) a dict style

```
        - volumes:
          /home/user1:
            bind: /mnt/vol2
            ro: true
          /var/www:
            bind: /mnt/vol1
            ro: false
```

volumes_from List of containers to share volumes with

dns List of DNS servers.

```
        - dns:
            - 127.0.0.1
```

network_mode

- `bridge': creates a new network stack for the container on the docker bridge
- `none': no networking for this container
- `container:[name|id]': reuses another container network stack)
- `host': use the host network stack inside the container

```
        - network_mode: host
```

restart_policy Restart policy to apply when a container exits (no, on-failure[:max-retry], always)

```
        - restart_policy:
            MaximumRetryCount: 5
            Name: on-failure
```

cap_add List of capabilities to add in a container.

cap_drop List of capabilities to drop in a container.

check_is_running Enable checking if a container should run or not. Useful for data-only containers that must be linked to another one. e.g. nginx <- static-files

cpu_shares CPU shares (relative weight)

```
        - cpu_shares: 2
```

cpuset CPUs in which to allow execution (`0-3' or `0,1')

```
        - cpuset: '0-3'
```

For other parameters, see salt.modules.dockerio execution module and the docker-py python bindings for docker documentation <https://github.com/dotcloud/docker-py#api>`_ for *docker.create_container*.

Note: This command does not verify that the named container is running the specified image.

---

salt.states.dockerio.**script**(**args*, ***kw*)

Placeholder function for a cmd.script alike.

Note: Not yet implemented. Its implementation might be very similar from `salt.states.dockerio.run`

22.27.33 salt.states.drac

Management of Dell DRAC

The DRAC module is used to create and manage DRAC cards on Dell servers

Ensure the user damian is present

```
    damian:
      drac.present:
        - name: damian
        - password: secret
        - permission: login,test_alerts,clear_logs
```

Ensure the user damian does not exist

```
    damian:
      drac.absent:
        - name: damian
```

Ensure DRAC network is in a consistent state

`salt.states.drac.`**`absent`**(*name*)
Ensure a user does not exist on the Dell DRAC

name: The users username

`salt.states.drac.`**`network`**(*ip*, *netmask*, *gateway*)
Ensure the DRAC network settings are consistent

`salt.states.drac.`**`present`**(*name*, *password*, *permission*)
Ensure the user exists on the Dell DRAC

name: The users username

password The password used to authenticate

permission The permissions that should be assigned to a user

22.27.34 salt.states.environ

Support for getting and setting the environment variables of the current salt process.

`salt.states.environ.`**`setenv`**(*name*, *value*, *false_unsets=False*, *clear_all=False*, *update_minion=False*)
Set the salt process environment variables.

name The environment key to set. Must be a string.

value Either a string or dict. When string, it will be the value set for the environment key of `name' above. When a dict, each key/value pair represents an environment variable to set.

false_unsets If a key's value is False and false_unsets is True, then the key will be removed from the salt processes environment dict entirely. If a key's value is False and false_unsets is not True, then the key's value will be set to an empty string. Default: False

clear_all USE WITH CAUTION! This option can unset environment variables needed for salt to function properly. If clear_all is True, then any environment variables not defined in the environ dict will be deleted. Default: False

update_minion If True, apply these environ changes to the main salt-minion process. If False, the environ changes will only affect the current salt subprocess. Default: False

CLI Example:

```
    a_string_env:
       environ.set:
         - name: foo
         - value: bar
         - update_minion: True

    a_dict_env:
       environ.set:
         - name: does_not_matter
         - value:
             foo: bar
             baz: quux
```

22.27.35 salt.states.eselect

Management of Gentoo configuration using eselect

A state module to manage Gentoo configuration via eselect

```
profile:
    eselect.set:
        target: hardened/linux/amd64
```

salt.states.eselect.**set_**(*name*, *target*, *module_parameter=None*, *action_parameter=None*)

Verify that the given module is set to the given target

name The name of the module

target The target to be set for this module

module_parameter additional params passed to the defined module

action_parameter additional params passed to the defined action

22.27.36 salt.states.event

Send events through Salt's event system during state runs

salt.states.event.**send**(*name*, *data=None*, *preload=None*, *with_env=False*, *with_grains=False*, *with_pillar=False*, ***kwargs*)

Send an event to the Salt Master

New in version 2014.7.0.

Accepts the same arguments as the `event.send` execution module of the same name.

Example:

```
    # ...snip bunch of states above

    mycompany/mystaterun/status/update:
      event.send:
        - data:
            status: "Half-way through the state run!"

    # ...snip bunch of states below
```

salt.states.event.**wait**(*name*, *sfun=None*)

Fire an event on the Salt master event bus if called from a watch statement

New in version 2014.7.0.

Example:

```
    # Stand up a new web server.
    apache:
      pkg:
        - installed
        - name: httpd
      service:
        - running
        - enable: True
        - name: httpd

    # Notify the load balancer to update the pool once Apache is running.
    refresh_pool:
      event:
        - wait
        - name: mycompany/loadbalancer/pool/update
        - data:
            new_web_server_ip: {{ grains['ipv4'] | first() }}
        - watch:
          - pkg: apache
```

22.27.37 salt.states.file

Operations on regular files, special files, directories, and symlinks

Salt States can aggressively manipulate files on a system. There are a number of ways in which files can be managed.

Regular files can be enforced with the `file.managed` state. This state downloads files from the salt master and places them on the target system. Managed files can be rendered as a jinja, mako, or wempy template, adding a dynamic component to file management. An example of `file.managed` which makes use of the jinja templating system would look like this:

```
/etc/http/conf/http.conf:
  file.managed:
    - source: salt://apache/http.conf
    - user: root
    - group: root
    - mode: 644
    - template: jinja
    - defaults:
        custom_var: "default value"
        other_var: 123
{% if grains['os'] == 'Ubuntu' %}
    - context:
        custom_var: "override"
{% endif %}
```

It is also possible to use the `py renderer` as a templating option. The template would be a python script which would need to contain a function called `run()`, which returns a string. The returned string will be the contents of the managed file. For example:

```
def run():
    lines = ('foo', 'bar', 'baz')
    return '\n\n'.join(lines)
```

Note: The `defaults` and `context` arguments require extra indentation (four spaces instead of the normal two) in order to create a nested dictionary. *More information*.

If using a template, any user-defined template variables in the file defined in `source` must be passed in using the `defaults` and/or `context` arguments. The general best practice is to place default values in `defaults`, with conditional overrides going into `context`, as seen above.

The template will receive a variable `custom_var`, which would be accessed in the template using `{{ custom_var }}`. If the operating system is Ubuntu, the value of the variable `custom_var` would be *override*, otherwise it is the default *default value*

The `source` parameter can be specified as a list. If this is done, then the first file to be matched will be the one that is used. This allows you to have a default file on which to fall back if the desired file does not exist on the salt fileserver. Here's an example:

```
/etc/foo.conf:
  file.managed:
    - source:
      - salt://foo.conf.{{ grains['fqdn'] }}
      - salt://foo.conf.fallback
    - user: foo
    - group: users
    - mode: 644
    - backup: minion
```

Note: Salt supports backing up managed files via the backup option. For more details on this functionality please review the *backup_mode documentation*.

The `source` parameter can also specify a file in another Salt environment. In this example `foo.conf` in the `dev` environment will be used instead.

```
/etc/foo.conf:
  file.managed:
    - source:
      - salt://foo.conf?saltenv=dev
    - user: foo
    - group: users
    - mode: '0644'
```

Warning: When using a mode that includes a leading zero you must wrap the value in single quotes. If the value is not wrapped in quotes it will be read by YAML as an integer and evaluated as an octal.

Special files can be managed via the `mknod` function. This function will create and enforce the permissions on a special file. The function supports the creation of character devices, block devices, and fifo pipes. The function will create the directory structure up to the special file if it is needed on the minion. The function will not overwrite or operate on (change major/minor numbers) existing special files with the exception of user, group, and permissions. In most cases the creation of some special files require root permisisons on the minion. This would require that the minion to be run as the root user. Here is an example of a character device:

```
/var/named/chroot/dev/random:
  file.mknod:
    - ntype: c
    - major: 1
    - minor: 8
    - user: named
    - group: named
    - mode: 660
```

Here is an example of a block device:

```
/var/named/chroot/dev/loop0:
  file.mknod:
    - ntype: b
    - major: 7
    - minor: 0
    - user: named
    - group: named
    - mode: 660
```

Here is an example of a fifo pipe:

```
/var/named/chroot/var/log/logfifo:
  file.mknod:
    - ntype: p
    - user: named
    - group: named
    - mode: 660
```

Directories can be managed via the `directory` function. This function can create and enforce the permissions on a directory. A directory statement will look like this:

```
/srv/stuff/substuf:
  file.directory:
    - user: fred
    - group: users
    - mode: 755
    - makedirs: True
```

If you need to enforce user and/or group ownership or permissions recursively on the directory's contents, you can do so by adding a `recurse` directive:

```
/srv/stuff/substuf:
  file.directory:
    - user: fred
    - group: users
    - mode: 755
    - makedirs: True
    - recurse:
      - user
      - group
      - mode
```

As a default, `mode` will resolve to `dir_mode` and `file_mode`, to specify both directory and file permissions, use this form:

```
/srv/stuff/substuf:
  file.directory:
    - user: fred
    - group: users
    - file_mode: 744
    - dir_mode: 755
    - makedirs: True
    - recurse:
      - user
      - group
      - mode
```

Symlinks can be easily created; the symlink function is very simple and only takes a few arguments:

```
/etc/grub.conf:
  file.symlink:
    - target: /boot/grub/grub.conf
```

Recursive directory management can also be set via the `recurse` function. Recursive directory management allows for a directory on the salt master to be recursively copied down to the minion. This is a great tool for deploying large code and configuration systems. A state using `recurse` would look something like this:

```
/opt/code/flask:
  file.recurse:
    - source: salt://code/flask
    - include_empty: True
```

A more complex `recurse` example:

```
{% set site_user = 'testuser' %}
{% set site_name = 'test_site' %}
{% set project_name = 'test_proj' %}
{% set sites_dir = 'test_dir' %}

django-project:
  file.recurse:
    - name: {{ sites_dir }}/{{ site_name }}/{{ project_name }}
    - user: {{ site_user }}
    - dir_mode: 2775
    - file_mode: '0644'
    - template: jinja
    - source: salt://project/templates_dir
    - include_empty: True
```

`salt.states.file.`**`absent`**(*name*)

Make sure that the named file or directory is absent. If it exists, it will be deleted. This will work to reverse any of the functions in the file state module.

name The path which should be deleted

`salt.states.file.`**`accumulated`**(*name*, *filename*, *text*, ***kwargs*)

Prepare accumulator which can be used in template in file.managed state. Accumulator dictionary becomes available in template. It can also be used in file.blockreplace.

name Accumulator name

filename Filename which would receive this accumulator (see file.managed state documentation about `name`)

text String or list for adding in accumulator

require_in / watch_in One of them required for sure we fill up accumulator before we manage the file. Probably the same as filename

Example:

Given the following:

```
animals_doing_things:
  file.accumulated:
    - filename: /tmp/animal_file.txt
    - text: ' jumps over the lazy dog.'
    - require_in:
      - file: animal_file

animal_file:
  file.managed:
    - name: /tmp/animal_file.txt
    - source: salt://animal_file.txt
    - template: jinja
```

One might write a template for `animal_file.txt` like the following:

```
The quick brown fox{% for animal in accumulator['animals_doing_things'] %}{{ animal }}{% endfor
```

Collectively, the above states and template file will produce:

```
The quick brown fox jumps over the lazy dog.
```

Multiple accumulators can be ``chained'' together.

Note: The `accumulator' data structure is a Python dictionary. Do not expect any loop over the keys in a deterministic order!

`salt.states.file.`**`append`**(*name*, *text=None*, *makedirs=False*, *source=None*, *source_hash=None*, *template='jinja'*, *sources=None*, *source_hashes=None*, *defaults=None*, *context=None*)

Ensure that some text appears at the end of a file.

The text will not be appended if it already exists in the file. A single string of text or a list of strings may be appended.

name The location of the file to append to.

text The text to be appended, which can be a single string or a list of strings.

makedirs If the file is located in a path without a parent directory, then the state will fail. If makedirs is set to True, then the parent directories will be created to facilitate the creation of the named file. Defaults to False.

source A single source file to append. This source file can be hosted on either the salt master server, or on an HTTP or FTP server. Both HTTPS and HTTP are supported as well as downloading directly from Amazon S3 compatible URLs with both pre-configured and automatic IAM credentials (see s3.get state documentation). File retrieval from Openstack Swift object storage is supported via swift://container/object_path URLs (see swift.get documentation).

For files hosted on the salt file server, if the file is located on the master in the directory named spam, and is called eggs, the source string is salt://spam/eggs.

If the file is hosted on an HTTP or FTP server, the source_hash argument is also required.

source_hash

This can be one of the following:

1. a source hash string
2. the URI of a file that contains source hash strings

The function accepts the first encountered long unbroken alphanumeric string of correct length as a valid hash, in order from most secure to least secure:

```
Type    Length
======  ======
sha512     128
sha384      96
sha256      64
sha224      56
sha1        40
md5         32
```

The file can contain several checksums for several files. Each line must contain both the file name and the hash. If no file name is matched, the first hash encountered will be used, otherwise the most secure hash with the correct source file name will be used.

Debian file type `*.dsc` is supported.

Examples:

```
/etc/rc.conf ef6e82e4006dee563d98ada2a2a80a27
sha254c8525aee419eb649f0233be91c151178b30f0dff8ebbdcc8de71b1d5c8bcc06a  /etc/resolv.conf
ead48423703509d37c4a90e6a0d53e143b6fc268
```

Known issues: If the remote server URL has the hash file as an apparent sub-directory of the source file, the module will discover that it has already cached a directory where a file should be cached. For example:

```
tomdroid-src-0.7.3.tar.gz:
  file.managed:
    - name: /tmp/tomdroid-src-0.7.3.tar.gz
    - source: https://launchpad.net/tomdroid/beta/0.7.3/+download/tomdroid-src-0.7.3.ta
    - source_hash: https://launchpad.net/tomdroid/beta/0.7.3/+download/tomdroid-src-0.7
```

template [`jinja`] The named templating engine will be used to render the appended-to file. Defaults to jinja.

sources A list of source files to append. If the files are hosted on an HTTP or FTP server, the source_hashes argument is also required.

source_hashes A list of source_hashes corresponding to the sources list specified in the sources argument.

defaults Default context passed to the template.

context Overrides default context variables passed to the template.

Multi-line example:

```
/etc/motd:
  file.append:
    - text: |
        Thou hadst better eat salt with the Philosophers of Greece,
        than sugar with the Courtiers of Italy.
        - Benjamin Franklin
```

```
```

Multiple lines of text:

```
/etc/motd:
  file.append:
    - text:
      - Trust no one unless you have eaten much salt with him.
      - "Salt is born of the purest of parents: the sun and the sea."
```

Gather text from multiple template files:

```
/etc/motd:
  file:
      - append
      - template: jinja
      - sources:
          - salt://motd/devops-messages.tmpl
          - salt://motd/hr-messages.tmpl
          - salt://motd/general-messages.tmpl
```

New in version 0.9.5.

`salt.states.file.`**`blockreplace`**(*name*, *marker_start='#-- start managed zone --`*, *marker_end='#-- end managed zone --`*, *content=''*, *append_if_not_found=False*, *prepend_if_not_found=False*, *backup='.bak'*, *show_changes=True*)

Maintain an edit in a file in a zone delimited by two line markers

New in version 2014.1.0.

A block of content delimited by comments can help you manage several lines entries without worrying about old entries removal. This can help you maintaining an un-managed file containing manual edits. Note: this function will store two copies of the file in-memory (the original version and the edited version) in order to detect changes and only edit the targeted file if necessary.

name Filesystem path to the file to be edited

marker_start The line content identifying a line as the start of the content block. Note that the whole line containing this marker will be considered, so whitespace or extra content before or after the marker is included in final output

marker_end The line content identifying a line as the end of the content block. Note that the whole line containing this marker will be considered, so whitespace or extra content before or after the marker is included in final output. Note: you can use file.accumulated and target this state. All accumulated data dictionaries content will be added as new lines in the content

content The content to be used between the two lines identified by marker_start and marker_stop

append_if_not_found If markers are not found and set to True then the markers and content will be appended to the file. Default is `False`

prepend_if_not_found If markers are not found and set to True then the markers and content will be prepended to the file. Default is `False`

backup The file extension to use for a backup of the file if any edit is made. Set this to `False` to skip making a backup.

dry_run Don't make any edits to the file

show_changes Output a unified diff of the old file and the new file. If `False` return a boolean if any changes were made

Example of usage with an accumulator and with a variable:

```
{% set myvar = 42 %}
hosts-config-block-{{ myvar }}:
  file.blockreplace:
    - name: /etc/hosts
    - marker_start: "# START managed zone {{ myvar }} -DO-NOT-EDIT-"
    - marker_end: "# END managed zone {{ myvar }} --"
    - content: 'First line of content'
    - append_if_not_found: True
    - backup: '.bak'
    - show_changes: True

hosts-config-block-{{ myvar }}-accumulated1:
  file.accumulated:
    - filename: /etc/hosts
    - name: my-accumulator-{{ myvar }}
    - text: "text 2"
    - require_in:
      - file: hosts-config-block-{{ myvar }}

hosts-config-block-{{ myvar }}-accumulated2:
  file.accumulated:
    - filename: /etc/hosts
    - name: my-accumulator-{{ myvar }}
    - text: |
        text 3
        text 4
    - require_in:
      - file: hosts-config-block-{{ myvar }}
```

will generate and maintain a block of content in `/etc/hosts`:

```
# START managed zone 42 -DO-NOT-EDIT-
First line of content
text 2
text 3
text 4
# END managed zone 42 --
```

`salt.states.file.`**`comment`**(*name*, *regex*, *char='#'*, *backup='.bak'*)

Comment out specified lines in a file.

name The full path to the file to be edited

regex A regular expression used to find the lines that are to be commented; this pattern will be wrapped in parenthesis and will move any preceding/trailing `^` or `$` characters outside the parenthesis (e.g., the pattern `^foo$` will be rewritten as `^(foo)$`) Note that you _need_ the leading ^, otherwise each time you run highstate, another comment char will be inserted.

char [`#`] The character to be inserted at the beginning of a line in order to comment it out

backup [`.bak`] The file will be backed up before edit with this file extension

> **Warning**: This backup will be overwritten each time `sed` / `comment` / `uncomment` is called. Meaning the backup will only be useful after the first invocation.

Usage:

```
    /etc/fstab:
      file.comment:
        - regex: ^bind 127.0.0.1
```

New in version 0.9.5.

salt.states.file.**copy**(*name*, *source*, *force=False*, *makedirs=False*, *preserve=False*, *user=None*, *group=None*, *mode=None*, *subdir=False*, ***kwargs*)

If the source file exists on the system, copy it to the named file. The named file will not be overwritten if it already exists unless the force option is set to True.

name The location of the file to copy to

source The location of the file to copy to the location specified with name

force If the target location is present then the file will not be moved, specify ``force: True'' to overwrite the target file

makedirs If the target subdirectories don't exist create them

preserve New in version 2015.5.0.

Set `preserve: True` to preserve user/group ownership and mode after copying. Default is `False`. If `preseve` is set to `True`, then user/group/mode attributes will be ignored.

user New in version 2015.5.0.

The user to own the copied file, this defaults to the user salt is running as on the minion. If `preserve` is set to `True`, then this will be ignored

group New in version 2015.5.0.

The group to own the copied file, this defaults to the group salt is running as on the minion. If `preserve` is set to `True` or on Windows this will be ignored

mode New in version 2015.5.0.

The permissions to set on the copied file, aka 644, `0775', `4664'. If `preserve` is set to `True`, then this will be ignored. Not supported on Windows

subdir New in version 2015.5.0.

If the name is a directory then place the file inside the named directory

salt.states.file.**directory**(*name*, *user=None*, *group=None*, *recurse=None*, *dir_mode=None*, *file_mode=None*, *makedirs=False*, *clean=False*, *require=None*, *exclude_pat=None*, *follow_symlinks=False*, *force=False*, *backup-name=None*, *allow_symlink=True*, ***kwargs*)

Ensure that a named directory is present and has the right perms

name The location to create or manage a directory

user The user to own the directory; this defaults to the user salt is running as on the minion

group The group ownership set for the directory; this defaults to the group salt is running as on the minion. On Windows, this is ignored

recurse Enforce user/group ownership and mode of directory recursively. Accepts a list of strings representing what you would like to recurse. If `mode` is defined, will recurse on both `file_mode` and `dir_mode` if they are defined. If `ignore_files` or `ignore_dirs` is included, files or directories will be left unchanged respectively. Example:

```
        /var/log/httpd:
            file.directory:
            - user: root
            - group: root
            - dir_mode: 755
            - file_mode: 644
            - recurse:
                - user
                - group
                - mode
```

Leave files or directories unchanged:

```
        /var/log/httpd:
            file.directory:
            - user: root
            - group: root
            - dir_mode: 755
            - file_mode: 644
            - recurse:
                - user
                - group
                - mode
                - ignore_files

        /var/log/httpd:
            file.directory:
            - user: root
            - group: root
            - dir_mode: 755
            - file_mode: 644
            - recurse:
                - user
                - group
                - mode
                - ignore_dirs
```

New in version 2015.5.0.

dir_mode / mode The permissions mode to set any directories created. Not supported on Windows

file_mode The permissions mode to set any files created if `mode' is run in `recurse'. This defaults to dir_mode. Not supported on Windows

makedirs If the directory is located in a path without a parent directory, then the state will fail. If makedirs is set to True, then the parent directories will be created to facilitate the creation of the named file.

clean Make sure that only files that are set up by salt and required by this function are kept. If this option is set then everything in this directory will be deleted unless it is required.

require Require other resources such as packages or files

exclude_pat When `clean' is set to True, exclude this pattern from removal list and preserve in the destination.

follow_symlinks [False] If the desired path is a symlink (or `recurse` is defined and a symlink is encountered while recursing), follow it and check the permissions of the directory/file to which the symlink points.

New in version 2014.1.4.

force If the name of the directory exists and is not a directory and force is set to False, the state will fail. If force is set to True, the file in the way of the directory will be deleted to make room for the directory,

unless backupname is set, then it will be renamed.

New in version 2014.7.0.

backupname If the name of the directory exists and is not a directory, it will be renamed to the backupname. If the backupname already exists and force is False, the state will fail. Otherwise, the backupname will be removed first.

New in version 2014.7.0.

allow_symlink [True] If allow_symlink is True and the specified path is a symlink, it will be allowed to remain if it points to a directory. If allow_symlink is False then the state will fail, unless force is also set to True, in which case it will be removed or renamed, depending on the value of the backupname argument.

New in version 2014.7.0.

`salt.states.file.`**`exists`**(*name*)

Verify that the named file or directory is present or exists. Ensures pre-requisites outside of Salt's purview (e.g., keytabs, private keys, etc.) have been previously satisfied before deployment.

name Absolute path which must exist

`salt.states.file.`**`managed`**(*name*, *source=None*, *source_hash=''*, *user=None*, *group=None*, *mode=None*, *template=None*, *makedirs=False*, *dir_mode=None*, *context=None*, *replace=True*, *defaults=None*, *env=None*, *backup=''*, *show_diff=True*, *create=True*, *contents=None*, *contents_pillar=None*, *contents_grains=None*, *contents_newline=True*, *follow_symlinks=True*, *check_cmd=None*, ***kwargs*)

Manage a given file, this function allows for a file to be downloaded from the salt master and potentially run through a templating system.

name The location of the file to manage

source The source file to download to the minion, this source file can be hosted on either the salt master server, or on an HTTP or FTP server. Both HTTPS and HTTP are supported as well as downloading directly from Amazon S3 compatible URLs with both pre-configured and automatic IAM credentials. (see s3.get state documentation) File retrieval from Openstack Swift object storage is supported via swift://container/object_path URLs, see swift.get documentation. For files hosted on the salt file server, if the file is located on the master in the directory named spam, and is called eggs, the source string is salt://spam/eggs. If source is left blank or None (use ~ in YAML), the file will be created as an empty file and the content will not be managed

If the file is hosted on a HTTP or FTP server then the source_hash argument is also required

A list of sources can also be passed in to provide a default source and a set of fallbacks. The first source in the list that is found to exist will be used and subsequent entries in the list will be ignored.

```
file_override_example:
  file.managed:
    - source:
      - salt://file_that_does_not_exist
      - salt://file_that_exists
```

source_hash

This can be one of the following:

1. a source hash string
2. the URI of a file that contains source hash strings

The function accepts the first encountered long unbroken alphanumeric string of correct length as a valid hash, in order from most secure to least secure:

```
Type    Length
======  ======
sha512     128
sha384      96
sha256      64
sha224      56
sha1        40
md5         32
```

Using a Source Hash File The file can contain several checksums for several files. Each line must contain both the file name and the hash. If no file name is matched, the first hash encountered will be used, otherwise the most secure hash with the correct source file name will be used.

When using a source hash file the source_hash argument needs to be a url, the standard download urls are supported, ftp, http, salt etc:

Example:

```
tomdroid-src-0.7.3.tar.gz:
  file.managed:
    - name: /tmp/tomdroid-src-0.7.3.tar.gz
    - source: https://launchpad.net/tomdroid/beta/0.7.3/+download/tomdroid-src-0.7.3.ta
    - source_hash: https://launchpad.net/tomdroid/beta/0.7.3/+download/tomdroid-src-0.7
```

The following is an example of the supported source_hash format:

```
/etc/rc.conf ef6e82e4006dee563d98ada2a2a80a27
sha254c8525aee419eb649f0233be91c151178b30f0dff8ebbdcc8de71b1d5c8bcc06a  /etc/resolv.con
ead48423703509d37c4a90e6a0d53e143b6fc268
```

Debian file type `*.dsc` files are also supported.

Inserting the Source Hash in the sls Data Examples:

```
tomdroid-src-0.7.3.tar.gz:
  file.managed:
    - name: /tmp/tomdroid-src-0.7.3.tar.gz
    - source: https://launchpad.net/tomdroid/beta/0.7.3/+download/tomdroid-src-0.7.3.ta
    - source_hash: md5=79eef25f9b0b2c642c62b7f737d4f53f
```

Known issues: If the remote server URL has the hash file as an apparent sub-directory of the source file, the module will discover that it has already cached a directory where a file should be cached. For example:

```
tomdroid-src-0.7.3.tar.gz:
  file.managed:
    - name: /tmp/tomdroid-src-0.7.3.tar.gz
    - source: https://launchpad.net/tomdroid/beta/0.7.3/+download/tomdroid-src-0.7.3.ta
    - source_hash: https://launchpad.net/tomdroid/beta/0.7.3/+download/tomdroid-src-0.7
```

user The user to own the file, this defaults to the user salt is running as on the minion

group The group ownership set for the file, this defaults to the group salt is running as on the minion On Windows, this is ignored

mode The permissions to set on this file, aka 644, 0775, 4664. Not supported on Windows

template If this setting is applied then the named templating engine will be used to render the downloaded file, currently jinja, mako, and wempy are supported

makedirs If the file is located in a path without a parent directory, then the state will fail. If makedirs is set to True, then the parent directories will be created to facilitate the creation of the named file.

dir_mode If directories are to be created, passing this option specifies the permissions for those directories. If this is not set, directories will be assigned permissions from the `mode' argument.

replace If this file should be replaced. If false, this command will not overwrite file contents but will enforce permissions if the file exists already. Default is True.

context Overrides default context variables passed to the template.

defaults Default context passed to the template.

backup Overrides the default backup mode for this specific file.

show_diff If set to False, the diff will not be shown.

create Default is True, if create is set to False then the file will only be managed if the file already exists on the system.

contents Default is None. If specified, will use the given string as the contents of the file. Should not be used in conjunction with a source file of any kind. Ignores hashes and does not use a templating engine.

contents_pillar New in version 0.17.0.

Operates like `contents`, but draws from a value stored in pillar, using the pillar path syntax used in `pillar.get`. This is useful when the pillar value contains newlines, as referencing a pillar variable using a jinja/mako template can result in YAML formatting issues due to the newlines causing indentation mismatches.

For example, the following could be used to deploy an SSH private key:

```
/home/deployer/.ssh/id_rsa:
  file.managed:
    - user: deployer
    - group: deployer
    - mode: 600
    - contents_pillar: userdata:deployer:id_rsa
```

This would populate `/home/deployer/.ssh/id_rsa` with the contents of `pillar['userdata']['deployer']['id_rsa']`. An example of this pillar setup would be like so:

```
userdata:
  deployer:
    id_rsa: |
      -----BEGIN RSA PRIVATE KEY-----
      MIIEowIBAAKCAQEAoQiwO3JhBquPAalQF9qP1lLZNXVjYMIswrMe2HcWUVBgh+vY
      U7sCwx/dH6+VvNwmCoqmNnP+8gTPKGl1vgAObJAnMT623dMXjVKwnEagZPRJIxDy
      B/HaAre9euNiY3LvIzBTWRSeMfT+rWvIKVBpvwlgGrfgz70m0pqxu+UyFbAGLin+
      GpxzZAMaFpZw4sSbIlRuissXZj/sHpQb8p9M5IeO4Z3rjkCP1cxI
      -----END RSA PRIVATE KEY-----
```

Note: The private key above is shortened to keep the example brief, but shows how to do multiline string in YAML. The key is followed by a pipe character, and the mutliline string is indented two more spaces.

contents_grains New in version 2014.7.0.

Same as contents_pillar, but with grains

contents_newline New in version 2014.7.0.

When using contents, contents_pillar, or contents_grains, this option ensures the file will have a newline at the end. When loading some data this newline is better left off. Setting contents_newline to False will omit this final newline.

follow_symlinks [True] New in version 2014.7.0.

If the desired path is a symlink follow it and make changes to the file to which the symlink points.

check_cmd New in version 2014.7.0.

The specified command will be run with the managed file as an argument. If the command exits with a nonzero exit code, the command will not be run.

`salt.states.file.`**`missing`**(*name*)

Verify that the named file or directory is missing, this returns True only if the named file is missing but does not remove the file if it is present.

name Absolute path which must NOT exist

`salt.states.file.`**`mknod`**(*name*, *ntype*, *major=0*, *minor=0*, *user=None*, *group=None*, *mode=`0600'*)

Create a special file similar to the `nix mknod command. The supported device types are `p` (fifo pipe), `c` (character device), and `b` (block device). Provide the major and minor numbers when specifying a character device or block device. A fifo pipe does not require this information. The command will create the necessary dirs if needed. If a file of the same name not of the same type/major/minor exists, it will not be overwritten or unlinked (deleted). This is logically in place as a safety measure because you can really shoot yourself in the foot here and it is the behavior of `nix `mknod`. It is also important to note that not just anyone can create special devices. Usually this is only done as root. If the state is executed as none other than root on a minion, you may receive a permission error.

name name of the file

ntype node type `p' (fifo pipe), `c' (character device), or `b' (block device)

major major number of the device does not apply to a fifo pipe

minor minor number of the device does not apply to a fifo pipe

user owning user of the device/pipe

group owning group of the device/pipe

mode permissions on the device/pipe

Usage:

```
/dev/chr:
  file.mknod:
    - ntype: c
    - major: 180
    - minor: 31
    - user: root
    - group: root
    - mode: 660

/dev/blk:
  file.mknod:
    - ntype: b
    - major: 8
    - minor: 999
    - user: root
    - group: root
```

```
        - mode: 660

    /dev/fifo:
      file.mknod:
        - ntype: p
        - user: root
        - group: root
        - mode: 660
```

New in version 0.17.0.

salt.states.file.**mod_run_check_cmd**(*cmd*, *filename*, ***check_cmd_opts*)

Execute the check_cmd logic.

Return a result dict if `check_cmd` succeeds (check_cmd == 0) otherwise return True

salt.states.file.**patch**(*name*, *source=None*, *hash=None*, *options=''*, *dry_run_first=True*, *env=None*, ***kwargs*)

Apply a patch to a file.

Note: A suitable `patch` executable must be available on the minion when using this state function.

---

name The file to with the patch will be applied.

source The source patch to download to the minion, this source file must be hosted on the salt master server. If the file is located in the directory named spam, and is called eggs, the source string is salt://spam/eggs. A source is required.

hash Hash of the patched file. If the hash of the target file matches this value then the patch is assumed to have been applied. The hash string is the hash algorithm followed by the hash of the file: md5=e138491e9d5b97023cea823fe17bac22

options Extra options to pass to patch.

dry_run_first [`True`] Run patch with `--dry-run` first to check if it will apply cleanly.

env Specify the environment from which to retrieve the patch file indicated by the `source` parameter. If not provided, this defaults to the environment from which the state is being executed.

Usage:

```
    # Equivalent to ``patch --forward /opt/file.txt file.patch``
    /opt/file.txt:
      file.patch:
        - source: salt://file.patch
        - hash: md5=e138491e9d5b97023cea823fe17bac22
```

salt.states.file.**prepend**(*name*, *text=None*, *makedirs=False*, *source=None*, *source_hash=None*, *template='jinja'*, *sources=None*, *source_hashes=None*, *defaults=None*, *context=None*)

Ensure that some text appears at the beginning of a file

The text will not be prepended again if it already exists in the file. You may specify a single line of text or a list of lines to append.

Multi-line example:

```
    /etc/motd:
      file.prepend:
        - text: |
```

```
            Thou hadst better eat salt with the Philosophers of Greece,
            than sugar with the Courtiers of Italy.
            - Benjamin Franklin
```

Multiple lines of text:

```
/etc/motd:
  file.prepend:
    - text:
      - Trust no one unless you have eaten much salt with him.
      - "Salt is born of the purest of parents: the sun and the sea."
```

Gather text from multiple template files:

```
/etc/motd:
  file:
      - prepend
      - template: jinja
      - sources:
          - salt://motd/devops-messages.tmpl
          - salt://motd/hr-messages.tmpl
          - salt://motd/general-messages.tmpl
```

New in version 2014.7.0.

salt.states.file.**recurse**(*name*, *source*, *clean=False*, *require=None*, *user=None*, *group=None*, *dir_mode=None*, *file_mode=None*, *sym_mode=None*, *template=None*, *context=None*, *defaults=None*, *env=None*, *include_empty=False*, *backup=''*, *include_pat=None*, *exclude_pat=None*, *maxdepth=None*, *keep_symlinks=False*, *force_symlinks=False*, ***kwargs*)

Recurse through a subdirectory on the master and copy said subdirectory over to the specified path.

name The directory to set the recursion in

source The source directory, this directory is located on the salt master file server and is specified with the salt:// protocol. If the directory is located on the master in the directory named spam, and is called eggs, the source string is salt://spam/eggs

clean Make sure that only files that are set up by salt and required by this function are kept. If this option is set then everything in this directory will be deleted unless it is required.

require Require other resources such as packages or files

user The user to own the directory. This defaults to the user salt is running as on the minion

group The group ownership set for the directory. This defaults to the group salt is running as on the minion. On Windows, this is ignored

dir_mode The permissions mode to set on any directories created. Not supported on Windows

file_mode The permissions mode to set on any files created. Not supported on Windows

sym_mode The permissions mode to set on any symlink created. Not supported on Windows

template If this setting is applied then the named templating engine will be used to render the downloaded file. Supported templates are: *jinja*, *mako* and *wempy*.

Note: The template option is required when recursively applying templates.

---

context Overrides default context variables passed to the template.

defaults Default context passed to the template.

include_empty Set this to True if empty directories should also be created (default is False)

include_pat When copying, include only this pattern from the source. Default is glob match; if prefixed with `E@', then regexp match. Example:

```
- include_pat: hello*       :: glob matches 'hello01', 'hello02'
                               ... but not 'otherhello'
- include_pat: E@hello      :: regexp matches 'otherhello',
                               'hello01' ...
```

exclude_pat Exclude this pattern from the source when copying. If both *include_pat* and *exclude_pat* are supplied, then it will apply conditions cumulatively. i.e. first select based on include_pat, and then within that result apply exclude_pat.

Also, when `clean=True', exclude this pattern from the removal list and preserve in the destination. Example:

```
- exclude_pat: APPDATA*               :: glob matches APPDATA.01,
                                         APPDATA.02,.. for exclusion
- exclude_pat: E@(APPDATA)|(TEMPDATA) :: regexp matches APPDATA
                                         or TEMPDATA for exclusion
```

maxdepth When copying, only copy paths which are of depth *maxdepth* from the source path. Example:

```
- maxdepth: 0      :: Only include files located in the source
                      directory
- maxdepth: 1      :: Only include files located in the source
                      or immediate subdirectories
```

keep_symlinks Keep symlinks when copying from the source. This option will cause the copy operation to terminate at the symlink. If desire behavior similar to rsync, then set this to True.

force_symlinks Force symlink creation. This option will force the symlink creation. If a file or directory is obstructing symlink creation it will be recursively removed so that symlink creation can proceed. This option is usually not needed except in special circumstances.

`salt.states.file.`**`rename`**(*name*, *source*, *force=False*, *makedirs=False*)

If the source file exists on the system, rename it to the named file. The named file will not be overwritten if it already exists unless the force option is set to True.

name The location of the file to rename to

source The location of the file to move to the location specified with name

force If the target location is present then the file will not be moved, specify ``force: True'' to overwrite the target file

makedirs If the target subdirectories don't exist create them

`salt.states.file.`**`replace`**(*name*, *pattern*, *repl*, *count=0*, *flags=0*, *bufsize=1*, *append_if_not_found=False*, *prepend_if_not_found=False*, *not_found_content=None*, *backup='.bak'*, *show_changes=True*)

Maintain an edit in a file.

New in version 0.17.0.

name Filesystem path to the file to be edited.

pattern Python's **`regular expression search<https://docs.python.org/2/library/re.html>`_**.

repl The replacement text.

count Maximum number of pattern occurrences to be replaced.

flags A list of flags defined in the *re module documentation*. Each list item should be a string that will correlate to the human-friendly flag name. E.g., `['IGNORECASE', 'MULTILINE']`. Note: multiline searches must specify `file` as the `bufsize` argument below. Defaults to 0 and can be a list or an int.

bufsize How much of the file to buffer into memory at once. The default value `1` processes one line at a time. The special value `file` may be specified which will read the entire file into memory before processing. Note: multiline searches must specify `file` buffering. Can be an int or a str.

append_if_not_found If pattern is not found and set to `True` then, the content will be appended to the file.

New in version 2014.7.0.

prepend_if_not_found If pattern is not found and set to `True` then, the content will be prepended to the file.

New in version 2014.7.0.

not_found_content Content to use for append/prepend if not found. If `None` (default), uses `repl`. Useful when `repl` uses references to group in pattern.

New in version 2014.7.0.

backup The file extension to use for a backup of the file before editing. Set to `False` to skip making a backup.

show_changes Output a unified diff of the old file and the new file. If `False` return a boolean if any changes were made. Returns a boolean or a string.

For complex regex patterns it can be useful to avoid the need for complex quoting and escape sequences by making use of YAML's multiline string syntax.

```
complex_search_and_replace:
  file.replace:
    # <...snip...>
    - pattern: |
        CentOS \(2.6.32[^\n]+\n\s+root[^\n]+\n\)+
```

salt.states.file.**serialize**(*name*, *dataset=None*, *dataset_pillar=None*, *user=None*, *group=None*, *mode=None*, *env=None*, *backup=''*, *makedirs=False*, *show_diff=True*, *create=True*, *merge_if_exists=False*, ***kwargs*)

Serializes dataset and store it into managed file. Useful for sharing simple configuration files.

name The location of the file to create

dataset The dataset that will be serialized

dataset_pillar Operates like `dataset`, but draws from a value stored in pillar, using the pillar path syntax used in `pillar.get`. This is useful when the pillar value contains newlines, as referencing a pillar variable using a jinja/mako template can result in YAML formatting issues due to the newlines causing indentation mismatches.

New in version FIXME.

formatter Write the data as this format. Supported output formats:

- JSON
- YAML
- Python (via pprint.pformat)

user The user to own the directory, this defaults to the user salt is running as on the minion

group The group ownership set for the directory, this defaults to the group salt is running as on the minion

mode The permissions to set on this file, aka 644, 0775, 4664

backup Overrides the default backup mode for this specific file.

makedirs Create parent directories for destination file.

New in version 2014.1.3.

show_diff If set to False, the diff will not be shown.

create Default is True, if create is set to False then the file will only be managed if the file already exists on the system.

merge_if_exists Default is False, if merge_if_exists is True then the existing file will be parsed and the dataset passed in will be merged with the existing content

New in version 2014.7.0.

For example, this state:

```
/etc/dummy/package.json:
  file.serialize:
    - dataset:
        name: naive
        description: A package using naive versioning
        author: A confused individual <iam@confused.com>
        dependencies:
            express: >= 1.2.0
            optimist: >= 0.1.0
        engine: node 0.4.1
    - formatter: json
```

will manage the file `/etc/dummy/package.json`:

```
{
  "author": "A confused individual <iam@confused.com>",
  "dependencies": {
    "express": ">= 1.2.0",
    "optimist": ">= 0.1.0"
  },
  "description": "A package using naive versioning",
  "engine": "node 0.4.1",
  "name": "naive"
}
```

`salt.states.file.`**`symlink`**(*name*, *target*, *force=False*, *backupname=None*, *makedirs=False*, *user=None*, *group=None*, *mode=None*, ***kwargs*)

Create a symlink

If the file already exists and is a symlink pointing to any location other than the specified target, the symlink will be replaced. If the symlink is a regular file or directory then the state will return False. If the regular file or directory is desired to be replaced with a symlink pass force: True, if it is to be renamed, pass a backupname.

name The location of the symlink to create

target The location that the symlink points to

force If the name of the symlink exists and is not a symlink and force is set to False, the state will fail. If force is set to True, the file or directory in the way of the symlink file will be deleted to make room for the symlink, unless backupname is set, when it will be renamed

backupname If the name of the symlink exists and is not a symlink, it will be renamed to the backupname. If the backupname already exists and force is False, the state will fail. Otherwise, the backupname will be removed first.

makedirs If the location of the symlink does not already have a parent directory then the state will fail, setting makedirs to True will allow Salt to create the parent directory

user The user to own the file, this defaults to the user salt is running as on the minion

group The group ownership set for the file, this defaults to the group salt is running as on the minion. On Windows, this is ignored

mode The permissions to set on this file, aka 644, 0775, 4664. Not supported on Windows

`salt.states.file.touch`(*name*, *atime=None*, *mtime=None*, *makedirs=False*)

Replicate the `nix ``touch'' command to create a new empty file or update the atime and mtime of an existing file.

Note that if you just want to create a file and don't care about atime or mtime, you should use `file.managed` instead, as it is more feature-complete. (Just leave out the `source`/`template`/`contents` arguments, and it will just create the file and/or check its permissions, without messing with contents)

name name of the file

atime atime of the file

mtime mtime of the file

makedirs whether we should create the parent directory/directories in order to touch the file

Usage:

```
/var/log/httpd/logrotate.empty:
  file.touch
```

New in version 0.9.5.

`salt.states.file.uncomment`(*name*, *regex*, *char='#'*, *backup='.bak'*)

Uncomment specified commented lines in a file

name The full path to the file to be edited

regex A regular expression used to find the lines that are to be uncommented. This regex should not include the comment character. A leading `^` character will be stripped for convenience (for easily switching between comment() and uncomment()). The regex will be searched for from the beginning of the line, ignoring leading spaces (we prepend `'^[ t]*'`)

char [#] The character to remove in order to uncomment a line

backup [`.bak`] The file will be backed up before edit with this file extension;

> **Warning**: This backup will be overwritten each time `sed` / `comment` / `uncomment` is called. Meaning the backup will only be useful after the first invocation.

Usage:

```
/etc/adduser.conf:
  file.uncomment:
    - regex: EXTRA_GROUPS
```

New in version 0.9.5.

22.27.38 salt.states.gem

Installation of Ruby modules packaged as gems

A state module to manage rubygems. Gems can be set up to be installed or removed. This module will use RVM or rbenv if they are installed. In that case, you can specify what ruby version and gemset to target.

```
addressable:
  gem.installed:
    - user: rvm
    - ruby: jruby@jgemset
```

`salt.states.gem.`**`installed`**(*name*, *ruby=None*, *gem_bin=None*, *user=None*, *version=None*, *rdoc=False*, *ri=False*, *pre_releases=False*, *proxy=None*)

Make sure that a gem is installed.

name The name of the gem to install

ruby: **None** Only for RVM or rbenv installations: the ruby version and gemset to target.

gem_bin: **None** Custom `gem` command to run instead of the default. Use this to install gems to a non-default ruby install. If you are using rvm or rbenv use the ruby argument instead.

user: **None** The user under which to run the `gem` command

New in version 0.17.0.

version [None] Specify the version to install for the gem. Doesn't play nice with multiple gems at once

rdoc [False] Generate RDoc documentation for the gem(s).

ri [False] Generate RI documentation for the gem(s).

pre_releases [False] Install pre-release version of gem(s) if available.

proxy [None] Use the specified HTTP proxy server for all outgoing traffic. Format: http://hostname[:port]

`salt.states.gem.`**`removed`**(*name*, *ruby=None*, *user=None*, *gem_bin=None*)

Make sure that a gem is not installed.

name The name of the gem to uninstall

gem_bin [None] Full path to `gem` binary to use.

ruby [None] If RVM or rbenv are installed, the ruby version and gemset to use. Ignored if `gem_bin` is specified.

user: **None** The user under which to run the `gem` command

New in version 0.17.0.

22.27.39 salt.states.git

Interaction with Git repositories

Important: Before using git over ssh, make sure your remote host fingerprint exists in ``~/.ssh/known_hosts'' file. To avoid requiring password authentication, it is also possible to pass private keys to use explicitly.

```
https://github.com/saltstack/salt.git:
  git.latest:
    - rev: develop
    - target: /tmp/salt
```

salt.states.git.**config**(*name*, *value*, *repo=None*, *user=None*, *is_global=False*)

New in version 2014.7.0.

Manage a git config setting for a user or repository

name Name of the git config value to set

value Value to set

repo [None] An optional location of a git repository for local operations

user [None] Optional name of a user as whom *git config* will be run

is_global [False] Whether or not to pass the *--global* option to *git config*

Local config example:

```
mylocalrepo:
  git.config:
    - name: user.email
    - value: fester@bestertester.net
    - repo: file://my/path/to/repo
```

Global config example:

```
mylocalrepo:
  git.config:
    - name: user.name
    - value: Esther Bestertester
    - user: ebestertester
    - is_global: True
```

salt.states.git.**latest**(*name*, *rev=None*, *target=None*, *user=None*, *force=None*, *force_checkout=False*, *force_reset=False*, *submodules=False*, *mirror=False*, *bare=False*, *remote_name='origin'*, *always_fetch=False*, *depth=None*, *identity=None*, *https_user=None*, *https_pass=None*, *onlyif=False*, *unless=False*)

Make sure the repository is cloned to the given directory and is up to date

name Address of the remote repository as passed to ``git clone''

rev The remote branch, tag, or revision ID to checkout after clone / before update

target Name of the target directory where repository is about to be cloned

user Name of the user performing repository management operations

New in version 0.17.0.

force Force git to clone into pre-existing directories (deletes contents)

force_checkout Force a checkout even if there might be overwritten changes (Default: False)

force_reset Force the checkout to `--reset hard` to the remote ref (Default: False)

submodules Update submodules on clone or branch change (Default: False)

mirror True if the repository is to be a mirror of the remote repository. This implies bare, and thus is incompatible with rev.

bare True if the repository is to be a bare clone of the remote repository. This is incompatible with rev, as nothing will be checked out.

remote_name defines a different remote name. For the first clone the given name is set to the default remote, else it is just a additional remote. (Default: `origin')

always_fetch If a tag or branch name is used as the rev a fetch will not occur until the tag or branch name changes. Setting this to true will force a fetch to occur. Only applies when rev is set. (Default: False)

depth Defines depth in history when git a clone is needed in order to ensure latest. E.g. `depth: 1` is usefull when deploying from a repository with a long history. Use rev to specify branch. This is not compatible with tags or revision IDs.(Default: `None`)

identity A path on the minion server to a private key to use over SSH

https_user HTTP Basic Auth username for HTTPS (only) clones

https_pass HTTP Basic Auth password for HTTPS (only) clones

onlyif A command to run as a check, run the named command only if the command passed to the `onlyif` option returns true

unless A command to run as a check, only run the named command if the command passed to the `unless` option returns false

---

Note: Clashing ID declarations can be avoided when including different branches from the same git repository in the same sls file by using the `name` declaration. The example below checks out the `gh-pages` and `gh-pages-prod` branches from the same repository into separate directories. The example also sets up the `ssh_known_hosts` ssh key required to perform the git checkout.

```
    gitlab.example.com:
      ssh_known_hosts:
        - present
        - user: root
        - enc: ecdsa
        - fingerprint: 4e:94:b0:54:c1:5b:29:a2:70:0e:e1:a3:51:ee:ee:e3

    git-website-staging:
      git.latest:
        - name: git@gitlab.example.com:user/website.git
        - rev: gh-pages
        - target: /usr/share/nginx/staging
        - identity: /root/.ssh/website_id_rsa
        - require:
            - pkg: git
            - ssh_known_hosts: gitlab.example.com

    git-website-prod:
      git.latest:
        - name: git@gitlab.example.com:user/website.git
        - rev: gh-pages-prod
        - target: /usr/share/nginx/prod
        - identity: /root/.ssh/website_id_rsa
        - require:
            - pkg: git
            - ssh_known_hosts: gitlab.example.com
```

`salt.states.git.`**`mod_run_check`**(*cmd_kwargs*, *onlyif*, *unless*)

Execute the onlyif and unless logic. Return a result dict if: * onlyif failed (onlyif != 0) * unless succeeded (unless == 0) else return True

salt.states.git.**present**(*name*, *bare=True*, *user=None*, *force=False*, *shared=None*)
Make sure the repository is present in the given directory

name Name of the directory where the repository is about to be created

bare Create a bare repository (Default: True)

user Name of the user performing repository management operations

New in version 0.17.0.

force Force-create a new repository into an pre-existing non-git directory (deletes contents)

shared

Specify the permission for sharing, see git-init for details (Default: None)

New in version XXXX.

22.27.40 salt.states.glusterfs

Manage glusterfs pool.

salt.states.glusterfs.**add_volume_bricks**(*name*, *bricks*)
Add brick(s) to an existing volume

name Volume name

bricks List of bricks to add to the volume

myvolume:

glusterfs.add_volume_bricks:

- **bricks**:
  - host1:/srv/gluster/drive1
  - host2:/srv/gluster/drive2

Replicated Volume:

glusterfs.add_volume_bricks:

- name: volume2
- bricks: - host1:/srv/gluster/drive2 - host2:/srv/gluster/drive3

salt.states.glusterfs.**created**(*name*, *bricks*, *stripe=False*, *replica=False*, *device_vg=False*, *transport='tcp'*, *start=False*)
Check if volume already exists

name name of the volume

```
myvolume:
  glusterfs.created:
    - bricks:
        - host1:/srv/gluster/drive1
        - host2:/srv/gluster/drive2

Replicated Volume:
  glusterfs.created:
```

```
        - name: volume2
        - bricks:
          - host1:/srv/gluster/drive2
          - host2:/srv/gluster/drive3
        - replica: 2
        - start: True
```

salt.states.glusterfs.**peered**(*name*)

Check if node is peered.

name The remote host with which to peer.

```
    peer-cluster:
      glusterfs.peered:
        - name: two

    peer-clusters:
      glusterfs.peered:
        - names:
          - one
          - two
          - three
          - four
```

salt.states.glusterfs.**started**(*name*)

Check if volume has been started

name name of the volume

```
    mycluster:
      glusterfs.started: []
```

22.27.41 salt.states.gnomedesktop

Configuration of the GNOME desktop

Control the GNOME settings

```
localdesktop_wm_prefs:
    gnomedesktop.wm_preferences:
        - user: username
        - audible_bell: false
        - action_double_click_titlebar: 'toggle-maximize'
        - visual_bell: true
        - num_workspaces: 6
localdesktop_lockdown:
    gnomedesktop.desktop_lockdown:
        - user: username
        - disable_user_switching: true
localdesktop_interface:
    gnomedesktop.desktop_interface:
        - user: username
        - clock_show_date: true
        - clock_format: 12h
```

salt.states.gnomedesktop.**desktop_interface**(*name*, *user=None*, *automatic_mnemonics=None*, *buttons_have_icons=None*, *can_change_accels=None*, *clock_format=None*, *clock_show_date=None*, *clock_show_seconds=None*, *cursor_blink=None*, *cursor_blink_time=None*, *cursor_blink_timeout=None*, *cursor_size=None*, *cursor_theme=None*, *document_font_name=None*, *enable_animations=None*, *font_name=None*, *gtk_color_palette=None*, *gtk_color_scheme=None*, *gtk_im_module=None*, *gtk_im_preedit_style=None*, *gtk_im_status_style=None*, *gtk_key_theme=None*, *gtk_theme=None*, *gtk_timeout_initial=None*, *gtk_timeout_repeat=None*, *icon_theme=None*, *menubar_accel=None*, *menubar_detachable=None*, *menus_have_icons=None*, *menus_have_tearoff=None*, *monospace_font_name=None*, *show_input_method_menu=None*, *show_unicode_menu=None*, *text_scaling_factor=None*, *toolbar_detachable=None*, *toolbar_icons_size=None*, *toolbar_style=None*, *toolkit_accessibility=None*, ***kwargs*)

desktop_interface: sets values in the org.gnome.desktop.interface schema

salt.states.gnomedesktop.**desktop_lockdown**(*name*, *user=None*, *disable_application_handlers=None*, *disable_command_line=None*, *disable_lock_screen=None*, *disable_log_out=None*, *disable_print_setup=None*, *disable_printing=None*, *disable_save_to_disk=None*, *disable_user_switching=None*, *user_administration_disabled=None*, ***kwargs*)

desktop_lockdown: sets values in the org.gnome.desktop.lockdown schema

salt.states.gnomedesktop.**wm_preferences**(*name*, *user=None*, *action_double_click_titlebar=None*, *action_middle_click_titlebar=None*, *action_right_click_titlebar=None*, *application_based=None*, *audible_bell=None*, *auto_raise=None*, *auto_raise_delay=None*, *button_layout=None*, *disable_workarounds=None*, *focus_mode=None*, *focus_new_windows=None*, *mouse_button_modifier=None*, *num_workspaces=None*, *raise_on_click=None*, *resize_with_right_button=None*, *theme=None*, *titlebar_font=None*, *titlebar_uses_system_font=None*, *visual_bell=None*, *visual_bell_type=None*, *workspace_names=None*, ***kwargs*)

wm_preferences: sets values in the org.gnome.desktop.wm.preferences schema

22.27.42 salt.states.grafana

Manage Grafana Dashboards

This module uses `elasticsearch`, which can be installed via package, or pip.

You can specify elasticsearch hosts directly to the module, or you can use an elasticsearch profile via pillars:

```
mygrafanaprofile:
  hosts:
    - es1.example.com:9200
    - es2.example.com:9200
  index: grafana-dash
```

```
# Basic usage (uses default pillar profile key 'grafana')
Ensure myservice dashboard is managed:
  grafana.dashboard_present:
    - name: myservice
    - dashboard_from_pillar: default
    - rows_from_pillar:
        - systemhealth
        - requests

# Passing hosts in
Ensure myservice dashboard is managed:
  grafana.dashboard_present:
    - name: myservice
    - dashboard_from_pillar: default
    - rows:
        - collapse: false
          editable: true
          height: 150px
          title: System Health
          panels:
            - aliasColors: {}
              id: 200000
              annotate:
                enable: false
              bars: false
```

```
            datasource: null
            editable: true
            error: false
            fill: 7
            grid:
              leftMax: 100
              leftMin: null
              rightMax: null
              rightMin: null
              threshold1: 60
              threshold1Color: rgb(216, 27, 27)
              threshold2: null
              threshold2Color: rgba(234, 112, 112, 0.22)
            leftYAxisLabel: ''
            legend:
              avg: false
              current: false
              max: false
              min: false
              show: false
              total: false
              values: false
            lines: true
            linewidth: 1
            nullPointMode: connected
            percentage: false
            pointradius: 5
            points: false
            renderer: flot
            resolution: 100
            scale: 1
            seriesOverrides: []
            span: 4
            stack: false
            steppedLine: false
            targets:
              - target: cloudwatch.aws.ec2.mysrv.cpuutilization.average
            title: CPU (asg average)
            tooltip:
              query_as_alias: true
              shared: false
              value_type: cumulative
            type: graph
            x-axis: true
            y-axis: true
            y_formats:
              - short
              - short
            zerofill: true
    - rows_from_pillar:
      - systemhealth
      - requests
    - profile:
        hosts:
          - es1.example.com:9200
          - es2.example.com:9200
        index: grafana-dash
```

```
# Using a profile from pillars
Ensure myservice dashboard is managed:
  grafana.dashboard_present:
    - name: myservice
    - dashboard:
        annotations:
          enable: true
          list: []
        editable: true
        hideAllLegends: false
        hideControls: false
        nav:
          - collapse: false
            enable: true
            notice: false
            now: true
            refresh_intervals:
              - 10s
              - 30s
              - 1m
              - 5m
              - 15m
              - 30m
              - 1h
              - 2h
              - 1d
            status: Stable
            time_options:
              - 5m
              - 15m
              - 1h
              - 2h
              - 3h
              - 4h
              - 6h
              - 12h
              - 1d
              - 2d
              - 4d
              - 7d
              - 16d
              - 30d
            type: timepicker
        originalTitle: dockerregistry
        refresh: 1m
        rows: []
        sharedCrosshair: false
        style: dark
        tags: []
        templating:
          enable: true
          list: []
        time:
          from: now-2h
          to: now
        timezone: browser
    - rows_from_pillars:
      - systemhealth
```

```
      - requests
    - profile: mygrafanaprofile
```

The behavior of this module is to create dashboards if they do not exist, to add rows if they do not exist in existing dashboards, and to update rows if they exist in dashboards. The module will not manage rows that are not defined,

allowing users to manage their own custom rows.

`salt.states.grafana.`**`dashboard_absent`**(*name*, *hosts=None*, *profile='grafana'*)

Ensure the named grafana dashboard is deleted.

name Name of the grafana dashboard.

profile A pillar key or dict that contains a list of hosts and an elasticsearch index to use.

`salt.states.grafana.`**`dashboard_present`**(*name*, *dashboard=None*, *dashboard_from_pillar=None*, *rows=None*, *rows_from_pillar=None*, *profile='grafana'*)

Ensure the grafana dashboard exists and is managed.

name Name of the grafana dashboard.

dashboard A dict that defines a dashboard that should be managed.

dashboard_from_pillar A pillar key that contains a grafana dashboard dict. Mutually exclusive with dashboard.

rows A list of grafana rows.

rows_from_pillar A list of pillar keys that contain lists of grafana dashboard rows. Rows defined in the pillars will be appended to the rows defined in the state.

profile A pillar key or dict that contains a list of hosts and an elasticsearch index to use.

22.27.43 salt.states.grains

Manage grains on the minion

This state allows for grains to be set. Grains set or altered this way are stored in the `grains' file on the minions, by default at: /etc/salt/grains

Note: This does NOT override any grains set in the minion file.

`salt.states.grains.`**`absent`**(*name*, *destructive=False*)

New in version 2014.7.0.

Delete a grain from the grains config file

name The grain name

Parameters **`destructive`** -- If destructive is True, delete the entire grain. If destructive is False, set the grain's value to None. Defaults to False.

```
    grain_name:
      grains.absent
```

`salt.states.grains.`**`append`**(*name*, *value*, *convert=False*)

New in version 2014.7.0.

Append a value to a list in the grains config file

name The grain name

value The value to append

Parameters **convert** -- If convert is True, convert non-list contents into a list. If convert is False and the grain contains non-list contents, an error is given. Defaults to False.

```
grain_name:
  grains.append:
    - value: to_be_appended
```

salt.states.grains.**list_absent**(*name*, *value*)

Delete a value from a grain formed as a list.

New in version 2014.1.0.

name The grain name.

value The value to delete from the grain list.

The grain should be list type

```
roles:
  grains.list_absent:
    - value: db
```

For multiple grains, the syntax looks like:

```
roles:
  grains.list_absent:
    - value:
      - web
      - dev
```

salt.states.grains.**list_present**(*name*, *value*)

New in version 2014.1.0.

Ensure the value is present in the list type grain.

name The grain name.

value The value is present in the list type grain.

The grain should be list type

```
roles:
  grains.list_present:
    - value: web
```

For multiple grains, the syntax looks like:

```
roles:
  grains.list_present:
    - value:
      - web
      - dev
```

salt.states.grains.**present**(*name*, *value*)

Ensure that a grain is set

name The grain name

value The value to set on the grain

If the grain with the given name exists, its value is updated to the new value. If the grain does not yet exist, a new grain is set to the given value.

```
    cheese:
      grains.present:
        - value: edam
```

22.27.44 salt.states.group

Management of user groups

The group module is used to create and manage unix group settings, groups can be either present or absent:

```
cheese:
  group.present:
    - gid: 7648
    - system: True
    - addusers:
      - user1
      - users2
    - delusers:
      - foo

cheese:
  group.present:
    - gid: 7648
    - system: True
    - members:
      - foo
      - bar
      - user1
      - user2
```

salt.states.group.**absent**(*name*)

Ensure that the named group is absent

name The name of the group to remove

salt.states.group.**present**(*name*, *gid=None*, *system=False*, *addusers=None*, *delusers=None*, *members=None*)

Ensure that a group is present

name The name of the group to manage

gid The group id to assign to the named group; if left empty, then the next available group id will be assigned

system Whether or not the named group is a system group. This is essentially the `-r' option of `groupadd'.

addusers List of additional users to be added as a group members.

delusers Ensure these user are removed from the group membership.

members Replace existing group members with a list of new members.

Note: Options `members' and `addusers/delusers' are mutually exclusive and can not be used together.

22.27.45 salt.states.hg

Interaction with Mercurial repositories

Before using hg over ssh, make sure the remote host fingerprint already exists in ~/.ssh/known_hosts, and the remote host has this host's public key.

```
https://bitbucket.org/example_user/example_repo:
    hg.latest:
      - rev: tip
      - target: /tmp/example_repo
```

salt.states.hg.**latest**(*name*, *rev=None*, *target=None*, *clean=False*, *user=None*, *identity=None*, *force=False*, *opts=False*)

Make sure the repository is cloned to the given directory and is up to date

name Address of the remote repository as passed to ``hg clone''

rev The remote branch, tag, or revision hash to clone/pull

target Target destination directory path on minion to clone into

clean Force a clean update with -C (Default: False)

user Name of the user performing repository management operations

New in version 0.17.0.

identity Private SSH key on the minion server for authentication (ssh://)

New in version 2015.5.0.

force Force hg to clone into pre-existing directories (deletes contents)

opts Include additional arguments and options to the hg command line

22.27.46 salt.states.hipchat

Send a message to Hipchat

This state is useful for sending messages to Hipchat during state runs.

New in version 2015.5.0.

```
hipchat-message:
  hipchat.send_message:
    - room_id: 123456
    - from_name: SuperAdmin
    - message: 'This state was executed successfully.'
    - api_key: peWcBiMOS9HrZG15peWcBiMOS9HrZG15
    - api_version: v1
```

The api key can be specified in the master or minion configuration like below: .. code-block:: yaml

hipchat: api_key: peWcBiMOS9HrZG15peWcBiMOS9HrZG15 api_version: v1

salt.states.hipchat.**send_message**(*name*, *room_id*, *from_name*, *message*, *api_key=None*, *api_version=None*, *message_color='yellow'*, *notify=False*)

Send a message to a Hipchat room.

```
    hipchat-message:
      hipchat.send_message:
        - room_id: 123456
        - from_name: SuperAdmin
        - message: 'This state was executed successfully.'
        - api_key: peWcBiMOS9HrZG15peWcBiMOS9HrZG15
        - api_version: v1
        - color: green
        - notify: True
```

The following parameters are required:

name The unique name for this event.

room_id The room to send the message to. Can either be the ID or the name.

from_name The name of that is to be shown in the ``from'' field. If not specified, defaults to.

message The message that is to be sent to the Hipchat room.

The following parameters are optional: api_key

The api key for Hipchat to use for authentication, if not specified in the configuration options of master or minion.

api_version The api version for Hipchat to use, if not specified in the configuration options of master or minion.

color The color the Hipchat message should be displayed in. One of the following, default: yellow ``yellow'', ``red'', ``green'', ``purple'', ``gray'', or ``random''.

notify Should a notification in the room be raised.

22.27.47 salt.states.host

Management of addresses and names in hosts file

The `/etc/hosts` file can be managed to contain definitions for specific hosts:

```
salt-master:
  host.present:
    - ip: 192.168.0.42
```

Or using the `names` directive, you can put several names for the same IP. (Do not try one name with space-separated values).

```
server1:
  host.present:
    - ip: 192.168.0.42
    - names:
      - server1
      - florida
```

Note: Changing the `names` in `host.present` does not cause an update to remove the old entry.

```
server1:
  host.present:
    - ip:
      - 192.168.0.42
      - 192.168.0.43
      - 192.168.0.44
    - names:
      - server1
```

salt.states.host.**absent**(*name*, *ip*)

Ensure that the named host is absent

name The host to remove

ip The ip addr(s) of the host to remove

salt.states.host.**present**(*name*, *ip*)

Ensures that the named host is present with the given ip

name The host to assign an ip to

ip The ip addr(s) to apply to the host

22.27.48 salt.states.htpasswd

Support for htpasswd module

New in version 2014.7.0.

```
username:
  webutil.user_exists:
    - password: secr3t
    - htpasswd_file: /etc/nginx/htpasswd
    - options: d
    - force: true
```

salt.states.htpasswd.**user_exists**(*name*, *password=None*, *htpasswd_file=None*, *options='`'*, *force=False*, ***kwargs*)

Make sure the user is inside the `/etc/nginx/htpasswd`

name username

password password of the user

htpasswd_file path to the file that htpasswd will handle

options see `salt.module.htpasswd.useradd`

force touch the file even if user already created

22.27.49 salt.states.http

HTTP monitoring states

Perform an HTTP query and statefully return the result

salt.states.http.**query**(*name*, *match=None*, *match_type='string'*, *status=None*, ***kwargs*)

Perform an HTTP query and statefully return the result

22.27.50 salt.states.incron

Management of incron, the inotify cron

The incron state module allows for user incrontabs to be cleanly managed.

Incron declarations require a number of parameters. The parameters needed to be declared: `path`, `mask`, and `cmd`. The `user` whose incrontab is to be edited also needs to be defined.

When making changes to an existing incron job, the `path` declaration is the unique factor, so if an existing cron that looks like this:

```
Watch for modifications in /home/user:
    incron.present:
        - user: root
        - path: /home/user
        - mask:
            - IN_MODIFY
        - cmd: 'echo "$$ $@"'
```

Is changed to this:

```
Watch for modifications and access in /home/user:
    incron.present:
        - user: root
        - path: /home/user
        - mask:
            - IN_MODIFY
            - IN_ACCESS
        - cmd: 'echo "$$ $@"'
```

Then the existing cron will be updated, but if the cron command is changed, then a new cron job will be added to the user's crontab.

New in version 0.17.0.

`salt.states.incron.`**`absent`**(*name*, *path*, *mask*, *cmd*, *user='root'*)

Verifies that the specified incron job is absent for the specified user; only the name is matched when removing a incron job.

name Unique comment describing the entry

path The path that should be watched

user The name of the user who's crontab needs to be modified, defaults to the root user

mask The mask of events that should be monitored for

cmd The cmd that should be executed

`salt.states.incron.`**`present`**(*name*, *path*, *mask*, *cmd*, *user='root'*)

Verifies that the specified incron job is present for the specified user. For more advanced information about what exactly can be set in the cron timing parameters, check your incron system's documentation. Most Unix-like systems' incron documentation can be found via the incrontab man page: `man 5 incrontab`.

name Unique comment describing the entry

path The path that should be watched

user The name of the user who's crontab needs to be modified, defaults to the root user

mask The mask of events that should be monitored for

cmd The cmd that should be executed

22.27.51 salt.states.influxdb_database

Management of InfluxDB databases

(compatible with InfluxDB version 0.5+)

New in version 2014.7.0.

`salt.states.influxdb_database.`**`absent`**(*name*, *user=None*, *password=None*, *host=None*, *port=None*)

Ensure that the named database is absent

name The name of the database to remove

user The user to connect as (must be able to remove the database)

password The password of the user

host The host to connect to

port The port to connect to

`salt.states.influxdb_database.`**`present`**(*name*, *user=None*, *password=None*, *host=None*, *port=None*)

Ensure that the named database is present

name The name of the database to create

user The user to connect as (must be able to remove the database)

password The password of the user

host The host to connect to

port The port to connect to

22.27.52 salt.states.influxdb_user

Management of InfluxDB users

(compatible with InfluxDB version 0.5+)

New in version 2014.7.0.

`salt.states.influxdb_user.`**`absent`**(*name*, *database=None*, *user=None*, *password=None*, *host=None*, *port=None*)

Ensure that the named cluster admin or database user is absent.

name The name of the user to remove

database The database to remove the user from

user The user to connect as (must be able to remove the user)

password The password of the user

host The host to connect to

port The port to connect to

salt.states.influxdb_user.**present**(*name*, *passwd*, *database=None*, *user=None*, *password=None*, *host=None*, *port=None*)

Ensure that the cluster admin or database user is present.

name The name of the user to manage

passwd The password of the user

database The database to create the user in

user The user to connect as (must be able to create the user)

password The password of the user

host The host to connect to

port The port to connect to

22.27.53 salt.states.ini_manage

Manage ini files

maintainer <akilesh1597@gmail.com>

maturity new

depends re

platform all

use section as DEFAULT_IMPLICIT if your ini file does not have any section for example /etc/sysctl.conf

salt.states.ini_manage.**options_absent**(*name*, *sections=None*)

```
    /home/saltminion/api-paste.ini:
      ini.options_present:
        - sections:
            test:
              - testkey
              - secondoption
            test1:
              - testkey1
```

options present in file and not specified in sections dict will be untouched

changes dict will contain the list of changes made

salt.states.ini_manage.**options_present**(*name*, *sections=None*)

```
    /home/saltminion/api-paste.ini:
      ini.options_present:
        - sections:
            test:
              testkey: 'testval'
              secondoption: 'secondvalue'
            test1:
              testkey1: 'testval121'
```

options present in file and not specified in sections dict will be untouched

changes dict will contain the list of changes made

salt.states.ini_manage.**sections_absent**(*name*, *sections=None*)

```
    /home/saltminion/api-paste.ini:
      ini.sections_absent:
        - sections:
            - test
            - test1
```

options present in file and not specified in sections will be deleted changes dict will contain the sections that changed

salt.states.ini_manage.**sections_present**(*name*, *sections=None*)

```
    /home/saltminion/api-paste.ini:
      ini.sections_present:
        - sections:
            test:
              testkey: testval
              secondoption: secondvalue
            test1:
              testkey1: 'testval121'
```

options present in file and not specified in sections will be deleted changes dict will contain the sections that changed

22.27.54 salt.states.ipset

Management of ipsets

This is an ipset-specific module designed to manage IPSets for use in IPTables Firewalls.

```
setname:
  ipset.set_present:
    - set_type: bitmap:ip
    - range: 192.168.0.0/16
    - comment: True

setname:
  ipset.set_absent:
    - set_type: bitmap:ip
    - range: 192.168.0.0/16
    - comment: True

setname_entries:
  ipset.present:
    - set_name: setname
    - entry: 192.168.0.3
    - comment: Hello
    - require:
        - ipset: baz

setname_entries:
  ipset.present:
    - set_name: setname
    - entry:
```

```
        - 192.168.0.3
        - 192.168.1.3
    - comment: Hello
    - require:
        - ipset: baz

setname_entries:
  ipset.absent:
    - set_name: setname
    - entry:
        - 192.168.0.3
        - 192.168.1.3
    - comment: Hello
    - require:
        - ipset: baz

setname:
  ipset.flush:
```

salt.states.ipset.**absent**(*name*, *entry=None*, *entries=None*, *family='ipv4'*, ***kwargs*)
New in version 2014.7.0.

Remove a entry or entries from a chain

name A user-defined name to call this entry by in another part of a state or formula. This should not be an actual entry.

family Network family, ipv4 or ipv6.

salt.states.ipset.**flush**(*name*, *family='ipv4'*, ***kwargs*)
New in version 2014.7.0.

Flush current ipset set

family Networking family, either ipv4 or ipv6

salt.states.ipset.**present**(*name*, *entry=None*, *family='ipv4'*, ***kwargs*)
New in version 2014.7.0.

Append a entry to a set

name A user-defined name to call this entry by in another part of a state or formula. This should not be an actual entry.

entry A single entry to add to a set or a list of entries to add to a set

family Network family, ipv4 or ipv6.

salt.states.ipset.**set_absent**(*name*, *family='ipv4'*, ***kwargs*)
New in version 2014.7.0.

Verify the set is absent.

family Networking family, either ipv4 or ipv6

salt.states.ipset.**set_present**(*name*, *set_type*, *family='ipv4'*, ***kwargs*)
New in version 2014.7.0.

Verify the chain is exist.

name A user-defined set name.

set_type The type for the set

family Networking family, either ipv4 or ipv6

22.27.55 salt.states.iptables

Management of iptables

This is an iptables-specific module designed to manage Linux firewalls. It is expected that this state module, and other system-specific firewall states, may at some point be deprecated in favor of a more generic *firewall* state.

```
httpd:
  iptables.append:
    - table: filter
    - chain: INPUT
    - jump: ACCEPT
    - match: state
    - connstate: NEW
    - dport: 80
    - proto: tcp
    - sport: 1025:65535
    - save: True

httpd:
  iptables.append:
    - table: filter
    - chain: INPUT
    - jump: ACCEPT
    - match:
        - state
        - comment
    - comment: "Allow HTTP"
    - connstate: NEW
    - dport: 80
    - proto: tcp
    - sport: 1025:65535
    - save: True

httpd:
  iptables.append:
    - table: filter
    - chain: INPUT
    - jump: ACCEPT
    - match:
        - state
        - comment
    - comment: "Allow HTTP"
    - connstate: NEW
    - source: '127.0.0.1'
    - dport: 80
    - proto: tcp
    - sport: 1025:65535
    - save: True

.. Invert Rule
httpd:
  iptables.append:
    - table: filter
```

```
    - chain: INPUT
    - jump: ACCEPT
    - match:
        - state
        - comment
    - comment: "Allow HTTP"
    - connstate: NEW
    - source: '! 127.0.0.1'
    - dport: 80
    - proto: tcp
    - sport: 1025:65535
    - save: True

httpd:
  iptables.append:
    - table: filter
    - chain: INPUT
    - jump: ACCEPT
    - match:
        - state
        - comment
    - comment: "Allow HTTP"
    - connstate: NEW
    - source: 'not 127.0.0.1'
    - dport: 80
    - proto: tcp
    - sport: 1025:65535
    - save: True

httpd:
  iptables.append:
    - table: filter
    - family: ipv6
    - chain: INPUT
    - jump: ACCEPT
    - match: state
    - connstate: NEW
    - dport: 80
    - proto: tcp
    - sport: 1025:65535
    - save: True

httpd:
  iptables.append:
    - table: filter
    - family: ipv4
    - chain: INPUT
    - jump: ACCEPT
    - match: state
    - connstate: NEW
    - dports:
        - 80
        - 443
    - proto: tcp
    - sport: 1025:65535
    - save: True

httpd:
```

```
  iptables.insert:
    - position: 1
    - table: filter
    - chain: INPUT
    - jump: ACCEPT
    - match: state
    - connstate: NEW
    - dport: 80
    - proto: tcp
    - sport: 1025:65535
    - save: True

httpd:
  iptables.insert:
    - position: 1
    - table: filter
    - family: ipv6
    - chain: INPUT
    - jump: ACCEPT
    - match: state
    - connstate: NEW
    - dport: 80
    - proto: tcp
    - sport: 1025:65535
    - save: True

httpd:
  iptables.delete:
    - table: filter
    - chain: INPUT
    - jump: ACCEPT
    - match: state
    - connstate: NEW
    - dport: 80
    - proto: tcp
    - sport: 1025:65535
    - save: True

httpd:
  iptables.delete:
    - position: 1
    - table: filter
    - chain: INPUT
    - jump: ACCEPT
    - match: state
    - connstate: NEW
    - dport: 80
    - proto: tcp
    - sport: 1025:65535
    - save: True

httpd:
  iptables.delete:
    - table: filter
    - family: ipv6
    - chain: INPUT
    - jump: ACCEPT
    - match: state
```

```
    - connstate: NEW
    - dport: 80
    - proto: tcp
    - sport: 1025:65535
    - save: True

default to accept:
  iptables.set_policy:
    - chain: INPUT
    - policy: ACCEPT
```

Note: Various functions of the `iptables` module use the `--check` option. If the version of `iptables` on the target system does not include this option, an alternate version of this check will be performed using the output of iptables-save. This may have unintended consequences on legacy releases of `iptables`.

`salt.states.iptables.`**`append`**(*name*, *family='ipv4'*, ***kwargs*)

New in version 0.17.0.

Append a rule to a chain

name A user-defined name to call this rule by in another part of a state or formula. This should not be an actual rule.

family Network family, ipv4 or ipv6.

All other arguments are passed in with the same name as the long option that would normally be used for iptables, with one exception: `--state` is specified as *connstate* instead of *state* (not to be confused with *ctstate*).

`salt.states.iptables.`**`chain_absent`**(*name*, *table='filter'*, *family='ipv4'*)

New in version 2014.1.0.

Verify the chain is absent.

family Networking family, either ipv4 or ipv6

`salt.states.iptables.`**`chain_present`**(*name*, *table='filter'*, *family='ipv4'*)

New in version 2014.1.0.

Verify the chain is exist.

name A user-defined chain name.

table The table to own the chain.

family Networking family, either ipv4 or ipv6

`salt.states.iptables.`**`delete`**(*name*, *family='ipv4'*, ***kwargs*)

New in version 2014.1.0.

Delete a rule to a chain

name A user-defined name to call this rule by in another part of a state or formula. This should not be an actual rule.

family Networking family, either ipv4 or ipv6

All other arguments are passed in with the same name as the long option that would normally be used for iptables, with one exception: `--state` is specified as *connstate* instead of *state* (not to be confused with *ctstate*).

salt.states.iptables.**flush**(*name*, *family='ipv4'*, ***kwargs*)
New in version 2014.1.0.

Flush current iptables state

family Networking family, either ipv4 or ipv6

salt.states.iptables.**insert**(*name*, *family='ipv4'*, ***kwargs*)
New in version 2014.1.0.

Insert a rule into a chain

name A user-defined name to call this rule by in another part of a state or formula. This should not be an actual rule.

family Networking family, either ipv4 or ipv6

All other arguments are passed in with the same name as the long option that would normally be used for iptables, with one exception: `--state` is specified as *connstate* instead of *state* (not to be confused with *ctstate*).

salt.states.iptables.**mod_aggregate**(*low*, *chunks*, *running*)
The mod_aggregate function which looks up all rules in the available low chunks and merges them into a single rules ref in the present low data

salt.states.iptables.**set_policy**(*name*, *family='ipv4'*, ***kwargs*)
New in version 2014.1.0.

Sets the default policy for iptables firewall tables

family Networking family, either ipv4 or ipv6

policy The requested table policy

22.27.56 salt.states.jboss7

Manage JBoss 7 Application Server via CLI interface

This state uses jboss-cli.sh script from JBoss installation and parses its output to determine execution result.

In order to run each state, jboss_config dictionary with the following properties must be passed:

```
jboss:
   cli_path: '/opt/jboss/jboss-7.0/bin/jboss-cli.sh'
   controller: 10.11.12.13:9999
   cli_user: 'jbossadm'
   cli_password: 'jbossadm'
```

If controller doesn't require password, then passing cli_user and cli_password parameters is not obligatory.

Example of application deployment:

```
application_deployed:
 jboss7.deployed:
  - artifact:
      artifactory_url: http://artifactory.intranet.company.com/artifactory
      repository: 'ext-release-local'
      artifact_id: 'webcomponent'
      group_id: 'com.company.application'
      packaging: 'war'
      version: '0.1'
```

```
      target_dir: '/tmp'
   - jboss_config:
      cli_path: '/opt/jboss/jboss-7.0/bin/jboss-cli.sh'
      controller: 10.11.12.13:9999
      cli_user: 'jbossadm'
      cli_password: 'jbossadm'
```

Since same dictionary with configuration will be used in all the states, it is much more convenient to move jboss configuration and other properties to pillar. For example, configuration of jboss server, artifactory address and application version could be moved to pillars:

```
application_deployed:
 jboss7.deployed:
  - artifact:
      artifactory_url: {{ pillar['artifactory']['url'] }}
      repository: {{ pillar['artifactory']['repository'] }}
      artifact_id: 'webcomponent'
      group_id: 'com.company.application'
      packaging: 'war'
      version: {{ pillar['webcomponent-artifact']['version'] }}
      latest_snapshot: {{ pillar['webcomponent-artifact']['latest_snapshot'] }}
      repository: {{ pillar['webcomponent-artifact']['repository'] }}
  - jboss_config: {{ pillar['jboss'] }}
```

Configuration in pillars:

```
artifactory:
   url: 'http://artifactory.intranet.company.com/artifactory'
   repository: 'libs-snapshots-local'

webcomponent-artifact:
   repository: 'libs-snapshots-local'
   latest_snapshot: True
   version: -1 #If latest_snapshot then version is ignored
```

For the sake of brevity, examples for each state assume that jboss_config is moved to pillars.

`salt.states.jboss7.`**`bindings_exist`**(*name*, *jboss_config*, *bindings*)

Ensures that given JNDI binding are present on the server. If a binding doesn't exist on the server it will be created. If it already exists its value will be changed.

jboss_config: Dict with connection properties (see state description)

bindings: Dict with bindings to set.

Example:

```
    jndi_entries_created:
      jboss7.bindings_exist:
       - bindings:
          'java:global/sampleapp/environment': 'DEV'
          'java:global/sampleapp/configurationFile': '/var/opt/sampleapp/config.properties'
       - jboss_config: {{ pillar['jboss'] }}
```

`salt.states.jboss7.`**`datasource_exists`**(*name*, *jboss_config*, *datasource_properties*, *recreate=False*)

Ensures that a datasource with given properties exist on the jboss instance. If datasource doesn't exist, it is created, otherwise only the properties that are different will be updated.

name Datasource property name

jboss_config Dict with connection properties (see state description)

datasource_properties Dict with datasource properties

recreate [False] If set to True and datasource exists it will be removed and created again. However, if there are deployments that depend on the datasource, it will not me possible to remove it.

Example:

```
sampleDS:
  jboss7.datasource_exists:
   - recreate: False
   - datasource_properties:
       driver-name: mysql
       connection-url: 'jdbc:mysql://localhost:3306/sampleDatabase'
       jndi-name: 'java:jboss/datasources/sampleDS'
       user-name: sampleuser
       password: secret
       min-pool-size: 3
       use-java-context: True
   - jboss_config: {{ pillar['jboss'] }}
```

`salt.states.jboss7.`**`deployed`**(*name*, *jboss_config*, *artifact=None*, *salt_source=None*)

Ensures that the given application is deployed on server.

jboss_config: Dict with connection properties (see state description)

artifact:

If set, the artifact will be fetched from artifactory. This is a Dict object with the following properties:

- artifactory_url: Full url to artifactory instance, for example: http://artifactory.intranet.company.com/artifactory
- repository: One of the repositories, for example: libs-snapshots, ext-release-local, etc..
- artifact_id: Artifact ID of the artifact
- group_id: Group ID of the artifact
- packaging: war/jar/ear, etc...
- version: Artifact version. If latest_snapshot is set to True, the value of this attribute will be ignored, and newest snapshot will be taken instead.
- latest_snapshot: If set to True and repository is a snapshot repository it will automatically select the newest snapshot.
- snapshot_version: Exact version of the snapshot (with timestamp). A snapshot version may have several builds and a way to differentiate is to provide a build timestamp.
- target_dir: Temporary directory on minion where artifacts will be downloaded

salt_source:

If set, the artifact to be deployed will be fetched from salt master. This is a Dict object with the following properties:

- source: File on salt master (eg. salt://application-web-0.39.war)
- target_file: Temporary file on minion to save file to (eg. `/tmp/application-web-0.39.war')

- undeploy: Regular expression to match against existing deployments. If any deployment matches the regular expression then it will be undeployed.

The deployment consists of the following steps:

•Fetch artifact (salt filesystem, artifact or filesystem on minion)

•Check if same artifact is not deployed yet (perhaps with different version)

•Undeploy the artifact if it is already deployed

•Deploy the new artifact

Examples:

Deployment of a file from Salt file system:

```
application_deployed:
  jboss7.deployed:
   - salt_source:
        source: salt://application-web-0.39.war
        target_file: '/tmp/application-web-0.39.war'
        undeploy: 'application-web-.*'
   - jboss_config: {{ pillar['jboss'] }}
```

Here, application-web-0.39.war file is downloaded from Salt file system to /tmp/application-web-0.39.war file on minion. Existing deployments are checked if any of them matches `application-web-.*' regular expression, and if so then it is undeployed before deploying the application. This is useful to automate deployment of new application versions.

JBoss state is capable of deploying artifacts directly from Artifactory repository. Here are some examples of deployments:

1.Deployment of released version of artifact from Artifactory.

```
application_deployed:
  jboss7.deployed:
   - artifact:
       artifactory_url: http://artifactory.intranet.company.com/artifactory
       repository: 'ext-release-local'
       artifact_id: 'webcomponent'
       group_id: 'com.company.application'
       packaging: 'war'
       version: '0.1'
       target_dir: '/tmp'
    - jboss_config: {{ pillar['jboss'] }}
```

This performs the following operations:

•Download artifact from artifactory. In the example above the artifact will be fetched from: http://artifactory.intranet.company.com/artifactory/ext-release-local/com/company/application/webcomponent/0.1/webcomponent-0.1.war
As a rule, for released versions the artifacts are downloaded from: artifactory_url/repository/group_id_with_slashed_instead_of_dots/artifact_id/version/artifact_id-version.packaging' This follows artifactory convention for artifact resolution. By default the artifact will be downloaded to /tmp directory on minion.

•Connect to JBoss via controller (defined in jboss_config dict) and check if the artifact is not deployed already. In case of artifactory it will check if any deployment's name starts with artifact_id value. If deployment already exists it will be undeployed

•Deploy the downloaded artifact to JBoss via cli interface.

2.Deployment of last updated version of given SNAPSHOT version of artifact from Artifactory.

```
application_deployed:
  jboss7.deployed:
   - artifact:
       artifactory_url: http://artifactory.intranet.company.com/artifactory
       repository: 'ext-snapshot-local'
       artifact_id: 'webcomponent'
       group_id: 'com.company.application'
       packaging: 'war'
       version: '0.1-SNAPSHOT'
    - jboss_config: {{ pillar['jboss'] }}
```

Deploying snapshot version involves an additional step of resolving the exact version of the artifact (including the timestamp), which is not necessary when deploying a release. In the example above first a request will be made to retrieve the update timestamp from: http://artifactory.intranet.company.com/artifactory/ext-snapshot-local/com/company/application/webcomponent/0.1-SNAPSHOT/maven-metadata.xml
Then the artifact will be fetched from http://artifactory.intranet.company.com/artifactory/ext-snapshot-local/com/company/application/webcomponent/0.1-SNAPSHOT/webcomponent-RESOLVED_SNAPSHOT_VERSION.war

---

Note: In order to perform a snapshot deployment you have to:

- Set repository to a snapshot repository.
- Choose a version that ends with ``SNAPSHOT'' string. Snapshot repositories have a different layout and provide some extra information that is needed for deployment of the last or a specific snapshot.

3.Deployment of SNAPSHOT version (with exact timestamp) of artifact from Artifactory.

If you need to deploy an exact version of the snapshot you may provide snapshot_version parameter.

```
application_deployed:
  jboss7.deployed:
   - artifact:
       artifactory_url: http://artifactory.intranet.company.com/artifactory
       repository: 'ext-snapshot-local'
       artifact_id: 'webcomponent'
       group_id: 'com.company.application'
       packaging: 'war'
       version: '0.1-SNAPSHOT'
       snapshot_version: '0.1-20141023.131756-19'
    - jboss_config: {{ pillar['jboss'] }}
```

In this example the artifact will be retrieved from: http://artifactory.intranet.company.com/artifactory/ext-snapshot-local/com/company/application/webcomponent/0.1-SNAPSHOT/webcomponent-0.1-20141023.131756-19.war

4.Deployment of latest snapshot of artifact from Artifactory.

```
application_deployed:
  jboss7.deployed:
   - artifact:
       artifactory_url: http://artifactory.intranet.company.com/artifactory
       repository: 'ext-snapshot-local'
       artifact_id: 'webcomponent'
       group_id: 'com.company.application'
       packaging: 'war'
```

```
            latest_snapshot: True
        - jboss_config: {{ pillar['jboss'] }}
```

Instead of providing an exact version of a snapshot it is sometimes more convenient to get the newest version. If artifact.latest_snapshot is set to True, then the newest snapshot will be downloaded from Artifactory. In this case it is not necessary to specify version. This is particulary useful when integrating with CI tools that will deploy the current snapshot to the Artifactory.

salt.states.jboss7.**reloaded**(*name*, *jboss_config*, *timeout=60*, *interval=5*)
Reloads configuration of jboss server.

jboss_config: Dict with connection properties (see state description)

timeout: Time to wait until jboss is back in running state. Default timeout is 60s.

interval: Interval between state checks. Default interval is 5s. Decreasing the interval may slightly decrease waiting time but be aware that every status check is a call to jboss-cli which is a java process. If interval is smaller than process cleanup time it may easily lead to excessive resource consumption.

This step performs the following operations:

- Ensures that server is in running or reload-required state (by reading server-state attribute)
- Reloads configuration
- Waits for server to reload and be in running state

Example:

```
    configuration_reloaded:
       jboss7.reloaded:
        - jboss_config: {{ pillar['jboss'] }}
```

22.27.57 salt.states.keyboard

Management of keyboard layouts

The keyboard layout can be managed for the system:

```
us:
  keyboard.system
```

Or it can be managed for XOrg:

```
us:
  keyboard.xorg
```

salt.states.keyboard.**system**(*name*)
Set the keyboard layout for the system

name The keyboard layout to use

salt.states.keyboard.**xorg**(*name*)
Set the keyboard layout for XOrg

layout The keyboard layout to use

22.27.58 salt.states.keystone

Management of Keystone users

depends

- keystoneclient Python module

configuration See `salt.modules.keystone` for setup instructions.

```
Keystone tenants:
  keystone.tenant_present:
    - names:
      - admin
      - demo
      - service

Keystone roles:
  keystone.role_present:
    - names:
      - admin
      - Member

admin:
  keystone.user_present:
    - password: R00T_4CC3SS
    - email: admin@domain.com
    - roles:
        admin:   # tenants
          - admin  # roles
        service:
          - admin
          - Member
    - require:
      - keystone: Keystone tenants
      - keystone: Keystone roles

nova:
  keystone.user_present:
    - password: '$up3rn0v4'
    - email: nova@domain.com
    - tenant: service
    - roles:
        service:
          - admin
    - require:
      - keystone: Keystone tenants
      - keystone: Keystone roles

demo:
  keystone.user_present:
    - password: 'd3m0n$trati0n'
    - email: demo@domain.com
    - tenant: demo
    - roles:
        demo:
          - Member
    - require:
```

```
      - keystone: Keystone tenants
      - keystone: Keystone roles

nova service:
  keystone.service_present:
    - name: nova
    - service_type: compute
    - description: OpenStack Compute Service
```

salt.states.keystone.**endpoint_absent**(*name*, *profile=None*, ***connection_args*)
Ensure that the endpoint for a service doesn't exist in Keystone catalog

name The name of the service whose endpoints should not exist

salt.states.keystone.**endpoint_present**(*name*, *publicurl=None*, *internalurl=None*, *adminurl=None*, *region='RegionOne'*, *profile=None*, ***connection_args*)
Ensure the specified endpoints exists for service

name The Service name

public url The public url of service endpoint

internal url The internal url of service endpoint

admin url The admin url of the service endpoint

region The region of the endpoint

salt.states.keystone.**role_absent**(*name*, *profile=None*, ***connection_args*)
Ensure that the keystone role is absent.

name The name of the role that should not exist

salt.states.keystone.**role_present**(*name*, *profile=None*, ***connection_args*)
` Ensures that the keystone role exists

name The name of the role that should be present

salt.states.keystone.**service_absent**(*name*, *profile=None*, ***connection_args*)
Ensure that the service doesn't exist in Keystone catalog

name The name of the service that should not exist

salt.states.keystone.**service_present**(*name*, *service_type*, *description=None*, *profile=None*, ***connection_args*)
Ensure service present in Keystone catalog

name The name of the service

service_type The type of Openstack Service

description (optional) Description of the service

salt.states.keystone.**tenant_absent**(*name*, *profile=None*, ***connection_args*)
Ensure that the keystone tenant is absent.

name The name of the tenant that should not exist

salt.states.keystone.**tenant_present**(*name*, *description=None*, *enabled=True*, *profile=None*, ***connection_args*)
Ensures that the keystone tenant exists

name The name of the tenant to manage

description The description to use for this tenant

enabled Availability state for this tenant

salt.states.keystone.**user_absent**(*name*, *profile=None*, ***connection_args*)
Ensure that the keystone user is absent.

name The name of the user that should not exist

salt.states.keystone.**user_present**(*name*, *password*, *email*, *tenant=None*, *enabled=True*, *roles=None*, *profile=None*, ***connection_args*)
Ensure that the keystone user is present with the specified properties.

name The name of the user to manage

password The password to use for this user

email The email address for this user

tenant The tenant for this user

enabled Availability state for this user

roles The roles the user should have under given tenants. Passed as a dictionary mapping tenant names to a list of roles in this tenant, i.e.:

```
        roles:
            admin:   # tenant
              - admin  # role
            service:
              - admin
              - Member
```

22.27.59 salt.states.kmod

Loading and unloading of kernel modules

The Kernel modules on a system can be managed cleanly with the kmod state module:

```
kvm_amd:
  kmod.present
pcspkr:
  kmod.absent
```

salt.states.kmod.**absent**(*name*, *persist=False*, *comment=True*)
Verify that the named kernel module is not loaded

name The name of the kernel module to verify is not loaded

persist Delete module from `/etc/modules`

comment Don't remove module from `/etc/modules`, only comment it

salt.states.kmod.**present**(*name*, *persist=False*)
Ensure that the specified kernel module is loaded

name The name of the kernel module to verify is loaded

persist Also add module to `/etc/modules`

22.27.60 salt.states.layman

Management of Gentoo Overlays using layman

A state module to manage Gentoo package overlays via layman

```
sunrise:
    layman.present
```

salt.states.layman.**absent**(*name*)
Verify that the overlay is absent

name The name of the overlay to delete

salt.states.layman.**present**(*name*)
Verify that the overlay is present

name The name of the overlay to add

22.27.61 salt.states.libvirt

Manage libvirt certificates

This state uses the external pillar in the master to call for the generation and signing of certificates for systems running libvirt:

```
libvirt_keys:
  libvirt.keys
```

salt.states.libvirt.**keys**(*name*, *basepath='/etc/pki'*)
Manage libvirt keys.

name The name variable used to track the execution

basepath Defaults to `/etc/pki`, this is the root location used for libvirt keys on the hypervisor

22.27.62 salt.states.linux_acl

Linux File Access Control Lists

Ensure a Linux ACL is present

Ensure a Linux ACL does not exist

salt.states.linux_acl.**absent**(*name*, *acl_type*, *acl_name*, *perms*, *recurse=False*)
Ensure a Linux ACL does not exist

salt.states.linux_acl.**present**(*name*, *acl_type*, *acl_name*, *perms*, *recurse=False*)
Ensure a Linux ACL is present

22.27.63 salt.states.locale

Management of languages/locales

The locale can be managed for the system:

```
en_US.UTF-8:
  locale.system
```

salt.states.locale.**present**(*name*)
Generate a locale if it is not present

New in version 2014.7.0.

name The name of the locale to be present

salt.states.locale.**system**(*name*)
Set the locale for the system

name The name of the locale to use

22.27.64 salt.states.lvm

Management of Linux logical volumes

A state module to manage LVMs

```
/dev/sda:
  lvm.pv_present

my_vg:
  lvm.vg_present:
    - devices: /dev/sda

lvroot:
  lvm.lv_present:
    - vgname: my_vg
    - size: 10G
    - stripes: 5
    - stripesize: 8K
```

salt.states.lvm.**lv_absent**(*name*, *vgname=None*)
Remove a given existing logical volume from a named existing volume group

name The logical volume to remove

vgname The volume group name

salt.states.lvm.**lv_present**(*name*, *vgname=None*, *size=None*, *extents=None*, *snapshot=None*, *pv='`*, ***kwargs*)
Create a new logical volume

name The name of the logical volume

vgname The volume group name for this logical volume

size The initial size of the logical volume

extents The number of logical extents to allocate

snapshot The name of the snapshot

pv The physical volume to use

kwargs Any supported options to lvcreate. See `linux_lvm` for more details.

`salt.states.lvm.`**`pv_absent`**(*name*)
Ensure that a Physical Device is not being used by lvm

name The device name to initialize.

`salt.states.lvm.`**`pv_present`**(*name*, ***kwargs*)
Set a physical device to be used as an LVM physical volume

name The device name to initialize.

kwargs Any supported options to pvcreate. See `linux_lvm` for more details.

`salt.states.lvm.`**`vg_absent`**(*name*)
Remove an LVM volume group

name The volume group to remove

`salt.states.lvm.`**`vg_present`**(*name*, *devices=None*, ***kwargs*)
Create an LVM volume group

name The volume group name to create

devices A list of devices that will be added to the volume group

kwargs Any supported options to vgcreate. See `linux_lvm` for more details.

22.27.65 salt.states.lvs_server

Management of LVS (Linux Virtual Server) Real Server

`salt.states.lvs_server.`**`absent`**(*name*, *protocol=None*, *service_address=None*, *server_address=None*)
Ensure the LVS Real Server in specified service is absent.

name The name of the LVS server.

protocol The service protocol(only support `tcp`, `udp` and `fwmark` service).

service_address The LVS service address.

server_address The LVS real server address.

`salt.states.lvs_server.`**`present`**(*name*, *protocol=None*, *service_address=None*, *server_address=None*, *packet_forward_method='dr'*, *weight=1*)
Ensure that the named service is present.

name The LVS server name

protocol The service protocol

service_address The LVS service address

server_address The real server address.

packet_forward_method The LVS packet forwarding method(`dr` for direct routing, `tunnel` for tunneling, `nat` for network access translation).

weight The capacity of a server relative to the others in the pool.

```
    lvsrs:
      lvs_server.present:
        - protocol: tcp
        - service_address: 1.1.1.1:80
        - server_address: 192.168.0.11:8080
        - packet_forward_method: dr
        - weight: 10
```

22.27.66 salt.states.lvs_service

Management of LVS (Linux Virtual Server) Service

salt.states.lvs_service.**absent**(*name*, *protocol=None*, *service_address=None*)
Ensure the LVS service is absent.

name The name of the LVS service

protocol The service protocol

service_address The LVS service address

salt.states.lvs_service.**present**(*name*, *protocol=None*, *service_address=None*, *scheduler='wlc'*)
Ensure that the named service is present.

name The LVS service name

protocol The service protocol

service_address The LVS service address

scheduler Algorithm for allocating TCP connections and UDP datagrams to real servers.

```
    lvstest:
      lvs_service.present:
        - service_address: 1.1.1.1:80
        - protocol: tcp
        - scheduler: rr
```

22.27.67 salt.states.lxc

Manage Linux Containers

salt.states.lxc.**absent**(*name*)
Ensure a container is not present, destroying it if present

name Name of the container to destroy

```
    web01:
      lxc.absent
```

salt.states.lxc.**cloned**(*name*, *orig*, *snapshot=True*, *size=None*, *vgname=None*, *profile=None*)
Deprecated since version 2015.5.0: Use `lxc.present`

salt.states.lxc.**created**(*name*, ***kwargs*)
Deprecated since version 2015.5.0: Use `lxc.present`

salt.states.lxc.**edited_conf**(*name*, *lxc_conf=None*, *lxc_conf_unset=None*)

> **Warning**: This state is unsuitable for setting parameters that appear more than once in an LXC config file, or parameters which must appear in a certain order (such as when configuring more than one network interface). It is slated to be replaced, and as of version 2015.5.0 it is deprecated.

Edit LXC configuration options

```
setconf:
  lxc.edited_conf:
    - name: ubuntu
    - lxc_conf:
        - network.ipv4.ip: 10.0.3.6
    - lxc_conf_unset:
        - lxc.utsname
```

salt.states.lxc.**frozen**(*name*, *start=True*)

New in version 2015.5.0.

Ensure that a container is frozen

Note: This state does not enforce the existence of the named container, it just freezes the container if it is running. To ensure that the named container exists, use `lxc.present`.

---

name The name of the container

start [True] Start container first, if necessary. If `False`, then this state will fail if the container is not running.

```
web01:
  lxc.frozen

web02:
  lxc.frozen:
    - start: False
```

salt.states.lxc.**present**(*name*, *running=None*, *clone_from=None*, *snapshot=False*, *profile=None*, *template=None*, *options=None*, *image=None*, *config=None*, *fstype=None*, *size=None*, *backing=None*, *vgname=None*, *lvname=None*)

Changed in version 2015.5.0: The `lxc.created` state has been renamed to `lxc.present`, and the `lxc.cloned` state has been merged into this state.

Create the named container if it does not exist

name The name of the container to be created

running [False]

- If `True`, ensure that the container is running
- If `False`, ensure that the container is stopped
- If `None`, do nothing with regards to the running state of the container

New in version 2015.5.0.

clone_from Create named container as a clone of the specified container

snapshot [False] Use Copy On Write snapshots (LVM). Only supported with `clone_from`.

profile Profile to use in container creation (see the *LXC Tutorial* for more information). Values in a profile will be overridden by the parameters listed below.

Container Creation Arguments

template The template to use. E.g., `ubuntu' or `fedora'. Conflicts with the `image` argument.

Note: The `download` template requires the following three parameters to be defined in `options`:

- **dist** - The name of the distribution
- **release** - Release name/version
- **arch** - Architecture of the container

The available images can be listed using the `lxc.images` function.

---

options New in version 2015.5.0.

Template-specific options to pass to the lxc-create command. These correspond to the long options (ones beginning with two dashes) that the template script accepts. For example:

```
web01:
  lxc.present:
    - template: download
    - options:
        dist: centos
        release: 6
        arch: amd64
```

Remember to double-indent the options, due to *how PyYAML works*.

image A tar archive to use as the rootfs for the container. Conflicts with the `template` argument.

backing The type of storage to use. Set to `lvm` to use an LVM group. Defaults to filesystem within /var/lib/lxc.

fstype Filesystem type to use on LVM logical volume

size Size of the volume to create. Only applicable if `backing` is set to `lvm`.

vgname [lxc] Name of the LVM volume group in which to create the volume for this container. Only applicable if `backing` is set to `lvm`.

lvname Name of the LVM logical volume in which to create the volume for this container. Only applicable if `backing` is set to `lvm`.

`salt.states.lxc.`**`running`**(*name*, *restart=False*)

Changed in version 2015.5.0: The `lxc.started` state has been renamed to `lxc.running`

Ensure that a container is running

Note: This state does not enforce the existence of the named container, it just starts the container if it is not running. To ensure that the named container exists, use `lxc.present`.

---

name The name of the container

restart [False] Restart container if it is already running

```
web01:
  lxc.running

web02:
  lxc.running:
    - restart: True
```

salt.states.lxc.**set_pass**(*name*, ***kwargs*)

Deprecated since version 2015.5.0.

This state function has been disabled, as it did not conform to design guidelines. Specifically, due to the fact that `lxc.set_password` uses `chpasswd(8)` to set the password, there was no method to make this action idempotent (in other words, the password would be changed every time). This makes this state redundant, since the following state will do the same thing:

```
setpass:
  module.run:
    - name: set_pass
    - m_name: root
    - password: secret
```

salt.states.lxc.**started**(*name*, *restart=False*)

Deprecated since version 2015.5.0: Use `lxc.running`

salt.states.lxc.**stopped**(*name*, *kill=False*)

Ensure that a container is stopped

Note: This state does not enforce the existence of the named container, it just stops the container if it running or frozen. To ensure that the named container exists, use `lxc.present`, or use the `lxc.absent` state to ensure that the container does not exist.

---

name The name of the container

kill [False] Do not wait for the container to stop, kill all tasks in the container. Older LXC versions will stop containers like this irrespective of this argument.

New in version 2015.5.0.

```
web01:
  lxc.stopped
```

22.27.68 salt.states.makeconf

Management of Gentoo make.conf

A state module to manage Gentoo's `make.conf` file

```
makeopts:
  makeconf.present:
    - value: '-j3'
```

salt.states.makeconf.**absent**(*name*)

Verify that the variable is not in the `make.conf`.

name The variable name. This will automatically be converted to upper case since variables in `make.conf` are in upper case

salt.states.makeconf.**present**(*name*, *value=None*, *contains=None*, *excludes=None*)

Verify that the variable is in the `make.conf` and has the provided settings. If value is set, contains and excludes will be ignored.

name The variable name. This will automatically be converted to upper case since variables in `make.conf` are in upper case

value Enforce that the value of the variable is set to the provided value

contains Enforce that the value of the variable contains the provided value

excludes Enforce that the value of the variable does not contain the provided value.

22.27.69 salt.states.mdadm

Managing software RAID with mdadm

A state module for creating or destroying software RAID devices.

```
/dev/md0:
  raid.present:
    - level: 5
    - devices:
      - /dev/xvdd
      - /dev/xvde
      - /dev/xvdf
    - chunk: 256
    - run: True
```

salt.states.mdadm.**absent**(*name*)

Verify that the raid is absent

name The name of raid device to be destroyed

```
    /dev/md0:
      raid:
        - absent
```

salt.states.mdadm.**present**(*name*, *level*, *devices*, ***kwargs*)

Verify that the raid is present

Changed in version 2014.7.0.

name The name of raid device to be created

level The RAID level to use when creating the raid.

devices A list of devices used to build the array.

Example:

```
    /dev/md0:
      raid.present:
        - level: 5
        - devices:
          - /dev/xvdd
          - /dev/xvde
          - /dev/xvdf
        - chunk: 256
        - run: True
```

22.27.70 salt.states.memcached

States for Management of Memcached Keys

New in version 2014.1.0.

`salt.states.memcached.`**`absent`**(*name*, *value=None*, *host=`127.0.0.1'*, *port=11211*, *time=0*)

Ensure that a memcached key is not present.

name The key

value [None] If specified, only ensure that the key is absent if it matches the specified value.

host The memcached server IP address

port The memcached server port

```
foo:
  memcached.absent

bar:
  memcached.absent:
    - host: 10.0.0.1
```

`salt.states.memcached.`**`managed`**(*name*, *value=None*, *host=`127.0.0.1'*, *port=11211*, *time=0*, *min_compress_len=0*)

Manage a memcached key.

name The key to manage

value The value to set for that key

host The memcached server IP address

port The memcached server port

```
foo:
  memcached.managed:
    - value: bar
```

22.27.71 salt.states.modjk

State to control Apache modjk

`salt.states.modjk.`**`worker_activated`**(*name*, *workers=None*, *profile='default'*)

Activate all the workers in the modjk load balancer

Example:

```
loadbalancer:
  modjk.worker_activated:
    - workers:
      - app1
      - app2
```

`salt.states.modjk.`**`worker_disabled`**(*name*, *workers=None*, *profile='default'*)

Disable all the workers in the modjk load balancer

Example:

```
loadbalancer:
  modjk.worker_disabled:
    - workers:
      - app1
      - app2
```

salt.states.modjk.**worker_recover**(*name*, *workers=None*, *profile='default'*)
Recover all the workers in the modjk load balancer

Example:

```
loadbalancer:
  modjk.worker_recover:
    - workers:
      - app1
      - app2
```

salt.states.modjk.**worker_stopped**(*name*, *workers=None*, *profile='default'*)
Stop all the workers in the modjk load balancer

Example:

```
loadbalancer:
  modjk.worker_stopped:
    - workers:
      - app1
      - app2
```

22.27.72 salt.states.modjk_worker

Manage modjk workers

Send commands to a **modjk** load balancer via the peer system.

This module can be used with the *prereq* requisite to remove/add the worker from the load balancer before deploying/restarting service.

Mandatory Settings:

- The minion needs to have permission to publish the **modjk**.* functions (see *here* for information on configuring peer publishing permissions)
- The modjk load balancer must be configured as stated in the **modjk** execution module documentation

salt.states.modjk_worker.**activate**(*name*, *lbn*, *target*, *profile='default'*, *expr_form='glob'*)
Activate the named worker from the lbn load balancers at the targeted minions

Example:

```
disable-before-deploy:
  modjk_worker.activate:
    - name: {{ grains['id'] }}
    - lbn: application
    - target: 'roles:balancer'
    - expr_form: grain
```

salt.states.modjk_worker.**disable**(*name*, *lbn*, *target*, *profile='default'*, *expr_form='glob'*)
Disable the named worker from the lbn load balancers at the targeted minions. The worker will get traffic only for current sessions and won't get new ones.

Example:

```
disable-before-deploy:
  modjk_worker.disable:
    - name: {{ grains['id'] }}
    - lbn: application
    - target: 'roles:balancer'
    - expr_form: grain
```

salt.states.modjk_worker.**stop**(*name*, *lbn*, *target*, *profile='default'*, *expr_form='glob'*)

Stop the named worker from the lbn load balancers at the targeted minions The worker won't get any traffic from the lbn

Example:

```
disable-before-deploy:
  modjk_worker.stop:
    - name: {{ grains['id'] }}
    - lbn: application
    - target: 'roles:balancer'
    - expr_form: grain
```

22.27.73 salt.states.module

Execution of Salt modules from within states

These states allow individual execution module calls to be made via states. To call a single module function use a `module.run` state:

```
mine.send:
  module.run:
    - name: network.interfaces
```

Note that this example is probably unnecessary to use in practice, since the `mine_functions` and `mine_interval` config parameters can be used to schedule updates for the mine (see *here* for more info).

It is sometimes desirable to trigger a function call after a state is executed, for this the `module.wait` state can be used:

```
mine.send:
  module.wait:
    - name: network.interfaces
    - watch:
      - file: /etc/network/interfaces
```

All arguments that the `module` state does not consume are passed through to the execution module function being executed:

```
fetch_out_of_band:
  module.run:
    - name: git.fetch
    - cwd: /path/to/my/repo
    - user: myuser
    - opts: '--all'
```

Due to how the state system works, if a module function accepts an argument called, `name`, then `m_name` must be used to specify that argument, to avoid a collision with the `name` argument. For example:

```
disable_nfs:
  module.run:
    - name: service.disable
    - m_name: nfs
```

Note that some modules read all or some of the arguments from a list of keyword arguments. For example:

```
mine.send:
  module.run:
    - func: network.ip_addrs
    - kwargs:
        interface: eth0
```

`salt.states.module.`**`mod_watch`**(*name*, ***kwargs*)
Run a single module function

`name` The module function to execute

`returner` Specify the returner to send the return of the module execution to

`**kwargs` Pass any arguments needed to execute the function

`salt.states.module.`**`run`**(*name*, ***kwargs*)
Run a single module function

`name` The module function to execute

`returner` Specify the returner to send the return of the module execution to

`**kwargs` Pass any arguments needed to execute the function

`salt.states.module.`**`wait`**(*name*, ***kwargs*)
Run a single module function only if the watch statement calls it

`name` The module function to execute

`**kwargs` Pass any arguments needed to execute the function

Note: Like the `cmd.run` state, this state will return `True` but not actually execute, unless one of the following two things happens:

1. The state has a *watch requisite*, and the state which it is watching changes.
2. Another state has a *watch_in requisite* which references this state, and the state wth the `watch_in` changes.

22.27.74 salt.states.mongodb_database

Management of Mongodb databases

Only deletion is supported, creation doesn't make sense and can be done using mongodb_user.present

`salt.states.mongodb_database.`**`absent`**(*name*, *user=None*, *password=None*, *host=None*, *port=None*)
Ensure that the named database is absent

name The name of the database to remove

user The user to connect as (must be able to create the user)

password The password of the user

host The host to connect to

port The port to connect to

22.27.75 salt.states.mongodb_user

Management of Mongodb users

Note: This module requires PyMongo to be installed.

`salt.states.mongodb_user.`**`absent`**(*name*, *user=None*, *password=None*, *host=None*, *port=None*, *database='admin'*)

Ensure that the named user is absent

name The name of the user to remove

user MongoDB user with sufficient privilege to create the user

password Password for the admin user specified by the `user` parameter

host The hostname/IP address of the MongoDB server

port The port on which MongoDB is listening

database The database from which to remove the user specified by the `name` parameter

`salt.states.mongodb_user.`**`present`**(*name*, *passwd*, *database='admin'*, *user=None*, *password=None*, *host='localhost'*, *port=27017*)

Ensure that the user is present with the specified properties

name The name of the user to manage

passwd The password of the user to manage

user MongoDB user with sufficient privilege to create the user

password Password for the admin user specified with the `user` parameter

host The hostname/IP address of the MongoDB server

port The port on which MongoDB is listening

database The database in which to create the user

Note: If the database doesn't exist, it will be created.

Example:

```
mongouser-myapp:
  mongodb_user.present:
  - name: myapp
  - passwd: password-of-myapp
  # Connect as admin:sekrit
  - user: admin
  - password: sekrit
```

22.27.76 salt.states.mount

Mounting of filesystems

Mount any type of mountable filesystem with the mounted function:

```
/mnt/sdb:
  mount.mounted:
    - device: /dev/sdb1
    - fstype: ext4
    - mkmnt: True
    - opts:
      - defaults

/srv/bigdata:
  mount.mounted:
    - device: UUID=066e0200-2867-4ebe-b9e6-f30026ca2314
    - fstype: xfs
    - opts: nobootwait,noatime,nodiratime,nobarrier,logbufs=8
    - dump: 0
    - pass_num: 2
    - persist: True
    - mkmnt: True
```

`salt.states.mount.`**`mod_watch`**(*name*, *user=None*, ***kwargs*)

The mounted watcher, called to invoke the watch command.

name The name of the mount point

`salt.states.mount.`**`mounted`**(*name*, *device*, *fstype*, *mkmnt=False*, *opts='defaults'*, *dump=0*, *pass_num=0*, *config='/etc/fstab'*, *persist=True*, *mount=True*, *user=None*, *match_on='auto'*)

Verify that a device is mounted

name The path to the location where the device is to be mounted

device The device name, typically the device node, such as `/dev/sdb1` or `UUID=066e0200-2867-4ebe-b9e6-f30026ca2314` or `LABEL=DATA`

fstype The filesystem type, this will be `xfs`, `ext2/3/4` in the case of classic filesystems, and `fuse` in the case of fuse mounts

mkmnt If the mount point is not present then the state will fail, set `mkmnt: True` to create the mount point if it is otherwise not present

opts A list object of options or a comma delimited list

dump The dump value to be passed into the fstab, Default is `0`

pass_num The pass value to be passed into the fstab, Default is `0`

config Set an alternative location for the fstab, Default is `/etc/fstab`

persist Set if the mount should be saved in the fstab, Default is `True`

mount Set if the mount should be mounted immediately, Default is `True`

user The user to own the mount; this defaults to the user salt is running as on the minion

match_on A name or list of fstab properties on which this state should be applied. Default is `auto`, a special value indicating to guess based on fstype. In general, `auto` matches on name for recognized special devices and device otherwise.

salt.states.mount.**swap**(*name*, *persist=True*, *config='/etc/fstab'*)
Activates a swap device

```
/root/swapfile:
  mount.swap
```

Note: `swap` does not currently support LABEL

salt.states.mount.**unmounted**(*name*, *device*, *config='/etc/fstab'*, *persist=False*, *user=None*)
New in version 0.17.0.

Verify that a device is not mounted

name The path to the location where the device is to be unmounted from

New in version 2015.5.0.

device The device to be unmounted.

config Set an alternative location for the fstab, Default is `/etc/fstab`

persist Set if the mount should be purged from the fstab, Default is `False`

user The user to own the mount; this defaults to the user salt is running as on the minion

22.27.77 salt.states.mysql_database

Management of MySQL databases (schemas)

depends

- MySQLdb Python module

configuration See `salt.modules.mysql` for setup instructions.

The mysql_database module is used to create and manage MySQL databases. Databases can be set as either absent or present.

```
frank:
  mysql_database.present
```

salt.states.mysql_database.**absent**(*name*, ***connection_args*)
Ensure that the named database is absent

name The name of the database to remove

salt.states.mysql_database.**present**(*name*, ***connection_args*)
Ensure that the named database is present with the specified properties

name The name of the database to manage

22.27.78 salt.states.mysql_grants

Management of MySQL grants (user permissions)

depends

- MySQLdb Python module

configuration See `salt.modules.mysql` for setup instructions.

The mysql_grants module is used to grant and revoke MySQL permissions.

The `name` you pass in purely symbolic and does not have anything to do with the grant itself.

The `database` parameter needs to specify a `priv_level' in the same specification as defined in the MySQL documentation:

- *
- **
- db_name.*
- db_name.tbl_name
- etc...

```
frank_exampledb:
   mysql_grants.present:
    - grant: select,insert,update
    - database: exampledb.*
    - user: frank
    - host: localhost

frank_otherdb:
  mysql_grants.present:
    - grant: all privileges
    - database: otherdb.*
    - user: frank

restricted_singletable:
  mysql_grants.present:
    - grant: select
    - database: somedb.sometable
    - user: joe
```

`salt.states.mysql_grants.`**`absent`**(*name*, *grant=None*, *database=None*, *user=None*, *host='localhost'*, *grant_option=False*, *escape=True*, ***connection_args*)

Ensure that the grant is absent

name The name (key) of the grant to add

grant The grant priv_type (i.e. select,insert,update OR all privileges)

database The database priv_level (i.e. db.tbl OR db.*)

user The user to apply the grant to

host The network/host that the grant should apply to

`salt.states.mysql_grants.`**`present`**(*name*, *grant=None*, *database=None*, *user=None*, *host='localhost'*, *grant_option=False*, *escape=True*, *revoke_first=False*, *ssl_option=False*, ***connection_args*)

Ensure that the grant is present with the specified properties

name The name (key) of the grant to add

grant The grant priv_type (i.e. select,insert,update OR all privileges)

database The database priv_level (i.e. db.tbl OR db.*)

user The user to apply the grant to

host The network/host that the grant should apply to

grant_option Adds the WITH GRANT OPTION to the defined grant. Default is `False`

escape Defines if the database value gets escaped or not. Default is `True`

revoke_first By default, MySQL will not do anything if you issue a command to grant privileges that are more restrictive than what's already in place. This effectively means that you cannot downgrade permissions without first revoking permissions applied to a db.table/user pair first.

To have Salt forcibly revoke perms before applying a new grant, enable the `revoke_first options.

WARNING: This will *remove* permissions for a database before attempting to apply new permissions. There is no guarantee that new permissions will be applied correctly which can leave your database security in an unknown and potentially dangerous state. Use with caution!

Default is `False`

ssl_option Adds the specified ssl options for the connecting user as requirements for this grant. Value is a list of single-element dicts corresponding to the list of ssl options to use.

Possible key/value pairings for the dicts in the value:

```
- SSL: True
- X509: True
- SUBJECT: <subject>
- ISSUER: <issuer>
- CIPHER: <cipher>
```

The non-boolean ssl options take a string as their values, which should be an appropriate value as specified by the MySQL documentation for these options.

Default is `False` (no ssl options will be used)

22.27.79 salt.states.mysql_query

Execution of MySQL queries

New in version 2014.7.0.

depends

- MySQLdb Python module

configuration See `salt.modules.mysql` for setup instructions.

The mysql_query module is used to execute queries on MySQL databases. Its output may be stored in a file or in a grain.

```
query_id:
  mysql_query.run
    - database: my_database
    - query:    "SELECT * FROM table;"
    - output:   "/tmp/query_id.txt"
```

`salt.states.mysql_query.`**`run`**(*name*, *database*, *query*, *output=None*, *grain=None*, *key=None*, *overwrite=True*, ***connection_args*)

Execute an arbitrary query on the specified database

name Used only as an ID

database The name of the database to execute the query on

query The query to execute

output grain: output in a grain other: the file to store results None: output to the result comment (default)

grain: grain to store the output (need output=grain)

key: the specified grain will be treated as a dictionary, the result of this state will be stored under the specified key.

overwrite: The file or grain will be overwritten if it already exists (default)

22.27.80 salt.states.mysql_user

Management of MySQL users

depends

- MySQLdb Python module

configuration See `salt.modules.mysql` for setup instructions.

```
frank:
  mysql_user.present:
    - host: localhost
    - password: bobcat
```

New in version 0.16.2: Authentication overrides have been added.

The MySQL authentication information specified in the minion config file can be overridden in states using the following arguments: `connection_host`, `connection_port`, `connection_user`, `connection_pass`, `connection_db`, `connection_unix_socket`, `connection_default_file` and `connection_charset`.

```
frank:
  mysql_user.present:
    - host: localhost
    - password: "bob@cat"
    - connection_user: someuser
    - connection_pass: somepass
    - connection_charset: utf8
    - saltenv:
      - LC_ALL: "en_US.utf8"
```

`salt.states.mysql_user.`**`absent`**(*name*, *host='localhost'*, ***connection_args*)

Ensure that the named user is absent

name The name of the user to remove

`salt.states.mysql_user.`**`present`**(*name*, *host='localhost'*, *password=None*, *password_hash=None*, *allow_passwordless=False*, *unix_socket=False*, ***connection_args*)

Ensure that the named user is present with the specified properties. A passwordless user can be configured by omitting `password` and `password_hash`, and setting `allow_passwordless` to `True`.

name The name of the user to manage

host Host for which this user/password combo applies

password The password to use for this user. Will take precedence over the `password_hash` option if both are specified.

password_hash The password in hashed form. Be sure to quote the password because YAML doesn't like the `*`. A password hash can be obtained from the mysql command-line client like so:

```
mysql> SELECT PASSWORD('mypass');
+-------------------------------------------+
| PASSWORD('mypass')                        |
+-------------------------------------------+
| *6C8989366EAF75BB670AD8EA7A7FC1176A95CEF4 |
+-------------------------------------------+
1 row in set (0.00 sec)
```

allow_passwordless If `True`, then `password` and `password_hash` can be omitted to permit a passwordless login.

New in version 0.16.2.

unix_socket If `True` and allow_passwordless is `True`, the unix_socket auth plugin will be used.

22.27.81 salt.states.network

Configuration of network interfaces

The network module is used to create and manage network settings, interfaces can be set as either managed or ignored. By default all interfaces are ignored unless specified.

Note: Prior to version 2014.1.0, only RedHat-based systems (RHEL, CentOS, Scientific Linux, etc.) are supported. Support for Debian/Ubuntu is new in 2014.1.0 and should be considered experimental.

Other platforms are not yet supported.

```
system:
    network.system:
      - enabled: True
      - hostname: server1.example.com
      - gateway: 192.168.0.1
      - gatewaydev: eth0
      - nozeroconf: True
      - nisdomain: example.com
      - require_reboot: True

eth0:
  network.managed:
    - enabled: True
    - type: eth
    - proto: none
    - ipaddr: 10.1.0.1
    - netmask: 255.255.255.0
    - dns:
      - 8.8.8.8
      - 8.8.4.4

routes:
  network.routes:
    - name: eth0
```

```
    - routes:
      - name: secure_network
        ipaddr: 10.2.0.0
        netmask: 255.255.255.0
        gateway: 10.1.0.3
      - name: HQ_network
        ipaddr: 10.100.0.0
        netmask: 255.255.0.0
        gateway: 10.1.0.10

eth2:
  network.managed:
    - enabled: True
    - type: slave
    - master: bond0

eth3:
  network.managed:
    - enabled: True
    - type: slave
    - master: bond0

eth4:
  network.managed:
    - enabled: True
    - type: eth
    - proto: dhcp
    - bridge: br0

bond0:
  network.managed:
    - type: bond
    - ipaddr: 10.1.0.1
    - netmask: 255.255.255.0
    - mode: active-backup
    - proto: static
    - dns:
      - 8.8.8.8
      - 8.8.4.4
    - ipv6:
    - enabled: False
    - slaves: eth2 eth3
    - require:
      - network: eth2
      - network: eth3
    - miimon: 100
    - arp_interval: 250
    - downdelay: 200
    - lacp_rate: fast
    - max_bonds: 1
    - updelay: 0
    - use_carrier: on
    - xmit_hash_policy: layer2
    - mtu: 9000
    - autoneg: on
    - speed: 1000
    - duplex: full
    - rx: on
```

```
    - tx: off
    - sg: on
    - tso: off
    - ufo: off
    - gso: off
    - gro: off
    - lro: off

bond0.2:
  network.managed:
    - type: vlan
    - ipaddr: 10.1.0.2
    - use:
      - network: bond0
    - require:
      - network: bond0

bond0.3:
  network.managed:
    - type: vlan
    - ipaddr: 10.1.0.3
    - use:
      - network: bond0
    - require:
      - network: bond0

bond0.10:
  network.managed:
    - type: vlan
    - ipaddr: 10.1.0.4
    - use:
      - network: bond0
    - require:
      - network: bond0

bond0.12:
  network.managed:
    - type: vlan
    - ipaddr: 10.1.0.5
    - use:
      - network: bond0
    - require:
      - network: bond0
br0:
  network.managed:
    - enabled: True
    - type: bridge
    - proto: dhcp
    - bridge: br0
    - delay: 0
    - ports: eth4
    - bypassfirewall: True
    - use:
      - network: eth4
    - require:
      - network: eth4

system:
```

```
    network.system:
      - enabled: True
      - hostname: server1.example.com
      - gateway: 192.168.0.1
      - gatewaydev: eth0
      - nozeroconf: True
      - nisdomain: example.com
      - require_reboot: True
      - apply_hostname: True

.. note::
    Apply changes to hostname immediately.

.. versionadded:: 2015.5.0
```

Note: When managing bridged interfaces on a Debian or Ubuntu based system, the ports argument is required. Red Hat systems will ignore the argument.

`salt.states.network.`**`managed`**(*name*, *type*, *enabled=True*, ***kwargs*)

Ensure that the named interface is configured properly.

name The name of the interface to manage

type Type of interface and configuration.

enabled Designates the state of this interface.

kwargs The IP parameters for this interface.

`salt.states.network.`**`routes`**(*name*, ***kwargs*)

Manage network interface static routes.

name Interface name to apply the route to.

kwargs Named routes

`salt.states.network.`**`system`**(*name*, ***kwargs*)

Ensure that global network settings are configured properly.

name Custom name to represent this configuration change.

kwargs The global parameters for the system.

22.27.82 salt.states.nftables

Management of nftables

This is an nftables-specific module designed to manage Linux firewalls. It is expected that this state module, and other system-specific firewall states, may at some point be deprecated in favor of a more generic *firewall* state.

```
httpd:
  nftables.append:
    - table: filter
    - chain: input
    - jump: accept
    - match: state
    - connstate: new
    - dport: 80
```

```
    - proto: tcp
    - sport: 1025:65535
    - save: True

httpd:
  nftables.append:
    - table: filter
    - family: ipv6
    - chain: INPUT
    - jump: ACCEPT
    - match: state
    - connstate: NEW
    - dport: 80
    - proto: tcp
    - sport: 1025:65535
    - save: True

httpd:
  nftables.insert:
    - position: 1
    - table: filter
    - chain: INPUT
    - jump: ACCEPT
    - match: state
    - connstate: NEW
    - dport: 80
    - proto: tcp
    - sport: 1025:65535
    - save: True

httpd:
  nftables.insert:
    - position: 1
    - table: filter
    - family: ipv6
    - chain: INPUT
    - jump: ACCEPT
    - match: state
    - connstate: NEW
    - dport: 80
    - proto: tcp
    - sport: 1025:65535
    - save: True

httpd:
  nftables.delete:
    - table: filter
    - chain: INPUT
    - jump: ACCEPT
    - match: state
    - connstate: NEW
    - dport: 80
    - proto: tcp
    - sport: 1025:65535
    - save: True

httpd:
  nftables.delete:
```

```
    - position: 1
    - table: filter
    - chain: INPUT
    - jump: ACCEPT
    - match: state
    - connstate: NEW
    - dport: 80
    - proto: tcp
    - sport: 1025:65535
    - save: True

httpd:
  nftables.delete:
    - table: filter
    - family: ipv6
    - chain: INPUT
    - jump: ACCEPT
    - match: state
    - connstate: NEW
    - dport: 80
    - proto: tcp
    - sport: 1025:65535
    - save: True
```

salt.states.nftables.**append**(*name*, *family='ipv4'*, ***kwargs*)

New in version 0.17.0.

Append a rule to a chain

name A user-defined name to call this rule by in another part of a state or formula. This should not be an actual rule.

family Network family, ipv4 or ipv6.

All other arguments are passed in with the same name as the long option that would normally be used for nftables, with one exception: *--state* is specified as *connstate* instead of *state* (not to be confused with *ctstate*).

salt.states.nftables.**chain_absent**(*name*, *table='filter'*, *family='ipv4'*)

New in version 2014.7.0.

Verify the chain is absent.

family Networking family, either ipv4 or ipv6

salt.states.nftables.**chain_present**(*name*, *table='filter'*, *table_type=None*, *hook=None*, *priority=None*, *family='ipv4'*)

New in version 2014.7.0.

Verify the chain is exist.

name A user-defined chain name.

table The table to own the chain.

family Networking family, either ipv4 or ipv6

salt.states.nftables.**delete**(*name*, *family='ipv4'*, ***kwargs*)

New in version 2014.7.0.

Delete a rule to a chain

name A user-defined name to call this rule by in another part of a state or formula. This should not be an actual rule.

family Networking family, either ipv4 or ipv6

All other arguments are passed in with the same name as the long option that would normally be used for nftables, with one exception: *--state* is specified as *connstate* instead of *state* (not to be confused with *ctstate*).

salt.states.nftables.**flush**(*name*, *family='ipv4'*, ***kwargs*)
New in version 2014.7.0.

Flush current nftables state

family Networking family, either ipv4 or ipv6

salt.states.nftables.**insert**(*name*, *family='ipv4'*, ***kwargs*)
New in version 2014.7.0.

Insert a rule into a chain

name A user-defined name to call this rule by in another part of a state or formula. This should not be an actual rule.

family Networking family, either ipv4 or ipv6

All other arguments are passed in with the same name as the long option that would normally be used for nftables, with one exception: *--state* is specified as *connstate* instead of *state* (not to be confused with *ctstate*).

22.27.83 salt.states.npm

Installation of NPM Packages

These states manage the installed packages for node.js using the Node Package Manager (npm). Note that npm must be installed for these states to be available, so npm states should include a requisite to a pkg.installed state for the package which provides npm (simply `npm` in most cases). Example:

```
npm:
  pkg.installed

yaml:
  npm.installed:
    - require:
      - pkg: npm
```

salt.states.npm.**bootstrap**(*name*, *user=None*)
Bootstraps a node.js application.

Will execute `npm install --json' on the specified directory.

user The user to run NPM with

New in version 0.17.0.

salt.states.npm.**installed**(*name*, *pkgs=None*, *dir=None*, *user=None*, *force_reinstall=False*, *registry=None*, *env=None*)
Verify that the given package is installed and is at the correct version (if specified).

```
    coffee-script:
      npm.installed:
        - user: someuser

    coffee-script@1.0.1:
      npm.installed: []
```

name The package to install

Changed in version 2014.7.2: This parameter is no longer lowercased by salt so that case-sensitive NPM package names will work.

pkgs A list of packages to install with a single npm invocation; specifying this argument will ignore the `name` argument

New in version 2014.7.0.

dir The target directory in which to install the package, or None for global installation

user The user to run NPM with

New in version 0.17.0.

registry The NPM registry from which to install the package

New in version 2014.7.0.

env A list of environment variables to be set prior to execution. The format is the same as the `cmd.run` state function.

New in version 2014.7.0.

force_reinstall Install the package even if it is already installed

`salt.states.npm.`**`removed`**(*name*, *dir=None*, *user=None*)

Verify that the given package is not installed.

dir The target directory in which to install the package, or None for global installation

user The user to run NPM with

New in version 0.17.0.

22.27.84 salt.states.ntp

Management of NTP servers

New in version 2014.1.0.

This state is used to manage NTP servers. Currently only Windows is supported.

```
win_ntp:
  ntp.managed:
    - servers:
      - pool.ntp.org
      - us.pool.ntp.org
```

`salt.states.ntp.`**`managed`**(*name*, *servers=None*)

Manage NTP servers

servers A list of NTP servers

22.27.85 salt.states.openstack_config

Manage OpenStack configuration file settings.

maintainer Jeffrey C. Ollie <jeff@ocjtech.us>

maturity new

depends

platform linux

salt.states.openstack_config.**absent**(*name*, *filename*, *section*, *parameter=None*)
Ensure a value is not set in an OpenStack configuration file.

filename The full path to the configuration file

section The section in which the parameter will be set

parameter (optional) The parameter to change. If the parameter is not supplied, the name will be used as the parameter.

salt.states.openstack_config.**present**(*name*, *filename*, *section*, *value*, *parameter=None*)
Ensure a value is set in an OpenStack configuration file.

filename The full path to the configuration file

section The section in which the parameter will be set

parameter (optional) The parameter to change. If the parameter is not supplied, the name will be used as the parameter.

value The value to set

22.27.86 salt.states.pagerduty

Create an Event in PagerDuty

New in version 2014.1.0.

This state is useful for creating events on the PagerDuty service during state runs.

```
server-warning-message:
  pagerduty.create_event:
    - name: 'This is a server warning message'
    - details: 'This is a much more detailed message'
    - service_key: 9abcd123456789efabcde362783cdbaf
    - profile: my-pagerduty-account
```

salt.states.pagerduty.**create_event**(*name*, *details*, *service_key*, *profile*)
Create an event on the PagerDuty service

```
    server-warning-message:
      pagerduty.create_event:
        - name: 'This is a server warning message'
        - details: 'This is a much more detailed message'
        - service_key: 9abcd123456789efabcde362783cdbaf
        - profile: my-pagerduty-account
```

The following parameters are required:

name This is a short description of the event.

details This can be a more detailed description of the event.

service_key This key can be found by using pagerduty.list_services.

profile This refers to the configuration profile to use to connect to the PagerDuty service.

22.27.87 salt.states.pecl

Installation of PHP Extensions Using pecl

These states manage the installed pecl extensions. Note that php-pear must be installed for these states to be available, so pecl states should include a requisite to a pkg.installed state for the package which provides pecl (`php-pear` in most cases). Example:

```
php-pear:
  pkg.installed

mongo:
  pecl.installed:
    - require:
      - pkg: php-pear
```

`salt.states.pecl.`**`installed`**(*name*, *version=None*, *defaults=False*, *force=False*, *preferred_state='stable'*)

New in version 0.17.0.

Make sure that a pecl extension is installed.

name The pecl extension name to install

version The pecl extension version to install. This option may be ignored to install the latest stable version.

defaults Use default answers for extensions such as pecl_http which ask questions before installation. Without this option, the pecl.installed state will hang indefinitely when trying to install these extensions.

force Whether to force the installed version or not

preferred_state The pecl extension state to install

`salt.states.pecl.`**`removed`**(*name*)

Make sure that a pecl extension is not installed.

name The pecl extension name to uninstall

22.27.88 salt.states.pip_state

Installation of Python Packages Using pip

These states manage system installed python packages. Note that pip must be installed for these states to be available, so pip states should include a requisite to a pkg.installed state for the package which provides pip (`python-pip` in most cases). Example:

```
python-pip:
  pkg.installed

virtualenvwrapper:
  pip.installed:
    - require:
      - pkg: python-pip
```

salt.states.pip_state.**installed**(*name*, *pkgs=None*, *pip_bin=None*, *requirements=None*, *env=None*, *bin_env=None*, *use_wheel=False*, *no_use_wheel=False*, *log=None*, *proxy=None*, *timeout=None*, *repo=None*, *editable=None*, *find_links=None*, *index_url=None*, *extra_index_url=None*, *no_index=False*, *mirrors=None*, *build=None*, *target=None*, *download=None*, *download_cache=None*, *source=None*, *upgrade=False*, *force_reinstall=False*, *ignore_installed=False*, *exists_action=None*, *no_deps=False*, *no_install=False*, *no_download=False*, *install_options=None*, *global_options=None*, *user=None*, *no_chown=False*, *cwd=None*, *activate=False*, *pre_releases=False*, *cert=None*, *allow_all_external=False*, *allow_external=None*, *allow_unverified=None*, *process_dependency_links=False*, *env_vars=None*, *use_vt=False*)

Make sure the package is installed

name The name of the python package to install. You can also specify version numbers here using the standard operators `==, >=, <=`. If `requirements` is given, this parameter will be ignored.

Example:

```
django:
  pip.installed:
    - name: django >= 1.6, <= 1.7
    - require:
      - pkg: python-pip
```

This will install the latest Django version greater than 1.6 but less than 1.7.

requirements Path to a pip requirements file. If the path begins with salt:// the file will be transferred from the master file server.

user The user under which to run pip

use_wheel [False] Prefer wheel archives (requires pip>=1.4)

no_use_wheel [False] Force to not use wheel archives (requires pip>=1.4)

log Log file where a complete (maximum verbosity) record will be kept

proxy Specify a proxy in the form user:passwd@proxy.server:port. Note that the user:password@ is optional and required only if you are behind an authenticated proxy. If you provide user@proxy.server:port then you will be prompted for a password.

timeout Set the socket timeout (default 15 seconds)

editable install something editable (i.e. git+https://github.com/worldcompany/djangoembed.git#egg=djangoembed)

find_links URL to look for packages at

index_url Base URL of Python Package Index

extra_index_url Extra URLs of package indexes to use in addition to `index_url`

no_index Ignore package index

mirrors Specific mirror URL(s) to query (automatically adds --use-mirrors)

build Unpack packages into `build` dir

target Install packages into `target` dir

download Download packages into `download` instead of installing them

download_cache Cache downloaded packages in `download_cache` dir

source Check out `editable` packages into `source` dir

upgrade Upgrade all packages to the newest available version

force_reinstall When upgrading, reinstall all packages even if they are already up-to-date.

ignore_installed Ignore the installed packages (reinstalling instead)

exists_action Default action when a path already exists: (s)witch, (i)gnore, (w)ipe, (b)ackup

no_deps Ignore package dependencies

no_install Download and unpack all packages, but don't actually install them

no_chown When user is given, do not attempt to copy and chown a requirements file

cwd Current working directory to run pip from

activate Activates the virtual environment, if given via bin_env, before running install.

> Deprecated since version 2014.7.2: If *bin_env* is given, pip will already be sourced from that virualenv, making *activate* effectively a noop.

pre_releases Include pre-releases in the available versions

cert Provide a path to an alternate CA bundle

allow_all_external Allow the installation of all externally hosted files

allow_external Allow the installation of externally hosted files (comma separated list)

allow_unverified Allow the installation of insecure and unverifiable files (comma separated list)

process_dependency_links Enable the processing of dependency links

bin_env [None] Absolute path to a virtual environment directory or absolute path to a pip executable. The example below assumes a virtual environment has been created at `/foo/.virtualenvs/bar`.

env_vars Add or modify environment variables. Useful for tweaking build steps, such as specifying INCLUDE or LIBRARY paths in Makefiles, build scripts or compiler calls.

use_vt Use VT terminal emulation (see ouptut while installing)

Example:

```
django:
  pip.installed:
    - name: django >= 1.6, <= 1.7
    - bin_env: /foo/.virtualenvs/bar
    - require:
      - pkg: python-pip
```

Or

Example:

```
django:
  pip.installed:
    - name: django >= 1.6, <= 1.7
    - bin_env: /foo/.virtualenvs/bar/bin/pip
    - require:
      - pkg: python-pip
```

Attention

The following arguments are deprecated, do not use.

pip_bin [None] Deprecated, use `bin_env`

env [None] Deprecated, use `bin_env`

Changed in version 0.17.0: `use_wheel` option added.

install_options

Extra arguments to be supplied to the setup.py install command. If you are using an option with a directory path, be sure to use absolute path.

Example:

```
django:
  pip.installed:
    - name: django
    - install_options:
      - --prefix=/blah
    - require:
      - pkg: python-pip
```

global_options Extra global options to be supplied to the setup.py call before the install command.

New in version 2014.1.3.

Attention

As of Salt 0.17.0 the pip state **needs** an importable pip module. This usually means having the system's pip package installed or running Salt from an active virtualenv.

The reason for this requirement is because `pip` already does a pretty good job parsing its own requirements. It makes no sense for Salt to do `pip` requirements parsing and validation before passing them to the `pip` library. It's functionality duplication and it's more error prone.

`salt.states.pip_state.`**`removed`**(*name*, *requirements=None*, *bin_env=None*, *log=None*, *proxy=None*, *timeout=None*, *user=None*, *cwd=None*, *use_vt=False*)

Make sure that a package is not installed.

name The name of the package to uninstall

user The user under which to run pip

bin_env [None] the pip executable or virtualenenv to use

use_vt Use VT terminal emulation (see ouptut while installing)

`salt.states.pip_state.`**`uptodate`**(*name*, *bin_env=None*, *user=None*, *cwd=None*, *use_vt=False*)

New in version 2015.5.0.

Verify that the system is completely up to date.

name The name has no functional value and is only used as a tracking reference

user The user under which to run pip

bin_env the pip executable or virtualenenv to use

use_vt Use VT terminal emulation (see ouptut while installing)

22.27.89 salt.states.pkg

Installation of packages using OS package managers such as yum or apt-get

Salt can manage software packages via the pkg state module, packages can be set up to be installed, latest, removed and purged. Package management declarations are typically rather simple:

```
vim:
  pkg.installed
```

A more involved example involves pulling from a custom repository. Note that the pkgrepo has a require_in clause. This is necessary and can not be replaced by a require clause in the pkg.

```
base:
  pkgrepo.managed:
    - humanname: Logstash PPA
    - name: ppa:wolfnet/logstash
    - dist: precise
    - file: /etc/apt/sources.list.d/logstash.list
    - keyid: 28B04E4A
    - keyserver: keyserver.ubuntu.com
    - require_in:
      - pkg: logstash

logstash:
  pkg.installed
```

`salt.states.pkg.`**`installed`**(*name*, *version=None*, *refresh=None*, *fromrepo=None*, *skip_verify=False*, *skip_suggestions=False*, *pkgs=None*, *sources=None*, *allow_updates=False*, *pkg_verify=False*, *normalize=True*, ***kwargs*)

Ensure that the package is installed, and that it is the correct version (if specified).

name The name of the package to be installed. This parameter is ignored if either ``pkgs'' or ``sources'' is used. Additionally, please note that this option can only be used to install packages from a software repository. To install a package file manually, use the ``sources'' option detailed below.

fromrepo Specify a repository from which to install

> **Note:** Distros which use APT (Debian, Ubuntu, etc.) do not have a concept of repositories, in the same way as YUM-based distros do. When a source is added, it is assigned to a given release. Consider the following source configuration:
>
> ```
> deb http://ppa.launchpad.net/saltstack/salt/ubuntu precise main
> ```
>
> The packages provided by this source would be made available via the `precise` release, therefore `fromrepo` would need to be set to `precise` for Salt to install the package from this source.
>
> Having multiple sources in the same release may result in the default install candidate being newer than what is desired. If this is the case, the desired version must be specified using the `version` parameter.
>
> If the `pkgs` parameter is being used to install multiple packages in the same state, then instead of using `version`, use the method of version specification described in the **Multiple Package Installation Options** section below.

Running the shell command `apt-cache policy pkgname` on a minion can help elucidate the APT configuration and aid in properly configuring states:

```
root@saltmaster:~# salt ubuntu01 cmd.run 'apt-cache policy ffmpeg'
ubuntu01:
    ffmpeg:
    Installed: (none)
    Candidate: 7:0.10.11-1~precise1
    Version table:
        7:0.10.11-1~precise1 0
            500 http://ppa.launchpad.net/jon-severinsson/ffmpeg/ubuntu/ precise/main amd64
        4:0.8.10-0ubuntu0.12.04.1 0
            500 http://us.archive.ubuntu.com/ubuntu/ precise-updates/main amd64 Packages
            500 http://security.ubuntu.com/ubuntu/ precise-security/main amd64 Packages
        4:0.8.1-0ubuntu1 0
            500 http://us.archive.ubuntu.com/ubuntu/ precise/main amd64 Packages
```

The release is located directly after the source's URL. The actual release name is the part before the slash, so to install version **4:0.8.10-0ubuntu0.12.04.1** either `precise-updates` or `precise-security` could be used for the `fromrepo` value.

---

skip_verify Skip the GPG verification check for the package to be installed

skip_suggestions Force strict package naming. Disables lookup of package alternatives.

New in version 2014.1.1.

version Install a specific version of a package. This option is ignored if either ``pkgs'' or ``sources'' is used. Currently, this option is supported for the following pkg providers: `apt`, `ebuild`, `pacman`, `yumpkg`, and `zypper`. The version number includes the release designation where applicable, to allow Salt to target a specific release of a given version. When in doubt, using the `pkg.latest_version` function for an uninstalled package will tell you the version available.

```
# salt myminion pkg.latest_version httpd
myminion:
    2.2.15-30.el6.centos
```

Also, while this function is not yet implemented for all pkg frontends, `pkg.list_repo_pkgs` will show all versions available in the various repositories for a given package, irrespective of whether or not it is installed.

```
# salt myminion pkg.list_repo_pkgs httpd
myminion:
    ----------
    base:
        |_
          ----------
          httpd:
              2.2.15-29.el6.centos
    updates:
        |_
          ----------
          httpd:
              2.2.15-30.el6.centos
```

The version strings returned by either of these functions can be used as version specifiers in pkg states.

refresh Update the repo database of available packages prior to installing the requested package.

hold Force the package to be held at the current installed version. Currently works with YUM & APT based systems.

New in version 2014.7.0.

allow_updates Allow the package to be updated outside Salt's control (e.g. auto updates on Windows). This means a package on the Minion can have a newer version than the latest available in the repository without enforcing a re-installation of the package.

New in version 2014.7.0.

Example:

```
httpd:
  pkg.installed:
    - fromrepo: mycustomrepo
    - skip_verify: True
    - skip_suggestions: True
    - version: 2.0.6~ubuntu3
    - refresh: True
    - hold: False
```

pkg_verify New in version 2014.7.0.

For requested packages that are already installed and would not be targeted for upgrade or downgrade, use pkg.verify to determine if any of the files installed by the package have been altered. If files have been altered, the reinstall option of pkg.install is used to force a reinstall. Types to ignore can be passed to pkg.verify (see example below). Currently, this option is supported for the following pkg providers: yumpkg.

Examples:

```
httpd:
  pkg.installed:
    - version: 2.2.15-30.el6.centos
    - pkg_verify: True
```

```
mypkgs:
  pkg.installed:
    - pkgs:
      - foo
      - bar: 1.2.3-4
      - baz
    - pkg_verify:
      - ignore_types: [config,doc]
```

normalize Normalize the package name by removing the architecture. Default is True. This is useful for poorly created packages which might include the architecture as an actual part of the name such as kernel modules which match a specific kernel version.

New in version 2014.7.0.

Example:

```
gpfs.gplbin-2.6.32-279.31.1.el6.x86_64:
  pkg.installed:
    - normalize: False
```

Multiple Package Installation Options: (not supported in Windows or pkgng)

pkgs A list of packages to install from a software repository. All packages listed under `pkgs` will be installed via a single command.

Example:

```
mypkgs:
  pkg.installed:
    - pkgs:
      - foo
      - bar
      - baz
    - hold: True
```

`NOTE:` For `apt`, `ebuild`, `pacman`, `yumpkg`, and `zypper`, version numbers can be specified in the `pkgs` argument. For example:

```
mypkgs:
  pkg.installed:
    - pkgs:
      - foo
      - bar: 1.2.3-4
      - baz
```

Additionally, `ebuild`, `pacman` and `zypper` support the <, <=, >=, and > operators for more control over what versions will be installed. For example:

```
mypkgs:
  pkg.installed:
    - pkgs:
      - foo
      - bar: '>=1.2.3-4'
      - baz
```

`NOTE:` When using comparison operators, the expression must be enclosed in quotes to avoid a YAML render error.

With `ebuild` is also possible to specify a use flag list and/or if the given packages should be in package.accept_keywords file and/or the overlay from which you want the package to be installed. For example:

```
mypkgs:
  pkg.installed:
    - pkgs:
      - foo: '~'
      - bar: '~>=1.2:slot::overlay[use,-otheruse]'
      - baz
```

names A list of packages to install from a software repository. Each package will be installed individually by the package manager.

> **Warning**: Unlike `pkgs`, the `names` parameter cannot specify a version. In addition, it makes a separate call to the package management frontend to install each package, whereas `pkgs` makes just a single call. It is therefore recommended to use `pkgs` instead of `names` to install multiple packages, both for the additional features and the performance improvement that it brings.

sources A list of packages to install, along with the source URI or local path from which to install each package. In the example below, `foo`, `bar`, `baz`, etc. refer to the name of the package, as it would appear in the output of the `pkg.version` or `pkg.list_pkgs` salt CLI commands.

```
mypkgs:
  pkg.installed:
    - sources:
      - foo: salt://rpms/foo.rpm
      - bar: http://somesite.org/bar.rpm
      - baz: ftp://someothersite.org/baz.rpm
      - qux: /minion/path/to/qux.rpm
```

install_recommends Whether to install the packages marked as recommended. Default is True. Currently only works with APT based systems.

New in version 2015.5.0.

```
httpd:
  pkg.installed:
    - install_recommends: False
```

only_upgrade Only upgrade the packages, if they are already installed. Default is False. Currently only works with APT based systems.

New in version 2015.5.0.

```
httpd:
  pkg.installed:
    - only_upgrade: True
```

salt.states.pkg.**latest**(*name*, *refresh=None*, *fromrepo=None*, *skip_verify=False*, *pkgs=None*, ***kwargs*)

Ensure that the named package is installed and the latest available package. If the package can be updated, this state function will update the package. Generally it is better for the `installed` function to be used, as `latest` will update the package whenever a new package is available.

name The name of the package to maintain at the latest available version. This parameter is ignored if ``pkgs'' is used.

fromrepo Specify a repository from which to install

skip_verify Skip the GPG verification check for the package to be installed

Multiple Package Installation Options:

(Not yet supported for: Windows, FreeBSD, OpenBSD, MacOS, and Solaris pkgutil)

pkgs A list of packages to maintain at the latest available version.

```
mypkgs:
  pkg.latest:
    - pkgs:
      - foo
      - bar
      - baz
```

install_recommends Whether to install the packages marked as recommended. Default is True. Currently only works with APT based systems.

New in version 2015.5.0.

```
    httpd:
      pkg.latest:
        - install_recommends: False
```

only_upgrade Only upgrade the packages, if they are already installed. Default is False. Currently only works with APT based systems.

New in version 2015.5.0.

```
    httpd:
      pkg.latest:
        - only_upgrade: True
```

`salt.states.pkg.`**`mod_aggregate`**(*low*, *chunks*, *running*)

The mod_aggregate function which looks up all packages in the available low chunks and merges them into a single pkgs ref in the present low data

`salt.states.pkg.`**`purged`**(*name*, *version=None*, *pkgs=None*, ***kwargs*)

Verify that a package is not installed, calling `pkg.purge` if necessary to purge the package. All configuration files are also removed.

name The name of the package to be purged.

version The version of the package that should be removed. Don't do anything if the package is installed with an unmatching version.

Multiple Package Options:

pkgs A list of packages to purge. Must be passed as a python list. The `name` parameter will be ignored if this option is passed. It accepts version numbers as well.

New in version 0.16.0.

`salt.states.pkg.`**`removed`**(*name*, *version=None*, *pkgs=None*, ***kwargs*)

Verify that a package is not installed, calling `pkg.remove` if necessary to remove the package.

name The name of the package to be removed.

version The version of the package that should be removed. Don't do anything if the package is installed with an unmatching version.

Multiple Package Options:

pkgs A list of packages to remove. Must be passed as a python list. The `name` parameter will be ignored if this option is passed. It accepts version numbers as well.

New in version 0.16.0.

`salt.states.pkg.`**`uptodate`**(*name*, *refresh=False*, ***kwargs*)

New in version 2014.7.0.

Verify that the system is completely up to date.

name The name has no functional value and is only used as a tracking reference

refresh refresh the package database before checking for new upgrades

kwargs Any keyword arguments to pass through to `pkg.upgrade`.

New in version 2015.5.0.

22.27.90 salt.states.pkgng

Manage package remote repo using FreeBSD pkgng

Salt can manage the URL pkgng pulls packages from. ATM the state and module are small so use cases are typically rather simple:

```
pkgng_clients:
  pkgng.update_packaging_site:
    - name: "http://192.168.0.2"
```

salt.states.pkgng.**update_packaging_site**(*name*)

22.27.91 salt.states.pkgrepo

Management of APT/YUM package repos

Package repositories for APT-based and YUM-based distros can be managed with these states. Here is some example SLS:

```
base:
  pkgrepo.managed:
    - humanname: CentOS-$releasever - Base
    - mirrorlist: http://mirrorlist.centos.org/?release=$releasever&arch=$basearch&repo=os
    - comments:
        - '#http://mirror.centos.org/centos/$releasever/os/$basearch/'
    - gpgcheck: 1
    - gpgkey: file:///etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-6
```

```
base:
  pkgrepo.managed:
    - humanname: Logstash PPA
    - name: deb http://ppa.launchpad.net/wolfnet/logstash/ubuntu precise main
    - dist: precise
    - file: /etc/apt/sources.list.d/logstash.list
    - keyid: 28B04E4A
    - keyserver: keyserver.ubuntu.com
    - require_in:
      - pkg: logstash

  pkg.latest:
    - name: logstash
    - refresh: True
```

```
base:
  pkgrepo.managed:
    - ppa: wolfnet/logstash
  pkg.latest:
    - name: logstash
    - refresh: True
```

Note: On Ubuntu systems, the `python-software-properties` package should be installed for better support of PPA repositories. To check if this package is installed, run `dpkg -l python-software-properties`.

Also, some Ubuntu releases have a bug in their `python-software-properties` package, a missing dependency on pycurl, so `python-pycurl` will need to be manually installed if it is not present once `python-software-properties` is installed.

On Ubuntu & Debian systems, the `python-apt` package is required to be installed. To check if this package is installed, run `dpkg -l python-software-properties`. `python-apt` will need to be manually installed if it is not present.

`salt.states.pkgrepo.`**`absent`**(*name*, ***kwargs*)

This function deletes the specified repo on the system, if it exists. It is essentially a wrapper around pkg.del_repo.

name The name of the package repo, as it would be referred to when running the regular package manager commands.

ppa On Ubuntu, you can take advantage of Personal Package Archives on Launchpad simply by specifying the user and archive name.

```
logstash-ppa:
  pkgrepo.absent:
    - ppa: wolfnet/logstash
```

ppa_auth For Ubuntu PPAs there can be private PPAs that require authentication to access. For these PPAs the username/password can be specified. This is required for matching if the name format uses the ``ppa:'' specifier and is private (requires username/password to access, which is encoded in the URI).

```
logstash-ppa:
  pkgrepo.absent:
    - ppa: wolfnet/logstash
    - ppa_auth: username:password
```

`salt.states.pkgrepo.`**`managed`**(*name*, ***kwargs*)

This function manages the configuration on a system that points to the repositories for the system's package manager.

name The name of the package repo, as it would be referred to when running the regular package manager commands.

For yum-based systems, take note of the following configuration values:

humanname On yum-based systems, this is stored as the ``name'' value in the .repo file in /etc/yum.repos.d/. On yum-based systems, this is required.

baseurl On yum-based systems, baseurl refers to a direct URL to be used for this yum repo. One of baseurl or mirrorlist is required.

mirrorlist a URL which contains a collection of baseurls to choose from. On yum-based systems. One of baseurl or mirrorlist is required.

comments Sometimes you want to supply additional information, but not as enabled configuration. Anything supplied for this list will be saved in the repo configuration with a comment marker (#) in front.

Additional configuration values, such as gpgkey or gpgcheck, are used verbatim to update the options for the yum repo in question.

For apt-based systems, take note of the following configuration values:

ppa On Ubuntu, you can take advantage of Personal Package Archives on Launchpad simply by specifying the user and archive name. The keyid will be queried from launchpad and everything else is set automatically. You can override any of the below settings by simply setting them as you would normally. For example:

```
logstash-ppa:
  pkgrepo.managed:
    - ppa: wolfnet/logstash
```

ppa_auth For Ubuntu PPAs there can be private PPAs that require authentication to access. For these PPAs the username/password can be passed as an HTTP Basic style username/password combination.

```
logstash-ppa:
  pkgrepo.managed:
    - ppa: wolfnet/logstash
    - ppa_auth: username:password
```

name On apt-based systems this must be the complete entry as it would be seen in the sources.list file. This can have a limited subset of components (i.e. `main') which can be added/modified with the ``comps'' option.

```
precise-repo:
  pkgrepo.managed:
    - name: deb http://us.archive.ubuntu.com/ubuntu precise main
```

disabled Toggles whether or not the repo is used for resolving dependencies and/or installing packages.

enabled Enables the repository, even if the repository has been disabled, in order for the respective package requiring the repository can be found and installed.

comps On apt-based systems, comps dictate the types of packages to be installed from the repository (e.g. main, nonfree, ...). For purposes of this, comps should be a comma-separated list.

file The filename for the .list that the repository is configured in. It is important to include the full-path AND make sure it is in a directory that APT will look in when handling packages

dist This dictates the release of the distro the packages should be built for. (e.g. unstable)

keyid The KeyID of the GPG key to install. This option also requires the `keyserver' option to be set.

keyserver This is the name of the keyserver to retrieve gpg keys from. The keyid option must also be set for this option to work.

key_url URL to retrieve a GPG key from.

consolidate If set to true, this will consolidate all sources definitions to the sources.list file, cleanup the now unused files, consolidate components (e.g. main) for the same URI, type, and architecture to a single line, and finally remove comments from the sources.list file. The consolidate will run every time the state is processed. The option only needs to be set on one repo managed by salt to take effect.

refresh_db If set to false this will skip refreshing the apt package database on debian based systems.

require_in Set this to a list of pkg.installed or pkg.latest to trigger the running of apt-get update prior to attempting to install these packages. Setting a require in the pkg will not work for this.

22.27.92 salt.states.portage_config

Management of Portage package configuration on Gentoo

A state module to manage Portage configuration on Gentoo

```
salt:
    portage_config.flags:
        - use:
            - openssl
```

salt.states.portage_config.**flags**(*name*, *use=None*, *accept_keywords=None*, *env=None*, *license=None*, *properties=None*, *unmask=False*, *mask=False*)

Enforce the given flags on the given package or DEPEND atom.

> **Warning**: In most cases, the affected package(s) need to be rebuilt in order to apply the changes.

name The name of the package or its DEPEND atom

use A list of USE flags

accept_keywords A list of keywords to accept. ~ARCH means current host arch, and will be translated into a line without keywords

env A list of environment files

license A list of accepted licenses

properties A list of additional properties

unmask A boolean to unmask the package

mask A boolean to mask the package

22.27.93 salt.states.ports

Manage software from FreeBSD ports

New in version 2014.1.0.

Note: It may be helpful to use a higher timeout when running a `ports.installed` state, since compiling the port may exceed Salt's timeout.

```
salt -t 1200 '*' state.highstate
```

---

salt.states.ports.**installed**(*name*, *options=None*)

Verify that the desired port is installed, and that it was compiled with the desired options.

options Make sure that the desired non-default options are set

> **Warning**: Any build options not passed here assume the default values for the port, and are not just differences from the existing cached options from a previous `make config`.

Example usage:

```
    security/nmap:
      ports.installed:
        - options:
          - IPV6: off
```

22.27.94 salt.states.postgres_database

Management of PostgreSQL databases

The postgres_database module is used to create and manage Postgres databases. Databases can be set as either absent or present

```
frank:
  postgres_database.present
```

salt.states.postgres_database.**absent**(*name*, *user=None*, *maintenance_db=None*, *db_password=None*, *db_host=None*, *db_port=None*, *db_user=None*)

Ensure that the named database is absent

name The name of the database to remove

db_user database username if different from config or defaul

db_password user password if any password for a specified user

db_host Database host if different from config or default

db_port Database port if different from config or default

user System user all operations should be performed on behalf of

New in version 0.17.0.

salt.states.postgres_database.**present**(*name*, *tablespace=None*, *encoding=None*, *lc_collate=None*, *lc_ctype=None*, *owner=None*, *template=None*, *user=None*, *maintenance_db=None*, *db_password=None*, *db_host=None*, *db_port=None*, *db_user=None*)

Ensure that the named database is present with the specified properties. For more information about all of these options see man createdb(1)

name The name of the database to manage

tablespace Default tablespace for the database

encoding The character encoding scheme to be used in this database

lc_collate The LC_COLLATE setting to be used in this database

lc_ctype The LC_CTYPE setting to be used in this database

owner The username of the database owner

template The template database from which to build this database

user System user all operations should be performed on behalf of

db_user database username if different from config or default

db_password user password if any password for a specified user

db_host Database host if different from config or default

db_port Database port if different from config or default

New in version 0.17.0.

22.27.95 salt.states.postgres_extension

Management of PostgreSQL extensions (e.g.: postgis)

The postgres_extensions module is used to create and manage Postgres extensions.

```
adminpack:
  postgres_extension.present
```

New in version 2014.7.0.

`salt.states.postgres_extension.`**`absent`**(*name*, *if_exists=None*, *restrict=None*, *cascade=None*, *user=None*, *maintenance_db=None*, *db_password=None*, *db_host=None*, *db_port=None*, *db_user=None*)

Ensure that the named extension is absent

name Extension name of the extension to remove

cascade Drop on cascade

if_exists Add if exist slug

restrict Add restrict slug

maintenance_db Database to act on

user System user all operations should be performed on behalf of

db_user database username if different from config or default

db_password user password if any password for a specified user

db_host Database host if different from config or default

db_port Database port if different from config or default

`salt.states.postgres_extension.`**`present`**(*name*, *if_not_exists=None*, *schema=None*, *ext_version=None*, *from_version=None*, *user=None*, *maintenance_db=None*, *db_password=None*, *db_host=None*, *db_port=None*, *db_user=None*)

Ensure that the named extension is present with the specified privileges

name The name of the extension to manage

if_not_exists Add a if_not_exists switch to the ddl statement

schema Schema to install the extension into

from_version Old extension version if already installed

ext_version version to install

user System user all operations should be performed on behalf of

maintenance_db Database to act on

db_user database username if different from config or default

db_password user password if any password for a specified user

db_host Database host if different from config or default

db_port Database port if different from config or default

22.27.96 salt.states.postgres_group

Management of PostgreSQL groups (roles)

The postgres_group module is used to create and manage Postgres groups.

```
frank:
  postgres_group.present
```

salt.states.postgres_group.**absent**(*name*, *user=None*, *maintenance_db=None*, *db_password=None*, *db_host=None*, *db_port=None*, *db_user=None*)

Ensure that the named group is absent

name The groupname of the group to remove

user System user all operations should be performed on behalf of

New in version 0.17.0.

db_user database username if different from config or defaul

db_password user password if any password for a specified user

db_host Database host if different from config or default

db_port Database port if different from config or default

salt.states.postgres_group.**present**(*name*, *createdb=None*, *createroles=None*, *createuser=None*, *encrypted=None*, *superuser=None*, *inherit=None*, *login=None*, *replication=None*, *password=None*, *refresh_password=None*, *groups=None*, *user=None*, *maintenance_db=None*, *db_password=None*, *db_host=None*, *db_port=None*, *db_user=None*)

Ensure that the named group is present with the specified privileges Please note that the user/group notion in postgresql is just abstract, we have roles, where users can be seens as roles with the LOGIN privilege and groups the others.

name The name of the group to manage

createdb Is the group allowed to create databases?

createroles Is the group allowed to create other roles/users

createuser Alias to create roles, and history problem, in pgsql normally createuser == superuser

encrypted Should the password be encrypted in the system catalog?

login Should the group have login perm

inherit Should the group inherit permissions

superuser Should the new group be a ``superuser''

replication Should the new group be allowed to initiate streaming replication

password The Group's password It can be either a plain string or a md5 postgresql hashed password:

```
'md5{MD5OF({password}{role}}'
```

If encrypted is None or True, the password will be automatically encrypted to the previous format if it is not already done.

refresh_password Password refresh flag

Boolean attribute to specify whether to password comparison check should be performed.

If refresh_password is None or False, the password will be automatically updated without extra password change check.

This behaviour makes it possible to execute in environments without superuser access available, e.g. Amazon RDS for PostgreSQL

groups A string of comma separated groups the group should be in

user System user all operations should be performed on behalf of

New in version 0.17.0.

db_user database username if different from config or defaul

db_password user password if any password for a specified user

db_host Database host if different from config or default

db_port Database port if different from config or default

22.27.97 salt.states.postgres_schema

Management of PostgreSQL schemas

The postgres_schemas module is used to create and manage Postgres schemas.

```
public:
  postgres_schema.present 'dbname' 'name'
```

salt.states.postgres_schema.**absent**(*dbname*, *name*, *db_user=None*, *db_password=None*, *db_host=None*, *db_port=None*)

Ensure that the named schema is absent

dbname The database's name will work on

name The name of the schema to remove

db_user database username if different from config or default

db_password user password if any password for a specified user

db_host Database host if different from config or default

db_port Database port if different from config or default

salt.states.postgres_schema.**present**(*dbname*, *name*, *owner=None*, *db_user=None*, *db_password=None*, *db_host=None*, *db_port=None*)

Ensure that the named schema is present in the database.

dbname The database's name will work on

name The name of the schema to manage

db_user database username if different from config or default

db_password user password if any password for a specified user

db_host Database host if different from config or default

db_port Database port if different from config or default

22.27.98 salt.states.postgres_user

Management of PostgreSQL users (roles)

The postgres_users module is used to create and manage Postgres users.

```
frank:
  postgres_user.present
```

salt.states.postgres_user.**absent**(*name*, *user=None*, *maintenance_db=None*, *db_password=None*, *db_host=None*, *db_port=None*, *db_user=None*)

Ensure that the named user is absent

name The username of the user to remove

user System user all operations should be performed on behalf of

New in version 0.17.0.

db_user database username if different from config or default

db_password user password if any password for a specified user

db_host Database host if different from config or default

db_port Database port if different from config or default

salt.states.postgres_user.**present**(*name*, *createdb=None*, *createroles=None*, *createuser=None*, *encrypted=None*, *superuser=None*, *replication=None*, *inherit=None*, *login=None*, *password=None*, *refresh_password=None*, *groups=None*, *user=None*, *maintenance_db=None*, *db_password=None*, *db_host=None*, *db_port=None*, *db_user=None*)

Ensure that the named user is present with the specified privileges Please note that the user/group notion in postgresql is just abstract, we have roles, where users can be seens as roles with the LOGIN privilege and groups the others.

name The name of the user to manage

createdb Is the user allowed to create databases?

createroles Is the user allowed to create other users?

createuser Alias to create roles

encrypted Should the password be encrypted in the system catalog?

login Should the group have login perm

inherit Should the group inherit permissions

superuser Should the new user be a ``superuser''

replication Should the new user be allowed to initiate streaming replication

password The user's password It can be either a plain string or a md5 postgresql hashed password:

```
'md5{MD5OF({password}{role}}'
```

If encrypted is None or True, the password will be automatically encrypted to the previous format if it is not already done.

refresh_password Password refresh flag

Boolean attribute to specify whether to password comparison check should be performed.

If refresh_password is None or False, the password will be automatically updated without extra password change check.

This behaviour makes it possible to execute in environments without superuser access available, e.g. Amazon RDS for PostgreSQL

groups A string of comma separated groups the user should be in

user System user all operations should be performed on behalf of

New in version 0.17.0.

db_user database username if different from config or default

db_password user password if any password for a specified user

db_host Database host if different from config or default

db_port Database port if different from config or default

22.27.99 salt.states.powerpath

Powerpath configuration support

Allows configuration of EMC Powerpath. Currently only addition/deletion of licenses is supported.

```
key:
  powerpath.license_present: []
```

`salt.states.powerpath.`**`license_absent`**(*name*)

Ensures that the specified PowerPath license key is absent on the host.

name The licnese key to ensure is absent

`salt.states.powerpath.`**`license_present`**(*name*)

Ensures that the specified PowerPath license key is present on the host.

name The licnese key to ensure is present

22.27.100 salt.states.process

Process Management

Ensure a process matching a given pattern is absent.

```
httpd-absent:
  process.absent:
    - name: apache2
```

`salt.states.process.`**`absent`**(*name*, *user=None*, *signal=None*)

Ensures that the named command is not running.

name The pattern to match.

user The user process belongs

signal Signal to send to the process(es).

22.27.101 salt.states.pyenv

Managing python installations with pyenv

This module is used to install and manage python installations with pyenv. Different versions of python can be installed, and uninstalled. pyenv will be installed automatically the first time it is needed and can be updated later. This module will *not* automatically install packages which pyenv will need to compile the versions of python.

If pyenv is run as the root user then it will be installed to /usr/local/pyenv, otherwise it will be installed to the users ~/.pyenv directory. To make pyenv available in the shell you may need to add the pyenv/shims and pyenv/bin directories to the users PATH. If you are installing as root and want other users to be able to access pyenv then you will need to add pyenv_ROOT to their environment.

This is how a state configuration could look like:

```
pyenv-deps:
  pkg.installed:
    - pkgs:
      - make
      - build-essential
      - libssl-dev
      - zlib1g-dev
      - libbz2-dev
      - libreadline-dev
      - libsqlite3-dev
      - wget
      - curl
      - llvm
python-2.6:
  pyenv.absent:
    - require:
      - pkg: pyenv-deps

python-2.7.6:
  pyenv.installed:
    - default: True
    - require:
      - pkg: pyenv-deps
```

`salt.states.pyenv.`**`absent`**(*name*, *user=None*)

Verify that the specified python is not installed with pyenv. pyenv is installed if necessary.

name The version of python to uninstall

user: **None** The user to run pyenv as.

New in version 0.17.0.

New in version 0.16.0.

`salt.states.pyenv.`**`install_pyenv`**(*name*, *user=None*)

Install pyenv if not installed. Allows you to require pyenv be installed prior to installing the plugins. Useful if you want to install pyenv plugins via the git or file modules and need them installed before installing any rubies.

Use the pyenv.root configuration option to set the path for pyenv if you want a system wide install that is not in a user home dir.

user: **None** The user to run pyenv as.

salt.states.pyenv.**installed**(*name*, *default=False*, *user=None*)
Verify that the specified python is installed with pyenv. pyenv is installed if necessary.

name The version of python to install

default [False] Whether to make this python the default.

user: **None** The user to run pyenv as.

New in version 0.17.0.

New in version 0.16.0.

22.27.102 salt.states.pyrax_queues

Manage Rackspace Queues

New in version 2015.5.0.

Create and destroy Rackspace queues. Be aware that this interacts with Rackspace's services, and so may incur charges.

This module uses `pyrax`, which can be installed via package, or pip. This module is greatly inspired by boto_* modules from SaltStack code source.

```
myqueue:
    pyrax_queues.present:
        - provider: my-pyrax

myqueue:
    pyrax_queues.absent:
        - provider: my-pyrax
```

salt.states.pyrax_queues.**absent**(*name*, *provider*)
Ensure the named Rackspace queue is deleted.

name Name of the Rackspace queue.

provider Salt Cloud provider

salt.states.pyrax_queues.**present**(*name*, *provider*)
Ensure the RackSpace queue exists.

name Name of the Rackspace queue.

provider Salt Cloud Provider

22.27.103 salt.states.quota

Management of POSIX Quotas

The quota can be managed for the system:

```
/:
  quota.mode:
    mode: off
    quotatype: user
```

salt.states.quota.**mode**(*name*, *mode*, *quotatype*)

Set the quota for the system

name The filesystem to set the quota mode on

mode Whether the quota system is on or off

quotatype Must be `user` or `group`

22.27.104 salt.states.rabbitmq_cluster

Manage RabbitMQ Clusters

Example:

```
rabbit@rabbit.example.com:
    rabbitmq_cluster.join:
      - user: rabbit
      - host: rabbit.example.com
```

salt.states.rabbitmq_cluster.**join**(*name*, *host*, *user='rabbit'*, *ram_node=None*, *runas='root'*)

Ensure the current node joined to a cluster with node user@host

name Irrelevant, not used (recommended: user@host)

user The user of node to join to (default: rabbit)

host The host of node to join to

ram_node Join node as a RAM node

runas The user to run the rabbitmq command as

salt.states.rabbitmq_cluster.**joined**(*name*, *host*, *user='rabbit'*, *ram_node=None*, *runas='root'*)

Ensure the current node joined to a cluster with node user@host

name Irrelevant, not used (recommended: user@host)

user The user of node to join to (default: rabbit)

host The host of node to join to

ram_node Join node as a RAM node

runas The user to run the rabbitmq command as

22.27.105 salt.states.rabbitmq_plugin

Manage RabbitMQ Plugins

New in version 2014.1.0.

Example:

```
some_plugin:
    rabbitmq_plugin.enabled: []
```

salt.states.rabbitmq_plugin.**disabled**(*name*, *runas=None*)
Ensure the RabbitMQ plugin is disabled.

name The name of the plugin

runas The user to run the rabbitmq-plugin command as

salt.states.rabbitmq_plugin.**enabled**(*name*, *runas=None*)
Ensure the RabbitMQ plugin is enabled.

name The name of the plugin

runas The user to run the rabbitmq-plugin command as

22.27.106 salt.states.rabbitmq_policy

Manage RabbitMQ Policies

maintainer Benn Eichhorn <benn@getlocalmeasure.com>

maturity new

platform all

Example:

```
rabbit_policy:
    rabbitmq_policy.present:
        - name: HA
        - pattern: '.*'
        - definition: '{"ha-mode": "all"}'
```

salt.states.rabbitmq_policy.**absent**(*name*, *vhost='/'*, *runas=None*)
Ensure the named policy is absent

Reference: http://www.rabbitmq.com/ha.html

name The name of the policy to remove

runas Name of the user to run the command as

salt.states.rabbitmq_policy.**present**(*name*, *pattern*, *definition*, *priority=0*, *vhost='/'*, *runas=None*)
Ensure the RabbitMQ policy exists.

Reference: http://www.rabbitmq.com/ha.html

name Policy name

pattern A regex of queues to apply the policy to

definition A json dict describing the policy

priority Priority (defaults to 0)

vhost Virtual host to apply to (defaults to `/')

runas Name of the user to run the command as

22.27.107 salt.states.rabbitmq_user

Manage RabbitMQ Users

Example:

```
rabbit_user:
    rabbitmq_user.present:
        - password: password
        - force: True
        - tags:
            - monitoring
            - user
        - perms:
          - '/':
            - '.*'
            - '.*'
            - '.*'
        - runas: rabbitmq
```

salt.states.rabbitmq_user.**absent**(*name*, *runas=None*)

Ensure the named user is absent

name The name of the user to remove

runas User to run the command

salt.states.rabbitmq_user.**present**(*name*, *password=None*, *force=False*, *tags=None*, *perms=()*, *runas=None*)

Ensure the RabbitMQ user exists.

name User name

password User's password, if one needs to be set

force If user exists, forcibly change the password

tags Optional list of tags for the user

perms A list of dicts with vhost keys and 3-tuple values

runas Name of the user to run the command

22.27.108 salt.states.rabbitmq_vhost

Manage RabbitMQ Virtual Hosts

Example:

```
virtual_host:
    rabbitmq_vhost.present:
        - user: rabbit_user
        - conf: .*
        - write: .*
        - read: .*
```

salt.states.rabbitmq_vhost.**absent**(*name*, *runas=None*)

Ensure the RabbitMQ Virtual Host is absent

name Name of the Virtual Host to remove

runas User to run the command

Deprecated since version Beryllium.

salt.states.rabbitmq_vhost.**present**(*name*, *user=None*, *owner=None*, *conf=None*, *write=None*, *read=None*, *runas=None*)

Ensure the RabbitMQ VHost exists.

name VHost name

user Initial user permission to set on the VHost, if present

Deprecated since version Beryllium.

owner Initial owner permission to set on the VHost, if present

conf Initial conf string to apply to the VHost and user. Defaults to .*

write Initial write permissions to apply to the VHost and user. Defaults to .*

read Initial read permissions to apply to the VHost and user. Defaults to .*

runas Name of the user to run the command

Deprecated since version Beryllium.

22.27.109 salt.states.rbenv

Managing Ruby installations with rbenv

This module is used to install and manage ruby installations with rbenv and the ruby-build plugin. Different versions of ruby can be installed, and uninstalled. Rbenv will be installed automatically the first time it is needed and can be updated later. This module will *not* automatically install packages which rbenv will need to compile the versions of ruby. If your version of ruby fails to install, refer to the ruby-build documentation to verify you are not missing any dependencies: https://github.com/sstephenson/ruby-build/wiki

If rbenv is run as the root user then it will be installed to /usr/local/rbenv, otherwise it will be installed to the users ~/.rbenv directory. To make rbenv available in the shell you may need to add the rbenv/shims and rbenv/bin directories to the users PATH. If you are installing as root and want other users to be able to access rbenv then you will need to add RBENV_ROOT to their environment.

The following state configuration demonstrates how to install Ruby 1.9.x and 2.x using rbenv on Ubuntu/Debian:

```
rbenv-deps:
  pkg.installed:
    - names:
      - bash
      - git
      - openssl
      - libssl-dev
      - make
      - curl
      - autoconf
      - bison
      - build-essential
      - libssl-dev
      - libyaml-dev
      - libreadline6-dev
      - zlib1g-dev
```

```
      - libncurses5-dev

ruby-1.9.3-p429:
  rbenv.absent:
    - require:
      - pkg: rbenv-deps

ruby-2.0.0-p598:
  rbenv.installed:
    - default: True
    - require:
      - pkg: rbenv-deps
```

salt.states.rbenv.**absent**(*name*, *user=None*)

Verify that the specified ruby is not installed with rbenv. Rbenv is installed if necessary.

name The version of ruby to uninstall

user: **None** The user to run rbenv as.

New in version 0.17.0.

New in version 0.16.0.

salt.states.rbenv.**install_rbenv**(*name*, *user=None*)

Install rbenv if not installed. Allows you to require rbenv be installed prior to installing the plugins. Useful if you want to install rbenv plugins via the git or file modules and need them installed before installing any rubies.

Use the rbenv.root configuration option to set the path for rbenv if you want a system wide install that is not in a user home dir.

user: **None** The user to run rbenv as.

salt.states.rbenv.**installed**(*name*, *default=False*, *user=None*)

Verify that the specified ruby is installed with rbenv. Rbenv is installed if necessary.

name The version of ruby to install

default [False] Whether to make this ruby the default.

user: **None** The user to run rbenv as.

New in version 0.17.0.

New in version 0.16.0.

22.27.110 salt.states.rdp

Manage RDP Service on Windows servers

salt.states.rdp.**disabled**(*name*)

Disable the RDP service

salt.states.rdp.**enabled**(*name*)

Enable the RDP service and make sure access to the RDP port is allowed in the firewall configuration

22.27.111 salt.states.redismod

Management of Redis server

New in version 2014.7.0.

depends

- redis Python module

configuration See `salt.modules.redis` for setup instructions.

```
key_in_redis:
  redis.string:
    - value: string data
```

The redis server information specified in the minion config file can be overridden in states using the following arguments: `host`, `post`, `db`, `password`.

```
key_in_redis:
  redis.string:
    - value: string data
    - host: localhost
    - port: 6379
    - db: 0
    - password: somuchkittycat
```

`salt.states.redismod.`**`absent`**(*name*, *keys=None*, ***connection_args*)

Ensure key absent from redis

name Key to ensure absent from redis

keys list of keys to ensure absent, name will be ignored if this is used

`salt.states.redismod.`**`string`**(*name*, *value*, *expire=None*, *expireat=None*, ***connection_args*)

Ensure that the key exists in redis with the value specified

name Redis key to manage

value Data to persist in key

expire Sets time to live for key in seconds

expireat Sets expiration time for key via UNIX timestamp, overrides *expire*

22.27.112 salt.states.reg

Manage the registry on Windows

`salt.states.reg.`**`absent`**(*name*)

Remove a registry key

Example:

```
    'HKEY_CURRENT_USER\SOFTWARE\Salt\version':
      reg.absent
```

salt.states.reg.**present**(*name*, *value*, *vtype='REG_DWORD'*, *reflection=True*)

Set a registry entry

Optionally set `reflection` to `False` to disable reflection. `reflection` has no effect on a 32-bit OS.

In the example below, this will prevent Windows from silently creating the key in: `HKEY_CURRENT_USER\SOFTWARE\Wow6432Node\Salt\version`

Example:

```
HKEY_CURRENT_USER\SOFTWARE\Salt\version:
  reg.present:
    - value: 0.15.3
    - vtype: REG_SZ
    - reflection: False
```

22.27.113 salt.states.rvm

Managing Ruby installations and gemsets with Ruby Version Manager (RVM)

This module is used to install and manage ruby installations and gemsets with RVM, the Ruby Version Manager. Different versions of ruby can be installed and gemsets created. RVM itself will be installed automatically if it's not present. This module will not automatically install packages that RVM depends on or ones that are needed to build ruby. If you want to run RVM as an unprivileged user (recommended) you will have to create this user yourself. This is how a state configuration could look like:

```
rvm:
  group.present: []
  user.present:
    - gid: rvm
    - home: /home/rvm
    - require:
      - group: rvm

rvm-deps:
  pkg.installed:
    - pkgs:
      - bash
      - coreutils
      - gzip
      - bzip2
      - gawk
      - sed
      - curl
      - git-core
      - subversion

mri-deps:
  pkg.installed:
    - pkgs:
      - build-essential
      - openssl
      - libreadline6
      - libreadline6-dev
      - curl
      - git-core
      - zlib1g
```

```
      - zlib1g-dev
      - libssl-dev
      - libyaml-dev
      - libsqlite3-0
      - libsqlite3-dev
      - sqlite3
      - libxml2-dev
      - libxslt1-dev
      - autoconf
      - libc6-dev
      - libncurses5-dev
      - automake
      - libtool
      - bison
      - subversion
      - ruby

jruby-deps:
  pkg.installed:
    - pkgs:
      - curl
      - g++
      - openjdk-6-jre-headless

ruby-1.9.2:
  rvm.installed:
    - default: True
    - user: rvm
    - require:
      - pkg: rvm-deps
      - pkg: mri-deps
      - user: rvm

jruby:
  rvm.installed:
    - user: rvm
    - require:
      - pkg: rvm-deps
      - pkg: jruby-deps
      - user: rvm

jgemset:
  rvm.gemset_present:
    - ruby: jruby
    - user: rvm
    - require:
      - rvm: jruby

mygemset:
  rvm.gemset_present:
    - ruby: ruby-1.9.2
    - user: rvm
    - require:
      - rvm: ruby-1.9.2
```

salt.states.rvm.**gemset_present**(*name*, *ruby='default'*, *user=None*)

Verify that the gemset is present.

name The name of the gemset.

ruby: **default** The ruby version this gemset belongs to.

user: **None** The user to run rvm as.

New in version 0.17.0.

`salt.states.rvm.`**`installed`**(*name*, *default=False*, *user=None*)

Verify that the specified ruby is installed with RVM. RVM is installed when necessary.

name The version of ruby to install

default [False] Whether to make this ruby the default.

user: **None** The user to run rvm as.

New in version 0.17.0.

22.27.114 salt.states.saltmod

Control the Salt command interface

This state is intended for use from the Salt Master. It provides access to sending commands down to minions as well as access to executing master-side modules. These state functions wrap Salt's *Python API*.

See also:

More Orchestrate documentation

- *Full Orchestrate Tutorial*
- `The Orchestrate runner`

`salt.states.saltmod.`**`function`**(*name*, *tgt*, *ssh=False*, *tgt_type=None*, *expr_form=None*, *ret='`*, *expect_minions=False*, *fail_minions=None*, *fail_function=None*, *arg=None*, *kwarg=None*, *timeout=None*)

Execute a single module function on a remote minion via salt or salt-ssh

name The name of the function to run, aka cmd.run or pkg.install

tgt The target specification, aka `*' for all minions

tgt_type | expr_form The target type, defaults to glob

arg The list of arguments to pass into the function

kwarg The list of keyword arguments to pass into the function

ret Optionally set a single or a list of returners to use

expect_minions An optional boolean for failing if some minions do not respond

fail_minions An optional list of targeted minions where failure is an option

fail_function An optional string that points to a salt module that returns True or False based on the returned data dict for individual minions

ssh Set to *True* to use the ssh client instaed of the standard salt client

`salt.states.saltmod.`**`runner`**(*name*, ***kwargs*)

Execute a runner module on the master

New in version 2014.7.

name The name of the function to run

kwargs Any keyword arguments to pass to the runner function

salt.states.saltmod.**state**(*name*, *tgt*, *ssh=False*, *tgt_type=None*, *expr_form=None*, *ret=''*, *highstate=None*, *sls=None*, *top=None*, *env=None*, *test=False*, *pillar=None*, *expect_minions=False*, *fail_minions=None*, *allow_fail=0*, *concurrent=False*, *timeout=None*)

Invoke a state run on a given target

name An arbitrary name used to track the state execution

tgt The target specification for the state run.

tgt_type | expr_form The target type to resolve, defaults to glob

ret Optionally set a single or a list of returners to use

highstate Defaults to None, if set to True the target systems will ignore any sls references specified in the sls option and call state.highstate on the targeted minions

top Should be the name of a top file. If set state.top is called with this top file instead of state.sls.

sls A group of sls files to execute. This can be defined as a single string containing a single sls file, or a list of sls files

test Pass `test=true` through to the state function

pillar Pass the `pillar` kwarg through to the state function

saltenv The default salt environment to pull sls files from

ssh Set to *True* to use the ssh client instaed of the standard salt client

roster In the event of using salt-ssh, a roster system can be set

expect_minions An optional boolean for failing if some minions do not respond

fail_minions An optional list of targeted minions where failure is an option

allow_fail Pass in the number of minions to allow for failure before setting the result of the execution to False

concurrent Allow multiple state runs to occur at once.

WARNING: This flag is potentially dangerous. It is designed for use when multiple state runs can safely be run at the same Do not use this flag for performance optimization.

Examples:

Run a list of sls files via `state.sls` on target minions:

```
webservers:
  salt.state:
    - tgt: 'web*'
    - sls:
      - apache
      - django
      - core
    - saltenv: prod
```

Run a full `state.highstate` on target mininons.

```
databases:
  salt.state:
    - tgt: role:database
    - tgt_type: grain
    - highstate: True
```

salt.states.saltmod.**wait_for_event**(*name*, *id_list*, *event_id='id'*, *timeout=300*)

Watch Salt's event bus and block until a condition is met

New in version 2014.7.

name An event tag to watch for; supports Reactor-style globbing.

id_list A list of event identifiers to watch for -- usually the minion ID. Each time an event tag is matched the event data is inspected for `event_id`, if found it is removed from `id_list`. When `id_list` is empty this function returns success.

event_id [id] The name of a key in the event data. Default is `id` for the minion ID, another common value is `name` for use with orchestrating salt-cloud events.

timeout [300] The maximum time in seconds to wait before failing.

The following example blocks until all the listed minions complete a restart and reconnect to the Salt master:

```
    reboot_all_minions:
      salt.function:
        - name: system.reboot
        - tgt: '*'

    wait_for_reboots:
      salt.wait_for_event:
        - name: salt/minion/*/start
        - id_list:
          - jerry
          - stuart
          - dave
          - phil
          - kevin
          - mike
        - require:
          - salt: reboot_all_minions
```

salt.states.saltmod.**wheel**(*name*, ***kwargs*)

Execute a wheel module on the master

New in version 2014.7.

name The name of the function to run

kwargs Any keyword arguments to pass to the wheel function

```
    accept_minion_key:
      salt.wheel:
        - name: key.accept
        - match: frank
```

22.27.115 salt.states.schedule

Management of the Salt scheduler

```
job3:
  schedule.present:
    - function: test.ping
    - seconds: 3600
    - splay: 10
```

```
This will schedule the command: test.ping every 3600 seconds
(every hour) splaying the time between 0 and 10 seconds

job2:
  schedule.present:
    - function: test.ping
    - seconds: 15
    - splay:
        - start: 10
        - end: 20

This will schedule the command: test.ping every 3600 seconds
(every hour) splaying the time between 10 and 20 seconds

job1:
  schedule.present:
    - function: state.sls
    - job_args:
      - httpd
    - job_kwargs:
        test: True
    - when:
        - Monday 5:00pm
        - Tuesday 3:00pm
        - Wednesday 5:00pm
        - Thursday 3:00pm
        - Friday 5:00pm

This will schedule the command: state.sls httpd test=True at 5pm on Monday,
Wednesday and Friday, and 3pm on Tuesday and Thursday.  Requires that
python-dateutil is installed on the minion.

job1:
  schedule.present:
    - function: state.sls
    - job_args:
      - httpd
    - job_kwargs:
        test: True
    - cron: '*/5 * * * *'

Scheduled jobs can also be specified using the format used by cron.  This will
schedule the command: state.sls httpd test=True to run every 5 minutes.  Requires
that python-croniter is installed on the minion.

job1:
  schedule.present:
    - function: state.sls
    - args:
      - httpd
    - kwargs:
      test: True
    - when:
        - Monday 5:00pm
        - Tuesday 3:00pm
        - Wednesday 5:00pm
        - Thursday 3:00pm
        - Friday 5:00pm
```

```
    - returner: xmpp
    - return_config: xmpp_state_run

This will schedule the command: state.sls httpd test=True at 5pm on Monday,
Wednesday and Friday, and 3pm on Tuesday and Thursday.  Using the xmpp returner
to return the results of the scheduled job, with the alternative configuration
options found in the xmpp_state_run section.
```

salt.states.schedule.**absent**(*name*, ***kwargs*)

Ensure a job is absent from the schedule

name The unique name that is given to the scheduled job.

salt.states.schedule.**present**(*name*, ***kwargs*)

Ensure a job is present in the schedule

name The unique name that is given to the scheduled job.

seconds The scheduled job will be executed after the specified number of seconds have passed.

minutes The scheduled job will be executed after the specified number of minutes have passed.

hours The scheduled job will be executed after the specified number of hours have passed.

days The scheduled job will be executed after the specified number of days have passed.

when This will schedule the job at the specified time(s). The when parameter must be a single value or a dictionary with the date string(s) using the dateutil format. Requires python-dateutil.

cron This will schedule the job at the specified time(s) using the crontab format. Requires python-croniter.

function The function that should be executed by the scheduled job.

job_args The arguments that will be used by the scheduled job.

job_kwargs The keyword arguments that will be used by the scheduled job.

maxrunning Ensure that there are no more than N copies of a particular job running.

jid_include Include the job into the job cache.

splay The amount of time in seconds to splay a scheduled job. Can be specified as a single value in seconds or as a dictionary range with `start' and `end' values.

range This will schedule the command within the range specified. The range parameter must be a dictionary with the date strings using the dateutil format. Requires python-dateutil.

returner The returner to use to return the results of the scheduled job.

return_config The alternative configuration to use for returner configuration options.

22.27.116 salt.states.selinux

Management of SELinux rules

If SELinux is available for the running system, the mode can be managed and booleans can be set.

```
enforcing:
    selinux.mode

samba_create_home_dirs:
    selinux.boolean:
```

```
    - value: True
    - persist: True
```

Note: Use of these states require that the `selinux` execution module is available.

`salt.states.selinux.`**`boolean`**(*name*, *value*, *persist=False*)
Set up an SELinux boolean

name The name of the boolean to set

value The value to set on the boolean

persist Defaults to False, set persist to true to make the boolean apply on a reboot

`salt.states.selinux.`**`mode`**(*name*)
Verifies the mode SELinux is running in, can be set to enforcing or permissive

name The mode to run SELinux in, permissive or enforcing

22.27.117 salt.states.serverdensity_device

Monitor Server with Server Density

New in version 2014.7.0.

Server Density Is a hosted monitoring service.

Warning: This state module is beta. It might be changed later to include more or less automation.

Note: This state module requires a pillar for authentication with Server Density:

```
serverdensity:
  api_token: "b97da80a41c4f61bff05975ee51eb1aa"
  account_url: "https://your-account.serverdensity.io"
```

Note: Although Server Density allows duplicate device names in its database, this module will raise an exception if you try monitoring devices with the same name.

Example:

```
'server_name':
  serverdensity_device.monitored
```

`salt.states.serverdensity_device.`**`monitored`**(*name*, *group=None*, *salt_name=True*, *salt_params=True*, ***params*)
Device is monitored with Server Density.

name Device name in Server Density.

salt_name If `True` (default), takes the name from the `id` grain. If `False`, the provided name is used.

group Group name under with device will appear in Server Density dashboard. Default - *None*.

salt_params If `True` (default), needed config parameters will be sourced from grains and from `status.all_status`.

params Add parameters that you want to appear in the Server Density dashboard. Will overwrite the *salt_params* parameters. For more info, see the API docs.

Usage example:

```
'server_name':
  serverdensity_device.monitored
```

```
'server_name':
  serverdensity_device.monitored:
    - group: web-servers
```

```
'my_special_server':
  serverdensity_device.monitored:
    - salt_name: False
    - group: web-servers
    - cpuCores: 2
    - os: CentOS
```

22.27.118 salt.states.service

Starting or restarting of services and daemons

Services are defined as system daemons typically started with system init or rc scripts, services can be defined as running or dead.

```
httpd:
  service.running: []
```

The service can also be set to be started at runtime via the enable option:

```
openvpn:
  service.running:
    - enable: True
```

By default if a service is triggered to refresh due to a watch statement the service is by default restarted. If the desired behavior is to reload the service, then set the reload value to True:

```
redis:
  service.running:
    - enable: True
    - reload: True
    - watch:
      - pkg: redis
```

Note: More details regarding `watch` can be found in the *Requisites* documentation.

`salt.states.service.`**`dead`**(*name*, *enable=None*, *sig=None*, ***kwargs*)

Ensure that the named service is dead by stopping the service if it is running

name The name of the init or rc script used to manage the service

enable Set the service to be enabled at boot time, `True` sets the service to be enabled, `False` sets the named service to be disabled. The default is `None`, which does not enable or disable anything.

sig The string to search for when looking for the service process with ps

`salt.states.service.`**`disabled`**(*name*, ***kwargs*)
Verify that the service is disabled on boot, only use this state if you don't want to manage the running process, remember that if you want to disable a service to use the enable: False option for the running or dead function.

name The name of the init or rc script used to manage the service

`salt.states.service.`**`enabled`**(*name*, ***kwargs*)
Verify that the service is enabled on boot, only use this state if you don't want to manage the running process, remember that if you want to enable a running service to use the enable: True option for the running or dead function.

name The name of the init or rc script used to manage the service

`salt.states.service.`**`mod_watch`**(*name*, *sfun=None*, *sig=None*, *reload=False*, *full_restart=False*, *force=False*, ***kwargs*)
The service watcher, called to invoke the watch command.

name The name of the init or rc script used to manage the service

sfun The original function which triggered the mod_watch call (*service.running*, for example).

sig The string to search for when looking for the service process with ps

`salt.states.service.`**`running`**(*name*, *enable=None*, *sig=None*, ***kwargs*)
Verify that the service is running

name The name of the init or rc script used to manage the service

enable Set the service to be enabled at boot time, True sets the service to be enabled, False sets the named service to be disabled. The default is None, which does not enable or disable anything.

sig The string to search for when looking for the service process with ps

22.27.119 salt.states.slack

Send a message to Slack

This state is useful for sending messages to Slack during state runs.

New in version 2015.5.0.

```
slack-message:
  slack.send_message:
    - channel: '#general'
    - from_name: SuperAdmin
    - message: 'This state was executed successfully.'
    - api_key: peWcBiMOS9HrZG15peWcBiMOS9HrZG15
```

The api key can be specified in the master or minion configuration like below: .. code-block:: yaml

slack: api_key: peWcBiMOS9HrZG15peWcBiMOS9HrZG15

`salt.states.slack.`**`post_message`**(*name*, *channel*, *from_name*, *message*, *api_key=None*)
Send a message to a Slack room.

```
    slack-message:
      slack.send_message:
        - channel: '#general'
        - from_name: SuperAdmin
        - message: 'This state was executed successfully.'
        - api_key: peWcBiMOS9HrZG15peWcBiMOS9HrZG15
```

The following parameters are required:

name The unique name for this event.

channel The room to send the message to. Can either be the ID or the name.

from_name The name of that is to be shown in the ``from'' field. If not specified, defaults to.

message The message that is to be sent to the Hipchat room.

The following parameters are optional: api_key

The api key for Slack to use for authentication, if not specified in the configuration options of master or minion.

22.27.120 salt.states.smtp

Sending Messages via SMTP

New in version 2014.7.0.

This state is useful for firing messages during state runs, using the XMPP protocol

```
server-warning-message:
  smtp.send_msg:
    - name: 'This is a server warning message'
    - profile: my-smtp-account
    - recipient: admins@example.com
```

salt.states.smtp.**send_msg**(*name*, *recipient*, *subject*, *sender*, *profile*, *use_ssl='True'*)

Send a message via SMTP

```
    server-warning-message:
      smtp.send_msg:
        - name: 'This is a server warning message'
        - profile: my-smtp-account
        - subject: 'Message from Salt'
        - recipient: admin@example.com
        - sender: admin@example.com
        - use_ssl: True
```

name The message to send via SMTP

22.27.121 salt.states.splunk_search

Splunk Search State Module

New in version 2015.5.0.

This state is used to ensure presence of splunk searches.

```
server-warning-message:
  splunk_search.present:
    - name: This is the splunk search name
    - search: index=main sourcetype=
```

salt.states.splunk_search.**absent**(*name*, *profile='splunk'*)

Ensure a search is absent

```
    API Error Search:
      splunk_search.absent
```

The following parameters are required:

name This is the name of the search in splunk

salt.states.splunk_search.**present**(*name*, *profile='splunk'*, ***kwargs*)

Ensure a search is present

```
    API Error Search:
      splunk_search.present:
        search: index=main sourcetype=blah
        template: alert_5min
```

The following parameters are required:

name This is the name of the search in splunk

22.27.122 salt.states.ssh_auth

Control of entries in SSH authorized_key files

The information stored in a user's SSH authorized key file can be easily controlled via the ssh_auth state. Defaults can be set by the enc, options, and comment keys. These defaults can be overridden by including them in the name.

Since the YAML specification limits the length of simple keys to 1024 characters, and since SSH keys are often longer than that, you may have to use a YAML `explicit key', as demonstrated in the second example below.

```
AAAAB3NzaC1kc3MAAACBAL0sQ9fJ5bYTEyY==:
  ssh_auth.present:
    - user: root
    - enc: ssh-dss

? AAAAB3NzaC1kc3MAAACBAL0sQ9fJ5bYTEyY==...
:
  ssh_auth.present:
    - user: root
    - enc: ssh-dss

thatch:
  ssh_auth.present:
    - user: root
    - source: salt://ssh_keys/thatch.id_rsa.pub

sshkeys:
  ssh_auth.present:
    - user: root
    - enc: ssh-rsa
```

```
    - options:
      - option1="value1"
      - option2="value2 flag2"
    - comment: myuser
    - names:
      - AAAAB3NzaC1kc3MAAACBAL0sQ9fJ5bYTEyY==
      - ssh-dss AAAAB3NzaCL0sQ9fJ5bYTEyY== user@domain
      - option3="value3" ssh-dss AAAAB3NzaC1kcQ9J5bYTEyY== other@testdomain
      - AAAAB3NzaC1kcQ9fJFF435bYTEyY== newcomment
```

salt.states.ssh_auth.**absent**(*name*, *user*, *enc='ssh-rsa'*, *comment=''*, *options=None*, *config='.ssh/authorized_keys'*)

Verifies that the specified SSH key is absent

name The SSH key to manage

user The user who owns the SSH authorized keys file to modify

enc Defines what type of key is being used; can be ed25519, ecdsa, ssh-rsa or ssh-dss

comment The comment to be placed with the SSH public key

options The options passed to the key, pass a list object

config The location of the authorized keys file relative to the user's home directory, defaults to ".ssh/authorized_keys"

salt.states.ssh_auth.**present**(*name*, *user*, *enc='ssh-rsa'*, *comment=''*, *source=''*, *options=None*, *config='.ssh/authorized_keys'*, ***kwargs*)

Verifies that the specified SSH key is present for the specified user

name The SSH key to manage

user The user who owns the SSH authorized keys file to modify

enc Defines what type of key is being used; can be ed25519, ecdsa, ssh-rsa or ssh-dss

comment The comment to be placed with the SSH public key

source The source file for the key(s). Can contain any number of public keys, in standard ``authorized_keys'' format. If this is set, comment, enc, and options will be ignored.

Note: The source file must contain keys in the format `<enc> <key> <comment>`. If you have generated a keypair using PuTTYgen, then you will need to do the following to retrieve an OpenSSH-compatible public key.

1. In PuTTYgen, click `Load`, and select the *private* key file (not the public key), and click `Open`.
2. Copy the public key from the box labeled `Public key for pasting into OpenSSH authorized_keys file`.
3. Paste it into a new file.

---

options The options passed to the key, pass a list object

config The location of the authorized keys file relative to the user's home directory, defaults to ".ssh/authorized_keys"

22.27.123 salt.states.ssh_known_hosts

Control of SSH known_hosts entries

Manage the information stored in the known_hosts files.

```
github.com:
  ssh_known_hosts:
    - present
    - user: root
    - fingerprint: 16:27:ac:a5:76:28:2d:36:63:1b:56:4d:eb:df:a6:48

example.com:
  ssh_known_hosts:
    - absent
    - user: root
```

`salt.states.ssh_known_hosts.`**`absent`**(*name*, *user=None*, *config=None*)

Verifies that the specified host is not known by the given user

name The host name

user The user who owns the ssh authorized keys file to modify

config The location of the authorized keys file relative to the user's home directory, defaults to ".ssh/known_hosts". If no user is specified, defaults to ``/etc/ssh/ssh_known_hosts". If present, must be an absolute path when a user is not specified.

`salt.states.ssh_known_hosts.`**`present`**(*name*, *user=None*, *fingerprint=None*, *key=None*, *port=None*, *enc=None*, *config=None*, *hash_hostname=True*)

Verifies that the specified host is known by the specified user

On many systems, specifically those running with openssh 4 or older, the `enc` option must be set, only openssh 5 and above can detect the key type.

name The name of the remote host (e.g. ``github.com")

user The user who owns the ssh authorized keys file to modify

enc Defines what type of key is being used, can be ed25519, ecdsa ssh-rsa or ssh-dss

fingerprint The fingerprint of the key which must be presented in the known_hosts file

port optional parameter, denoting the port of the remote host, which will be used in case, if the public key will be requested from it. By default the port 22 is used.

config The location of the authorized keys file relative to the user's home directory, defaults to ".ssh/known_hosts". If no user is specified, defaults to ``/etc/ssh/ssh_known_hosts". If present, must be an absolute path when a user is not specified.

hash_hostname [True] Hash all hostnames and addresses in the output.

22.27.124 salt.states.stateconf

Stateconf System

The stateconf system is intended for use only with the stateconf renderer. This State module presents the set function. This function does not execute any functionality, but is used to interact with the stateconf renderer.

salt.states.stateconf.**context**(*name*, ***kwargs*)
No-op state to support state config via the stateconf renderer.

salt.states.stateconf.**set**(*name*, ***kwargs*)
No-op state to support state config via the stateconf renderer.

22.27.125 salt.states.status

Minion status monitoring

Maps to the *status* execution module.

salt.states.status.**loadavg**(*name*, *maximum=None*, *minimum=None*)
Return the current load average for the specified minion. Available values for name are *1-min*, *5-min* and *15-min*. *minimum* and *maximum* values should be passed in as strings.

salt.states.status.**process**(*name*)
Return whether the specified signature is found in the process tree. This differs slightly from the services states, in that it may refer to a process that is not managed via the init system.

22.27.126 salt.states.supervisord

Interaction with the Supervisor daemon

```
wsgi_server:
  supervisord.running:
    - require:
      - pkg: supervisor
    - watch:
      - file: /etc/nginx/sites-enabled/wsgi_server.conf
```

salt.states.supervisord.**dead**(*name*, *user=None*, *conf_file=None*, *bin_env=None*)
Ensure the named service is dead (not running).

name Service name as defined in the supervisor configuration file

user Name of the user to run the supervisorctl command

New in version 0.17.0.

conf_file path to supervisorctl config file

bin_env path to supervisorctl bin or path to virtualenv with supervisor installed

salt.states.supervisord.**mod_watch**(*name*, *restart=True*, *update=False*, *user=None*, *conf_file=None*, *bin_env=None*, ***kwargs*)

salt.states.supervisord.**running**(*name*, *restart=False*, *update=False*, *user=None*, *conf_file=None*, *bin_env=None*)
Ensure the named service is running.

name Service name as defined in the supervisor configuration file

restart Whether to force a restart

update Whether to update the supervisor configuration.

user Name of the user to run the supervisorctl command

New in version 0.17.0.

conf_file path to supervisorctl config file

bin_env path to supervisorctl bin or path to virtualenv with supervisor installed

22.27.127 salt.states.svn

Manage SVN repositories

Manage repository checkouts via the svn vcs system:

```
http://unladen-swallow.googlecode.com/svn/trunk/:
  svn.latest:
    - target: /tmp/swallow
```

salt.states.svn.**dirty**(*name*, *target*, *user=None*, *username=None*, *password=None*, *ignore_unversioned=False*)

Determine if the working directory has been changed.

salt.states.svn.**export**(*name*, *target=None*, *rev=None*, *user=None*, *username=None*, *password=None*, *force=False*, *overwrite=False*, *externals=True*, *trust=False*)

Export a file or directory from an SVN repository

name Address and path to the file or directory to be exported.

target Name of the target directory where the checkout will put the working directory

rev [None] The name revision number to checkout. Enable ``force'' if the directory already exists.

user [None] Name of the user performing repository management operations

username [None] The user to access the name repository with. The svn default is the current user

password Connect to the Subversion server with this password

New in version 0.17.0.

force [False] Continue if conflicts are encountered

overwrite [False] Overwrite existing target

externals [True] Change to False to not checkout or update externals

trust [False] Automatically trust the remote server. SVN's --trust-server-cert

salt.states.svn.**latest**(*name*, *target=None*, *rev=None*, *user=None*, *username=None*, *password=None*, *force=False*, *externals=True*, *trust=False*)

Checkout or update the working directory to the latest revision from the remote repository.

name Address of the name repository as passed to ``svn checkout''

target Name of the target directory where the checkout will put the working directory

rev [None] The name revision number to checkout. Enable ``force'' if the directory already exists.

user [None] Name of the user performing repository management operations

username [None] The user to access the name repository with. The svn default is the current user

password Connect to the Subversion server with this password

New in version 0.17.0.

force [False] Continue if conflicts are encountered

externals [True] Change to False to not checkout or update externals

trust [False] Automatically trust the remote server. SVN's --trust-server-cert

22.27.128 salt.states.sysctl

Configuration of the Linux kernel using sysctl

Control the kernel sysctl system.

```
vm.swappiness:
  sysctl.present:
    - value: 20
```

`salt.states.sysctl.`**`present`**(*name*, *value*, *config=None*)

Ensure that the named sysctl value is set in memory and persisted to the named configuration file. The default sysctl configuration file is /etc/sysctl.conf

name The name of the sysctl value to edit

value The sysctl value to apply

config The location of the sysctl configuration file. If not specified, the proper location will be detected based on platform.

22.27.129 salt.states.syslog_ng

State module for syslog_ng

maintainer Tibor Benke <btibi@sch.bme.hu>

maturity new

depends cmd, ps, syslog_ng

platform all

Users can generate syslog-ng configuration files from YAML format or use plain ones and reload, start, or stop their syslog-ng by using this module.

Details

The service module is not available on all system, so this module includes `syslog_ng.reloaded`, `syslog_ng.stopped`, and `syslog_ng.started` functions. If the service module is available on the computers, users should use that.

Users can generate syslog-ng configuration with `syslog_ng.config` function. For more information see *syslog-ng state usage*.

Syslog-ng configuration file format

The syntax of a configuration snippet in syslog-ng.conf:

```
object_type object_id {<options>};
```

These constructions are also called statements. There are options inside of them:

```
option(parameter1, parameter2); option2(parameter1, parameter2);
```

You can find more information about syslog-ng's configuration syntax in the Syslog-ng Admin guide: http://www.balabit.com/sites/default/files/documents/syslog-ng-ose-3.5-guides/en/syslog-ng-ose-v3.5-guide-admin/html-single/index.html#syslog-ng.conf.5

`salt.states.syslog_ng.`**`config`**(*name*, *config*, *write=True*)
Builds syslog-ng configuration.

name : the id of the Salt document config : the parsed YAML code write : if True, it writes the config into the configuration file, otherwise just returns it

`salt.states.syslog_ng.`**`reloaded`**(*name*)
Reloads syslog-ng.

`salt.states.syslog_ng.`**`started`**(*name=None*, *user=None*, *group=None*, *chroot=None*, *caps=None*, *no_caps=False*, *pidfile=None*, *enable_core=False*, *fd_limit=None*, *verbose=False*, *debug=False*, *trace=False*, *yydebug=False*, *persist_file=None*, *control=None*, *worker_threads=None*, **args*, ***kwargs*)
Ensures, that syslog-ng is started via the given parameters.

Users shouldn't use this function, if the service module is available on their system.

`salt.states.syslog_ng.`**`stopped`**(*name=None*)
Kills syslog-ng.

22.27.130 salt.states.sysrc

`salt.states.sysrc.`**`absent`**(*name*, ***kwargs*)
Ensure a sysrc variable is absent.

name The variable name to set

file (optional) The rc file to add the variable to.

jail (option) the name or JID of the jail to set the value in.

`salt.states.sysrc.`**`managed`**(*name*, *value*, ***kwargs*)
Ensure a sysrc variable is set to a specific value.

name The variable name to set

value Value to set the variable to

file (optional) The rc file to add the variable to.

jail (option) the name or JID of the jail to set the value in.

22.27.131 salt.states.test

Test States

Provide test case states that enable easy testing of things to do with state calls, e.g. running, calling, logging, output filtering etc.

```
always-passes:
  test.succeed_without_changes:
    - name: foo

always-fails:
```

```
  test.fail_without_changes:
    - name: foo

always-changes-and-succeeds:
  test.succeed_with_changes:
    - name: foo

always-changes-and-fails:
  test.fail_with_changes:
    - name: foo

my-custom-combo:
  test.configurable_test_state:
    - name: foo
    - changes: True
    - result: False
    - comment: bar.baz
```

salt.states.test.**configurable_test_state**(*name*, *changes=True*, *result=True*, *comment=''*)

A configurable test state which determines its output based on the inputs.

New in version 2014.7.0.

name: A unique string.

changes: Do we return anything in the changes field? Accepts True, False, and `Random' Default is True

result: Do we return successfully or not? Accepts True, False, and `Random' Default is True

comment: String to fill the comment field with. Default is `'

salt.states.test.**fail_with_changes**(*name*)

Returns failure and changes is not empty.

New in version 2014.7.0.

name: A unique string.

salt.states.test.**fail_without_changes**(*name*)

Returns failure.

New in version 2014.7.0.

name: A unique string.

salt.states.test.**mod_watch**(*name*, *sfun=None*, ***kwargs*)

` Call this function via a watch statement

New in version 2014.7.0.

Any parameters in the state return dictionary can be customized by adding the keywords `result`, `comment`, and `changes`.

```
    this_state_will_return_changes:
      test.succeed_with_changes

    this_state_will_NOT_return_changes:
      test.succeed_without_changes

    this_state_is_watching_another_state:
      test.succeed_without_changes:
        - comment: 'This is a custom comment'
        - watch:
```

```
          - test: this_state_will_return_changes
          - test: this_state_will_NOT_return_changes

    this_state_is_also_watching_another_state:
      test.succeed_without_changes:
        - watch:
          - test: this_state_will_NOT_return_changes
```

salt.states.test.**succeed_with_changes**(*name*)

Returns successful and changes is not empty

New in version 2014.7.0.

name: A unique string.

salt.states.test.**succeed_without_changes**(*name*)

Returns successful.

New in version 2014.7.0.

name A unique string.

22.27.132 salt.states.timezone

Management of timezones

The timezone can be managed for the system:

```
America/Denver:
  timezone.system
```

The system and the hardware clock are not necessarily set to the same time. By default, the hardware clock is set to localtime, meaning it is set to the same time as the system clock. If *utc* is set to True, then the hardware clock will be set to UTC, and the system clock will be an offset of that.

```
America/Denver:
  timezone.system:
    - utc: True
```

The Ubuntu community documentation contains an explanation of this setting, as it applies to systems that dual-boot with Windows. This is explained in greater detail here.

salt.states.timezone.**system**(*name*, *utc=True*)

Set the timezone for the system.

name The name of the timezone to use (e.g.: America/Denver)

utc Whether or not to set the hardware clock to UTC (default is True)

22.27.133 salt.states.tomcat

This state uses the manager webapp to manage Apache tomcat webapps This state requires the manager webapp to be enabled

The following grains/pillar should be set:

```
tomcat-manager:user: admin user name
tomcat-manager:passwd: password
```

and also configure a user in the conf/tomcat-users.xml file:

```
<?xml version='1.0' encoding='utf-8'?>
<tomcat-users>
    <role rolename="manager-script"/>
    <user username="tomcat" password="tomcat" roles="manager-script"/>
</tomcat-users>
```

Notes:

- Not supported multiple version on the same context path
- **More information about tomcat manager**: http://tomcat.apache.org/tomcat-7.0-doc/manager-howto.html
- **if you use only this module for deployments you might want to restrict** access to the manager so its only accessible via localhost for more info: http://tomcat.apache.org/tomcat-7.0-doc/manager-howto.html#Configuring_Manager_Application_Access
- Tested on:

  JVM Vendor: Sun Microsystems Inc.

  JVM Version: 1.6.0_43-b01

  OS Architecture: amd64

  OS Name: Linux

  OS Version: 2.6.32-358.el6.x86_64

  Tomcat Version: Apache Tomcat/7.0.37

`salt.states.tomcat.`**`mod_watch`**(*name*, *url='http://localhost:8080/manager'*, *timeout=180*)
The tomcat watcher function. When called it will reload the webapp in question

`salt.states.tomcat.`**`undeployed`**(*name*, *url='http://localhost:8080/manager'*, *timeout=180*)
Enforce that the WAR will be un-deployed from the server

name the context path to deploy

url [http://localhost:8080/manager] the URL of the server manager webapp

timeout [180] timeout for HTTP request to the tomcat manager

Example:

```
    jenkins:
      tomcat.undeployed:
        - name: /ran
        - require:
          - service: application-service
```

`salt.states.tomcat.`**`wait`**(*name*, *url='http://localhost:8080/manager'*, *timeout=180*)
Wait for the tomcat manager to load

Notice that if tomcat is not running we won't wait for it start and the state will fail. This state can be required in the tomcat.war_deployed state to make sure tomcat is running and that the manager is running as well and ready for deployment

url [http://localhost:8080/manager] the URL of the server manager webapp

timeout [180] timeout for HTTP request to the tomcat manager

Example:

```
tomcat-service:
  service.running:
    - name: tomcat
    - enable: True

wait-for-tomcatmanager:
  tomcat.wait:
    - timeout: 300
    - require:
      - service: tomcat-service

jenkins:
  tomcat.war_deployed:
    - name: /ran
    - war: salt://jenkins-1.2.4.war
    - require:
      - tomcat: wait-for-tomcatmanager
```

salt.states.tomcat.**war_deployed**(*name*, *war*, *force=False*, *url='http://localhost:8080/manager'*, *timeout=180*, *temp_war_location=None*)

Enforce that the WAR will be deployed and started in the context path it will make use of WAR versions

for more info: http://tomcat.apache.org/tomcat-7.0-doc/config/context.html#Naming

name the context path to deploy

war absolute path to WAR file (should be accessible by the user running tomcat) or a path supported by the salt.modules.cp.get_url function

force force deploy even if version strings are the same, False by default.

url [http://localhost:8080/manager] the URL of the server manager webapp

timeout [180] timeout for HTTP request to the tomcat manager

temp_war_location [None] use another location to temporarily copy to war file by default the system's temp directory is used

Example:

```
jenkins:
  tomcat.war_deployed:
    - name: /ran
    - war: salt://jenkins-1.2.4.war
    - require:
      - service: application-service
```

22.27.134 salt.states.user

Management of user accounts

The user module is used to create and manage user settings, users can be set as either absent or present

```
fred:
  user.present:
    - fullname: Fred Jones
```

```
    - shell: /bin/zsh
    - home: /home/fred
    - uid: 4000
    - gid: 4000
    - groups:
      - wheel
      - storage
      - games

testuser:
  user.absent
```

salt.states.user.**absent**(*name*, *purge=False*, *force=False*)

Ensure that the named user is absent

name The name of the user to remove

purge Set purge to True to delete all of the user's files as well as the user, Default is `False`.

force If the user is logged in, the absent state will fail. Set the force option to True to remove the user even if they are logged in. Not supported in FreeBSD and Solaris, Default is `False`.

salt.states.user.**present**(*name*, *uid=None*, *gid=None*, *gid_from_name=False*, *groups=None*, *optional_groups=None*, *remove_groups=True*, *home=None*, *createhome=True*, *password=None*, *enforce_password=True*, *empty_password=False*, *shell=None*, *unique=True*, *system=False*, *fullname=None*, *roomnumber=None*, *workphone=None*, *homephone=None*, *loginclass=None*, *date=None*, *mindays=None*, *maxdays=None*, *inactdays=None*, *warndays=None*, *expire=None*)

Ensure that the named user is present with the specified properties

name The name of the user to manage

uid The user id to assign, if left empty then the next available user id will be assigned

gid The default group id

gid_from_name If True, the default group id will be set to the id of the group with the same name as the user, Default is `False`.

groups A list of groups to assign the user to, pass a list object. If a group specified here does not exist on the minion, the state will fail. If set to the empty list, the user will be removed from all groups except the default group.

optional_groups A list of groups to assign the user to, pass a list object. If a group specified here does not exist on the minion, the state will silently ignore it.

NOTE: If the same group is specified in both ``groups'' and ``optional_groups'', then it will be assumed to be required and not optional.

remove_groups Remove groups that the user is a member of that weren't specified in the state, Default is `True`.

home The custom login directory of user. Uses default value of underlying system if not set. Notice that this directory does not have to exists. This also the location of the home directory to create if createhome is set to True.

createhome If False, the home directory will not be created if it doesn't exist. Please note that directories leading up to the home directory will NOT be created, Default is `True`.

password A password hash to set for the user. This field is only supported on Linux, FreeBSD, NetBSD, OpenBSD, and Solaris.

Changed in version 0.16.0: BSD support added.

enforce_password Set to False to keep the password from being changed if it has already been set and the password hash differs from what is specified in the ``password'' field. This option will be ignored if ``password'' is not specified, Default is `True`.

empty_password Set to True to enable password-less login for user, Default is `False`.

shell The login shell, defaults to the system default shell

unique Require a unique UID, Default is `True`.

system Choose UID in the range of FIRST_SYSTEM_UID and LAST_SYSTEM_UID, Default is `False`.

loginclass The login class, defaults to empty (BSD only)

User comment field (GECOS) support (currently Linux, BSD, and MacOS only):

The below values should be specified as strings to avoid ambiguities when the values are loaded. (Especially the phone and room number fields which are likely to contain numeric data)

fullname The user's full name

roomnumber The user's room number (not supported in MacOS)

workphone The user's work phone number (not supported in MacOS)

homephone The user's home phone number (not supported in MacOS)

Changed in version 2014.7.0: Shadow attribute support added.

Shadow attributes support (currently Linux only):

The below values should be specified as integers.

date Date of last change of password, represented in days since epoch (January 1, 1970).

mindays The minimum number of days between password changes.

maxdays The maximum number of days between password changes.

inactdays The number of days after a password expires before an account is locked.

warndays Number of days prior to maxdays to warn users.

expire Date that account expires, represented in days since epoch (January 1, 1970).

22.27.135 salt.states.vbox_guest

VirtualBox Guest Additions installer state

`salt.states.vbox_guest.`**`additions_installed`**(*name*, *reboot=False*, *upgrade_os=False*)
Ensure that the VirtualBox Guest Additions are installed. Uses the CD, connected by VirtualBox.

name The name has no functional value and is only used as a tracking reference.

reboot [False] Restart OS to complete installation.

upgrade_os [False] Upgrade OS (to ensure the latests version of kernel and developer tools installed).

`salt.states.vbox_guest.`**`additions_removed`**(*name*, *force=False*)
Ensure that the VirtualBox Guest Additions are removed. Uses the CD, connected by VirtualBox.

To connect VirtualBox Guest Additions via VirtualBox graphical interface press `Host+D' (`Host' is usually `Right Ctrl').

name The name has no functional value and is only used as a tracking reference.

force Force VirtualBox Guest Additions removing.

salt.states.vbox_guest.**grant_access_to_shared_folders_to**(*name*, *users=None*)
Grant access to auto-mounted shared folders to the users.

User is specified by it's name. To grant access for several users use argument *users*.

name Name of the user to grant access to auto-mounted shared folders to.

users List of names of users to grant access to auto-mounted shared folders to. If specified, *name* will not be taken into account.

22.27.136 salt.states.virtualenv

Setup of Python virtualenv sandboxes

salt.states.virtualenv_mod.**managed**(*name*, *venv_bin=None*, *requirements=None*, *system_site_packages=False*, *distribute=False*, *use_wheel=False*, *clear=False*, *python=None*, *extra_search_dir=None*, *never_download=None*, *prompt=None*, *user=None*, *no_chown=False*, *cwd=None*, *index_url=None*, *extra_index_url=None*, *pre_releases=False*, *no_deps=False*, *pip_download=None*, *pip_download_cache=None*, *pip_exists_action=None*, *proxy=None*, *use_vt=False*, *env_vars=None*)
Create a virtualenv and optionally manage it with pip

name Path to the virtualenv

requirements Path to a pip requirements file. If the path begins with `salt://` the file will be transferred from the master file server.

cwd Path to the working directory where ``pip install'' is executed.

use_wheel [False] Prefer wheel archives (requires pip>=1.4)

no_deps: **False** Pass *--no-deps* to *pip*.

pip_exists_action: **None** Default action of pip when a path already exists: (s)witch, (i)gnore, (w)ipe, (b)ackup

proxy: **None** Proxy address which is passed to ``pip install''

env_vars Set environment variables that some builds will depend on. For example, a Python C-module may have a Makefile that needs INCLUDE_PATH set to pick up a header file while compiling.

Also accepts any kwargs that the virtualenv module will.

```
/var/www/myvirtualenv.com:
  virtualenv.managed:
    - system_site_packages: False
    - requirements: salt://REQUIREMENTS.txt
```

22.27.137 salt.states.win_dns_client

Module for configuring DNS Client on Windows systems

salt.states.win_dns_client.**dns_dhcp**(*name*, *interface='Local Area Connection'*)
Configure the DNS server list from DHCP Server

salt.states.win_dns_client.**dns_exists**(*name*, *servers=None*, *interface='Local Area Connection'*, *replace=False*)

Configure the DNS server list in the specified interface

Example:

```
config_dns_servers:
  win_dns_client.dns_exists:
    - replace: True #remove any servers not in the "servers" list, default is False
    - servers:
      - 8.8.8.8
      - 8.8.8.9
```

salt.states.win_dns_client.**primary_suffix**(*name*, *suffix=None*, *updates=False*)

New in version 2014.7.0.

Configure the global primary DNS suffix of a DHCP client.

suffix [None] The suffix which is advertised for this client when acquiring a DHCP lease When none is set, the explicitly configured DNS suffix will be removed.

updates [False] Allow syncing the DNS suffix with the AD domain when the client's AD domain membership changes

```
primary_dns_suffix:
    win_dns_client.primary_suffix:
        - suffix: sub.domain.tld
        - updates: True
```

22.27.138 salt.states.win_firewall

State for configuring Windows Firewall

salt.states.win_firewall.**add_rule**(*name*, *localport*, *protocol='tcp'*, *action='allow'*, *dir='in'*)

Add a new firewall rule (Windows only)

salt.states.win_firewall.**disabled**(*name*)

Disable all the firewall profiles (Windows only)

22.27.139 salt.states.win_network

Configuration of network interfaces on Windows hosts

New in version 2014.1.0.

This module provides the `network` state(s) on Windows hosts. DNS servers, IP addresses and default gateways can currently be managed.

Below is an example of the configuration for an interface that uses DHCP for both DNS servers and IP addresses:

```
Local Area Connection #2:
  network.managed:
    - dns_proto: dhcp
    - ip_proto: dhcp
```

Note: Both the `dns_proto` and `ip_proto` arguments are required.

Static DNS and IP addresses can be configured like so:

```
Local Area Connection #2:
  network.managed:
    - dns_proto: static
    - dns_servers:
      - 8.8.8.8
      - 8.8.4.4
    - ip_proto: static
    - ip_addrs:
      - 10.2.3.4/24
```

Note: IP addresses are specified using the format `<ip-address>/<subnet-length>`. Salt provides a convenience function called `ip.get_subnet_length` to calculate the subnet length from a netmask.

Optionally, if you are setting a static IP address, you can also specify the default gateway using the `gateway` parameter:

```
Local Area Connection #2:
  network.managed:
    - dns_proto: static
    - dns_servers:
      - 8.8.8.8
      - 8.8.4.4
    - ip_proto: static
    - ip_addrs:
      - 10.2.3.4/24
    - gateway: 10.2.3.1
```

salt.states.win_network.**managed**(*name*, *dns_proto=None*, *dns_servers=None*, *ip_proto=None*, *ip_addrs=None*, *gateway=None*, *enabled=True*, ***kwargs*)

Ensure that the named interface is configured properly.

name The name of the interface to manage

dns_proto [None] Set to `static` and use the `dns_servers` parameter to provide a list of DNS nameservers. set to `dhcp` to use DHCP to get the DNS servers.

dns_servers [None] A list of static DNS servers.

ip_proto [None] Set to `static` and use the `ip_addrs` and (optionally) `gateway` parameters to provide a list of static IP addresses and the default gateway. Set to `dhcp` to use DHCP.

ip_addrs [None] A list of static IP addresses.

gateway [None] A list of static IP addresses.

enabled [True] Set to `False` to ensure that this interface is disabled.

22.27.140 salt.states.win_path

Manage the Windows System PATH

salt.states.win_path.**absent**(*name*)

Remove the directory from the SYSTEM path

index: where the directory should be placed in the PATH (default: 0)

Example:

```
'C:\sysinternals':
  win_path.absent
```

salt.states.win_path.**exists**(*name*, *index=None*)

Add the directory to the system PATH at index location

index: where the directory should be placed in the PATH (default: None) [Note: Providing no index will append directory to PATH and will not enforce its location within the PATH.]

Example:

```
'C:\python27':
  win_path.exists

'C:\sysinternals':
  win_path.exists:
    index: 0
```

22.27.141 salt.states.win_servermanager

Manage Windows features via the ServerManager powershell module

salt.states.win_servermanager.**installed**(*name*, *recurse=False*, *force=False*)

Install the windows feature

name: short name of the feature (the right column in win_servermanager.list_available)

recurse: install all sub-features as well

force: if the feature is installed but on of its sub-features are not installed set this to True to force the installation of the sub-features

Note: Some features require reboot after un/installation. If so, until the server is restarted other features can not be installed!

Example:

Run `salt MinionName win_servermanager.list_available` to get a list of available roles and features. Use the name in the right column. Do not use the role or feature names mentioned in the PKGMGR documentation. In this example for IIS-WebServerRole the name to be used is Web-Server.

```
ISWebserverRole:
  win_servermanager.installed:
    - force: True
    - recurse: True
    - name: Web-Server
```

salt.states.win_servermanager.**removed**(*name*)

Remove the windows feature name:

short name of the feature (the right column in win_servermanager.list_available)

Note: Some features require a reboot after uninstallation. If so the feature will not be completly uninstalled until the server is restarted. Example: Run `salt MinionName win_servermanager.list_installed` to get a list of all features installed. Use the top name listed for each feature, not the indented one. Do not use the role or feature names mentioned in the PKGMGR documentation. .. code-block:: yaml

ISWebserverRole:

win_servermanager.removed:

- name: Web-Server

22.27.142 salt.states.win_system

Management of Windows system information

New in version 2014.1.0.

This state is used to manage system information such as the computer name and description.

```
ERIK-WORKSTATION:
  system.computer_name: []

This is Erik's computer, don't touch!:
  system.computer_desc: []
```

salt.states.win_system.**computer_desc**(*name*)
Manage the computer's description field

name The desired computer description

salt.states.win_system.**computer_name**(*name*)
Manage the computer's name

name The desired computer name

22.27.143 salt.states.win_update

Management of the windows update agent

New in version 2014.7.0.

Set windows updates to run by category. Default behavior is to install all updates that do not require user interaction to complete.

Optionally set `category` to a category of your choice to only install certain updates. Default is to set to install all available updates.

The following example will install all Security and Critical Updates, and download but not install standard updates.

```
updates:
  win_update.installed:
    - categories:
      - 'Critical Updates'
      - 'Security Updates'
  win_update.downloaded:
    - categories:
      - 'Updates'
```

You can also specify a number of features about the update to have a fine grain approach to specific types of updates. These are the following features/states of updates available for configuring:

```
'UI' - User interaction required, skipped by default
'downloaded' - Already downloaded, skipped by default (downloading)
'present' - Present on computer, included by default (installing)
'installed' - Already installed, skipped by default
```

```
'reboot' - Reboot required, included by default
'hidden' - skip those updates that have been hidden.
'software' - Software updates, included by default
'driver' - driver updates, skipped by default
```

The following example installs all driver updates that don't require a reboot:

```
gryffindor:
  win_update.installed:
    - includes:
      - driver: True
      - software: False
      - reboot: False
```

To just update your windows machine, add this your sls:

```
updates:
  win_update.installed
```

class salt.states.win_update.**PyWinUpdater**(*categories=None*, *skipUI=True*, *skipDownloaded=True*, *skipInstalled=True*, *skipReboot=False*, *skipPresent=True*, *softwareUpdates=True*, *driverUpdates=False*, *skipHidden=True*)

AutoSearch()

Download()

GetAvailableCategories()

GetCategories()

GetDownloadResults()

GetInstallationResults()

Install()

Search(*searchString*)

SetCategories(*categories*)

SetInclude(*include*, *state*)

SetIncludes(*includes*)

salt.states.win_update.**downloaded**(*name*, *categories=None*, *includes=None*, *retries=10*)

Cache updates for later install.

name: if categories is left empty, it will be assumed that you are passing the category option through the name. These are separate because you can only have one name, but can have multiple categories.

categories: the list of categories to be downloaded. These are simply strings in the update's information, so there is no enumeration of the categories available. Some known categories:

```
        Updates
        Windows 7
        Critical Updates
        Security Updates
        Update Rollups
```

includes: a list of features of the updates to cull by. Available features:

```
'UI' - User interaction required, skipped by default
'downloaded' - Already downloaded, skipped by default (downloading)
'present' - Present on computer, included by default (installing)
'installed' - Already installed, skipped by default
'reboot' - Reboot required, included by default
'hidden' - skip those updates that have been hidden.
'software' - Software updates, included by default
'driver' - driver updates, skipped by default
```

retries Number of retries to make before giving up. This is total, not per step.

salt.states.win_update.**installed**(*name*, *categories=None*, *includes=None*, *retries=10*)
Install specified windows updates.

name: if categories is left empty, it will be assumed that you are passing the category option through the name. These are separate because you can only have one name, but can have multiple categories.

categories: the list of categories to be downloaded. These are simply strings in the update's information, so there is no enumeration of the categories available. Some known categories:

```
Updates
Windows 7
Critical Updates
Security Updates
Update Rollups
```

includes: a list of features of the updates to cull by. Available features:

```
'UI' - User interaction required, skipped by default
'downloaded' - Already downloaded, skipped by default (downloading)
'present' - Present on computer, included by default (installing)
'installed' - Already installed, skipped by default
'reboot' - Reboot required, included by default
'hidden' - skip those updates that have been hidden.
'software' - Software updates, included by default
'driver' - driver updates, skipped by default
```

retries Number of retries to make before giving up. This is total, not per step.

22.27.144 salt.states.winrepo

Manage Windows Package Repository

salt.states.winrepo.**genrepo**(*name*, *force=False*, *allow_empty=False*)
Refresh the winrepo.p file of the repository (salt-run winrepo.genrepo)

if force is True no checks will be made and the repository will be generated if allow_empty is True then the state will not return an error if there are 0 packages

Example:

```
winrepo:
  winrepo.genrepo
```

22.27.145 salt.states.xmpp

Sending Messages over XMPP

New in version 2014.1.0.

This state is useful for firing messages during state runs, using the XMPP protocol

```
server-warning-message:
  xmpp.send_msg:
    - name: 'This is a server warning message'
    - profile: my-xmpp-account
    - recipient: admins@xmpp.example.com/salt
```

salt.states.xmpp.**send_msg**(*name*, *recipient*, *profile*)

Send a message to an XMPP user

```
    server-warning-message:
      xmpp.send_msg:
        - name: 'This is a server warning message'
        - profile: my-xmpp-account
        - recipient: admins@xmpp.example.com/salt
```

name The message to send to the XMPP user

22.27.146 salt.states.zcbuildout

Management of zc.buildout

This module is inspired from minitage's buildout maker (https://github.com/minitage/minitage/blob/master/src/minitage/core/makers/

New in version Boron.

Note: This state module is beta; the API is subject to change and no promise as to performance or functionality is yet present

---

Available Functions

- built

```
    installed1
      buildout.installed:
        - name: /path/to/buildout

    installed2
      buildout.installed:
        - name: /path/to/buildout
        - parts:
          - a
          - b
        - python: /path/to/pythonpath/bin/python
        - unless: /bin/test_something_installed
        - onlyif: /bin/test_else_installed
```

salt.states.zcbuildout.**installed**(*name*, *config='buildout.cfg'*, *quiet=False*, *parts=None*, *user=None*, *env=()*, *buildout_ver=None*, *test_release=False*, *distribute=None*, *new_st=None*, *offline=False*, *newest=False*, *python='/usr/bin/python2.7'*, *debug=False*, *verbose=False*, *unless=None*, *onlyif=None*, *use_vt=False*, *loglevel='debug'*)

Install buildout in a specific directory

It is a thin wrapper to modules.buildout.buildout

name directory to execute in

quiet

do not output console & logs

config buildout config to use (default: buildout.cfg)

parts specific buildout parts to run

user user used to run buildout as

New in version 2014.1.4.

env environment variables to set when running

buildout_ver force a specific buildout version (1 | 2)

test_release buildout accept test release

new_st Forcing use of setuptools >= 0.7

distribute use distribute over setuptools if possible

offline does buildout run offline

python python to use

debug run buildout with -D debug flag

onlyif Only execute cmd if statement on the host return 0

unless Do not execute cmd if statement on the host return 0

newest run buildout in newest mode

verbose run buildout in verbose mode (-vvvvv)

use_vt Use the new salt VT to stream output [experimental]

loglevel loglevel for buildout commands

22.27.147 salt.states.zk_concurrency

Control concurrency of steps within state execution using zookeeper

This module allows you to ``wrap'' a state's execution with concurrency control. This is useful to protect against all hosts executing highstate simultaneously if your services don't all HUP restart. The common way of protecting against this is to run in batch mode, but that doesn't protect from another person running the same batch command (and thereby having 2x the number of nodes deploying at once).

This module will bock while acquiring a slot, meaning that however the command gets called it will coordinate with zookeeper to ensure that no more than max_concurrency steps are executing with a single path.

```
acquire_lock:
  zk_concurrency.lock:
    - zk_hosts: 'zookeeper:2181'
    - path: /trafficserver
    - max_concurrency: 4
    - prereq:
        - service: trafficserver

trafficserver:
  service.running:
    - watch:
      - file: /etc/trafficserver/records.config

/etc/trafficserver/records.config:
  file.managed:
    - source: salt://records.config

release_lock:
  zk_concurrency.unlock:
    - path: /trafficserver
    - require:
        - service: trafficserver
```

This example would allow the file state to change, but would limit the concurrency of the trafficserver service restart to 4.

salt.states.zk_concurrency.**lock**(*path*, *zk_hosts*, *identifier=None*, *max_concurrency=1*, *timeout=None*, *ephemeral_lease=False*)

Block state execution until you are able to get the lock (or hit the timeout)

salt.states.zk_concurrency.**unlock**(*path*, *zk_hosts=None*, *identifier=None*, *max_concurrency=1*, *ephemeral_lease=False*)

Remove lease from semaphore

22.28 Execution Modules

Salt execution modules are the functions called by the **salt** command.

Note: Salt execution modules are different from state modules and cannot be called directly within state files. You must use the `module` state module to call execution modules within state runs.

See also:

Full list of builtin modules

Salt ships with many modules that cover a wide variety of tasks.

22.28.1 Modules Are Easy to Write!

Writing Salt execution modules is straightforward.

A Salt execution modules is a Python or Cython module placed in a directory called `_modules/` within the `file_roots` as specified by the master config file. By default this is */srv/salt/_modules* on Linux systems.

Modules placed in `_modules/` will be synced to the minions when any of the following Salt functions are called:

- state.highstate
- saltutil.sync_modules
- saltutil.sync_all

Note that a module's default name is its filename (i.e. `foo.py` becomes module `foo`), but that its name can be overridden by using a *__virtual__ function*.

If a Salt module has errors and cannot be imported, the Salt minion will continue to load without issue and the module with errors will simply be omitted.

If adding a Cython module the file must be named `<modulename>.pyx` so that the loader knows that the module needs to be imported as a Cython module. The compilation of the Cython module is automatic and happens when the minion starts, so only the `*.pyx` file is required.

22.28.2 Cross-Calling Modules

All of the Salt execution modules are available to each other and modules can call functions available in other execution modules.

The variable `__salt__` is packed into the modules after they are loaded into the Salt minion.

The `__salt__` variable is a *Python dictionary* containing all of the Salt functions. Dictionary keys are strings representing the names of the modules and the values are the functions themselves.

Salt modules can be cross-called by accessing the value in the `__salt__` dict:

```
def foo(bar):
    return __salt__['cmd.run'](bar)
```

This code will call the *run* function in the `cmd` and pass the argument `bar` to it.

22.28.3 Preloaded Execution Module Data

When interacting with execution modules often it is nice to be able to read information dynamically about the minion or to load in configuration parameters for a module.

Salt allows for different types of data to be loaded into the modules by the minion.

Grains Data

The values detected by the Salt Grains on the minion are available in a *dict* named `__grains__` and can be accessed from within callable objects in the Python modules.

To see the contents of the grains dictionary for a given system in your deployment run the `grains.items()` function:

```
salt 'hostname' grains.items --output=pprint
```

Any value in a grains dictionary can be accessed as any other Python dictionary. For example, the grain representing the minion ID is stored in the `id` key and from an execution module, the value would be stored in `__grains__['id']`.

Module Configuration

Since parameters for configuring a module may be desired, Salt allows for configuration information from the minion configuration file to be passed to execution modules.

Since the minion configuration file is a YAML document, arbitrary configuration data can be passed in the minion config that is read by the modules. It is therefore **strongly** recommended that the values passed in the configuration file match the module name. A value intended for the `test` execution module should be named `test.<value>`.

The test execution module contains usage of the module configuration and the default configuration file for the minion contains the information and format used to pass data to the modules. `salt.modules.test`, `conf/minion`.

22.28.4 Printout Configuration

Since execution module functions can return different data, and the way the data is printed can greatly change the presentation, Salt has a printout configuration.

When writing a module the `__outputter__` dictionary can be declared in the module. The `__outputter__` dictionary contains a mapping of function name to Salt Outputter.

```
__outputter__ = {
                'run': 'txt'
                }
```

This will ensure that the text outputter is used.

22.28.5 Virtual Modules

Sometimes an execution module should be presented in a generic way. A good example of this can be found in the package manager modules. The package manager changes from one operating system to another, but the Salt execution module that interfaces with the package manager can be presented in a generic way.

The Salt modules for package managers all contain a `__virtual__` function which is called to define what systems the module should be loaded on.

The `__virtual__` function is used to return either a *string* or `False`. If False is returned then the module is not loaded, if a string is returned then the module is loaded with the name of the string.

Note: Optionally, modules may additionally return a list of reasons that a module could not be loaded. For example, if a dependency for `my_mod' was not met, a __virtual__ function could do as follows:

return False, ['My Module must be installed before this module can be used.']

This means that the package manager modules can be presented as the `pkg` module regardless of what the actual module is named.

Since `__virtual__` is called before the module is loaded, `__salt__` will be unavailable as it will not have been packed into the module at this point in time.

The package manager modules are among the best example of using the `__virtual__` function. Some examples:

- pacman.py
- yumpkg.py
- aptpkg.py
- at.py

Note: Modules which return a string from `__virtual__` that is already used by a module that ships with Salt will _override_ the stock module.

22.28.6 Documentation

Salt execution modules are documented. The `sys.doc()` function will return the documentation for all available modules:

```
salt '*' sys.doc
```

The `sys.doc` function simply prints out the docstrings found in the modules; when writing Salt execution modules, please follow the formatting conventions for docstrings as they appear in the other modules.

Adding Documentation to Salt Modules

It is strongly suggested that all Salt modules have documentation added.

To add documentation add a Python docstring to the function.

```
def spam(eggs):
    '''
    A function to make some spam with eggs!

    CLI Example::

        salt '*' test.spam eggs
    '''
    return eggs
```

Now when the sys.doc call is executed the docstring will be cleanly returned to the calling terminal.

Documentation added to execution modules in docstrings will automatically be added to the online web-based documentation.

Add Execution Module Metadata

When writing a Python docstring for an execution module, add information about the module using the following field lists:

```
:maintainer:    Thomas Hatch <thatch@saltstack.com, Seth House <shouse@saltstack.com>
:maturity:      new
:depends:       python-mysqldb
:platform:      all
```

The maintainer field is a comma-delimited list of developers who help maintain this module.

The maturity field indicates the level of quality and testing for this module. Standard labels will be determined.

The depends field is a comma-delimited list of modules that this module depends on.

The platform field is a comma-delimited list of platforms that this module is known to run on.

22.28.7 Private Functions

In Salt, Python callable objects contained within an execution module are made available to the Salt minion for use. The only exception to this rule is a callable object with a name starting with an underscore _.

Objects Loaded Into the Salt Minion

```
def foo(bar):
    return bar

class baz:
    def __init__(self, quo):
        pass
```

Objects NOT Loaded into the Salt Minion

```
def _foobar(baz): # Preceded with an _
    return baz

cheese = {} # Not a callable Python object
```

Note: Some callable names also end with an underscore _, to avoid keyword clashes with Python keywords. When using execution modules, or state modules, with these in them the trailing underscore should be omitted.

22.28.8 Useful Decorators for Modules

Depends Decorator

When writing execution modules there are many times where some of the module will work on all hosts but some functions have an external dependency, such as a service that needs to be installed or a binary that needs to be present on the system.

Instead of trying to wrap much of the code in large try/except blocks, a decorator can be used.

If the dependencies passed to the decorator don't exist, then the salt minion will remove those functions from the module on that host.

If a ``fallback_function'' is defined, it will replace the function instead of removing it

```
import logging

from salt.utils.decorators import depends

log = logging.getLogger(__name__)

try:
    import dependency_that_sometimes_exists
except ImportError as e:
    log.trace('Failed to import dependency_that_sometimes_exists: {0}'.format(e))

@depends('dependency_that_sometimes_exists')
```

```
def foo():
    '''
    Function with a dependency on the "dependency_that_sometimes_exists" module,
    if the "dependency_that_sometimes_exists" is missing this function will not exist
    '''
    return True

def _fallback():
    '''
    Fallback function for the depends decorator to replace a function with
    '''
    return '"dependency_that_sometimes_exists" needs to be installed for this function to exist'

@depends('dependency_that_sometimes_exists', fallback_function=_fallback)
def foo():
    '''
    Function with a dependency on the "dependency_that_sometimes_exists" module.
    If the "dependency_that_sometimes_exists" is missing this function will be
    replaced with "_fallback"
    '''
    return True
```

In addition to global dependancies the depends decorator also supports raw booleans.

```
from salt.utils.decorators import depends

HAS_DEP = False
try:
    import dependency_that_sometimes_exists
    HAS_DEP = True
except ImportError:
    pass

@depends(HAS_DEP)
def foo():
    return True
```

22.29 Master Tops

Salt includes a number of built-in subsystems to generate top file data, they are listed listed at *Full list of builtin master tops modules*.

The source for the built-in Salt master tops can be found here: https://github.com/saltstack/salt/blob/develop/salt/tops

22.30 Full list of builtin master tops modules

cobbler	Cobbler Tops
ext_nodes	External Nodes Classifier
mongo	Read tops data from a mongodb collection
reclass_adapter	Read tops data from a reclass database

22.30.1 salt.tops.cobbler

Cobbler Tops

Cobbler Tops is a master tops subsystem used to look up mapping information from Cobbler via its API. The same cobbler.* parameters are used for both the Cobbler tops and Cobbler pillar modules.

```
master_tops:
  cobbler: {}
cobbler.url: https://example.com/cobbler_api #default is http://localhost/cobbler_api
cobbler.user: username # default is no username
cobbler.password: password # default is no password
```

Module Documentation

`salt.tops.cobbler.`**`top`**`(`*`**kwargs`*`)`

Look up top data in Cobbler for a minion.

22.30.2 salt.tops.ext_nodes

External Nodes Classifier

The External Nodes Classifier is a master tops subsystem that retrieves mapping information from major configuration management systems. One of the most common external nodes classifiers system is provided by Cobbler and is called `cobbler-ext-nodes`.

The cobbler-ext-nodes command can be used with this configuration:

```
master_tops:
  ext_nodes: cobbler-ext-nodes
```

It is noteworthy that the Salt system does not directly ingest the data sent from the `cobbler-ext-nodes` command, but converts the data into information that is used by a Salt top file.

Any command can replace the call to `cobbler-ext-nodes' above, but currently the data must be formatted in the same way that the standard `cobbler-ext-nodes' does.

See (admittedly degenerate and probably not complete) example:

```
classes:
  - basepackages
  - database
```

The above essentially is the same as a top.sls containing the following:

```
base:
  '*':
    - basepackages
    - database

base:
  '*':
```

```
    - basepackages
    - database
```

salt.tops.ext_nodes.**top**(***kwargs*)
Run the command configured

22.30.3 salt.tops.mongo

Read tops data from a mongodb collection

This module will load tops data from a mongo collection. It uses the node's id for lookups.

Salt Master Mongo Configuration

The module shares the same base mongo connection variables as `salt.returners.mongo_return`. These variables go in your master config file.

- `mongo.db` - The mongo database to connect to. Defaults to `'salt'`.
- `mongo.host` - The mongo host to connect to. Supports replica sets by specifying all hosts in the set, comma-delimited. Defaults to `'salt'`.
- `mongo.port` - The port that the mongo database is running on. Defaults to `27017`.
- `mongo.user` - The username for connecting to mongo. Only required if you are using mongo authentication. Defaults to `''`.
- `mongo.password` - The password for connecting to mongo. Only required if you are using mongo authentication. Defaults to `''`.

Configuring the Mongo Tops Subsystem

```
master_tops:
  mongo:
    collection: tops
    id_field: _id
    re_replace: ""
    re_pattern: \.example\.com
    states_field: states
    environment_field: environment
```

Module Documentation

salt.tops.mongo.**top**(***kwargs*)
Connect to a mongo database and read per-node tops data.

Parameters:

- *collection*: The mongodb collection to read data from. Defaults to `'tops'`.
- *id_field*: The field in the collection that represents an individual minion id. Defaults to `'_id'`.
- *re_pattern*: If your naming convention in the collection is shorter than the minion id, you can use this to trim the name. *re_pattern* will be used to match the name, and *re_replace* will be used to replace it. Backrefs are supported as they are in the Python standard library. If `None`, no mangling

of the name will be performed - the collection will be searched with the entire minion id. Defaults to None.

- *re_replace*: Use as the replacement value in node ids matched with *re_pattern*. Defaults to ``. Feel free to use backreferences here.
- *states_field*: The name of the field providing a list of states.
- *environment_field*: The name of the field providing the environment. Defaults to environment.

22.30.4 salt.tops.reclass_adapter

Read tops data from a reclass database

This *master_tops* plugin provides access to the **reclass** database, such that state information (top data) are retrieved from **reclass**.

You can find more information about **reclass** at http://reclass.pantsfullofunix.net.

To use the plugin, add it to the master_tops list in the Salt master config and tell **reclass** by way of a few options how and where to find the inventory:

```
master_tops:
  reclass:
    storage_type: yaml_fs
    inventory_base_uri: /srv/salt
```

This would cause **reclass** to read the inventory from YAML files in /srv/salt/nodes and /srv/salt/classes.

If you are also using **reclass** as ext_pillar plugin, and you want to avoid having to specify the same information for both, use YAML anchors (take note of the differing data types for ext_pillar and master_tops):

```
reclass: &reclass
  storage_type: yaml_fs
  inventory_base_uri: /srv/salt
  reclass_source_path: ~/code/reclass

ext_pillar:
  - reclass: *reclass

master_tops:
  reclass: *reclass
```

If you want to run reclass from source, rather than installing it, you can either let the master know via the PYTHONPATH environment variable, or by setting the configuration option, like in the example above.

salt.tops.reclass_adapter.**top**(***kwargs*)
Query **reclass** for the top data (states of the minions).

22.31 Full list of builtin wheel modules

config	Manage the master configuration file
error	Error generator to enable integration testing of salt wheel error handling

Continued on next page

Table 22.15 -- continued from previous page

file_roots	Read in files from the file_root and save files to the file root
key	Wheel system wrapper for key system
minions	Wheel system wrapper for connected minions
pillar_roots	The *pillar_roots* wheel module is used to manage files under the pillar roots directories on the master server.

22.31.1 salt.wheel.config

Manage the master configuration file

salt.wheel.config.**apply**(*key*, *value*)
Set a single key

Note: This will strip comments from your config file

salt.wheel.config.**update_config**(*file_name*, *yaml_contents*)
Update master config with `yaml_contents`.

Writes `yaml_contents` to a file named `file_name.conf` under the folder specified by `default_include`. This folder is named `master.d` by default. Please look at http://docs.saltstack.com/en/latest/ref/configuration/master.html#include-configuration for more information.

Example low data data = {

`username': `salt', `password': `salt', `fun': `config.update_config', `file_name': `gui', `yaml_contents': {`id': 1}, `client': `wheel', `eauth': `pam',

}

salt.wheel.config.**values**()
Return the raw values of the config file

22.31.2 salt.wheel.error

Error generator to enable integration testing of salt wheel error handling

salt.wheel.error.**error**(*name=None*, *message=''*)
If name is None Then return empty dict

Otherwise raise an exception with __name__ from name, message from message

CLI Example:

```
salt-wheel error
salt-wheel error.error name="Exception" message="This is an error."
```

22.31.3 salt.wheel.file_roots

Read in files from the file_root and save files to the file root

salt.wheel.file_roots.**find**(*path*, *saltenv='base'*, *env=None*)
Return a dict of the files located with the given path and environment

salt.wheel.file_roots.**list_env**(*saltenv='base'*, *env=None*)
Return all of the file paths found in an environment

salt.wheel.file_roots.**list_roots**()
Return all of the files names in all available environments

salt.wheel.file_roots.**read**(*path*, *saltenv='base'*, *env=None*)
Read the contents of a text file, if the file is binary then

salt.wheel.file_roots.**write**(*data*, *path*, *saltenv='base'*, *index=0*, *env=None*)
Write the named file, by default the first file found is written, but the index of the file can be specified to write to a lower priority file root

22.31.4 salt.wheel.key

Wheel system wrapper for key system

salt.wheel.key.**accept**(*match*)
Accept keys based on a glob match

salt.wheel.key.**accept_dict**(*match*)
Accept keys based on a dict of keys

salt.wheel.key.**delete**(*match*)
Delete keys based on a glob match

salt.wheel.key.**delete_dict**(*match*)
Delete keys based on a dict of keys

salt.wheel.key.**finger**(*match*)
Return the matching key fingerprints

salt.wheel.key.**gen**(*id_=None*, *keysize=2048*)
Generate a key pair. No keys are stored on the master, a keypair is returned as a dict containing pub and priv keys

salt.wheel.key.**gen_accept**(*id_*, *keysize=2048*, *force=False*)
Generate a key pair then accept the public key. This function returns the key pair in a dict, only the public key is preserved on the master.

salt.wheel.key.**key_str**(*match*)
Return the key strings

salt.wheel.key.**list_**(*match*)
List all the keys under a named status

salt.wheel.key.**list_all**()
List all the keys

salt.wheel.key.**reject**(*match*)
Reject keys based on a glob match

salt.wheel.key.**reject_dict**(*match*)
Reject keys based on a dict of keys

22.31.5 salt.wheel.minions

Wheel system wrapper for connected minions

salt.wheel.minions.**connected**()
List all connected minions on a salt-master

22.31.6 salt.wheel.pillar_roots

The *pillar_roots* wheel module is used to manage files under the pillar roots directories on the master server.

`salt.wheel.pillar_roots.`**`find`**(*path*, *saltenv='base'*, *env=None*)
Return a dict of the files located with the given path and environment

`salt.wheel.pillar_roots.`**`list_env`**(*saltenv='base'*, *env=None*)
Return all of the file paths found in an environment

`salt.wheel.pillar_roots.`**`list_roots`**()
Return all of the files names in all available environments

`salt.wheel.pillar_roots.`**`read`**(*path*, *saltenv='base'*, *env=None*)
Read the contents of a text file, if the file is binary then

`salt.wheel.pillar_roots.`**`write`**(*data*, *path*, *saltenv='base'*, *index=0*, *env=None*)
Write the named file, by default the first file found is written, but the index of the file can be specified to write to a lower priority file root

22.32 Full list of builtin beacon modules

btmp	Beacon to fire events at failed login of users
diskusage	Beacon to monitor disk usage.
inotify	Watch files and translate the changes into salt events
journald	A simple beacon to watch journald for specific entries
load	Beacon to emit system load averages
network_info	Beacon to monitor statistics from ethernet adapters
service	Send events covering service status
sh	Watch the shell commands being executed actively.
twilio_txt_msg	Beacon to emit Twilio text messages
wtmp	Beacon to fire events at login of users as registered in the wtmp file

22.32.1 salt.beacons.btmp

Beacon to fire events at failed login of users

```
beacons:
  btmp: {}
```

`salt.beacons.btmp.`**`beacon`**(*config*)
Read the last btmp file and return information on the failed logins

```
    beacons:
      btmp: {}
```

22.32.2 salt.beacons.diskusage

Beacon to monitor disk usage.

New in version 2015.5.0.

salt.beacons.diskusage.**beacon**(*config*)
Monitor the disk usage of the minion

Specify thresholds for for each load average and only emit a beacon if any of them are exceeded.

code_block:: yaml

beacons:

- diskusage: - /: 63% - /mnt/nfs: 50%

22.32.3 salt.beacons.inotify

Watch files and translate the changes into salt events

salt.beacons.inotify.**beacon**(*config*)
Watch the configured files

Example Config

```
beacons:
  inotify:
    /path/to/file/or/dir:
      mask:
        - open
        - create
        - close_write
      recurse: True
      auto_add: True
```

The mask can be a single option from: access attrib close_nowrite close_write create delete delete_self excl_unlink ignored modify moved_from moved_to move_self oneshot onlydir open unmount

recurse: Tell the beacon to recursively watch files in the directory

auto_add: Automatically start adding files that are created in the watched directory

22.32.4 salt.beacons.journald

A simple beacon to watch journald for specific entries

salt.beacons.journald.**beacon**(*config*)
The journald beacon allows for the systemd jornal to be parsed and linked objects to be turned into events.

This beacons config will return all sshd jornal entries

```
beacons:
    journald:
        sshd:
            SYSLOG_IDENTIFIER: sshd
            PRIORITY: 6
```

22.32.5 salt.beacons.load

Beacon to emit system load averages

salt.beacons.load.**beacon**(*config*)

Emit the load averages of this host.

Specify thresholds for for each load average and only emit a beacon if any of them are exceeded.

```
beacons:
  - load:
    - 1m:
      - 0.0
      - 2.0
    - 5m:
      - 0.0
      - 1.5
    - 15m:
      - 0.1
      - 1.0
```

22.32.6 salt.beacons.network_info

Beacon to monitor statistics from ethernet adapters

New in version 2015.5.0.

salt.beacons.network_info.**beacon**(*config*)

Emit the network statistics of this host.

Specify thresholds for for each network stat and only emit a beacon if any of them are exceeded.

code_block:: yaml

Emit beacon when any values are equal to configured values.

beacons:

network_info:

eth0:

- type: equal
- bytes_sent: 100000
- bytes_recv: 100000
- packets_sent: 100000
- packets_recv: 100000
- errin: 100
- errout: 100
- dropin: 100
- dropout: 100

Emit beacon when any values are greater than to configured values.

beacons:

network_info:

eth0:

- type: greater

- bytes_sent: 100000
- bytes_recv: 100000
- packets_sent: 100000
- packets_recv: 100000
- errin: 100
- errout: 100
- dropin: 100
- dropout: 100

22.32.7 salt.beacons.service

Send events covering service status

`salt.beacons.service.`**`beacon`**(*config*)
Scan for the configured services and fire events

Example Config

```
beacons:
  service:
    salt-master:
    mysql:
```

The config above sets up beacons to check for the salt-master and mysql services.

The config also supports two other parameters for each service:

onchangeonly: when *onchangeonly* is True the beacon will fire events only when the service status changes. Otherwise, it will fire an event at each beacon interval. The default is False.

uncleanshutdown: If *uncleanshutdown* is present it should point to the location of a pid file for the service. Most services will not clean up this pid file if they are shutdown uncleanly (e.g. via *kill -9*) or if they are terminated through a crash such as a segmentation fault. If the file is present, then the beacon will add *uncleanshutdown: True* to the event. If not present, the field will be False. The field is only added when the service is NOT running. Omitting the configuration variable altogether will turn this feature off.

Here is an example that will fire an event whenever the state of nginx changes and report an uncleanshutdown. This example is for Arch, which places nginx's pid file in */run*.

```
beacons:
  service:
    nginx:
      onchangeonly: True
      uncleanshutdown: /run/nginx.pid
```

22.32.8 salt.beacons.sh

Watch the shell commands being executed actively. This beacon requires strace.

`salt.beacons.sh.`**`beacon`**(*config*)
Scan the shell execve routines. This beacon will convert all login shells

```
    beacons:
      sh: {}
```

22.32.9 salt.beacons.twilio_txt_msg

Beacon to emit Twilio text messages

salt.beacons.twilio_txt_msg.**beacon**(*config*)
Emit a dict name ``texts'' whose value is a list of texts.

```
    beacons:
      twilio_txt_msg:
        account_sid: "<account sid>"
        auth_token: "<auth token>"
        twilio_number: "+15555555555"
        poll_interval: 10
```

poll_interval defaults to 10 seconds

22.32.10 salt.beacons.wtmp

Beacon to fire events at login of users as registered in the wtmp file

```
beacons:
  wtmp: {}
```

salt.beacons.wtmp.**beacon**(*config*)
Read the last wtmp file and return information on the logins

```
    beacons:
      wtmp: {}
```

22.33 Full list of builtin sdb modules

etcd_db	etcd Database Module
keyring_db	Keyring Database Module
memcached	Memcached sdb Module
sqlite3	SQLite sdb Module

22.33.1 salt.sdb.etcd_db

etcd Database Module

maintainer SaltStack

maturity New

depends python-etcd

platform all

New in version 2015.5.0.

This module allows access to the etcd database using an `sdb://` URI. This package is located at `https://pypi.python.org/pypi/python-etcd`.

Like all sdb modules, the etcd module requires a configuration profile to be configured in either the minion or master configuration file. This profile requires very little. In the example:

```
myetcd:
  driver: etcd
  etcd.host: 127.0.0.1
  etcd.port: 4001
```

The `driver` refers to the etcd module, `etcd.host` refers to the host that is hosting the etcd database and `etcd.port` refers to the port on that host.

```
password: sdb://myetcd/mypassword
```

`salt.sdb.etcd_db.`**`get`**(*key*, *service=None*, *profile=None*)
Get a value from the etcd service

`salt.sdb.etcd_db.`**`set_`**(*key*, *value*, *service=None*, *profile=None*)
Set a key/value pair in the etcd service

22.33.2 salt.sdb.keyring_db

Keyring Database Module

maintainer SaltStack

maturity New

depends keyring

platform all

This module allows access to the keyring package using an `sdb://` URI. This package is located at `https://pypi.python.org/pypi/keyring`.

Care must be taken when using keyring. Not all keyend backends are supported on all operating systems. Also, many backends require an agent to be running in order to work. For instance, the ``Secret Service'' backend requires a compatible agent such as `gnome-keyring-daemon` or `kwallet` to be running. The keyczar backend does not seem to enjoy the benefits of an agent, and so using it will require either that the password is typed in manually (which is unreasonable for the salt-minion and salt-master daemons, especially in production) or an agent is written for it.

Like all sdb modules, the keyring module requires a configuration profile to be configured in either the minion or master configuration file. This profile requires very little. In the example:

```
mykeyring:
  driver: keyring
  service: system
```

The `driver` refers to the keyring module, `service` refers to the service that will be used inside of keyring (which may be likened unto a database table) and `mykeyring` refers to the name that will appear in the URI:

```
password: sdb://mykeyring/mypassword
```

The underlying backend configuration must be configured via keyring itself. For examples and documentation, see keyring:

https://pypi.python.org/pypi/keyring

New in version 2014.1.4.

`salt.sdb.keyring_db.get(key, service=None, profile=None)`
Get a value from a keyring service

`salt.sdb.keyring_db.set_(key, value, service=None, profile=None)`
Set a key/value pair in a keyring service

22.33.3 salt.sdb.memcached

Memcached sdb Module

maintainer SaltStack

maturity New

depends python-memcached

platform all

This module allows access to memcached using an `sdb://` URI. This package is located at `https://pypi.python.org/pypi/python-memcached`.

Like all sdb modules, the memcached module requires a configuration profile to be configured in either the minion or master configuration file. This profile requires very little. In the example:

```
mymemcache:
  driver: memcached
  host: localhost
  port: 11211
```

The `driver` refers to the memcached module, `host` and `port` the memcached server to connect to (defaults to `localhost` and `11211`, and `mymemcached` refers to the name that will appear in the URI:

```
password: sdb://mymemcached/mykey
```

`salt.sdb.memcached.get(key, profile=None)`
Get a value from memcached

`salt.sdb.memcached.set_(key, value, profile=None)`
Set a key/value pair in memcached

22.33.4 salt.sdb.sqlite3

SQLite sdb Module

maintainer SaltStack

maturity New

platform all

This module allows access to sqlite3 using an `sdb://` URI

Like all sdb modules, the sqlite3 module requires a configuration profile to be configured in either the minion or master configuration file. This profile requires very little. For example:

```
mysqlite:
  driver: sqlite3
  database: /tmp/sdb.sqlite
  table: sdb
  create_table: True
```

The `driver` refers to the sqlite3 module, `database` refers to the sqlite3 database file. `table` is the table within the db that will hold keys and values (defaults to `sdb`). The database and table will be created if they do not exist.

Advanced Usage:

Instead of a table name, it is possible to provide custom SQL statements to create the table(s) and get and set values.

`salt.sdb.sqlite3.`**`get`**(*key*, *profile=None*)
Get a value from sqlite3

`salt.sdb.sqlite3.`**`set_`**(*key*, *value*, *profile=None*)
Set a key/value pair in sqlite3

CHAPTER 23

Salt Best Practices

Salt's extreme flexibility leads to many questions concerning the structure of configuration files.

This document exists to clarify these points through examples and code.

23.1 General rules

1. Modularity and clarity should be emphasized whenever possible.
2. Create clear relations between pillars and states.
3. Use variables when it makes sense but don't overuse them.
4. Store sensitive data in pillar.
5. Don't use grains for matching in your pillar top file for any sensitive pillars.

23.2 Structuring States and Formulas

When structuring Salt States and Formulas it is important to begin with the directory structure. A proper directory structure clearly defines the functionality of each state to the user via visual inspection of the state's name.

Reviewing the MySQL Salt Formula it is clear to see the benefits to the end-user when reviewing a sample of the available states:

```
/srv/salt/mysql/files/
/srv/salt/mysql/client.sls
/srv/salt/mysql/map.jinja
/srv/salt/mysql/python.sls
/srv/salt/mysql/server.sls
```

This directory structure would lead to these states being referenced in a top file in the following way:

```
base:
  'web*':
    - mysql.client
    - mysql.python
  'db*':
    - mysql.server
```

This clear definition ensures that the user is properly informed of what each state will do.

Another example comes from the vim-formula:

```
/srv/salt/vim/files/
/srv/salt/vim/absent.sls
/srv/salt/vim/init.sls
/srv/salt/vim/map.jinja
/srv/salt/vim/nerdtree.sls
/srv/salt/vim/pyflakes.sls
/srv/salt/vim/salt.sls
```

Once again viewing how this would look in a top file:

/srv/salt/top.sls:

```
base:
  'web*':
    - vim
    - vim.nerdtree
    - vim.pyflakes
    - vim.salt
  'db*':
    - vim.absent
```

The usage of a clear top-level directory as well as properly named states reduces the overall complexity and leads a user to both understand what will be included at a glance and where it is located.

In addition *Formulas* should be used as often as possible.

Note: Formulas repositories on the saltstack-formulas GitHub organization should not be pointed to directly from systems that automatically fetch new updates such as GitFS or similar tooling. Instead formulas repositories should be forked on GitHub or cloned locally, where unintended, automatic changes will not take place.

23.3 Structuring Pillar Files

Pillars are used to store secure and insecure data pertaining to minions. When designing the structure of the `/srv/pillar` directory, the pillars contained within should once again be focused on clear and concise data which users can easily review, modify, and understand.

The `/srv/pillar/` directory is primarily controlled by `top.sls`. It should be noted that the pillar `top.sls` is not used as a location to declare variables and their values. The `top.sls` is used as a way to include other pillar files and organize the way they are matched based on environments or grains.

An example `top.sls` may be as simple as the following:

/srv/pillar/top.sls:

```
base:
  '*':
    - packages
```

Or much more complicated, using a variety of matchers:

/srv/pillar/top.sls:

```
base:
  '*':
    - apache
dev:
  'os:Debian':
    - match: grain
    - vim
test:
  '* and not G@os: Debian':
    - match: compound
    - emacs
```

It is clear to see through these examples how the top file provides users with power but when used incorrectly it can lead to confusing configurations. This is why it is important to understand that the top file for pillar is not used for variable definitions.

Each SLS file within the `/srv/pillar/` directory should correspond to the states which it matches.

This would mean that the `apache` pillar file should contain data relevant to Apache. Structuring files in this way once again ensures modularity, and creates a consistent understanding throughout our Salt environment. Users can expect that pillar variables found in an Apache state will live inside of an Apache pillar:

`/srv/salt/pillar/apache.sls`:

```
apache:
  lookup:
    name: httpd
    config:
      tmpl: /etc/httpd/httpd.conf
```

While this pillar file is simple, it shows how a pillar file explicitly relates to the state it is associated with.

23.4 Variable Flexibility

Salt allows users to define variables in SLS files. When creating a state variables should provide users with as much flexibility as possible. This means that variables should be clearly defined and easy to manipulate, and that sane defaults should exist in the event a variable is not properly defined. Looking at several examples shows how these different items can lead to extensive flexibility.

Although it is possible to set variables locally, this is generally not preferred:

`/srv/salt/apache/conf.sls`:

```
{% set name = 'httpd' %}
{% set tmpl = 'salt://apache/files/httpd.conf' %}

include:
  - apache

apache_conf:
  file.managed:
    - name: {{ name }}
    - source: {{ tmpl }}
    - template: jinja
    - user: root
```

```
    - watch_in:
      - service: apache
```

When generating this information it can be easily transitioned to the pillar where data can be overwritten, modified, and applied to multiple states, or locations within a single state:

`/srv/pillar/apache.sls`:

```
apache:
  lookup:
    name: httpd
    config:
      tmpl: salt://apache/files/httpd.conf
```

`/srv/salt/apache/conf.sls`:

```
{% from "apache/map.jinja" import apache with context %}

include:
  - apache

apache_conf:
  file.managed:
    - name: {{ salt['pillar.get']('apache:lookup:name') }}
    - source: {{ salt['pillar.get']('apache:lookup:config:tmpl') }}
    - template: jinja
    - user: root
    - watch_in:
      - service: apache
```

This flexibility provides users with a centralized location to modify variables, which is extremely important as an environment grows.

23.5 Modularity Within States

Ensuring that states are modular is one of the key concepts to understand within Salt. When creating a state a user must consider how many times the state could be re-used, and what it relies on to operate. Below are several examples which will iteratively explain how a user can go from a state which is not very modular to one that is:

`/srv/salt/apache/init.sls`:

```
httpd:
  pkg.installed: []
  service.running:
    - enable: True

/etc/httpd/httpd.conf:
  file.managed:
    - source: salt://apache/files/httpd.conf
    - template: jinja
    - watch_in:
      - service: httpd
```

The example above is probably the worst-case scenario when writing a state. There is a clear lack of focus by naming both the pkg/service, and managed file directly as the state ID. This would lead to changing multiple requires within this state, as well as others that may depend upon the state.

Imagine if a require was used for the `httpd` package in another state, and then suddenly it's a custom package. Now changes need to be made in multiple locations which increases the complexity and leads to a more error prone configuration.

There is also the issue of having the configuration file located in the init, as a user would be unable to simply install the service and use the default conf file.

Our second revision begins to address the referencing by using `- name`, as opposed to direct ID references:

`/srv/salt/apache/init.sls`:

```
apache:
  pkg.installed:
    - name: httpd
  service.running:
    - name: httpd
    - enable: True

apache_conf:
  file.managed:
    - name: /etc/httpd/httpd.conf
    - source: salt://apache/files/httpd.conf
    - template: jinja
    - watch_in:
      - service: apache
```

The above init file is better than our original, yet it has several issues which lead to a lack of modularity. The first of these problems is the usage of static values for items such as the name of the service, the name of the managed file, and the source of the managed file. When these items are hard coded they become difficult to modify and the opportunity to make mistakes arises. It also leads to multiple edits that need to occur when changing these items (imagine if there were dozens of these occurrences throughout the state!). There is also still the concern of the configuration file data living in the same state as the service and package.

In the next example steps will be taken to begin addressing these issues. Starting with the addition of a map.jinja file (as noted in the *Formula documentation*), and modification of static values:

`/srv/salt/apache/map.jinja`:

```
{% set apache = salt['grains.filter_by']({
    'Debian': {
        'server': 'apache2',
        'service': 'apache2',
         'conf': '/etc/apache2/apache.conf',
    },
    'RedHat': {
        'server': 'httpd',
        'service': 'httpd',
        'conf': '/etc/httpd/httpd.conf',
    },
}, merge=salt['pillar.get']('apache:lookup')) %}
```

/srv/pillar/apache.sls:

```
apache:
  lookup:
    config:
      tmpl: salt://apache/files/httpd.conf
```

/srv/salt/apache/init.sls:

```
{% from "apache/map.jinja" import apache with context %}

apache:
  pkg.installed:
    - name: {{ apache.server }}
  service.running:
    - name: {{ apache.service }}
    - enable: True

apache_conf:
  file.managed:
    - name: {{ apache.conf }}
    - source: {{ salt['pillar.get']('apache:lookup:config:tmpl') }}
    - template: jinja
    - user: root
    - watch_in:
      - service: apache
```

The changes to this state now allow us to easily identify the location of the variables, as well as ensuring they are flexible and easy to modify. While this takes another step in the right direction, it is not yet complete. Suppose the user did not want to use the provided conf file, or even their own configuration file, but the default apache conf. With the current state setup this is not possible. To attain this level of modularity this state will need to be broken into two states.

/srv/salt/apache/map.jinja:

```
{% set apache = salt['grains.filter_by']({
    'Debian': {
        'server': 'apache2',
        'service': 'apache2',
         'conf': '/etc/apache2/apache.conf',
    },
    'RedHat': {
        'server': 'httpd',
        'service': 'httpd',
        'conf': '/etc/httpd/httpd.conf',
    },
}, merge=salt['pillar.get']('apache:lookup')) %}
```

/srv/pillar/apache.sls:

```
apache:
  lookup:
    config:
      tmpl: salt://apache/files/httpd.conf
```

/srv/salt/apache/init.sls:

```
{% from "apache/map.jinja" import apache with context %}

apache:
  pkg.installed:
    - name: {{ apache.server }}
  service.running:
    - name: {{ apache.service }}
    - enable: True
```

/srv/salt/apache/conf.sls:

```
{% from "apache/map.jinja" import apache with context %}

include:
  - apache

apache_conf:
  file.managed:
    - name: {{ apache.conf }}
    - source: {{ salt['pillar.get']('apache:lookup:config:tmpl') }}
    - template: jinja
    - user: root
    - watch_in:
      - service: apache
```

This new structure now allows users to choose whether they only wish to install the default Apache, or if they wish, overwrite the default package, service, configuration file location, or the configuration file itself. In addition to this the data has been broken between multiple files allowing for users to identify where they need to change the associated data.

23.6 Storing Secure Data

Secure data refers to any information that you would not wish to share with anyone accessing a server. This could include data such as passwords, keys, or other information.

As all data within a state is accessible by EVERY server that is connected it is important to store secure data within pillar. This will ensure that only those servers which require this secure data have access to it. In this example a use can go from an insecure configuration to one which is only accessible by the appropriate hosts:

/srv/salt/mysql/testerdb.sls:

```
testdb:
  mysql_database.present::
    - name: testerdb
```

/srv/salt/mysql/user.sls:

```
include:
  - mysql.testerdb

testdb_user:
  mysql_user.present:
    - name: frank
    - password: "test3rdb"
```

```
    - host: localhost
    - require:
      - sls: mysql.testerdb
```

Many users would review this state and see that the password is there in plain text, which is quite problematic. It results in several issues which may not be immediately visible.

The first of these issues is clear to most users -- the password being visible in this state. This means that any minion will have a copy of this, and therefore the password which is a major security concern as minions may not be locked down as tightly as the master server.

The other issue that can be encountered is access by users on the master. If everyone has access to the states (or their repository), then they are able to review this password. Keeping your password data accessible by only a few users is critical for both security and peace of mind.

There is also the issue of portability. When a state is configured this way it results in multiple changes needing to be made. This was discussed in the sections above but it is a critical idea to drive home. If states are not portable it may result in more work later!

Fixing this issue is relatively simple, the content just needs to be moved to the associated pillar:

`/srv/pillar/mysql.sls`:

```
mysql:
  lookup:
    name: testerdb
    password: test3rdb
    user: frank
    host: localhost
```

`/srv/salt/mysql/testerdb.sls`:

```
testdb:
  mysql_database.present:
    - name: {{ salt['pillar.get']('mysql:lookup:name') }}
```

`/srv/salt/mysql/user.sls`:

```
include:
  - mysql.testerdb

testdb_user:
  mysql_user.present:
    - name: {{ salt['pillar.get']('mysql:lookup:user') }}
    - password: {{ salt['pillar.get']('mysql:lookup:password') }}
    - host: {{ salt['pillar.get']('mysql:lookup:host') }}
    - require:
      - sls: mysql.testerdb
```

Now that the database details have been moved to the associated pillar file, only machines which are targeted via pillar will have access to these details. Access to users who should not be able to review these details can also be prevented while ensuring that they are still able to write states which take advantage of this information.

CHAPTER 24

Troubleshooting

The intent of the troubleshooting section is to introduce solutions to a number of common issues encountered by users and the tools that are available to aid in developing States and Salt code.

24.1 Troubleshooting the Salt Master

If your Salt master is having issues such as minions not returning data, slow execution times, or a variety of other issues, the following links contain details on troubleshooting the most common issues encountered:

24.1.1 Troubleshooting the Salt Master

Running in the Foreground

A great deal of information is available via the debug logging system, if you are having issues with minions connecting or not starting run the master in the foreground:

```
# salt-master -l debug
```

Anyone wanting to run Salt daemons via a process supervisor such as monit, runit, or supervisord, should omit the `-d` argument to the daemons and run them in the foreground.

What Ports does the Master Need Open?

For the master, TCP ports 4505 and 4506 need to be open. If you've put both your Salt master and minion in debug mode and don't see an acknowledgment that your minion has connected, it could very well be a firewall interfering with the connection. See our *firewall configuration* page for help opening the firewall on various platforms.

If you've opened the correct TCP ports and still aren't seeing connections, check that no additional access control system such as SELinux or AppArmor is blocking Salt.

Too many open files

The salt-master needs at least 2 sockets per host that connects to it, one for the Publisher and one for response port. Thus, large installations may, upon scaling up the number of minions accessing a given master, encounter:

```
12:45:29,289 [salt.master    ][INFO    ] Starting Salt worker process 38
Too many open files
sock != -1 (tcp_listener.cpp:335)
```

The solution to this would be to check the number of files allowed to be opened by the user running salt-master (root by default):

```
[root@salt-master ~]# ulimit -n
1024
```

If this value is not equal to at least twice the number of minions, then it will need to be raised. For example, in an environment with 1800 minions, the `nofile` limit should be set to no less than 3600. This can be done by creating the file `/etc/security/limits.d/99-salt.conf`, with the following contents:

```
root        hard    nofile        4096
root        soft    nofile        4096
```

Replace `root` with the user under which the master runs, if different.

If your master does not have an `/etc/security/limits.d` directory, the lines can simply be appended to `/etc/security/limits.conf`.

As with any change to resource limits, it is best to stay logged into your current shell and open another shell to run `ulimit -n` again and verify that the changes were applied correctly. Additionally, if your master is running upstart, it may be necessary to specify the `nofile` limit in `/etc/default/salt-master` if upstart isn't respecting your resource limits:

```
limit nofile 4096 4096
```

Note: The above is simply an example of how to set these values, and you may wish to increase them even further if your Salt master is doing more than just running Salt.

Salt Master Stops Responding

There are known bugs with ZeroMQ versions less than 2.1.11 which can cause the Salt master to not respond properly. If you're running a ZeroMQ version greater than or equal to 2.1.9, you can work around the bug by setting the sysctls `net.core.rmem_max` and `net.core.wmem_max` to 16777216. Next, set the third field in `net.ipv4.tcp_rmem` and `net.ipv4.tcp_wmem` to at least 16777216.

You can do it manually with something like:

```
# echo 16777216 > /proc/sys/net/core/rmem_max
# echo 16777216 > /proc/sys/net/core/wmem_max
# echo "4096 87380 16777216" > /proc/sys/net/ipv4/tcp_rmem
# echo "4096 87380 16777216" > /proc/sys/net/ipv4/tcp_wmem
```

Or with the following Salt state:

```
net.core.rmem_max:
  sysctl:
    - present
    - value: 16777216

```

```
net.core.wmem_max:
  sysctl:
    - present
    - value: 16777216

net.ipv4.tcp_rmem:
  sysctl:
    - present
    - value: 4096 87380 16777216

net.ipv4.tcp_wmem:
  sysctl:
    - present
    - value: 4096 87380 16777216
```

Live Python Debug Output

If the master seems to be unresponsive, a SIGUSR1 can be passed to the salt-master threads to display what piece of code is executing. This debug information can be invaluable in tracking down bugs.

To pass a SIGUSR1 to the master, first make sure the minion is running in the foreground. Stop the service if it is running as a daemon, and start it in the foreground like so:

```
# salt-master -l debug
```

Then pass the signal to the master when it seems to be unresponsive:

```
# killall -SIGUSR1 salt-master
```

When filing an issue or sending questions to the mailing list for a problem with an unresponsive daemon, be sure to include this information if possible.

Live Salt-Master Profiling

When faced with performance problems one can turn on master process profiling by sending it SIGUSR2.

```
# killall -SIGUSR2 salt-master
```

This will activate `yappi` profiler inside salt-master code, then after some time one must send SIGUSR2 again to stop profiling and save results to file. If run in foreground salt-master will report filename for the results, which are usually located under `/tmp` on Unix-based OSes and `c:\temp` on windows.

Results can then be analyzed with kcachegrind or similar tool.

Commands Time Out or Do Not Return Output

Depending on your OS (this is most common on Ubuntu due to apt-get) you may sometimes encounter times where your highstate, or other long running commands do not return output.

Note: A number of timing issues were resolved in the 2014.1 release of Salt. Upgrading to at least this version is strongly recommended if timeouts persist.

By default the timeout is set to 5 seconds. The timeout value can easily be increased by modifying the `timeout` line within your `/etc/salt/master` configuration file.

Passing the -c Option to Salt Returns a Permissions Error

Using the `-c` option with the Salt command modifies the configuration directory. When the configuration file is read it will still base data off of the `root_dir` setting. This can result in unintended behavior if you are expecting files such as `/etc/salt/pki` to be pulled from the location specified with `-c`. Modify the `root_dir` setting to address this behavior.

Salt Master Doesn't Return Anything While Running jobs

When a command being run via Salt takes a very long time to return (package installations, certain scripts, etc.) the master may drop you back to the shell. In most situations the job is still running but Salt has exceeded the set timeout before returning. Querying the job queue will provide the data of the job but is inconvenient. This can be resolved by either manually using the `-t` option to set a longer timeout when running commands (by default it is 5 seconds) or by modifying the master configuration file: `/etc/salt/master` and setting the `timeout` value to change the default timeout for all commands, and then restarting the salt-master service.

Salt Master Auth Flooding

In large installations, care must be taken not to overwhealm the master with authentication requests. Several options can be set on the master which mitigate the chances of an authentication flood from causing an interuption in service.

Note: recon_default:

The average number of seconds to wait between reconnection attempts.

recon_max: The maximum number of seconds to wait between reconnection attempts.

recon_randomize: A flag to indicate whether the recon_default value should be randomized.

acceptance_wait_time: The number of seconds to wait for a reply to each authentication request.

random_reauth_delay: The range of seconds across which the minions should attempt to randomize authentication attempts.

auth_timeout: The total time to wait for the authentication process to complete, regardless of the number of attempts.

24.2 Troubleshooting the Salt Minion

In the event that your Salt minion is having issues, a variety of solutions and suggestions are available. Please refer to the following links for more information:

24.2.1 Troubleshooting the Salt Minion

Running in the Foreground

A great deal of information is available via the debug logging system, if you are having issues with minions connecting or not starting run the minion in the foreground:

```
# salt-minion -l debug
```

Anyone wanting to run Salt daemons via a process supervisor such as monit, runit, or supervisord, should omit the `-d` argument to the daemons and run them in the foreground.

What Ports does the Minion Need Open?

No ports need to be opened on the minion, as it makes outbound connections to the master. If you've put both your Salt master and minion in debug mode and don't see an acknowledgment that your minion has connected, it could very well be a firewall interfering with the connection. See our *firewall configuration* page for help opening the firewall on various platforms.

If you have netcat installed, you can check port connectivity from the minion with the `nc` command:

```
$ nc -v -z salt.master.ip.addr 4505
Connection to salt.master.ip.addr 4505 port [tcp/unknown] succeeded!
$ nc -v -z salt.master.ip.addr 4506
Connection to salt.master.ip.addr 4506 port [tcp/unknown] succeeded!
```

The Nmap utility can also be used to check if these ports are open:

```
# nmap -sS -q -p 4505-4506 salt.master.ip.addr

Starting Nmap 6.40 ( http://nmap.org ) at 2013-12-29 19:44 CST
Nmap scan report for salt.master.ip.addr (10.0.0.10)
Host is up (0.0026s latency).
PORT     STATE  SERVICE
4505/tcp open   unknown
4506/tcp open   unknown
MAC Address: 00:11:22:AA:BB:CC (Intel)

Nmap done: 1 IP address (1 host up) scanned in 1.64 seconds
```

If you've opened the correct TCP ports and still aren't seeing connections, check that no additional access control system such as SELinux or AppArmor is blocking Salt.

Using salt-call

The `salt-call` command was originally developed for aiding in the development of new Salt modules. Since then, many applications have been developed for running any Salt module locally on a minion. These range from the original intent of salt-call, development assistance, to gathering more verbose output from calls like `state.highstate`.

When initially creating your state tree, it is generally recommended to invoke `state.highstate` from the minion with `salt-call`. This displays far more information about the highstate execution than calling it remotely. For even more verbosity, increase the loglevel with the same argument as `salt-minion`:

```
# salt-call -l debug state.highstate
```

The main difference between using `salt` and using `salt-call` is that `salt-call` is run from the minion, and it only runs the selected function on that minion. By contrast, `salt` is run from the master, and requires you to specify the minions on which to run the command using salt's *targeting system*.

Live Python Debug Output

If the minion seems to be unresponsive, a SIGUSR1 can be passed to the process to display what piece of code is executing. This debug information can be invaluable in tracking down bugs.

To pass a SIGUSR1 to the minion, first make sure the minion is running in the foreground. Stop the service if it is running as a daemon, and start it in the foreground like so:

```
# salt-minion -l debug
```

Then pass the signal to the minion when it seems to be unresponsive:

```
# killall -SIGUSR1 salt-minion
```

When filing an issue or sending questions to the mailing list for a problem with an unresponsive daemon, be sure to include this information if possible.

Multiprocessing in Execution Modules

As is outlined in github issue #6300, Salt cannot use python's multiprocessing pipes and queues from execution modules. Multiprocessing from the execution modules is perfectly viable, it is just necessary to use Salt's event system to communicate back with the process.

The reason for this difficulty is that python attempts to pickle all objects in memory when communicating, and it cannot pickle function objects. Since the Salt loader system creates and manages function objects this causes the pickle operation to fail.

Salt Minion Doesn't Return Anything While Running Jobs Locally

When a command being run via Salt takes a very long time to return (package installations, certain scripts, etc.) the minion may drop you back to the shell. In most situations the job is still running but Salt has exceeded the set timeout before returning. Querying the job queue will provide the data of the job but is inconvenient. This can be resolved by either manually using the `-t` option to set a longer timeout when running commands (by default it is 5 seconds) or by modifying the minion configuration file: `/etc/salt/minion` and setting the `timeout` value to change the default timeout for all commands, and then restarting the salt-minion service.

Note: Modifying the minion timeout value is not required when running commands from a Salt Master. It is only required when running commands locally on the minion.

24.3 Running in the Foreground

A great deal of information is available via the debug logging system, if you are having issues with minions connecting or not starting run the minion and/or master in the foreground:

```
salt-master -l debug
salt-minion -l debug
```

Anyone wanting to run Salt daemons via a process supervisor such as monit, runit, or supervisord, should omit the `-d` argument to the daemons and run them in the foreground.

24.4 What Ports do the Master and Minion Need Open?

No ports need to be opened up on each minion. For the master, TCP ports 4505 and 4506 need to be open. If you've put both your Salt master and minion in debug mode and don't see an acknowledgment that your minion has connected, it could very well be a firewall.

You can check port connectivity from the minion with the nc command:

```
nc -v -z salt.master.ip 4505
nc -v -z salt.master.ip 4506
```

There is also a *firewall configuration* document that might help as well.

If you've enabled the right TCP ports on your operating system or Linux distribution's firewall and still aren't seeing connections, check that no additional access control system such as SELinux or AppArmor is blocking Salt.

24.5 Using salt-call

The `salt-call` command was originally developed for aiding in the development of new Salt modules. Since then, many applications have been developed for running any Salt module locally on a minion. These range from the original intent of salt-call, development assistance, to gathering more verbose output from calls like `state.highstate`.

When creating your state tree, it is generally recommended to invoke `state.highstate` with `salt-call`. This displays far more information about the highstate execution than calling it remotely. For even more verbosity, increase the loglevel with the same argument as `salt-minion`:

```
salt-call -l debug state.highstate
```

The main difference between using `salt` and using `salt-call` is that `salt-call` is run from the minion, and it only runs the selected function on that minion. By contrast, `salt` is run from the master, and requires you to specify the minions on which to run the command using salt's *targeting system*.

24.6 Too many open files

The salt-master needs at least 2 sockets per host that connects to it, one for the Publisher and one for response port. Thus, large installations may, upon scaling up the number of minions accessing a given master, encounter:

```
12:45:29,289 [salt.master    ][INFO    ] Starting Salt worker process 38
Too many open files
sock != -1 (tcp_listener.cpp:335)
```

The solution to this would be to check the number of files allowed to be opened by the user running salt-master (root by default):

```
[root@salt-master ~]# ulimit -n
1024
```

And modify that value to be at least equal to the number of minions x 2. This setting can be changed in limits.conf as the nofile value(s), and activated upon new a login of the specified user.

So, an environment with 1800 minions, would need 1800 x 2 = 3600 as a minimum.

24.7 Salt Master Stops Responding

There are known bugs with ZeroMQ versions less than 2.1.11 which can cause the Salt master to not respond properly. If you're running a ZeroMQ version greater than or equal to 2.1.9, you can work around the bug by setting the sysctls `net.core.rmem_max` and `net.core.wmem_max` to 16777216. Next, set the third field in `net.ipv4.tcp_rmem` and `net.ipv4.tcp_wmem` to at least 16777216.

You can do it manually with something like:

```
# echo 16777216 > /proc/sys/net/core/rmem_max
# echo 16777216 > /proc/sys/net/core/wmem_max
# echo "4096 87380 16777216" > /proc/sys/net/ipv4/tcp_rmem
# echo "4096 87380 16777216" > /proc/sys/net/ipv4/tcp_wmem
```

Or with the following Salt state:

```
net.core.rmem_max:
  sysctl:
    - present
    - value: 16777216

net.core.wmem_max:
  sysctl:
    - present
    - value: 16777216

net.ipv4.tcp_rmem:
  sysctl:
    - present
    - value: 4096 87380 16777216

net.ipv4.tcp_wmem:
  sysctl:
    - present
    - value: 4096 87380 16777216
```

24.8 Salt and SELinux

Currently there are no SELinux policies for Salt. For the most part Salt runs without issue when SELinux is running in Enforcing mode. This is because when the minion executes as a daemon the type context is changed to `initrc_t`. The problem with SELinux arises when using salt-call or running the minion in the foreground, since the type context stays `unconfined_t`.

This problem is generally manifest in the rpm install scripts when using the pkg module. Until a full SELinux Policy is available for Salt the solution to this issue is to set the execution context of `salt-call` and `salt-minion` to rpm_exec_t:

```
# CentOS 5 and RHEL 5:
chcon -t system_u:system_r:rpm_exec_t:s0 /usr/bin/salt-minion
chcon -t system_u:system_r:rpm_exec_t:s0 /usr/bin/salt-call

# CentOS 6 and RHEL 6:
chcon system_u:object_r:rpm_exec_t:s0 /usr/bin/salt-minion
chcon system_u:object_r:rpm_exec_t:s0 /usr/bin/salt-call
```

This works well, because the `rpm_exec_t` context has very broad control over other types.

24.9 Red Hat Enterprise Linux 5

Salt requires Python 2.6 or 2.7. Red Hat Enterprise Linux 5 and its variants come with Python 2.4 installed by default. When installing on RHEL 5 from the EPEL repository this is handled for you. But, if you run Salt from git, be advised that its dependencies need to be installed from EPEL and that Salt needs to be run with the `python26` executable.

24.10 Common YAML Gotchas

An extensive list of YAML idiosyncrasies has been compiled:

24.10.1 YAML Idiosyncrasies

One of Salt's strengths, the use of existing serialization systems for representing SLS data, can also backfire. YAML is a general purpose system and there are a number of things that would seem to make sense in an sls file that cause YAML issues. It is wise to be aware of these issues. While reports or running into them are generally rare they can still crop up at unexpected times.

Spaces vs Tabs

YAML uses spaces, period. Do not use tabs in your SLS files! If strange errors are coming up in rendering SLS files, make sure to check that no tabs have crept in! In Vim, after enabling search highlighting with: `:set hlsearch`, you can check with the following key sequence in normal mode(you can hit *ESC* twice to be sure): `/`, *Ctrl-v*, *Tab*, then hit *Enter*. Also, you can convert tabs to 2 spaces by these commands in Vim: `:set tabstop=2 expandtab` and then `:retab`.

Indentation

The suggested syntax for YAML files is to use 2 spaces for indentation, but YAML will follow whatever indentation system that the individual file uses. Indentation of two spaces works very well for SLS files given the fact that the data is uniform and not deeply nested.

Nested Dictionaries

When *dicts* are nested within other data structures (particularly lists), the indentation logic sometimes changes. Examples of where this might happen include `context` and `default` options from the *file.managed* state:

```
/etc/http/conf/http.conf:
  file:
    - managed
    - source: salt://apache/http.conf
    - user: root
    - group: root
    - mode: 644
    - template: jinja
    - context:
        custom_var: "override"
    - defaults:
        custom_var: "default value"
        other_var: 123
```

Notice that while the indentation is two spaces per level, for the values under the `context` and `defaults` options there is a four-space indent. If only two spaces are used to indent, then those keys will be considered part of the same dictionary that contains the `context` key, and so the data will not be loaded correctly. If using a double indent is not desirable, then a deeply-nested dict can be declared with curly braces:

```
/etc/http/conf/http.conf:
  file:
    - managed
    - source: salt://apache/http.conf
    - user: root
    - group: root
    - mode: 644
    - template: jinja
    - context: {
      custom_var: "override" }
    - defaults: {
      custom_var: "default value",
      other_var: 123 }
```

Here is a more concrete example of how YAML actually handles these indentations, using the Python interpreter on the command line:

```
>>> import yaml
>>> yaml.safe_load('''mystate:
...   file.managed:
...     - context:
...         some: var''')
{'mystate': {'file.managed': [{'context': {'some': 'var'}}]}}
>>> yaml.safe_load('''mystate:
...   file.managed:
...     - context:
...       some: var''')
{'mystate': {'file.managed': [{'some': 'var', 'context': None}]}}
```

Note that in the second example, `some` is added as another key in the same dictionary, whereas in the first example, it's the start of a new dictionary. That's the distinction. `context` is a common example because it is a keyword arg for many functions, and should contain a dictionary.

True/False, Yes/No, On/Off

PyYAML will load these values as boolean `True` or `False`. Un-capitalized versions will also be loaded as booleans (`true`, `false`, `yes`, `no`, `on`, and `off`). This can be especially problematic when constructing Pillar data. Make sure that your Pillars which need to use the string versions of these values are enclosed in quotes.

Integers are Parsed as Integers

NOTE: This has been fixed in salt 0.10.0, as of this release passing an integer that is preceded by a 0 will be correctly parsed

When passing `integers` into an SLS file, they are passed as integers. This means that if a state accepts a string value and an integer is passed, that an integer will be sent. The solution here is to send the integer as a string.

This is best explained when setting the mode for a file:

```
/etc/vimrc:
  file:
    - managed
    - source: salt://edit/vimrc
    - user: root
    - group: root
    - mode: 644
```

Salt manages this well, since the mode is passed as 644, but if the mode is zero padded as 0644, then it is read by YAML as an integer and evaluated as an octal value, 0644 becomes 420. Therefore, if the file mode is preceded by a 0 then it needs to be passed as a string:

```
/etc/vimrc:
  file:
    - managed
    - source: salt://edit/vimrc
    - user: root
    - group: root
    - mode: '0644'
```

YAML does not like ``Double Short Decs''

If I can find a way to make YAML accept ``Double Short Decs'' then I will, since I think that double short decs would be awesome. So what is a ``Double Short Dec''? It is when you declare a multiple short decs in one ID. Here is a standard short dec, it works great:

```
vim:
  pkg.installed
```

The short dec means that there are no arguments to pass, so it is not required to add any arguments, and it can save space.

YAML though, gets upset when declaring multiple short decs, for the record...

THIS DOES NOT WORK:

```
vim:
  pkg.installed
  user.present
```

Similarly declaring a short dec in the same ID dec as a standard dec does not work either...

ALSO DOES NOT WORK:

```
fred:
  user.present
  ssh_auth.present:
    - name: AAAAB3NzaC...
    - user: fred
    - enc: ssh-dss
    - require:
      - user: fred
```

The correct way is to define them like this:

```
vim:
  pkg.installed: []
  user.present: []

fred:
  user.present: []
  ssh_auth.present:
    - name: AAAAB3NzaC...
    - user: fred
    - enc: ssh-dss
    - require:
      - user: fred
```

Alternatively, they can be defined the ``old way'', or with multiple ``full decs'':

```
vim:
  pkg:
    - installed
  user:
    - present

fred:
  user:
    - present
  ssh_auth:
    - present
    - name: AAAAB3NzaC...
    - user: fred
    - enc: ssh-dss
    - require:
      - user: fred
```

YAML support only plain ASCII

According to YAML specification, only ASCII characters can be used.

Within double-quotes, special characters may be represented with C-style escape sequences starting with a backslash (\).

Examples:

```
- micro: "\u00b5"
- copyright: "\u00A9"
- A: "\x41"
- alpha: "\u0251"
- Alef: "\u05d0"
```

List of usable Unicode characters will help you to identify correct numbers.

Python can also be used to discover the Unicode number for a character:

```
repr(u"Text with wrong characters i need to figure out")
```

This shell command can find wrong characters in your SLS files:

```
find . -name '*.sls'  -exec  grep --color='auto' -P -n '[^\x00-\x7F]' \{} \;
```

Alternatively you can toggle the *yaml_utf8* setting in your master configuration file. This is still an experimental setting but it should manage the right encoding conversion in salt after yaml states compilations.

Underscores stripped in Integer Definitions

If a definition only includes numbers and underscores, it is parsed by YAML as an integer and all underscores are stripped. To ensure the object becomes a string, it should be surrounded by quotes. More information here.

Here's an example:

```
>>> import yaml
>>> yaml.safe_load('2013_05_10')
20130510
>>> yaml.safe_load('"2013_05_10"')
'2013_05_10'
```

Automatic `datetime` conversion

If there is a value in a YAML file formatted `2014-01-20 14:23:23` or similar, YAML will automatically convert this to a Python `datetime` object. These objects are not msgpack serializable, and so may break core salt functionality. If values such as these are needed in a salt YAML file (specifically a configuration file), they should be formatted with surrounding strings to force YAML to serialize them as strings:

```
>>> import yaml
>>> yaml.safe_load('2014-01-20 14:23:23')
datetime.datetime(2014, 1, 20, 14, 23, 23)
>>> yaml.safe_load('"2014-01-20 14:23:23"')
'2014-01-20 14:23:23'
```

Additionally, numbers formatted like `XXXX-XX-XX` will also be converted (or YAML will attempt to convert them, and error out if it doesn't think the date is a real one). Thus, for example, if a minion were to have an ID of `4017-16-20` the minion would not start because YAML would complain that the date was out of range. The workaround is the same, surround the offending string with quotes:

```
>>> import yaml
>>> yaml.safe_load('4017-16-20')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/usr/local/lib/python2.7/site-packages/yaml/__init__.py", line 93, in safe_load
    return load(stream, SafeLoader)
  File "/usr/local/lib/python2.7/site-packages/yaml/__init__.py", line 71, in load
    return loader.get_single_data()
  File "/usr/local/lib/python2.7/site-packages/yaml/constructor.py", line 39, in get_single_data
    return self.construct_document(node)
  File "/usr/local/lib/python2.7/site-packages/yaml/constructor.py", line 43, in construct_document
    data = self.construct_object(node)
  File "/usr/local/lib/python2.7/site-packages/yaml/constructor.py", line 88, in construct_object
    data = constructor(self, node)
  File "/usr/local/lib/python2.7/site-packages/yaml/constructor.py", line 312, in construct_yaml_time
    return datetime.date(year, month, day)
ValueError: month must be in 1..12
>>> yaml.safe_load('"4017-16-20"')
'4017-16-20'
```

24.11 Live Python Debug Output

If the minion or master seems to be unresponsive, a SIGUSR1 can be passed to the processes to display where in the code they are running. If encountering a situation like this, this debug information can be invaluable. First make sure the master of minion are running in the foreground:

```
salt-master -l debug
salt-minion -l debug
```

Then pass the signal to the master or minion when it seems to be unresponsive:

```
killall -SIGUSR1 salt-master
killall -SIGUSR1 salt-minion
```

Also under BSD and Mac OS X in addition to SIGUSR1 signal, debug subroutine set up for SIGINFO which has an advantage of being sent by Ctrl+T shortcut.

When filing an issue or sending questions to the mailing list for a problem with an unresponsive daemon this information can be invaluable.

24.12 Salt 0.16.x minions cannot communicate with a 0.17.x master

As of release 0.17.1 you can no longer run different versions of Salt on your Master and Minion servers. This is due to a protocol change for security purposes. The Salt team will continue to attempt to ensure versions are as backwards compatible as possible.

24.13 Debugging the Master and Minion

A list of common *master* and *minion* troubleshooting steps provide a starting point for resolving issues you may encounter.

CHAPTER 25

Developing Salt

25.1 Overview

In its most typical use, Salt is a software application in which clients, called ``minions'' can be commanded and controlled from a central command server called a ``master''.

Commands are normally issued to the minions (via the master) by calling a a client script simply called, `salt'.

Salt features a pluggable transport system to issue commands from a master to minions. The default transport is ZeroMQ.

25.2 Salt Client

25.2.1 Overview

The salt client is run on the same machine as the Salt Master and communicates with the salt-master to issue commands and to receive the results and display them to the user.

The primary abstraction for the salt client is called `LocalClient'.

When LocalClient wants to publish a command to minions, it connects to the master by issuing a request to the master's ReqServer (TCP: 4506)

The LocalClient system listens to responses for its requests by listening to the master event bus publisher (master_event_pub.ipc).

25.3 Salt Master

25.3.1 Overview

The salt-master deamon runs on the designated Salt master and performs functions such as authenticating minions, sending, and receiving requests from connected minions and sending and receiving requests and replies to the `salt' CLI.

25.3.2 Moving Pieces

When a Salt master starts up, a number of processes are started, all of which are called `salt-master' in a process-list but have various role categories.

Among those categories are:

- Publisher
- EventPublisher
- MWorker

25.3.3 Publisher

The Publisher process is responsible for sending commands over the designated transport to connected minions. The Publisher is bound to the following:

- TCP: port 4505
- IPC: publish_pull.ipc

Each salt minion establishes a connection to the master Publisher.

25.3.4 EventPublisher

The EventPublisher publishes events onto the event bus. It is bound to the following:

- IPC: master_event_pull.ipc
- IPC: master_event_pub.ipc

25.3.5 MWorker

Worker processes manage the back-end operations for the Salt Master.

The number of workers is equivilient to the number of `worker_threads' specified in the master configuration and is always at least one.

Workers are bound to the following:

- IPC: workers.ipc

25.3.6 ReqServer

The Salt request server takes requests and distributes them to available MWorker processes for processing. It also receives replies back from minions.

The ReqServer is bound to the following:

- TCP: 4506
- IPC: workers.ipc

Each salt minion establishes a connection to the master ReqServer.

25.3.7 Job Flow

The Salt master works by always publishing commands to all connected minions and the minions decide if the command is meant for them by checking themselves against the command target.

The typical lifecycle of a salt job from the perspective of the master might be as follows:

1. A command is issued on the CLI. For example, `salt my_minion test.ping'.

2) The `salt' command uses LocalClient to generate a request to the salt master by connecting to the ReqServer on TCP:4506 and issuing the job.

3) The salt-master ReqServer sees the request and passes it to an available MWorker over workers.ipc.

4) A worker picks up the request and handles it. First, it checks to ensure that the requested user has permissions to issue the command. Then, it sends the publish command to all connected minions. For the curious, this happens in ClearFuncs.publish().

5) The worker announces on the master event bus that it is about to publish a job to conneceted minions. This happens by placing the event on the master event bus (master_event_pull.ipc) where the EventPublisher picks it up and distributes it to all connected event listeners on master_event_pub.ipc.

6) The message to the minions is encrypted and sent to the Publisher via IPC on publish_pull.ipc.

7) Connected minions have a TCP session established with the Publisher on TCP port 4505 where they await commands. When the Publisher receives the job over publish_pull, it sends the jobs across the wire to the minions for processing.

8) After the minions receive the request, they decrypt it and perform any requested work, if they determine that they are targeted to do so.

9) When the minion is ready to respond, it publishes the result of its job back to the master by sending the encrypted result back to the master on TCP 4506 where it is again picked up by the ReqServer and forwarded to an available MWorker for processing. (Again, this happens by passing this message across workers.ipc to an available worker.)

10) When the MWorker receives the job it decrypts it and fires an event onto the master event bus (master_event_pull.ipc). (Again for the curious, this happens in AESFuncs._return().

11) The EventPublisher sees this event and re-publishes it on the bus to all connected listeners of the master event bus (on master_event_pub.ipc). This is where the LocalClient has been waiting, listening to the event bus for minion replies. It gathers the job and stores the result.

12) When all targeted minions have replied or the timeout has been exceeded, the salt client displays the results of the job to the user on the CLI.

25.4 Salt Minion

25.4.1 Overview

The salt-minion is a single process that sits on machines to be managed by Salt. It can either operate as a stand-alone daemon which accepts commands locally via `salt-call' or it can connect back to a master and receive commands remotely.

When starting up, salt minions connect _back_ to a master defined in the minion config file. The connect to two ports on the master:

- **TCP: 4505** This is the connection to the master Publisher. It is on this port that the minion receives jobs from the master.

- **TCP: 4506** This is the connection to the master ReqServer. It is on this port that the minion sends job results back to the master.

25.4.2 Event System

Similar to the master, a salt-minion has its own event system that operates over IPC by default. The minion event system operates on a push/pull system with IPC files at minion_event_<unique_id>_pub.ipc and minion_event_<unique_id>_pull.ipc.

The astute reader might ask why have an event bus at all with a single-process daemon. The answer is that the salt-minion may fork other processes as required to do the work without blocking the main salt-minion process and this necessitates a mechanism by which those processes can communicate with each other. Secondarily, this provides a bus by which any user with sufficient permissions can read or write to the bus as a common interface with the salt minion.

25.4.3 Job Flow

When a salt minion starts up, it attempts to connect to the Publisher and the ReqServer on the salt master. It then attempts to authenticate and once the minion has successfully authenticated, it simply listens for jobs.

Jobs normally come either come from the `salt-call' script run by a local user on the salt minion or they can come directly from a master.

25.4.4 Master Job Flow

1) A master publishes a job that is received by a minion as outlined by the master's job flow above.

2) The minion is polling its receive socket that's connected to the master Publisher (TCP 4505 on master). When it detects an incoming message, it picks it up from the socket and decrypts it.

3) A new minion process or thread is created and provided with the contents of the decrypted message. The _thread_return() method is provided with the contents of the received message.

4) The new minion thread is created. The _thread_return() function starts up and actually calls out to the requested function contained in the job.

5. The requested function runs and returns a result. [Still in thread.]

6) The result of the function that's run is encrypted and returned to the master's ReqServer (TCP 4506 on master). [Still in thread.]

7) Thread exits. Because the main thread was only blocked for the time that it took to initialize the worker thread, many other requests could have been received and processed during this time.

25.5 A Note on ClearFuncs vs. AESFuncs

A common source of confusion is deteremining when messages are passed in the clear and when they are passed using encryption. There are two rules governing this behaviour:

1) ClearFuncs is used for intra-master communication and during the initial authentication handshake between a minion and master during the key exhange.

2. AESFuncs is used everywhere else.

25.6 Contributing

There is a great need for contributions to Salt and patches are welcome! The goal here is to make contributions clear, make sure there is a trail for where the code has come from, and most importantly, to give credit where credit is due!

There are a number of ways to contribute to Salt development.

For details on how to contribute documentation improvements please review *Writing Salt Documentation*.

25.6.1 Sending a GitHub pull request

Sending pull requests on GitHub is the preferred method for receiving contributions. The workflow advice below mirrors GitHub's own guide and is well worth reading.

1. Fork the saltstack/salt repository on GitHub.
2. Make a local clone of your fork.
3. Create a new branch in your clone.

   A branch should have one purpose. For example, ``Fix bug X,'' or ``Add feature Y.'' Multiple pull requests should be opened for unrelated changes.

   Choose a name for your branch that describes its purpose.

```
git checkout -b fixed-broken-thing
```

4. Make edits and changes locally.
5. Commit changes to this new branch.

   Edit the necessary files in your Salt clone and remember to add them to your commit. Write a descriptive commit message.

```
git add path/to/file1
git add path/to/file2
git commit -m "Fixed X in file1 and file2"
```

   If you get stuck there are many introductory Git resources on help.github.com.

6. Push your locally-committed changes to your GitHub fork.

```
git push --set-upstream origin fixed-broken-thing
```

7. Go to your fork on the GitHub website & find your branch.

   GitHub automatically displays a button with the text ``Compare & pull request'' for recently pushed branches.

   Otherwise click on the ``Branches'' tab at the top of your fork. A button with the text ``New pull request'' will be beside each branch.

8. Open a new pull request.
   (a) Click one of the pull request buttons from the previous step. GitHub will present a form and show a comparison of the changes in your pull request.
   (b) Write a descriptive comment, include links to any project issues related to the pull request.
   (c) Click ``Create pull request''.
9. The Salt repo managers will be notified of your pull request.

   If a reviewer asks for changes:

(a) Make the changes in your local clone on the same local branch.

(b) Push the branch to GitHub using the same command as before.

(c) The new commits will be reflected in the pull request automatically.

(d) Feel free to add a comment to the discussion.

Note: Jenkins

Whenever you make a pull request against the main Salt repository your changes will be tested on a variety of operating systems and configurations. On average these tests take 30 minutes to run and once they are complete a PASS/FAIL message will be added to your pull request. This message contains a link to http://jenkins.saltstack.com where you can review the test results. This message will also generate an email which will be sent to the email address associated with your GitHub account informing you of these results. It should be noted that a test failure does not necessarily mean there is an issue in the associated pull request as the entire development branch is tested.

25.6.2 Which Salt branch?

GitHub will open pull requests against Salt's main branch named `develop` by default. Most contributors can keep the default options. This section is for advanced contributors.

Each pull request should address a single concern, as mentioned in the section above. For example, ``Fix bug X,'' or ``Add feature Y.'' And a pull request should be opened against the branch that corresponds to that concern.

The current release branch

The current release branch is the most recent stable release. Pull requests containing bug fixes should be made against the release branch.

The branch name will be a date-based name such as `2014.7`.

Bug fixes are made on this branch so that minor releases can be cut from this branch without introducing surprises and new features. This approach maximizes stability.

The Salt development team will ``merge-forward'' any fixes made on the release branch to the `develop` branch once the pull request has been accepted. This keeps the fix in isolation on the release branch and also keeps the `develop` branch up-to-date.

Note: Closing GitHub issues from commits

This ``merge-forward'' strategy requires that the magic keywords to close a GitHub issue appear in the commit message text directly. Only including the text in a pull request will not close the issue.

GitHub will close the referenced issue once the *commit* containing the magic text is merged into the default branch (`develop`). Any magic text input only into the pull request description will not be seen at the Git-level when those commits are merged-forward. In other words, only the commits are merged-forward and not the pull request.

The `develop` branch

The `develop` branch is unstable and bleeding-edge. Pull requests containing feature additions or non-bug-fix changes should be made against the `develop` branch.

The Salt development team will back-port bug fixes made to `develop` to the current release branch if the contributor cannot create the pull request against that branch.

25.6.3 Keeping Salt Forks in Sync

Salt is advancing quickly. It is therefore critical to pull upstream changes from upstream into your fork on a regular basis. Nothing is worse than putting hard work into a pull request only to see bunches of merge conflicts because it has diverged too far from upstream.

See also:

GitHub Fork a Repo Guide

The following assumes `origin` is the name of your fork and `upstream` is the name of the main saltstack/salt repository.

1. View existing remotes.

```
git remote -v
```

2. Add the `upstream` remote.

```
# For ssh github
git remote add upstream git@github.com:saltstack/salt.git

# For https github
git remote add upstream https://github.com/saltstack/salt.git
```

3. Pull upstream changes into your clone.

```
git fetch upstream
```

4. Update your copy of the `develop` branch.

```
git checkout develop
git merge --ff-only upstream/develop
```

   If Git complains that a fast-forward merge is not possible, you have local commits.

   - Run `git pull --rebase origin develop` to rebase your changes on top of the upstream changes.
   - Or, run `git branch <branch-name>` to create a new branch with your commits. You will then need to reset your `develop` branch before updating it with the changes from upstream.

   If Git complains that local files will be overwritten, you have changes to files in your working directory. Run `git status` to see the files in question.

5. Update your fork.

```
git push origin develop
```

6. Repeat the previous two steps for any other branches you work with, such as the current release branch.

25.6.4 Posting patches to the mailing list

Patches will also be accepted by email. Format patches using git format-patch and send them to the salt-users mailing list. The contributor will then get credit for the patch, and the Salt community will have an archive of the patch and a place for discussion.

25.6.5 Backporting Pull Requests

If a bug is fixed on `develop` and the bug is also present on a currently-supported release branch it will need to be back-ported to all applicable branches.

Note: Most Salt contributors can skip these instructions

These instructions do not need to be read in order to contribute to the Salt project! The SaltStack team will back-port fixes on behalf of contributors in order to keep the contribution process easy.

These instructions are intended for frequent Salt contributors, advanced Git users, SaltStack employees, or independent souls who wish to back-port changes themselves.

It is often easiest to fix a bug on the oldest supported release branch and then merge that branch forward into `develop` (as described earlier in this document). When that is not possible the fix must be back-ported, or copied, into any other affected branches.

These steps assume a pull request `#1234` has been merged into `develop`. And `upstream` is the name of the remote pointing to the main Salt repo.

1. Identify the oldest supported release branch that is affected by the bug.
2. Create a new branch for the back-port by reusing the same branch from the original pull request.

   Name the branch `bp-<NNNN>` and use the number of the original pull request.

```
git fetch upstream refs/pull/1234/head:bp-1234
git checkout bp-1234
```

3. Find the parent commit of the original pull request.

   The parent commit of the original pull request must be known in order to rebase onto a release branch. The easiest way to find this is on GitHub.

   Open the original pull request on GitHub and find the first commit in the list of commits. Select and copy the SHA for that commit. The parent of that commit can be specified by appending `~1` to the end.

4. Rebase the new branch on top of the release branch.

   - `<release-branch>` is the branch identified in step #1.
   - `<orig-base>` is the SHA identified in step #3 -- don't forget to add `~1` to the end!

```
git rebase --onto <release-branch> <orig-base> bp-1234
```

   Note, release branches prior to `2014.7` will not be able to make use of rebase and must use cherry-picking instead.

5. Push the back-port branch to GitHub and open a new pull request.

   Opening a pull request for the back-port allows for the test suite and normal code-review process.

```
git push -u origin bp-1234
```

25.7 Deprecating Code

Salt should remain backwards compatible, though sometimes, this backwards compatibility needs to be broken because a specific feature and/or solution is no longer necessary or required. At first one might think, let me change this code, it seems that it's not used anywhere else so it should be safe to remove. Then, once there's a new release,

users complain about functionality which was removed and they where using it, etc. This should, at all costs, be avoided, and, in these cases, *that* specific code should be deprecated.

Depending on the complexity and usage of a specific piece of code, the deprecation time frame should be properly evaluated. As an example, a deprecation warning which is shown for 2 major releases, for example *0.17.0* and *2014.1.0*, gives users enough time to stop using the deprecated code and adapt to the new one.

For example, if you're deprecating the usage of a keyword argument to a function, that specific keyword argument should remain in place for the full deprecation time frame and if that keyword argument is used, a deprecation warning should be shown to the user.

To help in this deprecation task, salt provides `salt.utils.warn_until`. The idea behind this helper function is to show the deprecation warning until salt reaches the provided version. Once that provided version is equaled `salt.utils.warn_until` will raise a `RuntimeError` making salt stop its execution. This stoppage is unpleasant and will remind the developer that the deprecation limit has been reached and that the code can then be safely removed.

Consider the following example:

```
def some_function(bar=False, foo=None):
    if foo is not None:
        salt.utils.warn_until(
            (0, 18),
            'The \'foo\' argument has been deprecated and its '
            'functionality removed, as such, its usage is no longer '
            'required.'
        )
```

Consider that the current salt release is `0.16.0`. Whenever `foo` is passed a value different from `None` that warning will be shown to the user. This will happen in versions `0.16.2` to `2014.1.0`, after which a `RuntimeError` will be raised making us aware that the deprecated code should now be removed.

25.8 Dunder Dictionaries

Salt provides several special ``dunder'' dictionaries as a convenience for Salt development. These include `__opts__`, `__context__`, `__salt__`, and others. This document will describe each dictionary and detail where they exist and what information and/or functionality they provide.

25.8.1 __opts__

Available in

- All loader modules

The `__opts__` dictionary contains all of the options passed in the configuration file for the master or minion.

Note: In many places in salt, instead of pulling raw data from the __opts__ dict, configuration data should be pulled from the salt *get* functions such as config.get, aka - __salt__['config.get'](`foo:bar') The *get* functions also allow for dict traversal via the *:* delimiter. Consider using get functions whenever using __opts__ or __pillar__ and __grains__ (when using grains for configuration data)

The configuration file data made available in the `__opts__` dictionary is the configuration data relative to the running daemon. If the modules are loaded and executed by the master, then the master configuration data is

available, if the modules are executed by the minion, then the minion configuration is available. Any additional information passed into the respective configuration files is made available

25.8.2 __salt__

Available in

- Execution Modules
- State Modules
- Returners

`__salt__` contains the execution module functions. This allows for all functions to be called as they have been set up by the salt loader.

```
__salt__['cmd.run']('fdisk -l')
__salt__['network.ip_addrs']()
```

25.8.3 __grains__

Available in

- Execution Modules
- State Modules
- Returners
- External Pillar

The `__grains__` dictionary contains the grains data generated by the minion that is currently being worked with. In execution modules, state modules and returners this is the grains of the minion running the calls, when generating the external pillar the `__grains__` is the grains data from the minion that the pillar is being generated for.

25.8.4 __pillar__

Available in

- Execution Modules
- State Modules
- Returners

The `__pillar__` dictionary contains the pillar for the respective minion.

25.8.5 __context__

`__context__` exists in state modules and execution modules.

During a state run the `__context__` dictionary persists across all states that are run and then is destroyed when the state ends.

When running an execution module `__context__` persists across all module executions until the modules are refreshed; such as when `saltutils.sync_all` or `state.highstate` are executed.

A great place to see how to use `__context__` is in the cp.py module in salt/modules/cp.py. The fileclient authenticates with the master when it is instantiated and then is used to copy files to the minion. Rather than create a new fileclient for each file that is to be copied down, one instance of the fileclient is instantiated in the `__context__` dictionary and is reused for each file. Here is an example from salt/modules/cp.py:

```
if not 'cp.fileclient' in __context__:
    __context__['cp.fileclient'] = salt.fileclient.get_file_client(__opts__)
```

Note: Because __context__ may or may not have been destroyed, always be sure to check for the existence of the key in __context__ and generate the key before using it.

25.9 External Pillars

Salt provides a mechanism for generating pillar data by calling external pillar interfaces. This document will describe an outline of an ext_pillar module.

25.9.1 Location

Salt expects to find your `ext_pillar` module in the same location where it looks for other python modules. If the `extension_modules` option in your Salt master configuration is set, Salt will look for a `pillar` directory under there and load all the modules it finds. Otherwise, it will look in your Python site-packages `salt/pillar` directory.

25.9.2 Configuration

The external pillars that are called when a minion refreshes its pillars is controlled by the `ext_pillar` option in the Salt master configuration. You can pass a single argument, a list of arguments or a dictionary of arguments to your pillar:

```
ext_pillar:
  - example_a: some argument
  - example_b:
    - argumentA
    - argumentB
  - example_c:
      keyA: valueA
      keyB: valueB
```

25.9.3 The Module

25.9.4 Imports and Logging

Import modules your external pillar module needs. You should first include generic modules that come with stock Python:

```
import logging
```

And then start logging. This is an idiomatic way of setting up logging in Salt:

```
log = logging.getLogger(__name__)
```

Finally, load modules that are specific to what you are doing. You should catch import errors and set a flag that the `__virtual__` function can use later.

```
try:
    import weird_thing
    EXAMPLE_A_LOADED = True
except ImportError:
    EXAMPLE_A_LOADED = False
```

25.9.5 Options

If you define an `__opts__` dictionary, it will be merged into the `__opts__` dictionary handed to the `ext_pillar` function later. This is a good place to put default configuration items. The convention is to name things `module_name.option`.

```
__opts__ = { 'example_a.someconfig': 137 }
```

25.9.6 Initialization

If you define an `__init__` function, it will be called with the following signature:

```
def __init__( __opts__ ):
    # Do init work here
```

Note: The `__init__` function is ran every time a particular minion causes the external pillar to be called, so don't put heavy initialization code here. The `__init__` functionality is a side-effect of the Salt loader, so it may not be as useful in pillars as it is in other Salt items.

25.9.7 __virtual__

If you define a `__virtual__` function, you can control whether or not this module is visible. If it returns `False` then Salt ignores this module. If it returns a string, then that string will be how Salt identifies this external pillar in its `ext_pillar` configuration. If you're not renaming the module, simply return `True` in the `__virtual__` function, which is the same as if this function did not exist, then, the name Salt's `ext_pillar` will use to identify this module is its conventional name in Python.

This is useful to write modules that can be installed on all Salt masters, but will only be visible if a particular piece of software your module requires is installed.

```
# This external pillar will be known as `example_a`
def __virtual__():
    if EXAMPLE_A_LOADED:
```

```
        return True
    return False
```

```
# This external pillar will be known as `something_else`
__virtualname__ = 'something_else'

def __virtual__():
    if EXAMPLE_A_LOADED:
        return __virtualname__
    return False
```

25.9.8 ext_pillar

This is where the real work of an external pillar is done. If this module is active and has a function called `ext_pillar`, whenever a minion updates its pillar this function is called.

How it is called depends on how it is configured in the Salt master configuration. The first argument is always the current pillar dictionary, this contains pillar items that have already been added, starting with the data from `pillar_roots`, and then from any already-ran external pillars.

Using our example above:

```
ext_pillar( id, pillar, 'some argument' )                    # example_a
ext_pillar( id, pillar, 'argumentA', 'argumentB' )           # example_b
ext_pillar( id, pillar, keyA='valueA', keyB='valueB' } )     # example_c
```

In the `example_a` case, `pillar` will contain the items from the `pillar_roots`, in `example_b` `pillar` will contain that plus the items added by `example_a`, and in `example_c` `pillar` will contain that plus the items added by `example_b`. In all three cases, `id` will contain the ID of the minion making the pillar request.

This function should return a dictionary, the contents of which are merged in with all of the other pillars and returned to the minion. **Note**: this function is called once for each minion that fetches its pillar data.

```
def ext_pillar( minion_id, pillar, *args, **kwargs ):

    my_pillar = {}

    # Do stuff

    return my_pillar
```

You shouldn't just add items to `pillar` and return that, since that will cause Salt to merge data that already exists. Rather, just return the items you are adding or changing. You could, however, use `pillar` in your module to make some decision based on pillar data that already exists.

This function has access to some useful globals:

__opts__ A dictionary of mostly Salt configuration options. If you had an `__opts__` dictionary defined in your module, those values will be included.

__salt__ A dictionary of Salt module functions, useful so you don't have to duplicate functions that already exist. E.g. `__salt__['cmd.run']( 'ls -l' )` **Note**, runs on the *master*

__grains__ A dictionary of the grains of the minion making this pillar call.

25.9.9 Example configuration

As an example, if you wanted to add external pillar via the `cmd_json` external pillar, add something like this to your master config:

```
ext_pillar:
  - cmd_json: 'echo {\"arg\":\"value\"}'
```

25.9.10 Reminder

Just as with traditional pillars, external pillars must be refreshed in order for minions to see any fresh data:

```
salt '*' saltutil.refresh_pillar
```

25.10 Installing Salt for development

Clone the repository using:

```
git clone https://github.com/saltstack/salt
```

Note: tags

Just cloning the repository is enough to work with Salt and make contributions. However, fetching additional tags from git is required to have Salt report the correct version for itself. To do this, first add the git repository as an upstream source:

```
git remote add upstream https://github.com/saltstack/salt
```

Fetching tags is done with the git `fetch' utility:

```
git fetch --tags upstream
```

Create a new virtualenv:

```
virtualenv /path/to/your/virtualenv
```

On Arch Linux, where Python 3 is the default installation of Python, use the `virtualenv2` command instead of `virtualenv`.

Note: Using system Python modules in the virtualenv

To use already-installed python modules in virtualenv (instead of having pip download and compile new ones), run `virtualenv --system-site-packages` Using this method eliminates the requirement to install the salt dependencies again, although it does assume that the listed modules are all installed in the system PYTHONPATH at the time of virtualenv creation.

Activate the virtualenv:

```
source /path/to/your/virtualenv/bin/activate
```

Install Salt (and dependencies) into the virtualenv:

```
pip install M2Crypto    # Don't install on Debian/Ubuntu (see below)
pip install pyzmq PyYAML pycrypto msgpack-python jinja2 psutil
pip install -e ./salt   # the path to the salt git clone from above
```

Note: Installing M2Crypto

`swig` and `libssl-dev` are required to build M2Crypto. To fix the error `command 'swig' failed with exit status 1` while installing M2Crypto, try installing it with the following command:

```
env SWIG_FEATURES="-cpperraswarn -includeall -D__`uname -m`__ -I/usr/include/openssl" pip install M2C
```

Debian and Ubuntu systems have modified openssl libraries and mandate that a patched version of M2Crypto be installed. This means that M2Crypto needs to be installed via apt:

```
apt-get install python-m2crypto
```

This also means that pulling in the M2Crypto installed using apt requires using `--system-site-packages` when creating the virtualenv.

If you're using a platform other than Debian or Ubuntu, and you are installing M2Crypto via pip instead of a system package, then you will also need the `gcc` compiler.

Note: Installing psutil

Python header files are required to build this module, otherwise the pip install will fail. If your distribution separates binaries and headers into separate packages, make sure that you have the headers installed. In most Linux distributions which split the headers into their own package, this can be done by installing the `python-dev` or `python-devel` package. For other platforms, the package will likely be similarly named.

Note: Installing dependencies on OS X.

You can install needed dependencies on OS X using homebrew or macports. See *OS X Installation*

Warning: Installing on RedHat-based Distros
If installing from pip (or from source using `setup.py install`), be advised that the `yum-utils` package is needed for Salt to manage packages on RedHat-based systems.

25.10.1 Running a self-contained development version

During development it is easiest to be able to run the Salt master and minion that are installed in the virtualenv you created above, and also to have all the configuration, log, and cache files contained in the virtualenv as well.

Copy the master and minion config files into your virtualenv:

```
mkdir -p /path/to/your/virtualenv/etc/salt
cp ./salt/conf/master /path/to/your/virtualenv/etc/salt/master
cp ./salt/conf/minion /path/to/your/virtualenv/etc/salt/minion
```

Edit the master config file:

1. Uncomment and change the `user: root` value to your own user.
2. Uncomment and change the `root_dir: /` value to point to `/path/to/your/virtualenv`.
3. If you are running version 0.11.1 or older, uncomment, and change the `pidfile: /var/run/salt-master.pid` value to point to `/path/to/your/virtualenv/salt-master.pid`.
4. If you are also running a non-development version of Salt you will have to change the `publish_port` and `ret_port` values as well.

Edit the minion config file:

1. Repeat the edits you made in the master config for the `user` and `root_dir` values as well as any port changes.
2. If you are running version 0.11.1 or older, uncomment, and change the `pidfile: /var/run/salt-minion.pid` value to point to `/path/to/your/virtualenv/salt-minion.pid`.
3. Uncomment and change the `master: salt` value to point at `localhost`.
4. Uncomment and change the `id:` value to something descriptive like ``saltdev''. This isn't strictly necessary but it will serve as a reminder of which Salt installation you are working with.
5. If you changed the `ret_port` value in the master config because you are also running a non-development version of Salt, then you will have to change the `master_port` value in the minion config to match.

Note: Using *salt-call* with a *Standalone Minion*

If you plan to run *salt-call* with this self-contained development environment in a masterless setup, you should invoke *salt-call* with `-c /path/to/your/virtualenv/etc/salt` so that salt can find the minion config file. Without the `-c` option, Salt finds its config files in */etc/salt*.

Start the master and minion, accept the minion's key, and verify your local Salt installation is working:

```
cd /path/to/your/virtualenv
salt-master -c ./etc/salt -d
salt-minion -c ./etc/salt -d
salt-key -c ./etc/salt -L
salt-key -c ./etc/salt -A
salt -c ./etc/salt '*' test.ping
```

Running the master and minion in debug mode can be helpful when developing. To do this, add `-l debug` to the calls to `salt-master` and `salt-minion`. If you would like to log to the console instead of to the log file, remove the `-d`.

Once the minion starts, you may see an error like the following:

```
zmq.core.error.ZMQError: ipc path "/path/to/your/virtualenv/
var/run/salt/minion/minion_event_7824dcbcfd7a8f6755939af70b96249f_pub.ipc"
is longer than 107 characters (sizeof(sockaddr_un.sun_path)).
```

This means that the path to the socket the minion is using is too long. This is a system limitation, so the only workaround is to reduce the length of this path. This can be done in a couple different ways:

1. Create your virtualenv in a path that is short enough.

2. Edit the `sock_dir` minion config variable and reduce its length. Remember that this path is relative to the value you set in `root_dir`.

`NOTE:` The socket path is limited to 107 characters on Solaris and Linux, and 103 characters on BSD-based systems.

Note: File descriptor limits

Ensure that the system open file limit is raised to at least 2047:

```
# check your current limit
ulimit -n

# raise the limit. persists only until reboot
# use 'limit descriptors 2047' for c-shell
ulimit -n 2047
```

To set file descriptors on OSX, refer to the *OS X Installation* instructions.

25.10.2 Installing Salt from the Python Package Index

If you are installing using `easy_install`, you will need to define a **USE_SETUPTOOLS** environment variable, otherwise dependencies will not be installed:

```
USE_SETUPTOOLS=1 easy_install salt
```

25.10.3 Editing and previewing the documentation

You need `sphinx-build` command to build the docs. In Debian/Ubuntu this is provided in the `python-sphinx` package. Sphinx can also be installed to a virtualenv using pip:

```
pip install Sphinx
```

Change to salt documentation directory, then:

```
cd doc; make html
```

- This will build the HTML docs. Run `make` without any arguments to see the available make targets, which include **html**, **man**, and **text**.
- The docs then are built within the **docs/_build/** folder. To update the docs after making changes, run `make` again.
- The docs use reStructuredText for markup. See a live demo at http://rst.ninjs.org/.
- The help information on each module or state is culled from the python code that runs for that piece. Find them in `salt/modules/` or `salt/states/`.
- To build the docs on Arch Linux, the **python2-sphinx** package is required. Additionally, it is necessary to tell **make** where to find the proper **sphinx-build** binary, like so:

```
make SPHINXBUILD=sphinx-build2 html
```

- To build the docs on RHEL/CentOS 6, the **python-sphinx10** package must be installed from EPEL, and the following make command must be used:

```
make SPHINXBUILD=sphinx-1.0-build html
```

Once you've updated the documentation, you can run the following command to launch a simple Python HTTP server to see your changes:

```
cd _build/html; python -m SimpleHTTPServer
```

25.10.4 Running unit and integration tests

Run the test suite with following command:

```
./setup.py test
```

See *here* for more information regarding the test suite.

25.11 Logging Internals

TODO

25.12 Modular Systems

When first working with Salt, it is not always clear where all of the modular components are and what they do. Salt comes loaded with more modular systems than many users are aware of, making Salt very easy to extend in many places.

The most commonly used modular systems are execution modules and states. But the modular systems extend well beyond the more easily exposed components and are often added to Salt to make the complete system more flexible.

25.12.1 Execution Modules

Execution modules make up the core of the functionality used by Salt to interact with client systems. The execution modules create the core system management library used by all Salt systems, including states, which interact with minion systems.

Execution modules are completely open ended in their execution. They can be used to do anything required on a minion, from installing packages to detecting information about the system. The only restraint in execution modules is that the defined functions always return a JSON serializable object.

For a list of all built in execution modules, click *here*

For information on writing execution modules, see *this page*.

25.12.2 Interactive Debugging

Sometimes debugging with `print()` and extra logs sprinkled everywhere is not the best strategy.

IPython is a helpful debug tool that has an interactive python environment which can be embedded in python programs.

First the system will require IPython to be installed.

```
# Debian
apt-get install ipython

# Arch Linux
pacman -Syu ipython2

# RHEL/CentOS (via EPEL)
yum install python-ipython
```

Now, in the troubling python module, add the following line at a location where the debugger should be started:

```
test = 'test123'
import IPython; IPython.embed_kernel()
```

After running a Salt command that hits that line, the following will show up in the log file:

```
[CRITICAL] To connect another client to this kernel, use:
[IPKernelApp] --existing kernel-31271.json
```

Now on the system that invoked `embed_kernel`, run the following command from a shell:

```
# NOTE: use ipython2 instead of ipython for Arch Linux
ipython console --existing
```

This provides a console that has access to all the vars and functions, and even supports tab-completion.

```
print(test)
test123
```

To exit IPython and continue running Salt, press `Ctrl-d` to logout.

25.12.3 State Modules

State modules are used to define the state interfaces used by Salt States. These modules are restrictive in that they must follow a number of rules to function properly.

Note: State modules define the available routines in sls files. If calling an execution module directly is desired, take a look at the *module* state.

25.12.4 Auth

The auth module system allows for external authentication routines to be easily added into Salt. The *auth* function needs to be implemented to satisfy the requirements of an auth module. Use the `pam` module as an example.

25.12.5 Fileserver

The fileserver module system is used to create fileserver backends used by the Salt Master. These modules need to implement the functions used in the fileserver subsystem. Use the `gitfs` module as an example.

25.12.6 Grains

Grain modules define extra routines to populate grains data. All defined public functions will be executed and MUST return a Python dict object. The dict keys will be added to the grains made available to the minion.

25.12.7 Output

The output modules supply the outputter system with routines to display data in the terminal. These modules are very simple and only require the *output* function to execute. The default system outputter is the `nested` module.

25.12.8 Pillar

Used to define optional external pillar systems. The pillar generated via the filesystem pillar is passed into external pillars. This is commonly used as a bridge to database data for pillar, but is also the backend to the libvirt state used to generate and sign libvirt certificates on the fly.

25.12.9 Renderers

Renderers are the system used to render sls files into salt highdata for the state compiler. They can be as simple as the `py` renderer and as complex as `stateconf` and `pydsl`.

25.12.10 Returners

Returners are used to send data from minions to external sources, commonly databases. A full returner will implement all routines to be supported as an external job cache. Use the `redis` returner as an example.

25.12.11 Runners

Runners are purely master-side execution sequences. These range from simple reporting to orchestration engines like the overstate.

25.12.12 Tops

Tops modules are used to convert external data sources into top file data for the state system.

25.12.13 Wheel

The wheel system is used to manage master side management routines. These routines are primarily intended for the API to enable master configuration.

25.13 Package Providers

This page contains guidelines for writing package providers.

25.13.1 Package Functions

One of the most important features of Salt is package management. There is no shortage of package managers, so in the interest of providing a consistent experience in `pkg` states, there are certain functions that should be present in a package provider. Note that these are subject to change as new features are added or existing features are enhanced.

list_pkgs

This function should declare an empty dict, and then add packages to it by calling `pkg_resource.add_pkg`, like so:

```
__salt__['pkg_resource.add_pkg'](ret, name, version)
```

The last thing that should be done before returning is to execute `pkg_resource.sort_pkglist`. This function does not presently do anything to the return dict, but will be used in future versions of Salt.

```
__salt__['pkg_resource.sort_pkglist'](ret)
```

`list_pkgs` returns a dictionary of installed packages, with the keys being the package names and the values being the version installed. Example return data:

```
{'foo': '1.2.3-4',
 'bar': '5.6.7-8'}
```

latest_version

Accepts an arbitrary number of arguments. Each argument is a package name. The return value for a package will be an empty string if the package is not found or if the package is up-to-date. The only case in which a non-empty string is returned is if the package is available for new installation (i.e. not already installed) or if there is an upgrade available.

If only one argument was passed, this function return a string, otherwise a dict of name/version pairs is returned.

This function must also accept `**kwargs`, in order to receive the `fromrepo` and `repo` keyword arguments from pkg states. Where supported, these arguments should be used to find the install/upgrade candidate in the specified repository. The `fromrepo` kwarg takes precedence over `repo`, so if both of those kwargs are present, the repository specified in `fromrepo` should be used. However, if `repo` is used instead of `fromrepo`, it should still work, to preserve backwards compatibility with older versions of Salt.

version

Like `latest_version`, accepts an arbitrary number of arguments and returns a string if a single package name was passed, or a dict of name/value pairs if more than one was passed. The only difference is that the return values are the currently-installed versions of whatever packages are passed. If the package is not installed, an empty string is returned for that package.

upgrade_available

Deprecated and destined to be removed. For now, should just do the following:

```
return __salt__['pkg.latest_version'](name) != ''
```

install

The following arguments are required and should default to `None`:

1. name (for single-package pkg states)
2. pkgs (for multiple-package pkg states)
3. sources (for binary package file installation)

The first thing that this function should do is call `pkg_resource.parse_targets` (see below). This function will convert the SLS input into a more easily parsed data structure. `pkg_resource.parse_targets` may need to be modified to support your new package provider, as it does things like parsing package metadata which cannot be done for every package management system.

```
pkg_params, pkg_type = __salt__['pkg_resource.parse_targets'](name,
                                                              pkgs,
                                                              sources)
```

Two values will be returned to the **install** function. The first of them will be a dictionary. The keys of this dictionary will be package names, though the values will differ depending on what kind of installation is being done:

- If **name** was provided (and **pkgs** was not), then there will be a single key in the dictionary, and its value will be `None`. Once the data has been returned, if the **version** keyword argument was provided, then it should replace the `None` value in the dictionary.
- If **pkgs** was provided, then **name** is ignored, and the dictionary will contain one entry for each package in the **pkgs** list. The values in the dictionary will be `None` if a version was not specified for the package, and the desired version if specified. See the **Multiple Package Installation Options** section of the `pkg.installed` state for more info.
- If **sources** was provided, then **name** is ignored, and the dictionary values will be the path/URI for the package.

The second return value will be a string with two possible values: `repository` or `file`. The **install** function can use this value (if necessary) to build the proper command to install the targeted package(s).

Both before and after the installing the target(s), you should run **list_pkgs** to obtain a list of the installed packages. You should then return the output of `salt.utils.compare_dicts()`

```
return salt.utils.compare_dicts(old, new)
```

remove

Removes the passed package and return a list of the packages removed.

25.13.2 Package Repo Functions

There are some functions provided by `pkg` which are specific to package repositories, and not to packages themselves. When writing modules for new package managers, these functions should be made available as stated below, in order to provide compatibility with the `pkgrepo` state.

All repo functions should accept a basedir option, which defines which directory repository configuration should be found in. The default for this is dictated by the repo manager that is being used, and rarely needs to be changed.

```
basedir = '/etc/yum.repos.d'
__salt__['pkg.list_repos'](basedir)
```

list_repos

Lists the repositories that are currently configured on this system.

```
__salt__['pkg.list_repos']()
```

Returns a dictionary, in the following format:

```
{'reponame': 'config_key_1': 'config value 1',
             'config_key_2': 'config value 2',
             'config_key_3': ['list item 1 (when appropriate)',
                              'list item 2 (when appropriate)]}
```

get_repo

Displays all local configuration for a specific repository.

```
__salt__['pkg.get_repo'](repo='myrepo')
```

The information is formatted in much the same way as list_repos, but is specific to only one repo.

```
{'config_key_1': 'config value 1',
 'config_key_2': 'config value 2',
 'config_key_3': ['list item 1 (when appropriate)',
                  'list item 2 (when appropriate)]}
```

del_repo

Removes the local configuration for a specific repository. Requires a *repo* argument, which must match the locally configured name. This function returns a string, which informs the user as to whether or not the operation was a success.

```
__salt__['pkg.del_repo'](repo='myrepo')
```

mod_repo

Modify the local configuration for one or more option for a configured repo. This is also the way to create new repository configuration on the local system; if a repo is specified which does not yet exist, it will be created.

The options specified for this function are specific to the system; please refer to the documentation for your specific repo manager for specifics.

```
__salt__['pkg.mod_repo'](repo='myrepo', url='http://myurl.com/repo')
```

25.13.3 Low-Package Functions

In general, the standard package functions as describes above will meet your needs. These functions use the system's native repo manager (for instance, yum or the apt tools). In most cases, the repo manager is actually separate from the package manager. For instance, yum is usually a front-end for rpm, and apt is usually a front-end for dpkg. When possible, the package functions that use those package managers directly should do so through the low package functions.

It is normal and sane for `pkg` to make calls to `lowpkgs`, but `lowpkg` must never make calls to `pkg`. This is affects functions which are required by both `pkg` and `lowpkg`, but the technique in `pkg` is more performant than what is available to `lowpkg`. When this is the case, the `lowpkg` function that requires that technique must still use the `lowpkg` version.

list_pkgs

Returns a dict of packages installed, including the package name and version. Can accept a list of packages; if none are specified, then all installed packages will be listed.

```
installed = __salt__['lowpkg.list_pkgs']('foo', 'bar')
```

Example output:

```
{'foo': '1.2.3-4',
 'bar': '5.6.7-8'}
```

verify

Many (but not all) package management systems provide a way to verify that the files installed by the package manager have or have not changed. This function accepts a list of packages; if none are specified, all packages will be included.

```
installed = __salt__['lowpkg.verify']('httpd')
```

Example output:

```
{'/etc/httpd/conf/httpd.conf': {'mismatch': ['size', 'md5sum', 'mtime'],
                                'type': 'config'}}
```

file_list

Lists all of the files installed by all packages specified. If not packages are specified, then all files for all known packages are returned.

```
installed = __salt__['lowpkg.file_list']('httpd', 'apache')
```

This function does not return which files belong to which packages; all files are returned as one giant list (hence the *file_list* function name. However, This information is still returned inside of a dict, so that it can provide any errors to the user in a sane manner.

```
{'errors': ['package apache is not installed'],
  'files': ['/etc/httpd',
            '/etc/httpd/conf',
            '/etc/httpd/conf.d',
            '...SNIP...']}
```

file_dict

Lists all of the files installed by all packages specified. If not packages are specified, then all files for all known packages are returned.

```
installed = __salt__['lowpkg.file_dict']('httpd', 'apache', 'kernel')
```

Unlike *file_list*, this function will break down which files belong to which packages. It will also return errors in the same manner as *file_list*.

```
{'errors': ['package apache is not installed'],
 'packages': {'httpd': ['/etc/httpd',
                        '/etc/httpd/conf',
                        '...SNIP...'],
              'kernel': ['/boot/.vmlinuz-2.6.32-279.el6.x86_64.hmac',
                         '/boot/System.map-2.6.32-279.el6.x86_64',
                         '...SNIP...']}}
```

25.14 Community Projects That Use Salt

Below is a list of repositories that show real world Salt applications that you can use to get started. Please note that these projects do not adhere to any standards and express a wide variety of ideas and opinions on how an action can be completed with Salt.

https://github.com/terminalmage/djangocon2013-sls

https://github.com/jesusaurus/hpcs-salt-state

https://github.com/gravyboat/hungryadmin-sls

https://github.com/wunki/django-salted

25.15 Salt Topology

Salt is based on a powerful, asynchronous, network topology using ZeroMQ. Many ZeroMQ systems are in place to enable communication. The central idea is to have the fastest communication possible.

25.15.1 Servers

The Salt Master runs 2 network services. First is the ZeroMQ PUB system. This service by default runs on port `4505` and can be configured via the `publish_port` option in the master configuration.

Second is the ZeroMQ REP system. This is a separate interface used for all bi-directional communication with minions. By default this system binds to port `4506` and can be configured via the `ret_port` option in the master.

25.15.2 PUB/SUB

The commands sent out via the salt client are broadcast out to the minions via ZeroMQ PUB/SUB. This is done by allowing the minions to maintain a connection back to the Salt Master and then all connections are informed to download the command data at once. The command data is kept extremely small (usually less than 1K) so it is not a burden on the network.

25.15.3 Return

The PUB/SUB system is a one way communication, so once a publish is sent out the PUB interface on the master has no further communication with the minion. The minion, after running the command, then sends the command's return data back to the master via the `ret_port`.

25.16 Translating Documentation

If you wish to help translate the Salt documentation to your language, please head over to the Transifex website and signup for an account.

Once registered, head over to the Salt Translation Project, and either click on **Request Language** if you can't find yours, or, select the language for which you wish to contribute and click **Join Team**.

Transifex provides some useful reading resources on their support domain, namely, some useful articles directed to translators.

25.16.1 Building A Localized Version of the Documentation

While you're working on your translation on Transifex, you might want to have a look at how it's rendering.

Install The Transifex Client

To interact with the Transifex web service you will need to install the transifex-client:

```
pip install transifex-client
```

Configure The Transifex Client

Once installed, you will need to set it up on your computer. We created a script to help you with that:

```
.scripts/setup-transifex-config
```

Download Remote Translations

There's a little script which simplifies the download process of the translations(which isn't that complicated in the first place). So, let's assume you're translating `pt_PT`, Portuguese(Portugal). To download the translations, execute from the `doc/` directory of your Salt checkout:

```
make download-translations SPHINXLANG=pt_PT
```

To download `pt_PT`, Portuguese(Portugal), and `nl`, Dutch, you can use the helper script directly:

```
.scripts/download-translation-catalog pt_PT nl
```

Build Localized Documentation

After the download process finishes, which might take a while, the next step is to build a localized version of the documentation. Following the `pt_PT` example above:

```
make html SPHINXLANG=pt_PT
```

View Localized Documentation

Open your browser, point it to the local documentation path and check the localized output you've just build.

25.17 Running The Tests

There are requirements, in addition to Salt's requirements, which need to be installed in order to run the test suite. Install one of the lines below, depending on the relevant Python version:

```
pip install -r dev_requirements_python26.txt
pip install -r dev_requirements_python27.txt
```

Note: In Salt 0.17, testing libraries were migrated into their own repo. To install them:

```
pip install git+https://github.com/saltstack/salt-testing.git#egg=SaltTesting
```

Failure to install SaltTesting will result in import errors similar to the following:

```
ImportError: No module named salttesting
```

Once all require requirements are set, use `tests/runtests.py` to run all of the tests included in Salt's test suite. For more information, see `--help`.

An alternative way of invoking the test suite is available in `setup.py`:

```
./setup.py test
```

Instead of running the entire test suite, there are several ways to run only specific groups of tests or individual tests:

- Run unit tests only: `./tests/runtests.py --unit-tests`
- Run unit and integration tests for states: `./tests/runtests.py --state`
- Run integration tests for an individual module: `./tests/runtests.py -n integration.modules.virt`
- Run unit tests for an individual module: `./tests/runtests.py -n unit.modules.virt_test`
- Run an individual test by using the class and test name (this example is for the `test_default_kvm_profile` test in the `integration.module.virt`): `./tests/runtests.py -n ingtegration.module.virt.VirtTest.test_default_kvm_profile`

25.17.1 Running Unit Tests Without Integration Test Daemons

Since the unit tests do not require a master or minion to execute, it is often useful to be able to run unit tests individually, or as a whole group, without having to start up the integration testing daemons. Starting up the master, minion, and syndic daemons takes a lot of time before the tests can even start running and is unnecessary to run unit tests. To run unit tests without invoking the integration test daemons, simple remove the `/tests` portion of the `runtests.py` command:

```
./runtests.py --unit
```

All of the other options to run individual tests, entire classes of tests, or entire test modules still apply.

25.17.2 Running Destructive Integration Tests

Salt is used to change the settings and behavior of systems. In order to effectively test Salt's functionality, some integration tests are written to make actual changes to the underlying system. These tests are referred to as ``destructive tests''. Some examples of destructive tests are changes may be testing the addition of a user or installing packages. By default, destructive tests are disabled and will be skipped.

Generally, destructive tests should clean up after themselves by attempting to restore the system to its original state. For instance, if a new user is created during a test, the user should be deleted after the related test(s) have completed. However, no guarantees are made that test clean-up will complete successfully. Therefore, running destructive tests should be done with caution.

Note: Running destructive tests will change the underlying system. Use caution when running destructive tests.

To run tests marked as destructive, set the `--run-destructive` flag:

```
./tests/runtests.py --run-destructive
```

25.17.3 Running Cloud Provider Tests

Salt's testing suite also includes integration tests to assess the successful creation and deletion of cloud instances using *Salt-Cloud* for providers supported by Salt-Cloud.

The cloud provider tests are off by default and run on sample configuration files provided in `tests/integration/files/conf/cloud.providers.d/`. In order to run the cloud provider tests, valid credentials, which differ per provider, must be supplied. Each credential item that must be supplied is indicated by an empty string value and should be edited by the user before running the tests. For example, DigitalOcean requires a client key and an api key to operate. Therefore, the default cloud provider configuration file for DigitalOcean looks like this:

```
digitalocean-config:
  provider: digital_ocean
  client_key: ''
  api_key: ''
  location: New York 1
```

As indicated by the empty string values, the `client_key` and the `api_key` must be provided:

```
digitalocean-config:
  provider: digital_ocean
  client_key: wFGEwgregeqw3435gDger
  api_key: GDE43t43REGTrkilg43934t34qT43t4dgegerGEgg
  location: New York 1
```

Note: When providing credential information in cloud provider configuration files, do not include the single quotes.

Once all of the valid credentials for the cloud provider have been supplied, the cloud provider tests can be run by setting the `--cloud-provider-tests` flag:

```
./tests/runtests.py --cloud-provider-tests
```

25.17.4 Running The Tests In A Docker Container

The test suite can be executed under a docker container using the `--docked` option flag. The docker container must be properly configured on the system invoking the tests and the container must have access to the internet.

Here's a simple usage example:

```
tests/runtests.py --docked=ubuntu-12.04 -v
```

The full docker container repository can also be provided:

```
tests/runtests.py --docked=salttest/ubuntu-12.04 -v
```

The SaltStack team is creating some containers which will have the necessary dependencies pre-installed. Running the test suite on a container allows destructive tests to run without making changes to the main system. It also enables the test suite to run under a different distribution than the one the main system is currently using.

The current list of test suite images is on Salt's docker repository.

Custom docker containers can be provided by submitting a pull request against Salt's docker Salt test containers repository.

25.18 Automated Test Runs

SaltStack maintains a Jenkins server to allow for the execution of tests across supported platforms. The tests executed from Salt's Jenkins server create fresh virtual machines for each test run, then execute destructive tests on the new, clean virtual machine.

When a pull request is submitted to Salt's repository on GitHub, Jenkins runs Salt's test suite on a couple of virtual machines to gauge the pull request's viability to merge into Salt's develop branch. If these initial tests pass, the pull request can then merged into Salt's develop branch by one of Salt's core developers, pending their discretion. If the initial tests fail, core developers may request changes to the pull request. If the failure is unrelated to the changes in question, core developers may merge the pull request despite the initial failure.

Once the pull request is merged into Salt's develop branch, a new set of Jenkins virtual machines will begin executing the test suite. The develop branch tests have many more virtual machines to provide more comprehensive results.

There are a few other groups of virtual machines that Jenkins tests against, including past and current release branches. For a full list of currently running test environments, go to http://jenkins.saltstack.com.

25.18.1 Using Salt-Cloud on Jenkins

For testing Salt on Jenkins, SaltStack uses *Salt-Cloud* to spin up virtual machines. The script using Salt-Cloud to accomplish this is open source and can be found here: https://github.com/saltstack/salt/blob/develop/tests/jenkins.py

25.19 Writing Tests

Salt uses a test platform to verify functionality of components in a simple way. Two testing systems exist to enable testing salt functions in somewhat real environments. The two subsystems available are integration tests and unit tests.

Salt uses the python standard library unittest2 system for testing.

25.19.1 Naming Conventions

Any function in either integration test files or unit test files that is doing the actual testing, such as functions containing assertions, must start with `test_`:

```
def test_user_present(self):
```

When functions in test files are not prepended with `test_`, the function acts as a normal, helper function and is not run as a test by the test suite.

25.19.2 Integration Tests

The integration tests start up a number of salt daemons to test functionality in a live environment. These daemons include 2 salt masters, 1 syndic, and 2 minions. This allows the syndic interface to be tested and master/minion communication to be verified. All of the integration tests are executed as live salt commands sent through the started daemons.

Integration tests are particularly good at testing modules, states, and shell commands.

- *Writing integration tests*

25.19.3 Unit Tests

Direct unit tests are also available. These tests are good for testing internal functions.

- *Writing unit tests*

Integration Tests

The Salt integration tests come with a number of classes and methods which allow for components to be easily tested. These classes are generally inherited from and provide specific methods for hooking into the running integration test environment created by the integration tests.

It is noteworthy that since integration tests validate against a running environment that they are generally the preferred means to write tests.

The integration system is all located under `tests/integration` in the Salt source tree. Each directory within `tests/integration` corresponds to a directory in Salt's tree structure. For example, the integration tests for the `test.py` Salt module that is located in `salt/modules` should also be named `test.py` and reside in `tests/integration/modules`.

Adding New Directories

If the corresponding Salt directory does not exist within `tests/integration`, the new directory must be created along with the appropriate test file to maintain Salt's testing directory structure.

In order for Salt's test suite to recognize tests within the newly created directory, options to run the new integration tests must be added to `tests/runtests.py`. Examples of the necessary options that must be added can be found here: https://github.com/saltstack/salt/blob/develop/tests/runtests.py. The functions that need to be edited are `setup_additional_options`, `validate_options`, and `run_integration_tests`.

Integration Classes

The integration classes are located in `tests/integration/__init__.py` and can be extended therein. There are three classes available to extend:

ModuleCase Used to define executions run via the master to minions and to call single modules and states.

The available methods are as follows:

run_function: Run a single salt function and condition the return down to match the behavior of the raw function call. This will run the command and only return the results from a single minion to verify.

state_result: Return the result data from a single state return

run_state: Run the state.single command and return the state return structure

SyndicCase Used to execute remote commands via a syndic, only used to verify the capabilities of the Syndic.

The available methods are as follows:

run_function: Run a single salt function and condition the return down to match the behavior of the raw function call. This will run the command and only return the results from a single minion to verify.

ShellCase Shell out to the scripts which ship with Salt.

The available methods are as follows:

run_script: Execute a salt script with the given argument string

run_salt: Execute the salt command, pass in the argument string as it would be passed on the command line.

run_run: Execute the salt-run command, pass in the argument string as it would be passed on the command line.

run_run_plus: Execute Salt run and the salt run function and return the data from each in a dict

run_key: Execute the salt-key command, pass in the argument string as it would be passed on the command line.

run_cp: Execute salt-cp, pass in the argument string as it would be passed on the command line.

run_call: Execute salt-call, pass in the argument string as it would be passed on the command line.

Examples

Module Example via ModuleCase Class Import the integration module, this module is already added to the python path by the test execution. Inherit from the `integration.ModuleCase` class.

Now the workhorse method `run_function` can be used to test a module:

```
import os
import integration


class TestModuleTest(integration.ModuleCase):
    '''
    Validate the test module
    '''
    def test_ping(self):
        '''
        test.ping
        '''
        self.assertTrue(self.run_function('test.ping'))

    def test_echo(self):
        '''
        test.echo
        '''
        self.assertEqual(self.run_function('test.echo', ['text']), 'text')
```

Shell Example via ShellCase Validating the shell commands can be done via shell tests:

```
import sys
import shutil
import tempfile

import integration

class KeyTest(integration.ShellCase):
    '''
    Test salt-key script
    '''
```

```
    _call_binary_ = 'salt-key'

    def test_list(self):
        '''
        test salt-key -L
        '''
        data = self.run_key('-L')
        expect = [
                'Unaccepted Keys:',
                'Accepted Keys:',
                'minion',
                'sub_minion',
                'Rejected:', '']
        self.assertEqual(data, expect)
```

This example verifies that the `salt-key` command executes and returns as expected by making use of the `run_key` method.

Integration Test Files

Since using Salt largely involves configuring states, editing files, and changing system data, the integration test suite contains a directory named `files` to aid in testing functions that require files. Various Salt integration tests use these example files to test against instead of altering system files and data.

Each directory within `tests/integration/files` contain files that accomplish different tasks, based on the needs of the integration tests using those files. For example, `tests/integration/files/ssh` is used to bootstrap the test runner for salt-ssh testing, while `tests/integration/files/pillar` contains files storing data needed to test various pillar functions.

The `tests/integration/files` directory also includes an integration state tree. The integration state tree can be found at `tests/integration/files/file/base`.

The following example demonstrates how integration files can be used with ModuleCase to test states:

```
import os
import shutil
import integration

HFILE = os.path.join(integration.TMP, 'hosts')

class HostTest(integration.ModuleCase):
    '''
    Validate the host state
    '''

    def setUp(self):
        shutil.copyfile(os.path.join(integration.FILES, 'hosts'), HFILE)
        super(HostTest, self).setUp()

    def tearDown(self):
        if os.path.exists(HFILE):
            os.remove(HFILE)
        super(HostTest, self).tearDown()

    def test_present(self):
        '''
```

```
        host.present
        '''
        name = 'spam.bacon'
        ip = '10.10.10.10'
        ret = self.run_state('host.present', name=name, ip=ip)
        result = self.state_result(ret)
        self.assertTrue(result)
        with open(HFILE) as fp_:
            output = fp_.read()
            self.assertIn('{0}\t\t{1}'.format(ip, name), output)
```

To access the integration files, a variable named `integration.FILES` points to the `tests/integration/files` directory. This is where the referenced `host.present` sls file resides.

In addition to the static files in the integration state tree, the location `integration.TMP` can also be used to store temporary files that the test system will clean up when the execution finishes.

Destructive vs Non-Destructive Tests

Since Salt is used to change the settings and behavior of systems, one testing approach is to run tests that make actual changes to the underlying system. This is where the concept of destructive integration tests comes into play. Tests can be written to alter the system they are running on. This capability is what fills in the gap needed to properly test aspects of system management like package installation.

Any test that changes the underlying system in any way, such as creating or deleting users, installing packages, or changing permissions should include the `@destructive` decorator to signal system changes and should be written with care. System changes executed within a destructive test should also be restored once the related tests have completed. For example, if a new user is created to test a module, the same user should be removed after the test is completed to maintain system integrity.

To write a destructive test, import, and use the destructiveTest decorator for the test method:

```
import integration
from salttesting.helpers import destructiveTest

class DestructiveExampleModuleTest(integration.ModuleCase):
    '''
    Demonstrate a destructive test
    '''

    @destructiveTest
    @skipIf(os.geteuid() != 0, 'you must be root to run this test')
    def test_user_not_present(self):
        '''
        This is a DESTRUCTIVE TEST it creates a new user on the minion.
        And then destroys that user.
        '''
        ret = self.run_state('user.present', name='salt_test')
        self.assertSaltTrueReturn(ret)
        ret = self.run_state('user.absent', name='salt_test')
        self.assertSaltTrueReturn(ret)
```

Cloud Provider Tests

Cloud provider integration tests are used to assess *Salt-Cloud*`s ability to create and destroy cloud instances for various supported cloud providers. Cloud provider tests inherit from the ShellCase Integration Class.

Any new cloud provider test files should be added to the `tests/integration/cloud/providers/` directory. Each cloud provider test file also requires a sample cloud profile and cloud provider configuration file in the integration test file directory located at `tests/integration/files/conf/cloud.*.d/`.

The following is an example of the default profile configuration file for Digital Ocean, located at: `tests/integration/files/conf/cloud.profiles.d/digital_ocean.conf`:

```
digitalocean-test:
  provider: digitalocean-config
  image: Ubuntu 14.04 x64
  size: 512MB
```

Each cloud provider requires different configuration credentials. Therefore, sensitive information such as API keys or passwords should be omitted from the cloud provider configuration file and replaced with an empty string. The necessary credentials can be provided by the user by editing the provider configuration file before running the tests.

The following is an example of the default provider configuration file for Digital Ocean, located at: `tests/integration/files/conf/cloud.providers.d/digital_ocean.conf`:

```
digitalocean-config:
  provider: digital_ocean
  client_key: ''
  api_key: ''
  location: New York 1
```

In addition to providing the necessary cloud profile and provider files in the integration test suite file structure, appropriate checks for if the configuration files exist and contain valid information are also required in the test class's `setUp` function:

```
class LinodeTest(integration.ShellCase):
'''
Integration tests for the Linode cloud provider in Salt-Cloud
'''

def setUp(self):
    '''
    Sets up the test requirements
    '''
    super(LinodeTest, self).setUp()

    # check if appropriate cloud provider and profile files are present
    profile_str = 'linode-config:'
    provider = 'linode'
    providers = self.run_cloud('--list-providers')
    if profile_str not in providers:
        self.skipTest(
            'Configuration file for {0} was not found. Check {0}.conf files '
            'in tests/integration/files/conf/cloud.*.d/ to run these tests.'
            .format(provider)
        )

    # check if apikey and password are present
```

```
    path = os.path.join(integration.FILES,
                        'conf',
                        'cloud.providers.d',
                        provider + '.conf')
    config = cloud_providers_config(path)
    api = config['linode-config']['linode']['apikey']
    password = config['linode-config']['linode']['password']
    if api == '' or password == '':
        self.skipTest(
            'An api key and password must be provided to run these tests. Check '
            'tests/integration/files/conf/cloud.providers.d/{0}.conf'.format(
                provider
            )
        )
```

Repeatedly creating and destroying instances on cloud providers can be costly. Therefore, cloud provider tests are off by default and do not run automatically. To run the cloud provider tests, the `--cloud-provider-tests` flag must be provided:

```
./tests/runtests.py --cloud-provider-tests
```

Since cloud provider tests do not run automatically, all provider tests must be preceded with the `@expensiveTest` decorator. The expensive test decorator is necessary because it signals to the test suite that the `--cloud-provider-tests` flag is required to run the cloud provider tests.

To write a cloud provider test, import, and use the expensiveTest decorator for the test function:

```
from salttesting.helpers import expensiveTest

@expensiveTest
def test_instance(self):
    '''
    Test creating an instance on Linode
    '''
    name = 'linode-testing'

    # create the instance
    instance = self.run_cloud('-p linode-test {0}'.format(name))
    str = '        {0}'.format(name)

    # check if instance with salt installed returned as expected
    try:
        self.assertIn(str, instance)
    except AssertionError:
        self.run_cloud('-d {0} --assume-yes'.format(name))
        raise

    # delete the instance
    delete = self.run_cloud('-d {0} --assume-yes'.format(name))
    str = '            True'
    try:
        self.assertIn(str, delete)
    except AssertionError:
        raise
```

Writing Unit Tests

Introduction

Like many software projects, Salt has two broad-based testing approaches -- integration testing and unit testing. While integration testing focuses on the interaction between components in a sandboxed environment, unit testing focuses on the singular implementation of individual functions.

Preparing to Write a Unit Test

This guide assumes you've followed the directions for setting up salt testing.

Unit tests should be written to the following specifications:

- All the tests for a specific module at salt/.../<module>.py need to be contained in a file called tests/unit/.../<module>_test.py, e.g. the tests for salt/modules/fib.py need to be contained in a file called tests/unit/modules/fib_test.py.
- The tests within tests/unit/modules/fib_test.py file must be member functions of a class which subclasses salttesting.Testcase
- Each external resource used in the code to be tested, such as function calls, external data either globally available or passed in through the function arguments, file data, etc. needs to be mocked.
- Each raise and return statement of the code to be tested needs to be separately and independently tested.
- Test functions should contain only one test and contain all necessary mock data and mock code for that test.
- Test functions should be named test_<fcn>_<test-name> where <fcn> is the name of the function being tested and <test-name> describes which raise or return within the function is being tested and whether that raise or return statement is considered a success or a failure condition.

Most commonly, the following imports are necessary to create a unit test:

```
# Import Salt Testing libs
from salttesting import skipIf, TestCase
from salttesting.helpers import ensure_in_syspath
```

If you need mock support to your tests, please also import:

```
from salttesting.mock import NO_MOCK, NO_MOCK_REASON, MagicMock, patch, call
```

A Simple Example

Let's assume that we're testing a very basic function in an imaginary Salt execution module. Given a module called `fib.py` that has a function called `calculate(num_of_results)`, which given a `num_of_results`, produces a list of sequential Fibonacci numbers of that length.

A unit test to test this function might be commonly placed in a file called `tests/unit/modules/fib_test.py`. The convention is to place unit tests for Salt execution modules in `test/unit/modules/` and to name the tests module suffixed with `_test.py`.

Tests are grouped around test cases, which are logically grouped sets of tests against a piece of functionality in the tested software. Test cases are created as Python classes in the unit test module. To return to our example, here's how we might write the skeleton for testing `fib.py`:

```
# Import Salt Testing libs
from salttesting import TestCase

# Import Salt execution module to test
from salt.modules import fib

# Create test case class and inherit from Salt's customized TestCase
class FibTestCase(TestCase):

    '''
    This class contains a set of functions that test salt.modules.fib.
    '''

    def test_fib(self):
        '''
        To create a unit test, we should prefix the name with `test_' so
        that it's recognized by the test runner.
        '''
        fib_five = (0, 1, 1, 2, 3)
        self.assertEqual(fib.calculate(5), fib_five)
```

At this point, the test can now be run, either individually or as a part of a full run of the test runner. To ease development, a single test can be executed:

```
tests/runtests.py -n unit.modules.fib_test
```

This will produce output indicating the success or failure of the tests in given test case. For more detailed results, one can also include a flag to increase verbosity:

```
tests/runtests.py -n unit.modules.fib_test -v
```

To review the results of a particular run, take a note of the log location given in the output for each test:

```
Logging tests on /var/folders/nl/d809xbq577l3qrbj3ymtpbq80000gn/T/salt-runtests.log
```

Evaluating Truth

A longer discussion on the types of assertions one can make can be found by reading Python's documentation on unit testing.

Tests Using Mock Objects

In many cases, the very purpose of a Salt module is to interact with some external system, whether it be to control a database, manipulate files on a filesystem or many other examples. In these varied cases, it's necessary to design a unit test which can test the function whilst replacing functions which might actually call out to external systems. One might think of this as ``blocking the exits'' for code under tests and redirecting the calls to external systems with our own code which produces known results during the duration of the test.

To achieve this behavior, Salt makes heavy use of the MagicMock package.

To understand how one might integrate Mock into writing a unit test for Salt, let's imagine a scenario in which we're testing an execution module that's designed to operate on a database. Furthermore, let's imagine two separate methods, here presented in pseduo-code in an imaginary execution module called `db.py.

```
def create_user(username):
    qry = 'CREATE USER {0}'.format(username)
    execute_query(qry)

def execute_query(qry):
    # Connect to a database and actually do the query...
```

Here, let's imagine that we want to create a unit test for the *create_user* function. In doing so, we want to avoid any calls out to an external system and so while we are running our unit tests, we want to replace the actual interaction with a database with a function that can capture the parameters sent to it and return pre-defined values. Therefore, our task is clear -- to write a unit test which tests the functionality of *create_user* while also replacing `execute_query' with a mocked function.

To begin, we set up the skeleton of our class much like we did before, but with additional imports for MagicMock:

```
# Import Salt Testing libs
from salttesting import TestCase

# Import Salt execution module to test
from salt.modules import db

# NEW! -- Import Mock libraries
from salttesting.mock import NO_MOCK, NO_MOCK_REASON, MagicMock, patch, call

# Create test case class and inherit from Salt's customized TestCase

# Skip this test case if we don't have access to mock!
@skipIf(NO_MOCK, NO_MOCK_REASON)
class DbTestCase(TestCase):
    def test_create_user(self):
        # First, we replace 'execute_query' with our own mock function
        db.execute_query = MagicMock()

        # Now that the exits are blocked, we can run the function under test.

        db.create_user('testuser')

        # We could now query our mock object to see which calls were made
        # to it.
        ## print db.execute_query.mock_calls

        '''
        We want to test to ensure that the correct query was formed.  This
        is a contrived example, just designed to illustrate the concepts at
        hand.

        We're going to first construct a call() object that represents the
        way we expect our mocked execute_query() function to have been
        called.  Then, we'll examine the list of calls that were actually
        made to to execute_function().

        By comparing our expected call to execute_query() with
        create_user()'s call to execute_query(), we can determine the
```

```
        success or failure of our unit test.
        '''

        expected_call = call('CREATE USER testuser')

        # Do the comparison! Will assert False if execute_query() was not
        # called with the given call

        db.execute_query.assert_has_calls(expected_call)
```

Modifying `__salt__` In Place

At times, it becomes necessary to make modifications to a module's view of functions in its own `__salt__` dictionary. Luckily, this process is quite easy.

Below is an example that uses MagicMock's `patch` functionality to insert a function into `__salt__` that's actually a MagicMock instance.

```
def show_patch(self):
    with patch.dict(my_module.__salt__,
                    {'function.to_replace': MagicMock()}:
        # From this scope, carry on with testing, with a modified __salt__!
```

A More Complete Example

Consider the following function from salt/modules/linux_sysctl.py.

```
def get(name):
    '''
    Return a single sysctl parameter for this minion

    CLI Example:

    .. code-block:: bash

        salt '*' sysctl.get net.ipv4.ip_forward
    '''
    cmd = 'sysctl -n {0}'.format(name)
    out = __salt__['cmd.run'](cmd)
    return out
```

This function is very simple, comprising only four source lines of code and having only one return statement, so we know only one test is needed. There are also two inputs to the function, the `name` function argument and the call to `__salt__['cmd.run']()`, both of which need to be appropriately mocked. Mocking a function parameter is straightforward, whereas mocking a function call will require, in this case, the use of MagicMock. For added isolation, we will also redefine the `__salt__` dictionary such that it only contains `'cmd.run'`.

```
# Import Salt Libs
from salt.modules import linux_sysctl

# Import Salt Testing Libs
from salttesting import skipIf, TestCase
```

```
from salttesting.helpers import ensure_in_syspath
from salttesting.mock import (
    MagicMock,
    patch,
    NO_MOCK,
    NO_MOCK_REASON
)

ensure_in_syspath('../../')

# Globals
linux_sysctl.__salt__ = {}


@skipIf(NO_MOCK, NO_MOCK_REASON)
class LinuxSysctlTestCase(TestCase):
    '''
    TestCase for salt.modules.linux_sysctl module
    '''

    def test_get(self):
        '''
        Tests the return of get function
        '''
        mock_cmd = MagicMock(return_value=1)
        with patch.dict(linux_sysctl.__salt__, {'cmd.run': mock_cmd}):
            self.assertEqual(linux_sysctl.get('net.ipv4.ip_forward'), 1)


if __name__ == '__main__':
    from integration import run_tests
    run_tests(LinuxSysctlTestCase, needs_daemon=False)
```

Since `get()` has only one raise or return statement and that statement is a success condition, the test function is simply named `test_get()`. As described, the single function call parameter, `name` is mocked with `net.ipv4.ip_forward` and `__salt__['cmd.run']` is replaced by a MagicMock function object. We are only interested in the return value of `__salt__['cmd.run']`, which MagicMock allows to be specified via `return_value=1`. Finally, the test itself tests for equality between the return value of `get()` and the expected return of `1`. This assertion is expected to succeed because `get()` will determine its return value from `__salt__['cmd.run']`, which we have mocked to return `1`.

A Complex Example

Now consider the `assign()` function from the same salt/modules/linux_sysctl.py source file.

```
def assign(name, value):
    '''
    Assign a single sysctl parameter for this minion

    CLI Example:

    .. code-block:: bash

        salt '*' sysctl.assign net.ipv4.ip_forward 1
    '''
```

```
    value = str(value)
    sysctl_file = '/proc/sys/{0}'.format(name.replace('.', '/'))
    if not os.path.exists(sysctl_file):
        raise CommandExecutionError('sysctl {0} does not exist'.format(name))

    ret = {}
    cmd = 'sysctl -w {0}="{1}"'.format(name, value)
    data = __salt__['cmd.run_all'](cmd)
    out = data['stdout']
    err = data['stderr']

    # Example:
    #    # sysctl -w net.ipv4.tcp_rmem="4096 87380 16777216"
    #    net.ipv4.tcp_rmem = 4096 87380 16777216
    regex = re.compile(r'^{0}\s+=\s+{1}$'.format(re.escape(name),
                                                 re.escape(value)))

    if not regex.match(out) or 'Invalid argument' in str(err):
        if data['retcode'] != 0 and err:
            error = err
        else:
            error = out
        raise CommandExecutionError('sysctl -w failed: {0}'.format(error))
    new_name, new_value = out.split(' = ', 1)
    ret[new_name] = new_value
    return ret
```

This function contains two raise statements and one return statement, so we know that we will need (at least) three tests. It has two function arguments and many references to non-builtin functions. In the tests below you will see that MagicMock's `patch()` method may be used as a context manager or as a decorator.

There are three test functions, one for each raise and return statement in the source function. Each function is self-contained and contains all and only the mocks and data needed to test the raise or return statement it is concerned with.

```
# Import Salt Libs
from salt.modules import linux_sysctl
from salt.exceptions import CommandExecutionError

# Import Salt Testing Libs
from salttesting import skipIf, TestCase
from salttesting.helpers import ensure_in_syspath
from salttesting.mock import (
    MagicMock,
    patch,
    NO_MOCK,
    NO_MOCK_REASON
)

ensure_in_syspath('../../')

# Globals
linux_sysctl.__salt__ = {}


@skipIf(NO_MOCK, NO_MOCK_REASON)
class LinuxSysctlTestCase(TestCase):
    '''
```

```
    TestCase for salt.modules.linux_sysctl module
    '''

    @patch('os.path.exists', MagicMock(return_value=False))
    def test_assign_proc_sys_failed(self):
        '''
        Tests if /proc/sys/<kernel-subsystem> exists or not
        '''
        cmd = {'pid': 1337, 'retcode': 0, 'stderr': '',
               'stdout': 'net.ipv4.ip_forward = 1'}
        mock_cmd = MagicMock(return_value=cmd)
        with patch.dict(linux_sysctl.__salt__, {'cmd.run_all': mock_cmd}):
            self.assertRaises(CommandExecutionError,
                              linux_sysctl.assign,
                              'net.ipv4.ip_forward', 1)

    @patch('os.path.exists', MagicMock(return_value=True))
    def test_assign_cmd_failed(self):
        '''
        Tests if the assignment was successful or not
        '''
        cmd = {'pid': 1337, 'retcode': 0, 'stderr':
               'sysctl: setting key "net.ipv4.ip_forward": Invalid argument',
               'stdout': 'net.ipv4.ip_forward = backward'}
        mock_cmd = MagicMock(return_value=cmd)
        with patch.dict(linux_sysctl.__salt__, {'cmd.run_all': mock_cmd}):
            self.assertRaises(CommandExecutionError,
                              linux_sysctl.assign,
                              'net.ipv4.ip_forward', 'backward')

    @patch('os.path.exists', MagicMock(return_value=True))
    def test_assign_success(self):
        '''
        Tests the return of successful assign function
        '''
        cmd = {'pid': 1337, 'retcode': 0, 'stderr': '',
               'stdout': 'net.ipv4.ip_forward = 1'}
        ret = {'net.ipv4.ip_forward': '1'}
        mock_cmd = MagicMock(return_value=cmd)
        with patch.dict(linux_sysctl.__salt__, {'cmd.run_all': mock_cmd}):
            self.assertEqual(linux_sysctl.assign(
                'net.ipv4.ip_forward', 1), ret)

if __name__ == '__main__':
    from integration import run_tests
    run_tests(LinuxSysctlTestCase, needs_daemon=False)
```

25.20 raet

RAET # Reliable Asynchronous Event Transport Protocol

25.20.1 Protocol

Layering:

OSI Layers

7: Application: Format: Data (Stack to Application interface buffering etc) 6: Presentation: Format: Data (Encrypt-Decrypt convert to machine independent format) 5: Session: Format: Data (Interhost communications. Authentication. Groups) 4: Transport: Format: Segments (Reliable delivery of Message, Transactions, Segmentation, Error checking) 3: Network: Format: Packets/Datagrams (Addressing Routing) 2: Link: Format: Frames (Reliable per frame communications connection, Media access controller) 1: Physical: Bits (Transceiver communication connection not reliable)

Link is hidden from Raet Network is IP host address and Udp Port Transport is Raet transactions, service kind, tail error checking, Could include header signing as part of transport reliable delivery serialization of header Session is session id key exchange for signing. Grouping is Road (like 852 channel) Presentation is Encrypt Decrypt body Serialize Deserialize Body Application is body data dictionary

Header signing spans both the Transport and Session layers.

25.20.2 Header

JSON Header (Tradeoff some processing speed for extensibility, ease of use, readability)

Body initially JSON but support for ``packed'' binary body

25.20.3 Packet

Header ASCII Safe JSON Header termination: Empty line given by double pair of carriage return linefeed /r/n/r/n 10 13 10 13 ADAD 1010 1101 1010 1101

In json carriage return and newline characters cannot appear in a json encoded string unless they are escaped with backslash, so the 4 byte combination is illegal in valid json that does not have multi-byte unicode characters.

These means the header must be ascii safe so no multibyte utf-8 strings allowed in header.

Following Header Terminator is variable length signature block. This is binary and the length is provided in the header.

Following the signature block is the packet body or data. This may either be JSON or packed binary. The format is given in the json header

Finally is an optional tail block for error checking or encryption details

25.20.4 Header Fields

In UDP header

sh = source host sp = source port dh = destination host dp = destination port

In RAET Header

hk = header kind hl = header length

vn = version number

sd = Source Device ID dd = Destination Device ID cf = Corresponder Flag mf = Multicast Flag

si = Session ID ti = Transaction ID

sk = Service Kind pk = Packet Kind bf = Burst Flag (Send all Segments or Ordered packets without interleaved acks)

oi = Order Index dt = DateTime Stamp

sn = Segment Number sc = Segment Count

pf = Pending Segment Flag af = All Flag (Resent all Segments not just one)

nk = Auth header kind nl = Auth header length

bk = body kind bl = body length

tk = tail kind tl = tail length

fg = flags packed (Flags) Default `00' hex string 2 byte Hex string with bits (0, 0, af, pf, 0, bf, mf, cf) Zeros are TBD flags

25.20.5 Session Bootstrap

Minion sends packet with SID of Zero with public key of minions Public Private Key pair Master acks packet with SID of Zero to let minion know it received the request

Some time later Master sends packet with SID of zero that accepts the Minion

Minion

25.20.6 Session

Session is important for security. Want one session opened and then multiple transactions within session.

Session ID SID sid

GUID hash to guarantee uniqueness since no guarantee of nonvolatile storage or require file storage to keep last session ID used.

25.20.7 Service Types or Modular Services

Four Service Types

1. One or more maybe (unacknowledged repeat) maybe means no guarantee
2. **Exactly one at most (ack with retries) (duplicate detection idempotent)** at most means fixed number of retries has finite probability of failing B1) finite retries B2) infinite retries with exponential back-off up to a maximum delay
3. **Exactly one of sequence at most (sequence numbered)** Receiver requests retry of missing packet with same B1 or B2 retry type
4. **End to End (Application layer Request Response)** This is two B sub transactions

Initially unicast messaging Eventually support for Multicast

The use case for C) is to fragment large packets as once a UDP packet exceeds the frame size its reliability goes way down So its more reliable to fragment large packets.

Better approach might be to have more modularity. Services Levels

1. **Maybe one or more**
   (a) **Fire and forget** no transaction either side
   (b) **Repeat, no ack, no dupdet** repeat counter send side, no transaction on receive side
   (c) **Repeat, no Ack, dupdet** repeat counter send side, dup detection transaction receive side
2. **More or Less Once**

(a) **retry finite, ack no dupdet** retry timer send side, finite number of retires ack receive side no dupdet

3. **At most Once**

   (a) **retry finite, ack, dupdet** retry timer send side, finite number of retires ack receive side dupdet

4. **Exactly once**

   (a) **ack retry** retry timer send side, ack and duplicate detection receive side Infinite retries with exponential backoff

5. **Sequential sequence number**

   (a) reorder escrow

   (b) Segmented packets

6. request response to application layer

Service Features

1. repeats
2. ack retry transaction id
3. sequence number duplicate detection out of order detection sequencing
4. rep-req

Always include transaction id since multiple transactions on same port So get duplicate detection for free if keep transaction alive but if use

A) Maybe one or more B1) At Least One B2) Exactly One C) One of sequence D) End to End

A) Sender creates transaction id for number of repeats but receiver does not keep transaction alive

B1) Sender creates transaction id keeps it for retries. Receiver keeps it to send ack then kills so retry could be duplicate not detected

B2) Sender creates transaction id keeps for retries Receiver keeps tid for acks on any retires so no duplicates.

C) Sender creates TID and Sequence Number. Receiver checks for out of order sequence and can request retry.

D) Application layer sends response. So question is do we keep transaction open or have response be new transaction. No because then we need a rep-req ID so might as well use the same transaction id. Just keep alive until get response.

Little advantage to B1 vs B2 not having duplicates.

So 4 service types

1. Maybe one or more (unacknowledged repeat)
2. Exactly One (At most one) (ack with retry) (duplicate detection idempotent)
3. One of Sequence (sequence numbered)
4. End to End

Also multicast or unicast

Modular Transaction Table

Sender Side: Transaction ID plus transaction source sender or receiver generated transaction id Repeat Counter Retry Timer Retry Counter (finite retries) Redo Timer (infinite redos with exponential backoff) Sequence number without acks (look for resend requests) Sequence with ack (wait for ack before sending next in sequence) Segmentation

Receiver Side: Nothing just accept packet Acknowledge (can delete transaction after acknowledge) No duplicate detection Transaction timeout (keep transaction until timeout) Duplicate detection save transaction id duplicate detection timeout Request resend of missing packet in sequence Sequence reordering with escrow timeout wait escrow before requesting resend Unsegmentation (request resends of missing segment)

25.21 SaltStack Git Policy

The SaltStack team follows a git policy to maintain stability and consistency with the repository.

The git policy has been developed to encourage contributions and make contributing to Salt as easy as possible. Code contributors to SaltStack projects DO NOT NEED TO READ THIS DOCUMENT, because all contributions come into SaltStack via a single gateway to make it as easy as possible for contributors to give us code.

The primary rule of git management in SaltStack is to make life easy on contributors and developers to send in code. Simplicity is always a goal!

25.21.1 New Code Entry

All new SaltStack code is posted to the *develop* branch, which is the single point of entry. The only exception is when a bugfix to develop cannot be cleanly merged into a release branch and the bugfix needs to be rewritten for the release branch.

25.21.2 Release Branching

SaltStack maintains two types of releases, *Feature Releases* and *Point Releases*. A feature release is managed by incrementing the first or second release point number, so 0.10.5 -> 0.11.0 signifies a feature release and 0.11.0 -> 0.11.1 signifies a point release, also a hypothetical 0.42.7 -> 1.0.0 would also signify a feature release.

Feature Release Branching

Each feature release is maintained in a dedicated git branch derived from the last applicable release commit on develop. All file changes relevant to the feature release will be completed in the develop branch prior to the creation of the feature release branch. The feature release branch will be named after the relevant numbers to the feature release, which constitute the first two numbers. This means that the release branch for the 0.11.0 series is named 0.11.

A feature release branch is created with the following command:

```
# git checkout -b 0.11 # From the develop branch
# git push origin 0.11
```

Point Releases

Each point release is derived from its parent release branch. Constructing point releases is a critical aspect of Salt development and is managed by members of the core development team. Point releases comprise bug and security fixes which are cherry picked from develop onto the aforementioned release branch. At the time when a core developer accepts a pull request a determination needs to be made if the commits in the pull request need to be backported to the release branch. Some simple criteria are used to make this determination:

- Is this commit fixing a bug? Backport

- Does this commit change or add new features in any way? Don't backport
- Is this a PEP8 or code cleanup commit? Don't backport
- Does this commit fix a security issue? Backport

Determining when a point release is going to be made is up to the project leader (Thomas Hatch). Generally point releases are made every 1-2 weeks or if there is a security fix they can be made sooner.

The point release is only designated by tagging the commit on the release branch with release number using the existing convention (version 0.11.1 is tagged with v0.11.1). From the tag point a new source tarball is generated and published to PyPI, and a release announcement is made.

25.22 Salt Conventions

25.22.1 Writing Salt Documentation

Salt's documentation is built using the Sphinx documentation system. It can be build in a large variety of output formats including HTML, PDF, ePub, and manpage.

All the documentation is contained in the main Salt repository. Speaking broadly, most of the narrative documentation is contained within the https://github.com/saltstack/salt/blob/develop/doc subdirectory and most of the reference and API documentation is written inline with Salt's Python code and extracted using a Sphinx extension.

Style

The Salt project recommends the IEEE style guide as a general reference for writing guidelines. Those guidelines are not strictly enforced but rather serve as an excellent resource for technical writing questions. The NCBI style guide is another very approachable resource.

Point-of-view

Use third-person perspective and avoid ``I'', ``we'', ``you'' forms of address. Identify the addressee specifically e.g., ``users should'', ``the compiler does'', etc.

Active voice

Use active voice and present-tense. Avoid filler words.

Title capitalization

Document titles and section titles within a page should follow normal sentence capitalization rules. Words that are capitalized as part of a regular sentence should be capitalized in a title and otherwise left as lowercase. Punctuation can be omitted unless it aids the intent of the title (e.g., exclamation points or question marks).

For example:

```
This is a main heading
======================

Paragraph.
```

```
This is an exciting sub-heading!
--------------------------------

Paragraph.
```

Serial Commas

According to Wikipedia: In English punctuation, a serial comma or series comma (also called Oxford comma and Harvard comma) is a comma placed immediately before the coordinating conjunction (usually and, or, or nor) in a series of three or more terms. For example, a list of three countries might be punctuated either as ``France, Italy, and Spain'' (with the serial comma), or as ``France, Italy and Spain'' (without the serial comma).''

When writing a list that includes three or more items, the serial comma should always be used.

Documenting modules

Documentation for Salt's various module types is inline in the code. During the documentation build process it is extracted and formatted into the final HTML, PDF, etc format.

Inline documentation

Python has special multi-line strings called docstrings as the first element in a function or class. These strings allow documentation to live alongside the code and can contain special formatting. For example:

```
def myfunction(value):
    '''
    Upper-case the given value

    Usage:

    .. code-block:: python

        val = 'a string'
        new_val = myfunction(val)
        print(new_val) # 'A STRING'

    :param value: a string
    :return: a copy of ``value`` that has been upper-cased
    '''
    return value.upper()
```

Specify a release for additions or changes

New functions or changes to existing functions should include a marker that denotes what Salt release will be affected. For example:

```
def myfunction(value):
    '''
    Upper-case the given value
```

```
    .. versionadded:: 2014.7.0

    <...snip...>
    '''
    return value.upper()
```

For changes to a function:

```
def myfunction(value, strip=False):
    '''
    Upper-case the given value

    .. versionchanged:: Boron
        Added a flag to also strip whitespace from the string.

    <...snip...>
    '''
    if strip:
        return value.upper().strip()
    return value.upper()
```

Adding module documentation to the index

Each module type has an index listing all modules of that type. For example: *Full list of builtin execution modules*, *Full list of builtin state modules*, *Full list of builtin renderer modules*. New modules must be added to the index manually.

1. Edit the file for the module type: execution modules, state modules, renderer modules, etc.
2. Add the new module to the alphebetized list.
3. *Build the documentation* which will generate an `.rst` file for the new module in the same directory as the `index.rst`.
4. Commit the changes to `index.rst` and the new `.rst` file and send a pull request.

Cross-references

The Sphinx documentation system contains a wide variety of cross-referencing capabilities.

Glossary entries

Link to *glossary entries* using the term role. A cross-reference should be added the first time a Salt-specific term is used in a document.

```
A common way to encapsulate master-side functionality is by writing a
custom :term:`Runner Function`. Custom Runner Functions are easy to write.
```

Index entries

Sphinx automatically generates many kind of index entries but it is occasionally useful to manually add items to the index.

One method is to use the index directive above the document or section that should appear in the index.

```
.. index:: ! Event, event bus, event system
    see: Reactor; Event
```

Another method is to use the index role inline with the text that should appear in the index. The index entry is created and the target text is left otherwise intact.

```
Information about the :index:`Salt Reactor`
-------------------------------------------

Paragraph.
```

Documents and sections

Each document should contain a unique top-level label of the form:

```
.. _my-page:

My page
=======

Paragraph.
```

Unique labels can be linked using the ref role. This allows cross-references to survive document renames or movement.

```
For more information see :ref:`my-page`.
```

Note, the `:doc:` role should *not* be used to link documents together.

Modules

Cross-references to Salt modules can be added using Sphinx's Python domain roles. For example, to create a link to the `test.ping` function:

```
A useful execution module to test active communication with a minion is the
:py:func:`test.ping <salt.modules.test.ping>` function.
```

Salt modules can be referenced as well:

```
The :py:mod:`test module <salt.modules.test>` contains many useful
functions for inspecting an active Salt connection.
```

The same syntax works for all modules types:

```
One of the workhorse state module functions in Salt is the
:py:func:`file.managed <salt.states.file.managed>` function.
```

Settings

Individual settings in the Salt Master or Salt Minion configuration files are cross-referenced using two custom roles, `conf_master`, and `conf_minion`.

```
The :conf_minion:`minion ID <id>` setting is a unique identifier for a
single minion.
```

Building the documentation

1. Install Sphinx using a system package manager or pip. The package name is often of the form `python-sphinx`. There are no other dependencies.
2. Build the documentation using the provided Makefile or `.bat` file on Windows.

```
cd /path/to/salt/doc
make html
```

3. The generated documentation will be written to the `doc/_build/<format>` directory.
4. A useful method of viewing the HTML documentation locally is the start Python's built-in HTTP server:

```
cd /path/to/salt/doc/_build/html
python -m SimpleHTTPServer
```

Then pull up the documentation in a web browser at http://localhost:8000/.

25.22.2 Salt Formulas

Formulas are pre-written Salt States. They are as open-ended as Salt States themselves and can be used for tasks such as installing a package, configuring, and starting a service, setting up users or permissions, and many other common tasks.

All official Salt Formulas are found as separate Git repositories in the ``saltstack-formulas'' organization on GitHub:

https://github.com/saltstack-formulas

As a simple example, to install the popular Apache web server (using the normal defaults for the underlying distro) simply include the apache-formula from a top file:

```
base:
  'web*':
    - apache
```

Installation

Each Salt Formula is an individual Git repository designed as a drop-in addition to an existing Salt State tree. Formulas can be installed in the following ways.

Adding a Formula as a GitFS remote

One design goal of Salt's GitFS fileserver backend was to facilitate reusable States. GitFS is a quick and natural way to use Formulas.

1. *Install and configure GitFS*.
2. Add one or more Formula repository URLs as remotes in the `gitfs_remotes` list in the Salt Master configuration file:

```
gitfs_remotes:
  - https://github.com/saltstack-formulas/apache-formula
  - https://github.com/saltstack-formulas/memcached-formula
```

   We strongly recommend forking a formula repository into your own GitHub account to avoid unexpected changes to your infrastructure.

   Many Salt Formulas are highly active repositories so pull new changes with care. Plus any additions you make to your fork can be easily sent back upstream with a quick pull request!

3. Restart the Salt master.

Adding a Formula directory manually

Formulas are simply directories that can be copied onto the local file system by using Git to clone the repository or by downloading and expanding a tarball or zip file of the repository. The directory structure is designed to work with `file_roots` in the Salt master configuration.

1. Clone or download the repository into a directory:

```
mkdir -p /srv/formulas
cd /srv/formulas
git clone https://github.com/saltstack-formulas/apache-formula.git

# or

mkdir -p /srv/formulas
cd /srv/formulas
wget https://github.com/saltstack-formulas/apache-formula/archive/master.tar.gz
tar xf apache-formula-master.tar.gz
```

2. Add the new directory to `file_roots`:

```
file_roots:
  base:
    - /srv/salt
    - /srv/formulas/apache-formula
```

3. Restart the Salt Master.

Usage

Each Formula is intended to be immediately usable with sane defaults without any additional configuration. Many formulas are also configurable by including data in Pillar; see the `pillar.example` file in each Formula repository for available options.

Including a Formula in an existing State tree

Formula may be included in an existing `sls` file. This is often useful when a state you are writing needs to `require` or `extend` a state defined in the formula.

Here is an example of a state that uses the epel-formula in a `require` declaration which directs Salt to not install the `python26` package until after the EPEL repository has also been installed:

```
include:
  - epel

python26:
  pkg.installed:
    - require:
      - pkg: epel
```

Including a Formula from a Top File

Some Formula perform completely standalone installations that are not referenced from other state files. It is usually cleanest to include these Formula directly from a Top File.

For example the easiest way to set up an OpenStack deployment on a single machine is to include the openstack-standalone-formula directly from a `top.sls` file:

```
base:
  'myopenstackmaster':
    - openstack
```

Quickly deploying OpenStack across several dedicated machines could also be done directly from a Top File and may look something like this:

```
base:
  'controller':
    - openstack.horizon
    - openstack.keystone
  'hyper-*':
    - openstack.nova
    - openstack.glance
  'storage-*':
    - openstack.swift
```

Configuring Formula using Pillar

Salt Formulas are designed to work out of the box with no additional configuration. However, many Formula support additional configuration and customization through *Pillar*. Examples of available options can be found in a file named `pillar.example` in the root directory of each Formula repository.

Using Formula with your own states

Remember that Formula are regular Salt States and can be used with all Salt's normal state mechanisms. Formula can be required from other States with *require* declarations, they can be modified using `extend`, they can made to watch other states with *The _in versions of requisites*.

The following example uses the stock apache-formula alongside a custom state to create a vhost on a Debian/Ubuntu system and to reload the Apache service whenever the vhost is changed.

```
# Include the stock, upstream apache formula.
include:
  - apache

# Use the watch_in requisite to cause the apache service state to reload
# apache whenever the my-example-com-vhost state changes.
my-example-com-vhost:
  file:
    - managed
    - name: /etc/apache2/sites-available/my-example-com
    - watch_in:
      - service: apache
```

Don't be shy to read through the source for each Formula!

Reporting problems & making additions

Each Formula is a separate repository on GitHub. If you encounter a bug with a Formula please file an issue in the respective repository! Send fixes and additions as a pull request. Add tips and tricks to the repository wiki.

Writing Formulas

Each Formula is a separate repository in the saltstack-formulas organization on GitHub.

Note: Get involved creating new Formulas

The best way to create new Formula repositories for now is to create a repository in your own account on GitHub and notify a SaltStack employee when it is ready. We will add you to the contributors team on the saltstack-formulas organization and help you transfer the repository over. Ping a SaltStack employee on IRC (`#salt` on Freenode) or send an email to the salt-users mailing list.

There are a lot of repositories in that organization! Team members can manage which repositories they are subscribed to on GitHub's watching page: https://github.com/watching.

Style

Maintainability, readability, and reusability are all marks of a good Salt sls file. This section contains several suggestions and examples.

```
# Deploy the stable master branch unless version overridden by passing
# Pillar at the CLI or via the Reactor.

deploy_myapp:
  git.latest:
    - name: git@github.com/myco/myapp.git
    - version: {{ salt.pillar.get('myapp:version', 'master') }}
```

Use a descriptive State ID The ID of a state is used as a unique identifier that may be referenced via other states in *requisites*. It must be unique across the whole state tree (*it is a key in a dictionary*, after all).

In addition a state ID should be descriptive and serve as a high-level hint of what it will do, or manage, or change. For example, `deploy_webapp`, or `apache`, or `reload_firewall`.

Use `module.function` notation So-called ``short-declaration'' notation is preferred for referencing state modules and state functions. It provides a consistent pattern of `module.function` shared between Salt States, the Reactor, Overstate, Salt Mine, the Scheduler, as well as with the CLI.

```
# Do
apache:
  pkg.installed:
    - name: httpd

# Don't
apache:
  pkg:
    - installed
    - name: httpd
```

Salt's state compiler will transform ``short-decs'' into the longer format *when compiling the human-friendly highstate structure into the machine-friendly lowstate structure*.

Specify the `name` parameter Use a unique and permanent identifier for the state ID and reserve `name` for data with variability.

The *name declaration* is a required parameter for all state functions. The state ID will implicitly be used as `name` if it is not explicitly set in the state.

In many state functions the `name` parameter is used for data that varies such as OS-specific package names, OS-specific file system paths, repository addresses, etc. Any time the ID of a state changes all references to that ID must also be changed. Use a permanent ID when writing a state the first time to future-proof that state and allow for easier refactors down the road.

Comment state files YAML allows comments at varying indentation levels. It is a good practice to comment state files. Use vertical whitespace to visually separate different concepts or actions.

```
# Start with a high-level description of the current sls file.
# Explain the scope of what it will do or manage.

# Comment individual states as necessary.
update_a_config_file:
  # Provide details on why an unusual choice was made. For example:
  #
  # This template is fetched from a third-party and does not fit our
  # company norm of using Jinja. This must be processed using Mako.
  file.managed:
    - name: /path/to/file.cfg
    - source: salt://path/to/file.cfg.template
    - template: mako

  # Provide a description or explanation that did not fit within the state
  # ID. For example:
  #
```

```
  # Update the application's last-deployed timestamp.
  # This is a workaround until Bob configures Jenkins to automate RPM
  # builds of the app.
  cmd.run:
    # FIXME: Joe needs this to run on Windows by next quarter. Switch these
    # from shell commands to Salt's file.managed and file.replace state
    # modules.
    - name: |
        touch /path/to/file_last_updated
        sed -e 's/foo/bar/g' /path/to/file_environment
    - onchanges:
      - file: a_config_file
```

Be careful to use Jinja comments for commenting Jinja code and YAML comments for commenting YAML code.

```
# BAD EXAMPLE
# The Jinja in this YAML comment is still executed!
# {% set apache_is_installed = 'apache' in salt.pkg.list_pkgs() %}

# GOOD EXAMPLE
# The Jinja in this Jinja comment will not be executed.
{# {% set apache_is_installed = 'apache' in salt.pkg.list_pkgs() %} #}
```

Easy on the Jinja!

Jinja templating provides vast flexibility and power when building Salt sls files. It can also create an unmaintainable tangle of logic and data. Speaking broadly, Jinja is best used when kept apart from the states (as much as is possible).

Below are guidelines and examples of how Jinja can be used effectively.

Know the evaluation and execution order High-level knowledge of how Salt states are compiled and run is useful when writing states.

The default `renderer` setting in Salt is Jinja piped to YAML. Each is a separate step. Each step is not aware of the previous or following step. Jinja is not YAML aware, YAML is not Jinja aware; they cannot share variables or interact.

- Whatever the Jinja step produces must be valid YAML.
- Whatever the YAML step produces must be a valid *highstate data structure*. (This is also true of the final step for *any of the alternate renderers* in Salt.)
- Highstate can be thought of as a human-friendly data structure; easy to write and easy to read.
- Salt's state compiler validates the highstate and compiles it to low state.
- Low state can be thought of as a machine-friendly data structure. It is a list of dictionaries that each map directly to a function call.
- Salt's state system finally starts and executes on each ``chunk'' in the low state. Remember that requisites are evaluated at runtime.
- The return for each function call is added to the ``running'' dictionary which is the final output at the end of the state run.

The full evaluation and execution order:

```
Jinja -> YAML -> Highstate -> low state -> execution
```

Avoid changing the underlying system with Jinja Avoid calling commands from Jinja that change the underlying system. Commands run via Jinja do not respect Salt's dry-run mode (`test=True`)! This is usually in conflict with the idempotent nature of Salt states unless the command being run is also idempotent.

Inspect the local system A common use for Jinja in Salt states is to gather information about the underlying system. The `grains` dictionary available in the Jinja context is a great example of common data points that Salt itself has already gathered. Less common values are often found by running commands. For example:

```
{% set is_selinux_enabled = salt.cmd.run('sestatus') == '1' %}
```

This is usually best done with a variable assignment in order to separate the data from the state that will make use of the data.

Gather external data One of the most common uses for Jinja is to pull external data into the state file. External data can come from anywhere like API calls or database queries, but it most commonly comes from flat files on the file system or Pillar data from the Salt Master. For example:

```
{% set some_data = salt.pillar.get('some_data', {'sane default': True}) %}

{# or #}

{% load_json 'path/to/file.json' as some_data %}

{# or #}

{% load_text 'path/to/ssh_key.pub' as ssh_pub_key %}

{# or #}

{% from 'path/to/other_file.jinja' import some_data with context %}
```

This is usually best done with a variable assignment in order to separate the data from the state that will make use of the data.

Light conditionals and looping Jinja is extremely powerful for programatically generating Salt states. It is also easy to overuse. As a rule of thumb, if it is hard to read it will be hard to maintain!

Separate Jinja control-flow statements from the states as much as is possible to create readable states. Limit Jinja within states to simple variable lookups.

Below is a simple example of a readable loop:

```
{% for user in salt.pillar.get('list_of_users', []) %}

{# Ensure unique state IDs when looping. #}
{{ user.name }}-{{ loop.index }}:
  user.present:
    - name: {{ user.name }}
    - shell: {{ user.shell }}
```

```
{% endfor %}
```

Avoid putting a Jinja conditionals within Salt states where possible. Readability suffers and the correct YAML indentation is difficult to see in the surrounding visual noise. Parameterization (discussed below) and variables are both useful techniques to avoid this. For example:

```
{# ---- Bad example ---- #}

apache:
  pkg.installed:
    {% if grains.os_family == 'RedHat' %}
    - name: httpd
    {% elif grains.os_family == 'Debian' %}
    - name: apache2
    {% endif %}

{# ---- Better example ---- #}

{% if grains.os_family == 'RedHat' %}
{% set name = 'httpd' %}
{% elif grains.os_family == 'Debian' %}
{% set name = 'apache2' %}
{% endif %}

 apache:
  pkg.installed:
    - name: {{ name }}

{# ---- Good example ---- #}

{% set name = {
    'RedHat': 'httpd',
    'Debian': 'apache2',
}.get(grains.os_family) %}

 apache:
  pkg.installed:
    - name: {{ name }}
```

Dictionaries are useful to effectively ``namespace'' a collection of variables. This is useful with parameterization (discussed below). Dictionaries are also easily combined and merged. And they can be directly serialized into YAML which is often easier than trying to create valid YAML through templating. For example:

```
{# ---- Bad example ---- #}

haproxy_conf:
  file.managed:
    - name: /etc/haproxy/haproxy.cfg
    - template: jinja
    {% if 'external_loadbalancer' in grains.roles %}
    - source: salt://haproxy/external_haproxy.cfg
    {% elif 'internal_loadbalancer' in grains.roles %}
    - source: salt://haproxy/internal_haproxy.cfg
    {% endif %}
    - context:
        {% if 'external_loadbalancer' in grains.roles %}
```

```
        ssl_termination: True
        {% elif 'internal_loadbalancer' in grains.roles %}
        ssl_termination: False
        {% endif %}

{# ---- Better example ---- #}

{% load_yaml as haproxy_defaults %}
common_settings:
  bind_port: 80

internal_loadbalancer:
  source: salt://haproxy/internal_haproxy.cfg
  settings:
    bind_port: 8080
    ssl_termination: False

external_loadbalancer:
  source: salt://haproxy/external_haproxy.cfg
  settings:
    ssl_termination: True
{% endload %}

{% if 'external_loadbalancer' in grains.roles %}
{% set haproxy = haproxy_defaults['external_loadbalancer'] %}
{% elif 'internal_loadbalancer' in grains.roles %}
{% set haproxy = haproxy_defaults['internal_loadbalancer'] %}
{% endif %}

{% do haproxy.settings.update(haproxy_defaults.common_settings) %}

haproxy_conf:
  file.managed:
    - name: /etc/haproxy/haproxy.cfg
    - template: jinja
    - source: {{ haproxy.source }}
    - context: {{ haproxy.settings | yaml() }}
```

There is still room for improvement in the above example. For example, extracting into an external file or replacing the if-elif conditional with a function call to filter the correct data more succinctly. However, the state itself is simple and legible, the data is separate and also simple and legible. And those suggested improvements can be made at some future date without altering the state at all!

Avoid heavy logic and programming Jinja is not Python. It was made by Python programmers and shares many semantics and some syntax but it does not allow for abitrary Python function calls or Python imports. Jinja is a fast and efficient templating language but the syntax can be verbose and visually noisy.

Once Jinja use within an sls file becomes slightly complicated -- long chains of if-elif-elif-else statements, nested conditionals, complicated dictionary merges, wanting to use sets -- instead consider using a different Salt renderer, such as the Python renderer. As a rule of thumb, if it is hard to read it will be hard to maintain -- switch to a format that is easier to read.

Using alternate renderers is very simple to do using Salt's ``she-bang'' syntax at the top of the file. The Python renderer must simply return the correct *highstate data structure*. The following example is a state tree of two sls files, one simple and one complicated.

`/srv/salt/top.sls`:

```
base:
  '*':
    - common_configuration
    - roles_configuration
```

/srv/salt/common_configuration.sls:

```
common_users:
  user.present:
    - names: [larry, curly, moe]
```

/srv/salt/roles_configuration:

```
#!py
def run():
    list_of_roles = set()

    # This example has the minion id in the form 'web-03-dev'.
    # Easily access the grains dictionary:
    try:
        app, instance_number, environment = __grains__['id'].split('-')
        instance_number = int(instance_number)
    except ValueError:
        app, instance_number, environment = ['Unknown', 0, 'dev']

    list_of_roles.add(app)

    if app == 'web' and environment == 'dev':
        list_of_roles.add('primary')
        list_of_roles.add('secondary')
    elif app == 'web' and environment == 'staging':
        if instance_number == 0:
            list_of_roles.add('primary')
        else:
            list_of_roles.add('secondary')

    # Easily cross-call Salt execution modules:
    if __salt__['myutils.query_valid_ec2_instance']():
        list_of_roles.add('is_ec2_instance')

    return {
        'set_roles_grains': {
            'grains.present': [
                {'name': 'roles'},
                {'value': list(list_of_roles)},
            ],
        },
    }
```

Jinja Macros In Salt sls files Jinja macros are useful for one thing and one thing only: creating mini templates that can be reused and rendered on demand. Do not fall into the trap of thinking of macros as functions; Jinja is not Python (see above).

Macros are useful for creating reusable, parameterized states. For example:

```
{% macro user_state(state_id, user_name, shell='/bin/bash', groups=[]) %}
{{ state_id }}:
  user.present:
    - name: {{ user_name }}
    - shell: {{ shell }}
    - groups: {{ groups | json() }}
{% endmacro %}

{% for user_info in salt.pillar.get('my_users', []) %}
{{ user_state('user_number_' ~ loop.index, **user_info) }}
{% endfor %}
```

Macros are also useful for creating one-off ``serializers'' that can accept a data structure and write that out as a domain-specific configuration file. For example, the following macro could be used to write a php.ini config file:

`/srv/salt/php.sls`:

```
php_ini:
  file.managed:
    - name: /etc/php.ini
    - source: salt://php.ini.tmpl
    - template: jinja
    - context:
        php_ini_settings: {{ salt.pillar.get('php_ini', {}) | json() }}
```

`/srv/pillar/php.sls`:

```
PHP:
  engine: 'On'
  short_open_tag: 'Off'
  error_reporting: 'E_ALL & ~E_DEPRECATED & ~E_STRICT'
```

`/srv/salt/php.ini.tmpl`:

```
{% macro php_ini_serializer(data) %}
{% for section_name, name_val_pairs in data.items() %}
[{{ section }}]
{% for name, val in name_val_pairs.items() %}
{{ name }} = "{{ val }}"
{% endfor %}
{% endfor %}
{% endmacro %}

; File managed by Salt at <{{ source }}>.
; Your changes will be overwritten.

{{ php_ini_serializer(php_ini_settings) }}
```

Abstracting static defaults into a lookup table

Separate data that a state uses from the state itself to increases the flexibility and reusability of a state.

An obvious and common example of this is platform-specific package names and file system paths. Another example is sane defaults for an application, or common settings within a company or organization. Organizing such data as a

dictionary (aka hash map, lookup table, associative array) often provides a lightweight namespacing and allows for quick and easy lookups. In addition, using a dictionary allows for easily merging and overriding static values within a lookup table with dynamic values fetched from Pillar.

A strong convention in Salt Formulas is to place platform-specific data, such as package names and file system paths, into a file named `map.jinja` that is placed alongside the state files.

The following is an example from the MySQL Formula. The `grains.filter_by` function performs a lookup on that table using the `os_family` grain (by default).

The result is that the `mysql` variable is assigned to a *subset* of the lookup table for the current platform. This allows states to reference, for example, the name of a package without worrying about the underlying OS. The syntax for referencing a value is a normal dictionary lookup in Jinja, such as `{{ mysql['service'] }}` or the shorthand `{{ mysql.service }}`.

`map.jinja`:

```
{% set mysql = salt['grains.filter_by']({
    'Debian': {
        'server': 'mysql-server',
        'client': 'mysql-client',
        'service': 'mysql',
        'config': '/etc/mysql/my.cnf',
        'python': 'python-mysqldb',
    },
    'RedHat': {
        'server': 'mysql-server',
        'client': 'mysql',
        'service': 'mysqld',
        'config': '/etc/my.cnf',
        'python': 'MySQL-python',
    },
    'Gentoo': {
        'server': 'dev-db/mysql',
        'client': 'dev-db/mysql',
        'service': 'mysql',
        'config': '/etc/mysql/my.cnf',
        'python': 'dev-python/mysql-python',
    },
}, merge=salt['pillar.get']('mysql:lookup')) %}
```

Values defined in the map file can be fetched for the current platform in any state file using the following syntax:

```
{% from "mysql/map.jinja" import mysql with context %}

mysql-server:
  pkg.installed:
    - name: {{ mysql.server }}
  service.running:
    - name: {{ mysql.service }}
```

Collecting common values Common values can be collected into a *base* dictionary. This minimizes repetition of identical values in each of the `lookup_dict` sub-dictionaries. Now only the values that are different from the base must be specified of the alternates:

`map.jinja`:

```
{% set mysql = salt['grains.filter_by']({
    'default': {
        'server': 'mysql-server',
        'client': 'mysql-client',
        'service': 'mysql',
        'config': '/etc/mysql/my.cnf',
        'python': 'python-mysqldb',
    },
    'Debian': {
    },
    'RedHat': {
        'client': 'mysql',
        'service': 'mysqld',
        'config': '/etc/my.cnf',
        'python': 'MySQL-python',
    },
    'Gentoo': {
        'server': 'dev-db/mysql',
        'client': 'dev-db/mysql',
        'python': 'dev-python/mysql-python',
    },
},
merge=salt['pillar.get']('mysql:lookup'), base='default') %}
```

Overriding values in the lookup table Allow static values within lookup tables to be overridden. This is a simple pattern which once again increases flexibility and reusability for state files.

The `merge` argument in `filter_by` specifies the location of a dictionary in Pillar that can be used to override values returned from the lookup table. If the value exists in Pillar it will take precedence.

This is useful when software or configuration files is installed to non-standard locations or on unsupported platforms. For example, the following Pillar would replace the `config` value from the call above.

```
mysql:
  lookup:
    config: /usr/local/etc/mysql/my.cnf
```

Note: Protecting Expansion of Content with Special Characters

When templating keep in mind that YAML does have special characters for quoting, flows, and other special structure and content. When a Jinja substitution may have special characters that will be incorrectly parsed by YAML care must be taken. It is a good policy to use the `yaml_encode` or the `yaml_dquote` Jinja filters:

```
{%- set foo = 7.7 %}
{%- set bar = none %}
{%- set baz = true %}
{%- set zap = 'The word of the day is "salty".' %}
{%- set zip = '"The quick brown fox . . ."' %}

foo: {{ foo|yaml_encode }}
bar: {{ bar|yaml_encode }}
baz: {{ baz|yaml_encode }}
zap: {{ zap|yaml_encode }}
zip: {{ zip|yaml_dquote }}
```

The above will be rendered as below:

```
foo: 7.7
bar: null
baz: true
zap: "The word of the day is \"salty\"."
zip: "\"The quick brown fox . . .\""
```

The `filter_by` function performs a simple dictionary lookup but also allows for fetching data from Pillar and overriding data stored in the lookup table. That same workflow can be easily performed without using `filter_by`; other dictionaries besides data from Pillar can also be used.

```
{% set lookup_table = {...} %}
{% do lookup_table.update(salt.pillar.get('my:custom:data')) %}
```

When to use lookup tables The `map.jinja` file is only a convention within Salt Formulas. This greater pattern is useful for a wide variety of data in a wide variety of workflows. This pattern is not limited to pulling data from a single file or data source. This pattern is useful in States, Pillar, the Reactor, and Overstate as well.

Working with a data structure instead of, say, a config file allows the data to be cobbled together from multiple sources (local files, remote Pillar, database queries, etc), combined, overridden, and searched.

Below are a few examples of what lookup tables may be useful for and how they may be used and represented.

Platform-specific information An obvious pattern and one used heavily in Salt Formulas is extracting platform-specific information such as package names and file system paths in a file named `map.jinja`. The pattern is explained in detail above.

Sane defaults Application settings can be a good fit for this pattern. Store default settings along with the states themselves and keep overrides and sensitive settings in Pillar. Combine both into a single dictionary and then write the application config or settings file.

The example below stores most of the Apache Tomcat `server.xml` file alongside the Tomcat states and then allows values to be updated or augmented via Pillar. (This example uses the BadgerFish format for transforming JSON to XML.)

`/srv/salt/tomcat/defaults.yaml`:

```
Server:
  '@port': '8005'
  '@shutdown': SHUTDOWN
  GlobalNamingResources:
    Resource:
      '@auth': Container
      '@description': User database that can be updated and saved
      '@factory': org.apache.catalina.users.MemoryUserDatabaseFactory
      '@name': UserDatabase
      '@pathname': conf/tomcat-users.xml
      '@type': org.apache.catalina.UserDatabase
  # <...snip...>
```

`/srv/pillar/tomcat.sls`:

```
appX:
  server_xml_overrides:
    Server:
      Service:
        '@name': Catalina
        Connector:
          '@port': '8009'
          '@protocol': AJP/1.3
          '@redirectPort': '8443'
          # <...snip...>
```

/srv/salt/tomcat/server_xml.sls:

```
{% load_yaml 'tomcat/defaults.yaml' as server_xml_defaults %}
{% set server_xml_final_values = salt.pillar.get(
    'appX:server_xml_overrides',
    default=server_xml_defaults,
    merge=True)
%}

appX_server_xml:
  file.serialize:
    - name: /etc/tomcat/server.xml
    - dataset: {{ server_xml_final_values | json() }}
    - formatter: xml_badgerfish
```

The `file.serialize` state can provide a shorthand for creating some files from data structures. There are also many examples within Salt Formulas of creating one-off ``serializers'' (often as Jinja macros) that reformat a data structure to a specific config file format. For example, **`Nginx vhosts`__** or the **`php.ini`__**

__: https://github.com/saltstack-formulas/nginx-formula/blob/5cad4512/nginx/ng/vhosts_config.sls __: https://github.com/saltstack-formulas/php-formula/blob/82e2cd3a/php/ng/files/php.ini

Environment specific information A single state can be reused when it is parameterized as described in the section below, by separating the data the state will use from the state that performs the work. This can be the difference between deploying *Application X* and *Application Y*, or the difference between production and development. For example:

/srv/salt/app/deploy.sls:

```
{# Load the map file. #}
{% load_yaml 'app/defaults.yaml' as app_defaults %}

{# Extract the relevant subset for the app configured on the current
   machine (configured via a grain in this example). #}
{% app = app_defaults.get(salt.grains.get('role') %}

{# Allow values from Pillar to (optionally) update values from the lookup
   table. #}
{% do app_defaults.update(salt.pillar.get('myapp', {}) %}

deploy_application:
  git.latest:
    - name: {{ app.repo_url }}
    - version: {{ app.version }}
```

```
    - target: {{ app.deploy_dir }}

myco/myapp/deployed:
  event.send:
    - data:
        version: {{ app.version }}
    - onchanges:
      - git: deploy_application
```

`/srv/salt/app/defaults.yaml`:

```
appX:
  repo_url: git@github.com/myco/appX.git
  target: /var/www/appX
  version: master
appY:
  repo_url: git@github.com/myco/appY.git
  target: /var/www/appY
  version: v1.2.3.4
```

Single-purpose SLS files

Each sls file in a Formula should strive to do a single thing. This increases the reusability of this file by keeping unrelated tasks from getting coupled together.

As an example, the base Apache formula should only install the Apache httpd server and start the httpd service. This is the basic, expected behavior when installing Apache. It should not perform additional changes such as set the Apache configuration file or create vhosts.

If a formula is single-purpose as in the example above, other formulas, and also other states can `include` and use that formula with *Requisites and Other Global State Arguments* without also including undesirable or unintended side-effects.

The following is a best-practice example for a reusable Apache formula. (This skips platform-specific options for brevity. See the full apache-formula for more.)

```
# apache/init.sls
apache:
  pkg.installed:
    [...]
  service.running:
    [...]

# apache/mod_wsgi.sls
include:
  - apache

mod_wsgi:
  pkg.installed:
    [...]
    - require:
      - pkg: apache

# apache/conf.sls
include:
  - apache
```

```
apache_conf:
  file.managed:
    [...]
    - watch_in:
      - service: apache
```

To illustrate a bad example, say the above Apache formula installed Apache and also created a default vhost. The mod_wsgi state would not be able to include the Apache formula to create that dependency tree without also installing the unneeded default vhost.

Formulas should be reusable. Avoid coupling unrelated actions together.

Parameterization

Parameterization is a key feature of Salt Formulas and also for Salt States. Parameterization allows a single Formula to be reused across many operating systems; to be reused across production, development, or staging environments; and to be reused by many people all with varying goals.

Writing states, specifying ordering and dependencies is the part that takes the longest to write and to test. Filling those states out with data such as users or package names or file locations is the easy part. How many users, what those users are named, or where the files live are all implementation details that **should be parameterized**. This separation between a state and the data that populates a state creates a reusable formula.

In the example below the data that populates the state can come from anywhere -- it can be hard-coded at the top of the state, it can come from an external file, it can come from Pillar, it can come from an execution function call, or it can come from a database query. The state itself doesn't change regardless of where the data comes from. Production data will vary from development data will vary from data from one company to another, however the state itself stays the same.

```
{% set user_list = [
    {'name': 'larry', 'shell': 'bash'},
    {'name': 'curly', 'shell': 'bash'},
    {'name': 'moe', 'shell': 'zsh'},
] %}

{# or #}

{% set user_list = salt['pillar.get']('user_list') %}

{# or #}

{% load_json "default_users.json" as user_list %}

{# or #}

{% set user_list = salt['acme_utils.get_user_list']() %}

{% for user in list_list %}
{{ user.name }}:
  user.present:
    - name: {{ user.name }}
    - shell: {{ user.shell }}
{% endfor %}
```

Configuration

Formulas should strive to use the defaults of the underlying platform, followed by defaults from the upstream project, followed by sane defaults for the formula itself.

As an example, a formula to install Apache **should not** change the default Apache configuration file installed by the OS package. However, the Apache formula **should** include a state to change or override the default configuration file.

Pillar overrides

Pillar lookups must use the safe `get()` and must provide a default value. Create local variables using the Jinja `set` construct to increase redability and to avoid potentially hundreds or thousands of function calls across a large state tree.

```
{% from "apache/map.jinja" import apache with context %}
{% set settings = salt['pillar.get']('apache', {}) %}

mod_status:
  file.managed:
    - name: {{ apache.conf_dir }}
    - source: {{ settings.get('mod_status_conf', 'salt://apache/mod_status.conf') }}
    - template: {{ settings.get('template_engine', 'jinja') }}
```

Any default values used in the Formula must also be documented in the `pillar.example` file in the root of the repository. Comments should be used liberally to explain the intent of each configuration value. In addition, users should be able copy-and-paste the contents of this file into their own Pillar to make any desired changes.

Scripting

Remember that both State files and Pillar files can easily call out to Salt *execution modules* and have access to all the system grains as well.

```
{% if '/storage' in salt['mount.active']() %}
/usr/local/etc/myfile.conf:
  file:
    - symlink
    - target: /storage/myfile.conf
{% endif %}
```

Jinja macros to encapsulate logic or conditionals are discouraged in favor of *writing custom execution modules* in Python.

Repository structure

A basic Formula repository should have the following layout:

```
foo-formula
|-- foo/
|   |-- map.jinja
|   |-- init.sls
|   `-- bar.sls
```

```
|-- CHANGELOG.rst
|-- LICENSE
|-- pillar.example
|-- README.rst
`-- VERSION
```

See also:

template-formula

The template-formula repository has a pre-built layout that serves as the basic structure for a new formula repository. Just copy the files from there and edit them.

README.rst

The README should detail each available `.sls` file by explaining what it does, whether it has any dependencies on other formulas, whether it has a target platform, and any other installation or usage instructions or tips.

A sample skeleton for the `README.rst` file:

```
===
foo
===

Install and configure the FOO service.

.. note::

    See the full `Salt Formulas installation and usage instructions
    <http://docs.saltstack.com/en/latest/topics/development/conventions/formulas.html>`_.

Available states
================

.. contents::
    :local:

``foo``
-------

Install the ``foo`` package and enable the service.

``foo.bar``
-----------

Install the ``bar`` package.
```

CHANGELOG.rst

The `CHANGELOG.rst` file should detail the individual versions, their release date and a set of bullet points for each version highlighting the overall changes in a given version of the formula.

A sample skeleton for the *CHANGELOG.rst* file:

`CHANGELOG.rst`:

```
foo formula
===========

0.0.2 (2013-01-01)

- Re-organized formula file layout
- Fixed filename used for upstart logger template
- Allow for pillar message to have default if none specified
```

Versioning

Formula are versioned according to Semantic Versioning, http://semver.org/.

Note: Given a version number MAJOR.MINOR.PATCH, increment the:

1. MAJOR version when you make incompatible API changes,
2. MINOR version when you add functionality in a backwards-compatible manner, and
3. PATCH version when you make backwards-compatible bug fixes.

Additional labels for pre-release and build metadata are available as extensions to the MAJOR.MINOR.PATCH format.

Formula versions are tracked using Git tags as well as the `VERSION` file in the formula repository. The `VERSION` file should contain the currently released version of the particular formula.

Testing Formulas

A smoke-test for invalid Jinja, invalid YAML, or an invalid Salt state structure can be performed by with the `state.show_sls` function:

```
salt '*' state.show_sls apache
```

Salt Formulas can then be tested by running each `.sls` file via `state.sls` and checking the output for the success or failure of each state in the Formula. This should be done for each supported platform.

25.22.3 SaltStack Packaging Guide

Since Salt provides a powerful toolkit for system management and automation, the package can be spit into a number of sub-tools. While packaging Salt as a single package containing all components is perfectly acceptable, the split packages should follow this convention.

Patching Salt For Distributions

The occasion may arise where Salt source and default configurations may need to be patched. It is preferable if Salt is only patched to include platform specific additions or to fix release time bugs. It is preferable that configuration settings and operations remain in the default state, as changes here lowers the user experience for users moving across distributions.

In the event where a packager finds a need to change the default configuration it is advised to add the files to the master.d or minion.d directories.

Source Files

Release packages should always be built from the source tarball distributed via pypi. Release packages should *NEVER* use a git checkout as the source for distribution.

Single Package

Shipping Salt as a single package, where the minion, master, and all tools are together is perfectly acceptable and practiced by distributions such as FreeBSD.

Split Package

Salt Should always be split in a standard way, with standard dependencies, this lowers cross distribution confusion about what components are going to be shipped with specific packages. These packages can be defined from the Salt Source as of Salt 2014.1.0:

Salt Common

The *salt-common* or *salt* package should contain the files provided by the salt python package, or all files distributed from the `salt/` directory in the source distribution packages. The documentation contained under the `doc/` directory can be a part of this package but splitting out a doc package is preferred. Since salt-call is the entry point to utilize the libs and is useful for all salt packages it is included in the salt-common package.

Name

- *salt* OR *salt-common*

Files

- *salt/**
- *man/salt.7*
- *scripts/salt-call*
- *tests/**
- *man/salt-call.1*

Depends

- *Python 2.6-2.7*
- *PyYAML*
- *Jinja2*

Salt Master

The *salt-master* package contains the applicable scripts, related man pages and init information for the given platform.

Name

- *salt-master*

Files

- *scripts/salt-master*
- *scripts/salt*
- *scripts/salt-run*
- *scripts/salt-key*
- *scripts/salt-cp*
- *pkg/<master init data>*
- *man/salt.1*
- *man/salt-master.1*
- *man/salt-run.1*
- *man/salt-key.1*
- *man/salt-cp.1*
- *conf/master*

Depends

- *Salt Common*
- *ZeroMQ* >= 3.2
- *PyZMQ* >= 2.10
- *PyCrypto*
- *M2Crypto*
- *Python MessagePack* (Messagepack C lib, or msgpack-pure)

Salt Syndic

The Salt Syndic package can be rolled completely into the Salt Master package. Platforms which start services as part of the package deployment need to maintain a separate *salt-syndic* package (primarily Debian based platforms).

The Syndic may optionally not depend on the anything more than the Salt Master since the master will bring in all needed dependencies, but fall back to the platform specific packaging guidelines.

Name

- *salt-syndic*

Files

- *scripts/salt-syndic*
- *pkg/<syndic init data>*
- *man/salt-syndic.1*

Depends

- *Salt Common*
- *Salt Master*
- *ZeroMQ* >= 3.2
- *PyZMQ* >= 2.10
- *PyCrypto*
- *M2Crypto*
- *Python MessagePack* (Messagepack C lib, or msgpack-pure)

Salt Minion

The Minion is a standalone package and should not be split beyond the *salt-minion* and *salt-common* packages.

Name

- *salt-minion*

Files

- *scripts/salt-minion*
- *pkg/<minion init data>*
- *man/salt-minion.1*
- *conf/minion*

Depends

- *Salt Common*
- *ZeroMQ* >= 3.2
- *PyZMQ* >= 2.10
- *PyCrypto*
- *M2Crypto*
- *Python MessagePack* (Messagepack C lib, or msgpack-pure)

Salt SSH

Since Salt SSH does not require the same dependencies as the minion and master, it should be split out.

Name

- *salt-ssh*

Files

- *scripts/salt-ssh*
- *man/salt-ssh.1*
- *conf/cloud**

Depends

- *Salt Common*
- *Python MessagePack* (Messagepack C lib, or msgpack-pure)

Salt Cloud

As of Salt 2014.1.0 Salt Cloud is included in the same repo as Salt. This can be split out into a separate package or it can be included in the salt-master package.

Name

- *salt-cloud*

Files

- *scripts/salt-cloud*
- *man/salt-cloud.1*

Depends

- *Salt Common*
- *apache libcloud* >= 0.14.0

Salt Doc

The documentation package is very distribution optional. A completely split package will split out the documentation, but some platform conventions do not prefer this. If the documentation is not split out, it should be included with the *Salt Common* package.

Name

- *salt-doc*

Files

- *doc/**

Optional Depends

- *Salt Common*
- *Python Sphinx*
- *Make*

25.22.4 Salt Release Process

The goal for Salt projects is to cut a new feature release every four to six weeks. This document outlines the process for these releases, and the subsequent bug fix releases which follow.

Feature Release Process

When a new release is ready to be cut, the person responsible for cutting the release will follow the following steps (written using the 0.16 release as an example):

1. All open issues on the release milestone should be moved to the next release milestone. (e.g. from the `0.16` milestone to the `0.17` milestone)
2. Release notes should be created documenting the major new features and bugfixes in the release.
3. Create an annotated tag with only the major and minor version numbers, preceded by the letter `v`. (e.g. `v0.16`) This tag will reside on the `develop` branch.
4. Create a branch for the new release, using only the major and minor version numbers. (e.g. `0.16`)
5. On this new branch, create an annotated tag for the first revision release, which is generally a release candidate. It should be preceded by the letter `v`. (e.g. `v0.16.0RC`)
6. The release should be packaged from this annotated tag and uploaded to PyPI as well as the GitHub releases page for this tag.
7. The packagers should be notified on the salt-packagers mailing list so they can create packages for all the major operating systems. (note that release candidates should go in the testing repositories)
8. After the packagers have been given a few days to compile the packages, the release is announced on the salt-users mailing list.
9. Log into RTD and add the new release there. (Have to do it manually)

Maintenance and Bugfix Releases

Once a release has been cut, regular cherry-picking sessions should begin to cherry-pick any bugfixes from the `develop` branch to the release branch (e.g. `0.16`). Once major bugs have been fixes and cherry-picked, a bugfix release can be cut:

1. On the release branch (i.e. `0.16`), create an annotated tag for the revision release. It should be preceded by the letter `v`. (e.g. `v0.16.2`) Release candidates are unnecessary for bugfix releases.
2. The release should be packaged from this annotated tag and uploaded to PyPI.
3. The packagers should be notified on the salt-packagers mailing list so they can create packages for all the major operating systems.
4. After the packagers have been given a few days to compile the packages, the release is announced on the salt-users mailing list.

Cherry-Picking Process for Bugfixes

Bugfixes should be made on the `develop` branch. If the bug also applies to the current release branch, then on the pull request against `develop`, the user should mention `@basepi` and ask for the pull request to be cherry-picked. If it is verified that the fix is a bugfix, then the `Bugfix -- Cherry-Pick` label will be applied to the pull request. When those commits are cherry-picked, the label will be switched to the `Bugfix -- [Done] Cherry-Pick` label. This allows easy recognition of which pull requests have been cherry-picked, and which are still pending to be cherry-picked. All cherry-picked commits will be present in the next release.

Features will not be cherry-picked, and will be present in the next feature release.

25.22.5 Salt Coding Style

Salt is developed with a certain coding style, while the style is dominantly PEP 8 it is not completely PEP 8. It is also noteworthy that a few development techniques are also employed which should be adhered to. In the end, the code is made to be ``Salty''.

Most importantly though, we will accept code that violates the coding style and KINDLY ask the contributor to fix it, or go ahead and fix the code on behalf of the contributor. Coding style is NEVER grounds to reject code contributions, and is never grounds to talk down to another member of the community (There are no grounds to treat others without respect, especially people working to improve Salt)‼

Linting

Most Salt style conventions are codified in Salt's `.pylintrc` file. This file is found in the root of the Salt project and can be passed as an argument to the pylint program as follows:

```
pylint --rcfile=/path/to/salt/.pylintrc salt/dir/to/lint
```

Strings

Salt follows a few rules when formatting strings:

Single Quotes

In Salt, all strings use single quotes unless there is a good reason not to. This means that docstrings use single quotes, standard strings use single quotes etc.:

```
def foo():
    '''
    A function that does things
    '''
    name = 'A name'
    return name
```

Formatting Strings

All strings which require formatting should use the *.format* string method:

```
data = 'some text'
more = '{0} and then some'.format(data)
```

Make sure to use indices or identifiers in the format brackets, since empty brackets are not supported by python 2.6.

Please do NOT use printf formatting.

Docstring Conventions

Docstrings should always add a newline, docutils takes care of the new line and it makes the code cleaner and more vertical:

GOOD:

```
def bar():
    '''
    Here lies a docstring with a newline after the quotes and is the salty
    way to handle it! Vertical code is the way to go!
    '''
    return
```

BAD:

```
def baz():
    '''This is not ok!'''
    return
```

When adding a new function or state, where possible try to use a `versionadded` directive to denote when the function or state was added.

```
def new_func(msg=''):
    '''
    .. versionadded:: 0.16.0

    Prints what was passed to the function.

    msg : None
        The string to be printed.
    '''
    print msg
```

If you are uncertain what version should be used, either consult a core developer in IRC or bring this up when opening your *pull request* and a core developer will add the proper version once your pull request has been merged. Bugfixes will be available in a bugfix release (i.e. 0.17.1, the first bugfix release for 0.17.0), while new features are held for feature releases, and this will affect what version number should be used in the `versionadded` directive.

Similar to the above, when an existing function or state is modified (for example, when an argument is added), then under the explanation of that new argument a `versionadded` directive should be used to note the version in which the new argument was added. If an argument's function changes significantly, the `versionchanged` directive can be used to clarify this:

```
def new_func(msg='', signature=''):
    '''
    .. versionadded:: 0.16.0
```

```
    Prints what was passed to the function.

    msg : None
        The string to be printed. Will be prepended with 'Greetings! '.

    .. versionchanged:: 0.17.1

    signature : None
        An optional signature.

    .. versionadded 0.17.0
    '''
    print 'Greetings! {0}\n\n{1}'.format(msg, signature)
```

Dictionaries

Dictionaries should be initialized using *{}* instead of *dict()*.

See here for an in-depth discussion of this topic.

Imports

Salt code prefers importing modules and not explicit functions. This is both a style and functional preference. The functional preference originates around the fact that the module import system used by pluggable modules will include callable objects (functions) that exist in the direct module namespace. This is not only messy, but may unintentionally expose code python libs to the Salt interface and pose a security problem.

To say this more directly with an example, this is *GOOD*:

```
import os

def minion_path():
    path = os.path.join(self.opts['cachedir'], 'minions')
    return path
```

This on the other hand is *DISCOURAGED*:

```
from os.path import join

def minion_path():
    path = join(self.opts['cachedir'], 'minions')
    return path
```

The time when this is changed is for importing exceptions, generally directly importing exceptions is preferred:

This is a good way to import exceptions:

```
from salt.exceptions import CommandExecutionError
```

Absolute Imports

Although absolute imports seems like an awesome idea, please do not use it. Extra care would be necessary all over salt's code in order for absolute imports to work as supposed. Believe it, it has been tried before and, as a tried example, by renaming `salt.modules.sysmod` to `salt.modules.sys`, all other salt modules which needed to import `sys` would have to also import `absolute_import`, which should be avoided.

Vertical is Better

When writing Salt code, vertical code is generally preferred. This is not a hard rule but more of a guideline. As PEP 8 specifies, Salt code should not exceed 79 characters on a line, but it is preferred to separate code out into more newlines in some cases for better readability:

```
import os

os.chmod(
        os.path.join(self.opts['sock_dir'],
            'minion_event_pub.ipc'),
        448
        )
```

Where there are more line breaks, this is also apparent when constructing a function with many arguments, something very common in state functions for instance:

```
def managed(name,
        source=None,
        source_hash='',
        user=None,
        group=None,
        mode=None,
        template=None,
        makedirs=False,
        context=None,
        replace=True,
        defaults=None,
        env=None,
        backup='',
        **kwargs):
```

Note: Making function and class definitions vertical is only required if the arguments are longer then 80 characters. Otherwise, the formatting is optional and both are acceptable.

Line Length

For function definitions and function calls, Salt adheres to the PEP-8 specification of at most 80 characters per line.

Non function definitions or function calls, please adopt a soft limit of 120 characters per line. If breaking the line reduces the code readability, don't break it. Still, try to avoid passing that 120 characters limit and remember, **vertical is better... unless it isn't**

Indenting

Some confusion exists in the python world about indenting things like function calls, the above examples use 8 spaces when indenting comma-delimited constructs.

The confusion arises because the pep8 program INCORRECTLY flags this as wrong, where PEP 8, the document, cites only using 4 spaces here as wrong, as it doesn't differentiate from a new indent level.

Right:

```
def managed(name,
        source=None,
        source_hash='',
        user=None)
```

WRONG:

```
def managed(name,
    source=None,
    source_hash='',
    user=None)
```

Lining up the indent is also correct:

```
def managed(name,
            source=None,
            source_hash='',
            user=None)
```

This also applies to function calls and other hanging indents.

pep8 and Flake8 (and, by extension, the vim plugin Syntastic) will complain about the double indent for hanging indents. This is a known conflict between pep8 (the script) and the actual PEP 8 standard. It is recommended that this particular warning be ignored with the following lines in `~/.config/flake8`:

```
[flake8]
ignore = E226,E241,E242,E126
```

Make sure your Flake8/pep8 are up to date. The first three errors are ignored by default and are present here to keep the behavior the same. This will also work for pep8 without the Flake8 wrapper -- just replace all instances of `flake8' with `pep8', including the filename.

Code Churn

Many pull requests have been submitted that only churn code in the name of PEP 8. Code churn is a leading source of bugs and is strongly discouraged. While style fixes are encouraged they should be isolated to a single file per commit, and the changes should be legitimate, if there are any questions about whether a style change is legitimate please reference this document and the official PEP 8 (http://legacy.python.org/dev/peps/pep-0008/) document before changing code. Many claims that a change is PEP 8 have been invalid, please double check before committing fixes.

CHAPTER 26

Release notes

See the `version numbers` page for more information about the version numbering scheme.

26.1 Latest Stable Release

26.1.1 Salt 2014.7.0 Release Notes - Codename Helium

This release is the largest Salt release ever, with more features and commits then any previous release of Salt. Everything from the new RAET transport to major updates in Salt Cloud and the merging of Salt API into the main project.

Important: The Fedora/RHEL/CentOS **salt-master** package has been modified for this release. The following components of Salt have been broken out and placed into their own packages:

- salt-syndic
- salt-cloud
- salt-ssh

When the **salt-master** package is upgraded, these components will be removed, and they will need to be manually installed.

Important: Compound/pillar matching have been temporarily disabled for the `mine` and `publish` modules for this release due to the possibility of inferring pillar data using pillar glob matching. A proper fix is now in the 2014.7 branch and scheduled for the 2014.7.1 release, and compound matching and non-globbing pillar matching will be re-enabled at that point.

Compound and pillar matching for normal salt commands are unaffected.

New Transport!

RAET Transport Option

This has been a HUGE amount of work, but the beta release of Salt with RAET is ready to go. RAET is a reliable queuing transport system that has been developed in partnership with a number of large enterprises to give Salt an alternative to ZeroMQ and a way to get Salt to scale well beyond tens of thousands of servers. Unlike ZeroMQ, RAET is completely asynchronous in every aspect of its operation and has been developed using the flow programming paradigm. This allows for many new capabilities to be added to Salt in the upcoming releases.

Please keep in mind that this is a beta release of RAET and we hope for bugs to be worked out, performance to be better realized and more in the 2015.5.0 release.

Simply stated, users running Salt with RAET should expect some hiccups as we hammer out the update. This is a BETA release of Salt RAET.

For information about how to use Salt with RAET please see the `tutorial`.

Salt SSH Enhancements

Salt SSH has just entered a new league, with substantial updates and improvements to make salt-ssh more reliable and easier then ever! From new features like the ansible roster and fileserver backends to the new pypi salt-ssh installer to lowered deps and a swath of bugfixes, salt-ssh is basically reborn!

Install salt-ssh Using pip

Salt-ssh is now pip-installable!

https://pypi.python.org/pypi/salt-ssh/

Pip will bring in all of the required deps, and while some deps are compiled, they all include pure python implementations, meaning that any compile errors which may be seen can be safely ignored.

```
pip install salt-ssh
```

Fileserver Backends

Salt-ssh can now use the salt fileserver backend system. This allows for the gitfs, hgfs, s3, and many more ways to centrally store states to be easily used with salt-ssh. This also allows for a distributed team to easily use a centralized source.

Saltfile Support

The new saltfile system makes it easy to have a user specific custom extended configuration.

Ext Pillar

Salt-ssh can now use the external pillar system. Making it easier then ever to use salt-ssh with teams.

No More sshpass

Thanks to the enhancements in the salt vt system, salt-ssh no longer requires sshpass to send passwords to ssh. This also makes the manipulation of ssh calls substantially more flexible, allowing for intercepting ssh calls in a much more fluid way.

Pure Python Shim

The salt-ssh call originally used a shell script to discover what version of python to execute with and determine the state of the ssh code deployment. This shell script has been replaced with a pure python version making it easy to increase the capability of the code deployment without causing platform inconsistency issues with different shell interpreters.

Custom Module Delivery

Custom modules are now seamlessly delivered. This makes the deployment of custom grains, states, execution modules and returners a seamless process.

CP Module Support

Salt-ssh now makes simple file transfers easier then ever! The *cp* module allows for files to be conveniently sent from the salt fileserver system down to systems.

More Thin Directory Options

Salt ssh functions by copying a subset of the salt code, or *salt thin* down to the target system. In the past this was always transferred to /tmp/.salt and cached there for subsequent commands.

Now, salt thin can be sent to a random directory and removed when the call is complete with the *-W* option. The new *-W* option still uses a static location but will clean up that location when finished.

The default *salt thin* location is now user defined, allowing multiple users to cleanly access the same systems.

State System Enhancements

New Imperative State Keyword ``Listen''

The new `listen` and `listen_in` keywords allow for completely imperative states by calling the `mod_watch()` routine after all states have run instead of re-ordering the states.

Mod Aggregate Runtime Manipulator

The new `mod_aggregate` system allows for the state system to rewrite the state data during execution. This allows for state definitions to be aggregated dynamically at runtime.

The best example is found in the `pkg` state. If `mod_aggregate` is turned on, then when the first pkg state is reached, the state system will scan all of the other running states for pkg states and take all other packages set for install and install them all at once in the first pkg state.

These runtime modifications make it easy to run groups of states together. In future versions, we hope to fill out the `mod_aggregate` system to build in more and more optimizations.

For more documentation on `mod_aggregate`, see *the documentation*.

New Requisites: onchanges and onfail

The new `onchanges` and `onchanges_in` requisites make a state apply only if there are changes in the required state. This is useful to execute post hooks after changes occur on a system.

The other new requisites, `onfail`, and `onfail_in`, allow for a state to run in reaction to the failure of another state.

For more information about these new requisites, see the *requisites documentation*.

Global onlyif and unless

The `onlyif` and `unless` options can now be used for any state declaration.

Use `names` to expand and override values

The *names declaration* in Salt's state system can now override or add values to the expanded data structure. For example:

```
my_users:
  user.present:
    - names:
      - larry
      - curly
      - moe:
        - shell: /bin/zsh
        - groups:
          - wheel
    - shell: /bin/bash
```

Major Features

Scheduler Additions

The Salt scheduler system has received MAJOR enhancements, allowing for cron-like scheduling and much more granular timing routines. See `here` for more info.

Red Hat 7 Family Support

All the needed additions have been made to run Salt on RHEL 7 and derived OSes like CentOS and Scientific.

Fileserver Backends in salt-call

Fileserver backends like gitfs can now be used without a salt master! Just add the fileserver backend configuration to the minion config and execute salt-call. This has been a much-requested feature and we are happy to finally bring it to our users.

Amazon Execution Modules

An entire family of execution modules further enhancing Salt's Amazon Cloud support. They include the following:

- `Autoscale Groups` (includes `state support`) -- related: `Launch Control` states
- `Cloud Watch` (includes `state support`)
- `Elastic Cache` (includes `state support`)
- `Elastic Load Balancer` (includes `state support`)
- `IAM Identity and Access Management` (includes `state support`)
- `Route53 DNS` (includes `state support`)
- `Security Groups` (includes `state support`)
- `Simple Queue Service` (includes `state support`)

LXC Runner Enhancements

BETA The Salt LXC management system has received a number of enhancements which make running an LXC cloud entirely from Salt an easy proposition.

Next Gen Docker Management

The Docker support in Salt has been increased at least ten fold. The Docker API is now completely exposed and Salt ships with Docker data tracking systems which make automating Docker deployments very easy.

Peer System Performance Improvements

The peer system communication routines have been refined to make the peer system substantially faster.

SDB

Encryption at rest for configs

GPG Renderer

Encrypted pillar at rest

OpenStack Expansion

Lots of new OpenStack stuff

Queues System

Ran change external queue systems into Salt events

Multi Master Failover Additions

Connecting to multiple masters is more dynamic then ever

Chef Execution Module

Managing Chef with Salt just got even easier!

salt-api Project Merge

The `salt-api` project has been merged into Salt core and is now available as part of the regular `salt-master` package install. No API changes were made, the **`salt-api`** script and init scripts remain intact.

`salt-api` has always provided Yet Another Pluggable Interface to Salt (TM) in the form of ``netapi'' modules. These are modules that bind to a port and start a service. Like many of Salt's other module types, netapi modules often have library and configuration dependencies. See the documentation for each module for instructions.

See also:

The full list of netapi modules.

Synchronous and Asynchronous Execution of Runner and Wheel Modules `salt.runner.RunnerClient` and `salt.wheel.WheelClient` have both gained complimentary `cmd_sync` and `cmd_async` methods allowing for synchronous and asynchronous execution of any Runner or Wheel module function, all protected using Salt's *external authentication* system. `salt-api` benefits from this addition as well.

`rest_cherrypy` Additions The `rest_cherrypy` netapi module provides the main REST API for Salt.

Web Hooks This release of course includes the Web Hook additions from the most recent `salt-api` release, which allows external services to signal actions within a Salt infrastructure. External services such as Amazon SNS, Travis-CI, or GitHub, as well as internal services that cannot or should not run a Salt minion daemon can be used as first-class components in Salt's rich orchestration capabilities.

The raw HTTP request body is now available in the event data. This is sometimes required information for checking an HMAC signature in order to verify a HTTP request. As an example, Amazon or GitHub requests are signed this way.

Generating and Accepting Minion Keys The `/key` convenience URL generates a public and private key for a minion, automatically pre-accepts the public key on the Salt Master, and returns both keys as a tarball for download.

This allows for easily bootstrapping the key on a new minion with a single HTTP call, such as with a Kickstart script, all using regular shell tools.

```
curl -sS http://salt-api.example.com:8000/keys \
        -d mid=jerry \
        -d username=kickstart \
        -d password=kickstart \
        -d eauth=pam \
        -o jerry-salt-keys.tar
```

Fileserver Backend Enhancements

All of the fileserver backends have been overhauled to be faster, lighter, and more reliable. The VCS backends (`gitfs`, `hgfs`, and `svnfs`) have also received a **lot** of new features.

Additionally, most config parameters for the VCS backends can now be configured on a per-remote basis, allowing for global config parameters to be overridden for a specific gitfs/hgfs/svnfs remote.

New `gitfs` Features

Pygit2 and Dulwich In addition to supporting GitPython, support for pygit2 (0.20.3 and newer) and dulwich have been added. Provided a compatible version of pygit2 is installed, it will now be the default provider. The config parameter `gitfs_provider` has been added to allow one to choose a specific provider for gitfs.

Mountpoints Prior to this release, to serve a file from gitfs at a salt fileserver URL of `salt://foo/bar/baz.txt`, it was necessary to ensure that the parent directories existed in the repository. A new config parameter `gitfs_mountpoint` allows gitfs remotes to be exposed starting at a user-defined `salt://` URL.

Environment Whitelisting/Blacklisting By default, gitfs will expose all branches and tags as Salt fileserver environments. Two new config parameters, `gitfs_env_whitelist`, and `gitfs_env_blacklist`, allow more control over which branches and tags are exposed. More detailed information on how these two options work can be found in the *Gitfs Walkthrough*.

Expanded Authentication Support As of pygit2 0.20.3, both http(s) and SSH key authentication are supported, and Salt now also supports both authentication methods when using pygit2. Keep in mind that pygit2 0.20.3 is not yet available on many platforms, so those who had been using authenticated git repositories with a passphraseless key should stick to GitPython if a new enough pygit2 is not yet available for the platform on which the master is running.

A full explanation of how to use authentication can be found in the *Gitfs Walkthrough*.

New `hgfs` Features

Mountpoints This feature works exactly like its *gitfs counterpart*. The new config parameter is called `hgfs_mountpoint`.

Environment Whitelisting/Blacklisting This feature works exactly like its *gitfs counterpart*. The new config parameters are called `hgfs_env_whitelist` and `hgfs_env_blacklist`.

New `svnfs` Features

Mountpoints This feature works exactly like its *gitfs counterpart*. The new config parameter is called `svnfs_mountpoint`.

Environment Whitelisting/Blacklisting This feature works exactly like its *gitfs counterpart*. The new config parameters are called `svnfs_env_whitelist` and `svnfs_env_blacklist`.

Configurable Trunk/Branches/Tags Paths Prior to this release, the paths where trunk, branches, and tags were located could only be in directores named ``trunk'', ``branches'', and ``tags'' directly under the root of the repository. Three new config parameters (`svnfs_trunk`, `svnfs_branches`, and `svnfs_tags`) allow SVN repositories which are laid out differently to be used with svnfs.

New `minionfs` Features

Mountpoint This feature works exactly like its *gitfs counterpart*. The new config parameter is called `minionfs_mountpoint`. The one major difference is that, as minionfs doesn't use multiple remotes (it just serves up files pushed to the master using `cp.push`) there is no such thing as a per-remote configuration for `minionfs_mountpoint`.

Changing the Saltenv from Which Files are Served A new config parameter (`minionfs_env`) allows minionfs files to be served from a Salt fileserver environment other than `base`.

Minion Whitelisting/Blacklisting By default, minionfs will expose the pushed files from all minions. Two new config parameters, `minionfs_whitelist`, and `minionfs_blacklist`, allow minionfs to be restricted to serve files from only the desired minions.

Pyobjects Renderer

Salt now ships with with the `Pyobjects Renderer` that allows for construction of States using pure Python with an idiomatic object interface.

New Modules

In addition to the Amazon modules mentioned above, there are also several other new execution modules:

- `Oracle`
- `Random`
- `Redis`
- `Amazon Simple Queue Service`
- `Block Device Management`
- `CoreOS etcd`
- `Genesis`
- `InfluxDB`
- `Server Density`
- `Twilio Notifications`
- `Varnish`
- `ZNC IRC Bouncer`
- `SMTP`

New Runners

- Map/Reduce Style
- Queue

New External Pillars

- CoreOS etcd

New Salt-Cloud Providers

- Aliyun ECS Cloud
- LXC Containers
- Proxmox (OpenVZ containers & KVM)

Salt Call Change

When used with a returner, salt-call now contacts a master if `--local` is not specicified.

Deprecations

salt.modules.virtualenv_mod

- Removed deprecated `memoize` function from `salt/utils/__init__.py` (deprecated)
- Removed deprecated `no_site_packages` argument from `create` function (deprecated)
- Removed deprecated `check_dns` argument from `minion_config` and `apply_minion_config` functions (deprecated)
- Removed deprecated `OutputOptionsWithTextMixIn` class from `salt/utils/parsers.py` (deprecated)
- Removed the following deprecated functions from `salt/modules/ps.py`: - `physical_memory_usage` (deprecated) - `virtual_memory_usage` (deprecated) - `cached_physical_memory` (deprecated) - `physical_memory_buffers` (deprecated)
- Removed deprecated cloud arguments from `cloud_config` function in `salt/config.py`: - `vm_config` (deprecated) - `vm_config_path` (deprecated)
- Removed deprecated `libcloud_version` function from `salt/cloud/libcloudfuncs.py` (deprecated)
- Removed deprecated `CloudConfigMixIn` class from `salt/utils/parsers.py` (deprecated)

26.2 Previous Releases

26.2.1 Salt 2014.7.0 Release Notes - Codename Helium

This release is the largest Salt release ever, with more features and commits then any previous release of Salt. Everything from the new RAET transport to major updates in Salt Cloud and the merging of Salt API into the main

project.

Important: The Fedora/RHEL/CentOS **salt-master** package has been modified for this release. The following components of Salt have been broken out and placed into their own packages:

- salt-syndic
- salt-cloud
- salt-ssh

When the **salt-master** package is upgraded, these components will be removed, and they will need to be manually installed.

Important: Compound/pillar matching have been temporarily disabled for the `mine` and `publish` modules for this release due to the possibility of inferring pillar data using pillar glob matching. A proper fix is now in the 2014.7 branch and scheduled for the 2014.7.1 release, and compound matching and non-globbing pillar matching will be re-enabled at that point.

Compound and pillar matching for normal salt commands are unaffected.

New Transport!

RAET Transport Option

This has been a HUGE amount of work, but the beta release of Salt with RAET is ready to go. RAET is a reliable queuing transport system that has been developed in partnership with a number of large enterprises to give Salt an alternative to ZeroMQ and a way to get Salt to scale well beyond tens of thousands of servers. Unlike ZeroMQ, RAET is completely asynchronous in every aspect of its operation and has been developed using the flow programming paradigm. This allows for many new capabilities to be added to Salt in the upcoming releases.

Please keep in mind that this is a beta release of RAET and we hope for bugs to be worked out, performance to be better realized and more in the 2015.5.0 release.

Simply stated, users running Salt with RAET should expect some hiccups as we hammer out the update. This is a BETA release of Salt RAET.

For information about how to use Salt with RAET please see the `tutorial`.

Salt SSH Enhancements

Salt SSH has just entered a new league, with substantial updates and improvements to make salt-ssh more reliable and easier then ever! From new features like the ansible roster and fileserver backends to the new pypi salt-ssh installer to lowered deps and a swath of bugfixes, salt-ssh is basically reborn!

Install salt-ssh Using pip

Salt-ssh is now pip-installable!

https://pypi.python.org/pypi/salt-ssh/

Pip will bring in all of the required deps, and while some deps are compiled, they all include pure python implementations, meaning that any compile errors which may be seen can be safely ignored.

```
pip install salt-ssh
```

Fileserver Backends

Salt-ssh can now use the salt fileserver backend system. This allows for the gitfs, hgfs, s3, and many more ways to centrally store states to be easily used with salt-ssh. This also allows for a distributed team to easily use a centralized source.

Saltfile Support

The new saltfile system makes it easy to have a user specific custom extended configuration.

Ext Pillar

Salt-ssh can now use the external pillar system. Making it easier then ever to use salt-ssh with teams.

No More sshpass

Thanks to the enhancements in the salt vt system, salt-ssh no longer requires sshpass to send passwords to ssh. This also makes the manipulation of ssh calls substantially more flexible, allowing for intercepting ssh calls in a much more fluid way.

Pure Python Shim

The salt-ssh call originally used a shell script to discover what version of python to execute with and determine the state of the ssh code deployment. This shell script has been replaced with a pure python version making it easy to increase the capability of the code deployment without causing platform inconsistency issues with different shell interpreters.

Custom Module Delivery

Custom modules are now seamlessly delivered. This makes the deployment of custom grains, states, execution modules and returners a seamless process.

CP Module Support

Salt-ssh now makes simple file transfers easier then ever! The *cp* module allows for files to be conveniently sent from the salt fileserver system down to systems.

More Thin Directory Options

Salt ssh functions by copying a subset of the salt code, or *salt thin* down to the target system. In the past this was always transferred to /tmp/.salt and cached there for subsequent commands.

Now, salt thin can be sent to a random directory and removed when the call is complete with the *-W* option. The new *-W* option still uses a static location but will clean up that location when finished.

The default *salt thin* location is now user defined, allowing multiple users to cleanly access the same systems.

State System Enhancements

New Imperative State Keyword ``Listen''

The new `listen` and `listen_in` keywords allow for completely imperative states by calling the `mod_watch()` routine after all states have run instead of re-ordering the states.

Mod Aggregate Runtime Manipulator

The new `mod_aggregate` system allows for the state system to rewrite the state data during execution. This allows for state definitions to be aggregated dynamically at runtime.

The best example is found in the `pkg` state. If `mod_aggregate` is turned on, then when the first pkg state is reached, the state system will scan all of the other running states for pkg states and take all other packages set for install and install them all at once in the first pkg state.

These runtime modifications make it easy to run groups of states together. In future versions, we hope to fill out the `mod_aggregate` system to build in more and more optimizations.

For more documentation on `mod_aggregate`, see *the documentation*.

New Requisites: onchanges and onfail

The new `onchanges` and `onchanges_in` requisites make a state apply only if there are changes in the required state. This is useful to execute post hooks after changes occur on a system.

The other new requisites, `onfail`, and `onfail_in`, allow for a state to run in reaction to the failure of another state.

For more information about these new requisites, see the *requisites documentation*.

Global onlyif and unless

The `onlyif` and `unless` options can now be used for any state declaration.

Use `names` to expand and override values

The *names declaration* in Salt's state system can now override or add values to the expanded data structure. For example:

```
my_users:
  user.present:
    - names:
      - larry
      - curly
      - moe:
        - shell: /bin/zsh
        - groups:
          - wheel
    - shell: /bin/bash
```

Major Features

Scheduler Additions

The Salt scheduler system has received MAJOR enhancements, allowing for cron-like scheduling and much more granular timing routines. See here for more info.

Red Hat 7 Family Support

All the needed additions have been made to run Salt on RHEL 7 and derived OSes like CentOS and Scientific.

Fileserver Backends in salt-call

Fileserver backends like gitfs can now be used without a salt master! Just add the fileserver backend configuration to the minion config and execute salt-call. This has been a much-requested feature and we are happy to finally bring it to our users.

Amazon Execution Modules

An entire family of execution modules further enhancing Salt's Amazon Cloud support. They include the following:

- Autoscale Groups (includes state support) -- related: Launch Control states
- Cloud Watch (includes state support)
- Elastic Cache (includes state support)
- Elastic Load Balancer (includes state support)
- IAM Identity and Access Management (includes state support)
- Route53 DNS (includes state support)
- Security Groups (includes state support)
- Simple Queue Service (includes state support)

LXC Runner Enhancements

BETA The Salt LXC management system has received a number of enhancements which make running an LXC cloud entirely from Salt an easy proposition.

Next Gen Docker Management

The Docker support in Salt has been increased at least ten fold. The Docker API is now completely exposed and Salt ships with Docker data tracking systems which make automating Docker deployments very easy.

Peer System Performance Improvements

The peer system communication routines have been refined to make the peer system substantially faster.

SDB

Encryption at rest for configs

GPG Renderer

Encrypted pillar at rest

OpenStack Expansion

Lots of new OpenStack stuff

Queues System

Ran change external queue systems into Salt events

Multi Master Failover Additions

Connecting to multiple masters is more dynamic then ever

Chef Execution Module

Managing Chef with Salt just got even easier!

salt-api Project Merge

The `salt-api` project has been merged into Salt core and is now available as part of the regular `salt-master` package install. No API changes were made, the **`salt-api`** script and init scripts remain intact.

`salt-api` has always provided Yet Another Pluggable Interface to Salt (TM) in the form of ``netapi'' modules. These are modules that bind to a port and start a service. Like many of Salt's other module types, netapi modules often have library and configuration dependencies. See the documentation for each module for instructions.

See also:

The full list of netapi modules.

Synchronous and Asynchronous Execution of Runner and Wheel Modules `salt.runner.RunnerClient` and `salt.wheel.WheelClient` have both gained complimentary `cmd_sync` and `cmd_async` methods allowing for synchronous and asynchronous execution of any Runner or Wheel module function, all protected using Salt's *external authentication* system. `salt-api` benefits from this addition as well.

`rest_cherrypy` Additions The `rest_cherrypy` netapi module provides the main REST API for Salt.

Web Hooks This release of course includes the Web Hook additions from the most recent `salt-api` release, which allows external services to signal actions within a Salt infrastructure. External services such as Amazon SNS, Travis-CI, or GitHub, as well as internal services that cannot or should not run a Salt minion daemon can be used as first-class components in Salt's rich orchestration capabilities.

The raw HTTP request body is now available in the event data. This is sometimes required information for checking an HMAC signature in order to verify a HTTP request. As an example, Amazon or GitHub requests are signed this way.

Generating and Accepting Minion Keys The `/key` convenience URL generates a public and private key for a minion, automatically pre-accepts the public key on the Salt Master, and returns both keys as a tarball for download.

This allows for easily bootstrapping the key on a new minion with a single HTTP call, such as with a Kickstart script, all using regular shell tools.

```
curl -sS http://salt-api.example.com:8000/keys \
        -d mid=jerry \
        -d username=kickstart \
        -d password=kickstart \
        -d eauth=pam \
        -o jerry-salt-keys.tar
```

Fileserver Backend Enhancements

All of the fileserver backends have been overhauled to be faster, lighter, and more reliable. The VCS backends (`gitfs`, `hgfs`, and `svnfs`) have also received a **lot** of new features.

Additionally, most config parameters for the VCS backends can now be configured on a per-remote basis, allowing for global config parameters to be overridden for a specific gitfs/hgfs/svnfs remote.

New `gitfs` Features

Pygit2 and Dulwich In addition to supporting GitPython, support for pygit2 (0.20.3 and newer) and dulwich have been added. Provided a compatible version of pygit2 is installed, it will now be the default provider. The config parameter `gitfs_provider` has been added to allow one to choose a specific provider for gitfs.

Mountpoints Prior to this release, to serve a file from gitfs at a salt fileserver URL of `salt://foo/bar/baz.txt`, it was necessary to ensure that the parent directories existed in the repository. A new config parameter `gitfs_mountpoint` allows gitfs remotes to be exposed starting at a user-defined `salt://` URL.

Environment Whitelisting/Blacklisting By default, gitfs will expose all branches and tags as Salt fileserver environments. Two new config parameters, `gitfs_env_whitelist`, and `gitfs_env_blacklist`, allow more control over which branches and tags are exposed. More detailed information on how these two options work can be found in the *Gitfs Walkthrough*.

Expanded Authentication Support As of pygit2 0.20.3, both http(s) and SSH key authentication are supported, and Salt now also supports both authentication methods when using pygit2. Keep in mind that pygit2 0.20.3 is not yet available on many platforms, so those who had been using authenticated git repositories with a passphraseless key should stick to GitPython if a new enough pygit2 is not yet available for the platform on which the master is running.

A full explanation of how to use authentication can be found in the *Gitfs Walkthrough*.

New `hgfs` Features

Mountpoints This feature works exactly like its *gitfs counterpart*. The new config parameter is called `hgfs_mountpoint`.

Environment Whitelisting/Blacklisting This feature works exactly like its *gitfs counterpart*. The new config parameters are called `hgfs_env_whitelist` and `hgfs_env_blacklist`.

New `svnfs` Features

Mountpoints This feature works exactly like its *gitfs counterpart*. The new config parameter is called `svnfs_mountpoint`.

Environment Whitelisting/Blacklisting This feature works exactly like its *gitfs counterpart*. The new config parameters are called `svnfs_env_whitelist` and `svnfs_env_blacklist`.

Configurable Trunk/Branches/Tags Paths Prior to this release, the paths where trunk, branches, and tags were located could only be in directores named ``trunk'', ``branches'', and ``tags'' directly under the root of the repository. Three new config parameters (`svnfs_trunk`, `svnfs_branches`, and `svnfs_tags`) allow SVN repositories which are laid out differently to be used with svnfs.

New `minionfs` Features

Mountpoint This feature works exactly like its *gitfs counterpart*. The new config parameter is called `minionfs_mountpoint`. The one major difference is that, as minionfs doesn't use multiple remotes (it just serves up files pushed to the master using `cp.push`) there is no such thing as a per-remote configuration for `minionfs_mountpoint`.

Changing the Saltenv from Which Files are Served A new config parameter (`minionfs_env`) allows minionfs files to be served from a Salt fileserver environment other than `base`.

Minion Whitelisting/Blacklisting By default, minionfs will expose the pushed files from all minions. Two new config parameters, `minionfs_whitelist`, and `minionfs_blacklist`, allow minionfs to be restricted to serve files from only the desired minions.

Pyobjects Renderer

Salt now ships with with the `Pyobjects Renderer` that allows for construction of States using pure Python with an idiomatic object interface.

New Modules

In addition to the Amazon modules mentioned above, there are also several other new execution modules:

- Oracle
- Random
- Redis
- Amazon Simple Queue Service
- Block Device Management
- CoreOS etcd
- Genesis
- InfluxDB
- Server Density
- Twilio Notifications
- Varnish
- ZNC IRC Bouncer
- SMTP

New Runners

- Map/Reduce Style
- Queue

New External Pillars

- CoreOS etcd

New Salt-Cloud Providers

- Aliyun ECS Cloud
- LXC Containers
- Proxmox (OpenVZ containers & KVM)

Salt Call Change

When used with a returner, salt-call now contacts a master if `--local` is not specicified.

Deprecations

salt.modules.virtualenv_mod

- Removed deprecated `memoize` function from `salt/utils/__init__.py` (deprecated)
- Removed deprecated `no_site_packages` argument from `create` function (deprecated)

- Removed deprecated `check_dns` argument from `minion_config` and `apply_minion_config` functions (deprecated)
- Removed deprecated `OutputOptionsWithTextMixIn` class from `salt/utils/parsers.py` (deprecated)
- Removed the following deprecated functions from `salt/modules/ps.py`: - `physical_memory_usage` (deprecated) - `virtual_memory_usage` (deprecated) - `cached_physical_memory` (deprecated) - `physical_memory_buffers` (deprecated)
- Removed deprecated cloud arguments from `cloud_config` function in `salt/config.py`: - `vm_config` (deprecated) - `vm_config_path` (deprecated)
- Removed deprecated `libcloud_version` function from `salt/cloud/libcloudfuncs.py` (deprecated)
- Removed deprecated `CloudConfigMixIn` class from `salt/utils/parsers.py` (deprecated)

26.2.2 Salt 2014.1.0 Release Notes - Codename Hydrogen

Note: Due to a change in master to minion communication, 2014.1.0 minions are not compatible with older-version masters. Please upgrade masters first. More info on backwards-compatibility policy *here*, under the ``Upgrading Salt'' subheading.

Note: A change in the grammar in the state compiler makes `module.run` in requisites illegal syntax. Its use is replaced simply with the word `module`. In other words you will need to change requisites like this:

```
require:
    module.run: some_module_name
```

to:

```
require:
    module: some_module_name
```

This is a breaking change. We apologize for the inconvenience, we needed to do this to remove some ambiguity in parsing requisites.

release 2014-02-24

The 2014.1.0 release of Salt is a major release which not only increases stability but also brings new capabilities in virtualization, cloud integration, and more. This release brings a great focus on the expansion of testing making roughly double the coverage in the Salt tests, and comes with many new features.

2014.1.0 is the first release to follow the new date-based release naming system. See the `version numbers` page for more details.

Major Features

Salt Cloud Merged into Salt

Salt Cloud is a tool for provisioning salted minions across various cloud providers. Prior to this release, Salt Cloud was a separate project but this marks its full integration with the Salt distribution. A Getting Started guide and additional documentation for Salt Cloud can be found *here*:

Google Compute Engine

Alongside Salt Cloud comes new support for the Google Compute Engine. Salt Stack can now deploy and control GCE virtual machines and the application stacks that they run.

For more information on Salt Stack and GCE, please see this blog post.

Documentation for Salt and GCE can be found *here*.

Salt Virt

Salt Virt is a cloud controller that supports virtual machine deployment, inspection, migration, and integration with many aspects of Salt.

Salt Virt has undergone a major overhaul with this release and now supports many more features and includes a number of critical improvements.

Docker Integration

Salt now ships with `states` and an `execution module` to manage Docker containers.

Substantial Testing Expansion

Salt continues to increase its unit/regression test coverage. This release includes over 300 new tests.

BSD Package Management

BSD package management has been entirely rewritten. FreeBSD 9 and older now default to using pkg_add, while FreeBSD 10 and newer will use pkgng. FreeBSD 9 can be forced to use pkgng, however, by specifying the following option in the minion config file:

```
providers:
  pkg: pkgng
```

In addition, support for installing software from the ports tree has been added. See the documentation for the ports `state` and `execution module` for more information.

Network Management for Debian/Ubuntu

Initial support for management of network interfaces on Debian-based distros has been added. See the documentation for the `network state` and the `debian_ip` for more information.

IPv6 Support for iptables State/Module

The iptables `state` and `module` now have IPv6 support. A new parameter `family` has been added to the states and execution functions, to distinguish between IPv4 and IPv6. The default value for this parameter is `ipv4`, specifying `ipv6` will use ip6tables to manage firewall rules.

GitFS Improvements

Several performance improvements have been made to the `Git fileserver backend`. Additionally, file states can now use any any SHA1 commit hash as a fileserver environment:

```
/etc/httpd/httpd.conf:
  file.managed:
    - source: salt://webserver/files/httpd.conf
    - saltenv: 45af879
```

This applies to the functions in the `cp module` as well:

```
salt '*' cp.get_file salt://readme.txt /tmp/readme.txt saltenv=45af879
```

MinionFS

This new fileserver backend allows files which have been pushed from the minion to the master (using `cp.push`) to be served up from the salt fileserver. The path for these files takes the following format:

```
salt://minion-id/path/to/file
```

`minion-id` is the id of the ``source'' minion, the one from which the files were pushed to the master. `/path/to/file` is the full path of the file.

The *MinionFS Walkthrough* contains a more thorough example of how to use this backend.

saltenv

To distinguish between fileserver environments and execution functions which deal with environment variables, fileserver environments are now specified using the `saltenv` parameter. `env` will continue to work, but is deprecated and will be removed in a future release.

Grains Caching

A caching layer has been added to the Grains system, which can help speed up minion startup. Disabled by default, it can be enabled by setting the minion config option `grains_cache`:

```
grains_cache: True

# Seconds before grains cache is considered to be stale.
grains_cache_expiration: 300
```

If set to `True`, the grains loader will read from/write to a msgpack-serialized file containing the grains data.

Additional command-line parameters have been added to salt-call, mainly for testing purposes:

- `--skip-grains` will completely bypass the grains loader when salt-call is invoked.
- `--refresh-grains-cache` will force the grains loader to bypass the grains cache and refresh the grains, writing a new grains cache file.

Improved Command Logging Control

When using the `cmd module`, either on the CLI or when developing Salt execution modules, a new keyword argument `output_loglevel` allows for greater control over how (or even if) the command and its output are logged. For example:

```
salt '*' cmd.run 'tail /var/log/messages' output_loglevel=debug
```

The package management modules (`apt`, `yumpkg`, etc.) have been updated to log the copious output generated from these commands at loglevel `debug`.

Note: To keep a command from being logged, `output_loglevel=quiet` can be used.

Prior to this release, this could be done using `quiet=True`. This argument is still supported, but will be removed in a future Salt release.

PagerDuty Support

Initial support for firing events via PagerDuty has been added. See the documentation for the `pagerduty` module.

Virtual Terminal

Sometimes the subprocess module is not good enough, and, in fact, not even `askpass` is. This virtual terminal is still in it's infant childhood, needs quite some love, and was originally created to replace `askpass`, but, while developing it, it immediately proved that it could do so much more. It's currently used by salt-cloud when bootstrapping salt on clouds which require the use of a password.

Proxy Minions

Initial basic support for Proxy Minions is in this release. Documentation can be found *here*.

Proxy minions are a developing feature in Salt that enables control of devices that cannot run a minion. Examples include network gear like switches and routers that run a proprietary OS but offer an API, or ``dumb'' devices that just don't have the horsepower or ability to handle a Python VM.

Proxy minions can be difficult to write, so a simple REST-based example proxy is included. A Python bottle-based webserver can be found at https://github.com/cro/salt-proxy-rest as an endpoint for this proxy.

This is an ALPHA-quality feature. There are a number of issues with it currently, mostly centering around process control, logging, and inability to work in a masterless configuration.

Additional Bugfixes (Release Candidate Period)

Below are many of the fixes that were implemented in salt during the release candidate phase.

- Fix mount.mounted leaving conflicting entries in fstab (issue 7079)
- Fix mysql returner serialization to use json (issue 9590)
- Fix `ZMQError: Operation cannot be accomplished in current state` errors (issue 6306)
- Rbenv and ruby improvements

- Fix quoting issues with mysql port (issue 9568)
- Update mount module/state to support multiple swap partitions (issue 9520)
- Fix `archive` state to work with `bsdtar`
- Clarify logs for minion ID caching
- Add numeric revision support to git state (issue 9718)
- Update `master_uri` with `master_ip` (issue 9694)
- Add comment to Debian `mod_repo` (issue 9923)
- Fix potential undefined loop variable in rabbitmq state (issue 8703)
- Fix for salt-virt runner to delete key on VM deletion
- Fix for `salt-run -d` to limit results to specific runner or function (issue 9975)
- Add tracebacks to jinja renderer when applicable (issue 10010)
- Fix parsing in monit module (issue 10041)
- Fix highstate output from syndic minions (issue 9732)
- Quiet logging when dealing with passwords/hashes (issue 10000)
- Fix for multiple remotes in git_pillar (issue 9932)
- Fix npm installed command (issue 10109)
- Add safeguards for utf8 errors in zcbuildout module
- Fix compound commands (issue 9746)
- Add systemd notification when master is started
- Many doc improvements

26.2.3 Salt 2014.1.1 Release Notes

release 2014-03-18

Version 2014.1.1 is a bugfix release for *2014.1.0*. The changes include:

- Various doc fixes, including up-to-date Salt Cloud installation documentation.
- Renamed state.sls runner to state.orchestrate, to reduce confusion with the `state.sls` execution function
- Fix various bugs in the `dig` module (issue 10367)
- Add retry for query on certain EC2 status codes (issue 10154)
- Fix various bugs in mongodb_user state module (issue 10430)
- Fix permissions on `~/.salt_token` (issue 10422)
- Add PyObjects support
- Fix launchctl module crash with missing files
- Fix `saltutil.find_job` for Windows (issue 10581)
- Fix OS detection for OpenSolaris (issue 10601)
- Fix broken salt-ssh key_deploy
- Add support for multiline cron comments (issue 10721)

- Fix timezone module for Arch (issue 10789)
- Fix symlink support for `file.recurse` (issue 10809)
- Fix multi-master bugs (issue 10732 and issue 10969)
- Fix file.patch to error when source file is unavailable (issue 10380)
- Fix pkg to handle packages set as `purge` in `pkg.installed` (issue 10719)
- Add `zmqversion` grain
- Fix highstate summary for masterless minions (issue 10945)
- Fix `saltutil.find_job` for 2014.1 masters talking to 0.17 minions (issue 11020)
- Fix `file.recurse` states with trailing slashes in source (issue 11002)
- Fix `pkg states` to allow `pkgname.x86_64` (issue 7306)
- Make `iptables states` set a default table for flush (issue 11037)
- Added iptables `--reject-with` after final iptables call in `iptables states` (issue:*10757*)
- Fix improper passing of "family" in `iptables states` (issue 10774)
- Fix traceback in `iptables.insert` states (issue 10988)
- Fix zombie processes (issue 10867 and others)
- Fix batch mode to obey `--return` settings (issue 9146)
- Fix localclient issue that was causing batch mode breakage (issue 11094, issue 10470, and others)
- Multiple salt-ssh fixes
- FreeBSD: look in /usr/local/etc/salt for configuration by default, if installed using `pip --editable`.
- Add a `skip_suggestions` parameter to pkg.installed states which allows pre-flight check to be skipped (issue 11106)
- Fixed tag-based gitfs fileserver environments regression (issue 10956)
- Yum: fix cache of available pkgs not cleared when repos are changed (issue 11001)
- Yum: fix for plugin-provided repositories (i.e. RHN/Spacewalk) (issue 11145)
- Fix regression in `chocolatey.bootstrap` (issue 10541)
- Fix fail on unknown target in `jobs runner` (issue 11151)
- Don't log errors for commands which are expected to sometimes exit with non-zero exit status (issue 11154, issue 11090)
- Fix `test=True` CLI override of config option (issue 10877)
- Log sysctl key listing at loglevel TRACE (issue 10931)

26.2.4 Salt 2014.1.10 Release Notes

release 2014-08-01

Note: Version 2014.1.9 contained a regression which caused inaccurate Salt version detection, and thus was never packaged for general release. This version contains the version detection fix, but is otherwise identical to 2014.1.9.

Version 2014.1.10 is another bugfix release for *2014.1.0*. Changes include:

- Ensure salt-ssh will not continue if permissions on a temporary directory are not correct.
- Use the bootstrap script distributed with Salt instead of relying on an external resource
- Remove unused testing code
- Ensure salt states are placed into the `.salt` directory in salt-ssh
- Use a randomized path for temporary files in a salt-cloud deployment
- Clean any stale directories to ensure a fresh copy of salt-ssh during a deployment

Salt 2014.1.10 fixes security issues documented by CVE-2014-3563: ``Insecure tmp-file creation in seed.py, salt-ssh, and salt-cloud.'' Upgrading is recommended.

26.2.5 Salt 2014.1.11 Release Notes

release 2014-08-29

Version 2014.1.11 is another bugfix release for *2014.1.0*. Changes include:

- Fix for minion_id with byte-order mark (BOM) (issue 12296)
- Fix `runas` deprecation in `at` module
- Fix trailing slash befhavior for `file.makedirs_` (issue 14019)
- Fix chocolatey path (issue 13870)
- Fix git_pillar infinite loop issues (issue 14671)
- Fix json outputter `null` case
- Fix for minion error if one of multiple masters are down (issue 14099)

26.2.6 Salt 2014.1.12 Release Notes

release 2014-10-08

Version 2014.1.12 is another bugfix release for *2014.1.0*. Changes include:

- Fix `scp_file` always failing (which broke salt-cloud) (issue 16437)
- Fix regression in pillar in masterless (issue 16210, issue 16416, issue 16428)

26.2.7 Salt 2014.1.13 Release Notes

release 2014-10-14

Version 2014.1.13 is another bugfix release for *2014.1.0*. Changes include:

- Fix `sftp_file` by checking the exit status code of scp (which broke salt-cloud) (issue 16599)

26.2.8 Salt 2014.1.2 Release Notes

release 2014-04-15

Version 2014.1.2 is another bugfix release for *2014.1.0*. The changes include:

- Fix username detection when su'ed to root on FreeBSD (issue 11628)

- Fix minionfs backend for file.recurse states
- Fix 32-bit packages of different arches than the CPU arch, on 32-bit RHEL/CentOS (issue 11822)
- Fix bug with specifying alternate home dir on user creation (FreeBSD) (issue 11790)
- Don't reload `site` module on module refresh for MacOS
- Fix regression with running execution functions in Pillar SLS (issue 11453)
- Fix some modules missing from Windows installer
- Don't log an error for yum commands that return nonzero exit status on non-failure (issue 11645)
- Fix bug in rabbitmq state (issue 8703)
- Fix missing ssh config options (issue 10604)
- Fix top.sls ordering (issue 10810 and issue 11691)
- Fix `salt-key --list all` (issue 10982)
- Fix win_servermanager install/remove function (issue 11038)
- Fix interaction with tokens when running commands as root (issue 11223)
- Fix overstate bug with `find_job` and `**kwargs` (issue 10503)
- Fix `saltenv` for `aptpkg.mod_repo` from `pkgrepo` state
- Fix environment issue causing file caching problems (issue 11189)
- Fix bug in `__parse_key` in registry state (issue 11408)
- Add minion auth retry on rejection (issue 10763)
- Fix publish_session updating the encryption key (issue 11493)
- Fix for bad `AssertionError` raised by GitPython (issue 11473)
- Fix `debian_ip` to allow disabling and enabling networking on Ubuntu (issue 11164)
- Fix potential memory leak caused by saved (and unused) events (issue 11582)
- Fix exception handling in the MySQL module (issue 11616)
- Fix environment-related error (issue 11534)
- Include `psutil` on Windows
- Add `file.replace` and `file.search` to Windows (issue 11471)
- Add additional `file` module helpers to Windows (issue 11235)
- Add `pid` to netstat output on Windows (issue 10782)
- Fix Windows not caching new versions of installers in winrepo (issue 10597)
- Fix hardcoded md5 hashing
- Fix kwargs in salt-ssh (issue 11609)
- Fix file backup timestamps (issue 11745)
- Fix stacktrace on `sys.doc` with invalid eauth (issue 11293)
- Fix `git.latest` with `test=True` (issue 11595)
- Fix `file.check_perms` hardcoded `follow_symlinks` (issue 11387)
- Fix certain `pkg` states for RHEL5/Cent5 machines (issue 11719)

26.2.9 Salt 2014.1.3 Release Notes

release 2014-04-15

Version 2014.1.3 is another bugfix release for *2014.1.0*. It was created as a hotfix for a regression found in 2014.1.2, which was not distributed. The only change made was as follows:

- Fix regression that caused `saltutil.find_job` to fail, causing premature terminations of salt CLI commands.

Changes in the not-distributed 2014.1.2, also included in 2014.1.3:

- Fix username detection when su'ed to root on FreeBSD (issue 11628)
- Fix minionfs backend for file.recurse states
- Fix 32-bit packages of different arches than the CPU arch, on 32-bit RHEL/CentOS (issue 11822)
- Fix bug with specifying alternate home dir on user creation (FreeBSD) (issue 11790)
- Don't reload `site` module on module refresh for MacOS
- Fix regression with running execution functions in Pillar SLS (issue 11453)
- Fix some modules missing from Windows installer
- Don't log an error for yum commands that return nonzero exit status on non-failure (issue 11645)
- Fix bug in rabbitmq state (issue 8703)
- Fix missing ssh config options (issue 10604)
- Fix top.sls ordering (issue 10810 and issue 11691)
- Fix `salt-key --list all` (issue 10982)
- Fix win_servermanager install/remove function (issue 11038)
- Fix interaction with tokens when running commands as root (issue 11223)
- Fix overstate bug with `find_job` and `**kwargs` (issue 10503)
- Fix `saltenv` for `aptpkg.mod_repo` from `pkgrepo` state
- Fix environment issue causing file caching problems (issue 11189)
- Fix bug in `__parse_key` in registry state (issue 11408)
- Add minion auth retry on rejection (issue 10763)
- Fix publish_session updating the encryption key (issue 11493)
- Fix for bad `AssertionError` raised by GitPython (issue 11473)
- Fix `debian_ip` to allow disabling and enabling networking on Ubuntu (issue 11164)
- Fix potential memory leak caused by saved (and unused) events (issue 11582)
- Fix exception handling in the MySQL module (issue 11616)
- Fix environment-related error (issue 11534)
- Include `psutil` on Windows
- Add `file.replace` and `file.search` to Windows (issue 11471)
- Add additional `file` module helpers to Windows (issue 11235)
- Add `pid` to netstat output on Windows (issue 10782)

- Fix Windows not caching new versions of installers in winrepo (issue 10597)
- Fix hardcoded md5 hashing
- Fix kwargs in salt-ssh (issue 11609)
- Fix file backup timestamps (issue 11745)
- Fix stacktrace on `sys.doc` with invalid eauth (issue 11293)
- Fix `git.latest` with `test=True` (issue 11595)
- Fix `file.check_perms` hardcoded `follow_symlinks` (issue 11387)
- Fix certain `pkg` states for RHEL5/Cent5 machines (issue 11719)

26.2.10 Salt 2014.1.4 Release Notes

release 2014-05-05

Version 2014.1.4 is another bugfix release for *2014.1.0*. Changes include:

- Fix setup.py dependency issue (issue 12031)
- Fix handling for IOErrors under certain circumstances (issue 11783 and issue 11853)
- Fix fatal exception when `/proc/1/cgroup` is not readable (issue 11619)
- Fix os grains for OpenSolaris (issue 11907)
- Fix `lvs.zero` module argument pass-through (issue 9001)
- Fix bug in `debian_ip` interaction with `network.system` state (issue 11164)
- Remove bad binary package verification code (issue 12177)
- Fix traceback in solaris package installation (issue 12237)
- Fix `file.directory` state symlink handling (issue 12209)
- Remove `external_ip` grain
- Fix `file.managed` makedirs issues (issue 10446)
- Fix hang on non-existent Windows drive letter for `file` module (issue 9880)
- Fix salt minion caching all users on the server (issue 9743)
- Add strftime formatting for `ps.boot_time` (issue 12428)

26.2.11 Salt 2014.1.5 Release Notes

release 2014-06-11

Version 2014.1.5 is another bugfix release for *2014.1.0*. Changes include:

- Add function for finding cached job on the minion
- Fix iptables save file location for Debian (issue 11730)
- Fix for minion caching jobs when master is down
- Bump default `syndic_wait` to 5 to fix syndic-related problems (issue 12262)
- Add OpenBSD, FreeBSD, and NetBSD support for `network.netstat` (issue 12121)
- Fix false positive error in logs for `makeconf` state (issue 9762)

- Fix for yum `fromrepo` package installs when repo is disabled by default (issue 12466)
- Fix for extra blank lines in `file.blockreplace` (issue 12422)
- Fix grain detection for OpenVZ guests (issue 11877)
- Fix `get_dns_servers` function for Windows `win_dns_client`
- Use system locale for ports package installations
- Use correct stop/restart procedure for Debian networking in `debian_ip` (issue 12614)
- Fix for `cmd_iter`/`cmd_iter_no_block` blocking issues (issue 12617)
- Fix traceback when syncing custom types (issue 12883)
- Fix cleaning directory symlinks in `file.directory`
- Add performance optimizations for `saltutil.sync_all` and `state.highstate`
- Fix possible error in `saltutil.running`
- Fix for kmod modules with dashes (issue 13239)
- Fix possible race condition for Windows minions in state module reloading (issue 12370)
- Fix bug with roster for `passwd` option that is loaded as a non-string object (issue 13249)
- Keep duplicate version numbers from showing up in `pkg.list_pkgs` output
- Fixes for Jinja renderer, timezone `module`/`state` (issue 12724)
- Fix timedatectl parsing for systemd>=210 (issue 12728)
- Fix `saltenv` being written to YUM repo config files (issue 12887)
- Removed the deprecated external nodes classifier (originally accessible by setting a value for external_nodes in the master configuration file). Note that this functionality has been marked deprecated for some time and was replaced by the more general *master tops* system.
- More robust escaping of ldap filter strings.
- Fix trailing slash in `gitfs_root` causing files not to be available (issue 13185)

26.2.12 Salt 2014.1.6 Release Notes

release 2014-07-08

Version 2014.1.6 is another bugfix release for *2014.1.0*. Changes include:

- Fix extra `iptables --help` output (Sorry!) (issue 13648, issue 13507, issue 13527, issue 13607)
- Fix `mount.active` for Solaris
- Fix support for `allow-hotplug` statement in debian_ip network module
- Add sqlite3 to esky builds
- Fix `jobs.active` output (issue 9526)
- Fix the `virtual` grain for Xen (issue 13534)
- Fix eauth for batch mode (issue 9605)
- Fix force-related issues with `tomcat` support (issue 12889)
- Fix KeyError when cloud mapping
- Fix salt-minion restart loop in Windows (issue 12086)

- Fix detection of `service` virtual module on Fedora minions
- Fix traceback with missing ipv4 grain (issue 13838)
- Fix issue in roots backend with invalid data in mtime_map (issue 13836)
- Fix traceback in `jobs.active` (issue 11151)
- Fix `master_tops` and `_ext_nodes` issue (issue 13535, issue 13673)

26.2.13 Salt 2014.1.7 Release Notes

release 2014-07-09

Version 2014.1.7 is another bugfix release for *2014.1.0*. Changes include:

- Fix batch mode regression (issue 14046)

This release was a hotfix release for the regression listed above which was present in the 2014.1.6 release. The changes included in 2014.1.6 are listed below:

- Fix extra `iptables --help` output (Sorry!) (issue 13648, issue 13507, issue 13527, issue 13607)
- Fix `mount.active` for Solaris
- Fix support for `allow-hotplug` statement in debian_ip network module
- Add sqlite3 to esky builds
- Fix `jobs.active` output (issue 9526)
- Fix the `virtual` grain for Xen (issue 13534)
- Fix eauth for batch mode (issue 9605)
- Fix force-related issues with `tomcat` support (issue 12889)
- Fix KeyError when cloud mapping
- Fix salt-minion restart loop in Windows (issue 12086)
- Fix detection of `service` virtual module on Fedora minions
- Fix traceback with missing ipv4 grain (issue 13838)
- Fix issue in roots backend with invalid data in mtime_map (issue 13836)
- Fix traceback in `jobs.active` (issue 11151)
- Fix `master_tops` and `_ext_nodes` issue (issue 13535, issue 13673)

26.2.14 Salt 2014.1.8 Release Notes

release 2014-07-30

Note: This release contained a regression which caused inaccurate Salt version detection, and thus was never packaged for general release. Please use version 2014.1.10 instead.

Version 2014.1.8 is another bugfix release for *2014.1.0*. Changes include:

- Ensure salt-ssh will not continue if permissions on a temporary directory are not correct.
- Use the bootstrap script distributed with Salt instead of relying on an external resource
- Remove unused testing code

- Ensure salt states are placed into the `.salt` directory in salt-ssh
- Use a randomized path for temporary files in a salt-cloud deployment
- Clean any stale directories to ensure a fresh copy of salt-ssh during a deployment

26.2.15 Salt 2014.1.9 Release Notes

release 2014-07-31

Note: This release contained a regression which caused inaccurate Salt version detection, and thus was never packaged for general release. Please use version 2014.1.10 instead.

Note: Version 2014.1.8 contained a regression which caused inaccurate Salt version detection, and thus was never packaged for general release. This version contains the version detection fix, but is otherwise identical to 2014.1.8.

Version 2014.1.9 is another bugfix release for *2014.1.0*. Changes include:

- Ensure salt-ssh will not continue if permissions on a temporary directory are not correct.
- Use the bootstrap script distributed with Salt instead of relying on an external resource
- Remove unused testing code
- Ensure salt states are placed into the `.salt` directory in salt-ssh
- Use a randomized path for temporary files in a salt-cloud deployment
- Clean any stale directories to ensure a fresh copy of salt-ssh during a deployment

26.2.16 Salt 0.10.0 Release Notes

release 2012-06-16

0.10.0 has arrived! This release comes with MANY bug fixes, and new capabilities which greatly enhance performance and reliability. This release is primarily a bug fix release with many new tests and many repaired bugs. This release also introduces a few new key features which were brought in primarily to repair bugs and some limitations found in some of the components of the original architecture.

Major Features

Event System

The Salt Master now comes equipped with a new event system. This event system has replaced some of the back end of the Salt client and offers the beginning of a system which will make plugging external applications into Salt. The event system relies on a local ZeroMQ publish socket and other processes can connect to this socket and listen for events. The new events can be easily managed via Salt's event library.

Unprivileged User Updates

Some enhancements have been added to Salt for running as a user other than root. These new additions should make switching the user that the Salt Master is running as very painless, simply change the `user` option in the master configuration and restart the master, Salt will take care of all of the particulars for you.

Peer Runner Execution

Salt has long had the peer communication system used to allow minions to send commands via the salt master. 0.10.0 adds a new capability here, now the master can be configured to allow for minions to execute Salt runners via the `peer_run` option in the salt master configuration.

YAML Parsing Updates

In the past the YAML parser for sls files would return the incorrect numbers when the file mode was set with a preceding 0. The YAML parser used in Salt has been modified to no longer convert these number into octal but to keep them as the correct value so that sls files can be a little cleaner to write.

State Call Data Files

It was requested that the minion keep a local cache of the most recent executed state run. This has been added and now with state runs the data is stored in a msgpack file in the minion's cachedir.

Turning Off the Job Cache

A new option has been added to the master configuration file. In previous releases the Salt client would look over the Salt job cache to read in the minion return data. With the addition of the event system the Salt client can now watch for events directly from the master worker processes.

This means that the job cache is no longer a hard requirement. Keep in mind though, that turning off the job cache means that historic job execution data cannot be retrieved.

Test Updates

Minion Swarms Are Faster

To continue our efforts with testing Salt's ability to scale the minionswarm script has been updated. The minionswarm can now start up minions much faster than it could before and comes with a new feature allowing modules to be disabled, thus lowering the minion's footprint when making a swarm. These new updates have allows us to test

```
# python minionswarm.py -m 20 --master salt-master
```

Many Fixes

To get a good idea for the number of bugfixes this release offers take a look at the closed tickets for 0.10.0, this is a very substantial update:

https://github.com/saltstack/salt/issues?milestone=12&state=closed

Master and Minion Stability Fixes

As Salt deployments grow new ways to break Salt are discovered. 0.10.0 comes with a number of fixes for the minions and master greatly improving Salt stability.

26.2.17 Salt 0.10.1 Release Notes

release 2012-06-19

26.2.18 Salt 0.10.2 Release Notes

release 2012-07-30

0.10.2 is out! This release comes with enhancements to the pillar interface, cleaner ways to access the salt-call capabilities in the API, minion data caching and the event system has been added to salt minions.

There have also been updates to the ZeroMQ functions, many more tests (thanks to sponsors, the code sprint and many contributors) and a swath of bug fixes.

Major Features

Ext Pillar Modules

The ranks of available Salt modules directories sees a new member in 0.10.2. With the popularity of pillar a higher demand has arisen for `ext_pillar` interfaces to be more like regular Salt module additions. Now ext_pillar interfaces can be added in the same way as other modules, just drop it into the pillar directory in the salt source.

Minion Events

In 0.10.0 an event system was added to the Salt master. 0.10.2 adds the event system to the minions as well. Now event can be published on a local minion as well.

The minions can also send events back up to the master. This means that Salt is able to communicate individual events from the minions back up to the Master which are not associated with command.

Minion Data Caching

When pillar was introduced the landscape for available data was greatly enhanced. The minion's began sending grain data back to the master on a regular basis.

The new config option on the master called `minion_data_cache` instructs the Salt master to maintain a cache of the minion's grains and pillar data in the cachedir. This option is turned off by default to avoid hitting the disk more, but when enabled the cache is used to make grain matching from the salt command more powerful, since the minions that will match can be predetermined.

Backup Files

By default all files replaced by the file.managed and file.recurse states we simply deleted. 0.10.2 adds a new option. By setting the backup option to `minion` the files are backed up before they are replaced.

The backed up files are located in the cachedir under the file_backup directory. On a default system this will be at: `/var/cache/salt/file_backup`

Configuration files

`salt-master` and `salt-minion` automatically load additional configuration files from `master.d/*.conf` respective `minion.d/*.conf` where `master.d`/`minion.d` is a directory in the same directory as the main configuration file.

Salt Key Verification

A number of users complained that they had inadvertently deleted the wrong salt authentication keys. 0.10.2 now displays what keys are going to be deleted and verifies that they are the keys that are intended for deletion.

Key auto-signing

If `autosign_file` is specified in the configuration file incoming keys will be compared to the list of keynames in `autosign_file`. Regular expressions as well as globbing is supported.

The file must only be writable by the user otherwise the file will be ignored. To relax the permission and allow group write access set the `permissive_pki_access` option.

Module changes

Improved OpenBSD support New modules for managing services and packages were provided by Joshua Elsasser to further improve the support for OpenBSD.

Existing modules like the *disk* module were also improved to support OpenBSD.

SQL Modules The MySQL and PostgreSQL modules have both received a number of additions thanks to the work of Avi Marcus and Roman Imankulov.

ZFS Support on FreeBSD A new ZFS module has been added by Kurtis Velarde for FreeBSD supporting various ZFS operations like creating, extending or removing zpools.

Augeas A new Augeas module by Ulrich Dangel for editing and verifying config files.

Native Debian Service module The support for the Debian was further improved with an new service module for Debian by Ahmad Khayyat supporting *disable* and *enable*.

Cassandra Cassandra support has been added by Adam Garside. Currently only status and diagnostic information are supported.

Networking The networking support for *RHEL* has been improved and supports bonding support as well as zero-conf configuration.

Monit Basic monit support by Kurtis Velarde to control services via monit.

nzbget Basic support for controlling nzbget by Joseph Hall

Bluetooth Baisc `bluez` support for managing and controlling Bluetooth devices. Supports scanning as well as pairing/unpairing by Joseph Hall.

Test Updates

Consistency Testing

Another testing script has been added. A bug was found in pillar when many minions generated pillar data at the same time. The new `consist.py` script is the tests directory was created to reproduce bugs where data should always be consistent.

Many Fixes

To get a good idea for the number of bugfixes this release offers take a look at the closed tickets for 0.10.2, this is a very substantial update:

https://github.com/saltstack/salt/issues?milestone=24&page=1&state=closed

Master and Minion Stability Fixes

As Salt deployments grow new ways to break Salt are discovered. 0.10.2 comes with a number of fixes for the minions and master greatly improving Salt stability.

26.2.19 Salt 0.10.3 Release Notes

release 2012-09-30

The latest taste of Salt has come, this release has many fixes and feature additions. Modifications have been made to make ZeroMQ connections more reliable, the beginning of the ACL system is in place, a new command line parsing system has been added, dynamic module distribution has become more environment aware, the new *master_finger* option and many more!

Major Features

ACL System

The new ACL system has been introduced. The ACL system allows for system users other than root to execute salt commands. Users can be allowed to execute specific commands in the same way that minions are opened up to the peer system.

The configuration value to open up the ACL system is called `client_acl` and is configured like so:

```
client_acl:
  fred:
    - test..*
    - pkg.list_pkgs
```

Where *fred* is allowed access to functions in the test module and to the `pkg.list_pkgs` function.

Master Finger Option

The *master_finger* option has been added to improve the security of minion provisioning. The *master_finger* option allows for the fingerprint of the master public key to be set in the configuration file to double verify that the master is valid. This option was added in response to a motivation to pre-authenticate the master when provisioning new minions to help prevent man in the middle attacks in some situations.

Salt Key Fingerprint Generation

The ability to generate fingerprints of keys used by Salt has been added to `salt-key`. The new option *finger* accepts the name of the key to generate and display a fingerprint for.

```
salt-key -F master
```

Will display the fingerprints for the master public and private keys.

Parsing System

Pedro Algavio, aka s0undt3ch, has added a substantial update to the command line parsing system that makes the help message output much cleaner and easier to search through. Salt parsers now have *--versions-report* besides usual *--version* info which you can provide when reporting any issues found.

Key Generation

We have reduced the requirements needed for *salt-key* to generate minion keys. You're no longer required to have salt configured and it's common directories created just to generate keys. This might prove useful if you're batch creating keys to pre-load on minions.

Startup States

A few configuration options have been added which allow for states to be run when the minion daemon starts. This can be a great advantage when deploying with Salt because the minion can apply states right when it first runs. To use startup states set the `startup_states` configuration option on the minion to *highstate*.

New Exclude Declaration

Some users have asked about adding the ability to ensure that other sls files or ids are excluded from a state run. The exclude statement will delete all of the data loaded from the specified sls file or will delete the specified id:

```
exclude:
  - sls: http
  - id: /etc/vimrc
```

Max Open Files

While we're currently unable to properly handle ZeroMQ's abort signals when the max open files is reached, due to the way that's handled on ZeroMQ's, we have minimized the chances of this happening without at least warning the user.

More State Output Options

Some major changes have been made to the state output system. In the past state return data was printed in a very verbose fashion and only states that failed or made changes were printed by default. Now two options can be passed to the master and minion configuration files to change the behavior of the state output. State output can be set to verbose (default) or non-verbose with the `state_verbose` option:

```
state_verbose: False
```

It is noteworthy that the state_verbose option used to be set to *False* by default but has been changed to *True* by default in 0.10.3 due to many requests for the change.

Te next option to be aware of new and called `state_output`. This option allows for the state output to be set to *full* (default) or *terse*.

The *full* output is the standard state output, but the new *terse* output will print only one line per state making the output much easier to follow when executing a large state system.

```
state_output: terse
```

state.file.append Improvements

The salt state *file.append()* tries *not* to append existing text. Previously the matching check was being made line by line. While this kind of check might be enough for most cases, if the text being appended was multi-line, the check would not work properly. This issue is now properly handled, the match is done as a whole ignoring any white space addition or removal except inside commas. For those thinking that, in order to properly match over multiple lines, salt will load the whole file into memory, that's not true. For most cases this is not important but an erroneous order to read a 4GB file, if not properly handled, like salt does, could make salt chew that amount of memory. Salt has a buffered file reader which will keep in memory a maximum of 256KB and iterates over the file in chunks of 32KB to test for the match, more than enough, if not, explain your usage on a ticket. With this change, also *salt.modules.file.contains()*, *salt.modules.file.contains_regex()*, *salt.modules.file.contains_glob()* and *salt.utils.find* now do the searching and/or matching using the buffered chunks approach explained above.

Two new keyword arguments were also added, *makedirs*, and *source*. The first, *makedirs* will create the necessary directories in order to append to the specified file, of course, it only applies if we're trying to append to a non-existing file on a non-existing directory:

```
/tmp/salttest/file-append-makedirs:
    file.append:
        text: foo
        makedirs: True
```

The second, *source*, allows one to append the contents of a file instead of specifying the text.

```
/tmp/salttest/file-append-source:

file.append:
    - source: salt://testfile
```

Security Fix

A timing vulnerability was uncovered in the code which decrypts the AES messages sent over the network. This has been fixed and upgrading is strongly recommended.

26.2.20 Salt 0.10.4 Release Notes

release 2012-10-23

Salt 0.10.4 is a monumental release for the Salt team, with two new module systems, many additions to allow granular access to Salt, improved platform support and much more.

This release is also exciting because we have been able to shorten the release cycle back to under a month. We are working hard to keep up the aggressive pace and look forward to having releases happen more frequently!

This release also includes a serious security fix and all users are very strongly recommended to upgrade. As usual, upgrade the master first, and then the minion to ensure that the process is smooth.

Major Features

External Authentication System

The new external authentication system allows for Salt to pass through authentication to any authentication system to determine if a user has permission to execute a Salt command. The Unix PAM system is the first supported system with more to come!

The external authentication system allows for specific users to be granted access to execute specific functions on specific minions. Access is configured in the master configuration file, and uses the new access control system:

```
external_auth:
  pam:
    thatch:
      - 'web*':
        - test.*
        - network.*
```

The configuration above allows the user *thatch* to execute functions in the test and network modules on minions that match the web* target.

Access Control System

All Salt systems can now be configured to grant access to non-administrative users in a granular way. The old configuration continues to work. Specific functions can be opened up to specific minions from specific users in the case of external auth and client ACLs, and for specific minions in the case of the peer system.

Access controls are configured like this:

```
client_acl:
  fred:
    - web\*:
      - pkg.list_pkgs
      - test.*
      - apache.*
```

Target by Network

A new matcher has been added to the system which allows for minions to be targeted by network. This new matcher can be called with the *-S* flag on the command line and is available in all places that the matcher system is available. Using it is simple:

```
$ salt -S '192.168.1.0/24' test.ping
$ salt -S '192.168.1.100' test.ping
```

Nodegroup Nesting

Previously a nodegroup was limited by not being able to include another nodegroup, this restraint has been lifted and now nodegroups will be expanded within other nodegroups with the *N@* classifier.

Salt Key Delete by Glob

The ability to delete minion keys by glob has been added to `salt-key`. To delete all minion keys whose minion name starts with `web':

```
$ salt-key -d 'web*'
```

Master Tops System

The *external_nodes* system has been upgraded to allow for modular subsystems to be used to generate the top file data for a highstate run.

The *external_nodes* option still works but will be deprecated in the future in favor of the new *master_tops* option.

Example of using *master_tops*:

```
master_tops:
  ext_nodes: cobbler-external-nodes
```

Next Level Solaris Support

A lot of work has been put into improved Solaris support by Romeo Theriault. Packaging modules (pkgadd/pkgrm and pkgutil) and states, cron support and user and group management have all been added and improved upon. These additions along with SMF (Service Management Facility) service support and improved Solaris grain detection in 0.10.3 add up to Salt becoming a great tool to manage Solaris servers with.

Security

A vulnerability in the security handshake was found and has been repaired, old minions should be able to connect to a new master, so as usual, the master should be updated first and then the minions.

Pillar Updates

The pillar communication has been updated to add some extra levels of verification so that the intended minion is the only one allowed to gather the data. Once all minions and the master are updated to salt 0.10.4 please activate pillar *2* by changing the *pillar_version* in the master config to *2*. This will be set to *2* by default in a future release.

26.2.21 Salt 0.10.5 Release Notes

release 2012-11-15

Salt 0.10.5 is ready, and comes with some great new features. A few more interfaces have been modularized, like the outputter system. The job cache system has been made more powerful and can now store and retrieve jobs archived in external databases. The returner system has been extended to allow minions to easily retrieve data from a returner interface.

As usual, this is an exciting release, with many noteworthy additions!

Major Features

External Job Cache

The external job cache is a system which allows for a returner interface to also act as a job cache. This system is intended to allow users to store job information in a central location for longer periods of time and to make the act of looking up information from jobs executed on other minions easier.

Currently the external job cache is supported via the mongo and redis returners:

```
ext_job_cache: redis
redis.host: salt
```

Once the external job cache is turned on the new *ret* module can be used on the minions to retrieve return information from the job cache. This can be a great way for minions to respond and react to other minions.

OpenStack Additions

OpenStack integration with Salt has been moving forward at a blistering pace. The new *nova*, *glance*, and *keystone* modules represent the beginning of ongoing OpenStack integration.

The Salt team has had many conversations with core OpenStack developers and is working on linking to OpenStack in powerful new ways.

Wheel System

A new API was added to the Salt Master which allows the master to be managed via an external API. This new system allows Salt API to easily hook into the Salt Master and manage configs, modify the state tree, manage the pillar and more. The main motivation for the wheel system is to enable features needed in the upcoming web UI so users can manage the master just as easily as they manage minions.

The wheel system has also been hooked into the external auth system. This allows specific users to have granular access to manage components of the Salt Master.

Render Pipes

Jack Kuan has added a substantial new feature. The render pipes system allows Salt to treat the render system like unix pipes. This new system enables sls files to be passed through specific render engines. While the default renderer is still recommended, different engines can now be more easily merged. So to pipe the output of Mako used in YAML use this shebang line:

#!mako|yaml

Salt Key Overhaul

The Salt Key system was originally developed as only a CLI interface, but as time went on it was pressed into becoming a clumsy API. This release marks a complete overhaul of Salt Key. Salt Key has been rewritten to function purely from an API and to use the outputter system. The benefit here is that the outputter system works much more cleanly with Salt Key now, and the internals of Salt Key can be used much more cleanly.

Modular Outputters

The outputter system is now loaded in a modular way. This means that output systems can be more easily added by dropping a python file down on the master that contains the function *output*.

Gzip from Fileserver

Gzip compression has been added as an option to the cp.get_file and cp.get_dir commands. This will make file transfers more efficient and faster, especially over slower network links.

Unified Module Configuration

In past releases of Salt, the minions needed to be configured for certain modules to function. This was difficult because it required pre-configuring the minions. 0.10.5 changes this by making all module configs on minions search the master config file for values.

Now if a single database server is needed, then it can be defined in the master config and all minions will become aware of the configuration value.

Salt Call Enhancements

The `salt-call` command has been updated in a few ways. Now, `salt-call` can take the --return option to send the data to a returner. Also, `salt-call` now reports executions in the minion proc system, this allows the master to be aware of the operation salt-call is running.

Death to pub_refresh and sub_timeout

The old configuration values *pub_refresh* and *sub_timeout* have been removed. These options were in place to alleviate problems found in earlier versions of ZeroMQ which have since been fixed. The continued use of these options has proven to cause problems with message passing and have been completely removed.

Git Revision Versions

When running Salt directly from git (for testing or development, of course) it has been difficult to know exactly what code is being executed. The new versioning system will detect the git revision when building and how many commits have been made since the last release. A release from git will look like this:

0.10.4-736-gec74d69

Svn Module Addition

Anthony Cornehl (twinshadow) contributed a module that adds Subversion support to Salt. This great addition helps round out Salt's VCS support.

Noteworthy Changes

Arch Linux Defaults to Systemd

Arch Linux recently changed to use systemd by default and discontinued support for init scripts. Salt has followed suit and defaults to systemd now for managing services in Arch.

Salt, Salt Cloud and Openstack

With the releases of Salt 0.10.5 and Salt Cloud 0.8.2, OpenStack becomes the first (non-OS) piece of software to include support both on the user level (with Salt Cloud) and the admin level (with Salt). We are excited to continue to extend support of other platforms at this level.

26.2.22 Salt 0.11.0 Release Notes

release 2012-12-14

Salt 0.11.0 is here, with some highly sought after and exciting features. These features include the new overstate system, the reactor system, a new state run scope component called __context__, the beginning of the search system (still needs a great deal of work), multiple package states, the MySQL returner and a better system to arbitrarily reference outputters.

It is also noteworthy that we are changing how we mark release numbers. For the life of the project we have been pushing every release with features and fixes as point releases. We will now be releasing point releases for only bug fixes on a more regular basis and major feature releases on a slightly less regular basis. This means that the next release will be a bugfix only release with a version number of 0.11.1. The next feature release will be named 0.12.0 and will mark the end of life for the 0.11 series.

Major Features

OverState

The overstate system is a simple way to manage rolling state executions across many minions. The overstate allows for a state to depend on the successful completion of another state.

Reactor System

The new reactor system allows for a reactive logic engine to be created which can respond to events within a salted environment. The reactor system uses sls files to match events fired on the master with actions, enabling Salt to react to problems in an infrastructure.

Your load-balanced group of webservers is under extra load? Spin up a new VM and add it to the group. Your fileserver is filling up? Send a notification to your sysadmin on call. The possibilities are endless!

Module Context

A new component has been added to the module loader system. The module context is a data structure that can hold objects for a given scope within the module.

This allows for components that are initialized to be stored in a persistent context which can greatly speed up ongoing connections. Right now the best example can be found in the *cp* execution module.

Multiple Package Management

A long desired feature has been added to package management. By definition Salt States have always installed packages one at a time. On most platforms this is not the fastest way to install packages. Erik Johnson, aka terminalmage, has modified the package modules for many providers and added new capabilities to install groups of packages. These package groups can be defined as a list of packages available in repository servers:

```
python_pkgs:
  pkg.installed:
    - pkgs:
      - python-mako
      - whoosh
      - python-git
```

or specify based on the location of specific packages:

```
python_pkgs:
  pkg.installed:
    - sources:
      - python-mako: http://some-rpms.org/python-mako.rpm
      - whoosh: salt://whoosh/whoosh.rpm
      - python-git: ftp://companyserver.net/python-git.rpm
```

Search System

The bones to the search system have been added. This is a very basic interface that allows for search backends to be added as search modules. The first supported search module is the whoosh search backend. Right now only the basic paths for the search system are in place, making this very experimental. Further development will involve improving the search routines and index routines for whoosh and other search backends.

The search system has been made to allow for searching through all of the state and pillar files, configuration files and all return data from minion executions.

Notable Changes

All previous versions of Salt have shared many directories between the master and minion. The default locations for keys, cached data and sockets has been shared by master and minion. This has created serious problems with running a master and a minion on the same systems. 0.11.0 changes the defaults to be separate directories. Salt will also attempt to migrate all of the old key data into the correct new directories, but if it is not successful it may need to be done manually. If your keys exhibit issues after updating make sure that they have been moved from `/etc/salt/pki` to `/etc/salt/pki/{master,minion}`.

The old setup will look like this:

```
/etc/salt/pki
|-- master.pem
|-- master.pub
|-- minions
|   `-- ragnarok.saltstack.net
|-- minions_pre
|-- minion.pem
|-- minion.pub
|-- minion_master.pub
|-- minions_pre
`-- minions_rejected
```

With the accepted minion keys in `/etc/salt/pki/minions`, the new setup places the accepted minion keys in `/etc/salt/pki/master/minions`.

```
/etc/salt/pki
|-- master
|   |-- master.pem
|   |-- master.pub
|   |-- minions
|   |   `-- ragnarok.saltstack.net
|   |-- minions_pre
|   `-- minions_rejected
|-- minion
|   |-- minion.pem
|   |-- minion.pub
|   `-- minion_master.pub
```

26.2.23 Salt 0.11.1 Release Notes

release 2012-12-19

26.2.24 Salt 0.12.0 Release Notes

release 2013-01-15

Another feature release of Salt is here! Some exciting additions are included with more ways to make salt modular and even easier management of the salt file server.

Major Features

Modular Fileserver Backend

The new modular fileserver backend allows for any external system to be used as a salt file server. The main benefit here is that it is now possible to tell the master to directly use a git remote location, or many git remote locations, automatically mapping git branches and tags to salt environments.

Windows is First Class!

A new Salt Windows installer is now available! Much work has been put in to improve Windows support. With this much easier method of getting Salt on your Windows machines, we hope even more development and progress will occur. Please file bug reports on the Salt GitHub repo issue tracker so we can continue improving.

One thing that is missing on Windows that Salt uses extensively is a software package manager and a software package repository. The Salt pkg state allows sys admins to install software across their infrastructure and across operating systems. Software on Windows can now be managed in the same way. The SaltStack team built a package manager that interfaces with the standard Salt pkg module to allow for installing and removing software on Windows. In addition, a software package repository has been built on top of the Salt fileserver. A small YAML file provides the information necessary for the package manager to install and remove software.

An interesting feature of the new Salt Windows software package repository is that one or more remote git repositories can supplement the master's local repository. The repository can point to software on the master's fileserver or on an HTTP, HTTPS, or ftp server.

New Default Outputter

Salt displays data to the terminal via the outputter system. For a long time the default outputter for Salt has been the python pretty print library. While this has been a generally reasonable outputter, it did have many failings. The new default outputter is called ``nested'', it recursively scans return data structures and prints them out cleanly.

If the result of the new nested outputter is not desired any other outputter can be used via the --out option, or the output option can be set in the master and minion configs to change the default outputter.

Internal Scheduler

The internal Salt scheduler is a new capability which allows for functions to be executed at given intervals on the minion, and for runners to be executed at given intervals on the master. The scheduler allows for sequences such as executing state runs (locally on the minion or remotely via an overstate) or continually gathering system data to be run at given intervals.

The configuration is simple, add the schedule option to the master or minion config and specify jobs to run, this in the master config will execute the state.over runner every 60 minutes:

```
schedule:
  overstate:
    function: state.over
    minutes: 60
```

This example for the minion configuration will execute a highstate every 30 minutes:

```
schedule:
  highstate:
    function: state.highstate
    minutes: 30
```

Optional DSL for SLS Formulas

Jack Kuan, our renderer expert, has created something that is astonishing. Salt, now comes with an optional Python based DSL, this is a very powerful interface that makes writing SLS files in pure python easier than it was with the raw py renderer. As usual this can be used with the renderer shebang line, so a single sls can be written with the DSL if pure python power is needed while keeping other sls files simple with YAML.

Set Grains Remotely

A new execution function and state module have been added that allows for grains to be set on the minion. Now grains can be set via a remote execution or via states. Use the *grains.present* state or the *grains.setval* execution functions.

Gentoo Additions

Major additions to Gentoo specific components have been made. The encompasses executions modules and states ranging from supporting the make.conf file to tools like layman.

26.2.25 Salt 0.12.1 Release Notes

release 2013-01-21

26.2.26 Salt 0.13.0 Release Notes

release 2013-02-12

The lucky number 13 has turned the corner! From CLI notifications when quitting a salt command, to substantial improvements on Windows, Salt 0.13.0 has arrived!

Major Features

Improved file.recurse Performance

The file.recurse system has been deployed and used in a vast array of situations. Fixes to the file state and module have led towards opening up new ways of running file.recurse to make it faster. Now the file.recurse state will download fewer files and will run substantially faster.

Windows Improvements

Minion stability on Windows has improved. Many file operations, including file.recurse, have been fixed and improved. The network module works better, to include network.interfaces. Both 32bit and 64bit installers are now available.

Nodegroup Targeting in Peer System

In the past, nodegroups were not available for targeting via the peer system. This has been fixed, allowing the new nodegroup expr_form argument for the publish.publish function:

```
salt-call publish.publish group1 test.ping expr_form=nodegroup
```

Blacklist Additions

Additions allowing more granular blacklisting are available in 0.13.0. The ability to blacklist users and functions in client_acl have been added, as well as the ability to exclude state formulas from the command line.

Command Line Pillar Embedding

Pillar data can now be embedded on the command line when calling `state.sls` and `state.highstate`. This allows for on the fly changes or settings to pillar and makes parameterizing state formulas even easier. This is done via the keyword argument:

```
salt '*' state.highstate pillar='{"cheese": "spam"}'
```

The above example will extend the existing pillar to hold the `cheese` key with a value of `spam`. If the `cheese` key is already specified in the minion's pillar then it will be overwritten.

CLI Notifications

In the past hitting ctrl-C and quitting from the `salt` command would just drop to a shell prompt, this caused confusion with users who expected the remote executions to also quit. Now a message is displayed showing what command can be used to track the execution and what the job id is for the execution.

Version Specification in Multiple-Package States

Versions can now be specified within multiple-package `pkg.installed` states. An example can be found below:

```
mypkgs:
  pkg.installed:
    - pkgs:
      - foo
      - bar: 1.2.3-4
      - baz
```

Noteworthy Changes

The configuration subsystem in Salt has been overhauled to make the `opts` dict used by Salt applications more portable, the problem is that this is an incompatible change with salt-cloud, and salt-cloud will need to be updated to the latest git to work with Salt 0.13.0. Salt Cloud 0.8.5 will also require Salt 0.13.0 or later to function.

The SaltStack team is sorry for the inconvenience here, we work hard to make sure these sorts of things do not happen, but sometimes hard changes get in.

26.2.27 Salt 0.13.1 Release Notes

release 2013-02-15

26.2.28 Salt 0.13.2 Release Notes

release 2013-03-13

26.2.29 Salt 0.13.3 Release Notes

release 2013-03-18

26.2.30 Salt 0.14.0 Release Notes

release 2013-03-23

Salt 0.14.0 is here! This release was held up primarily by PyCon, Scale, and illness, but has arrived! 0.14.0 comes with many new features and is breaking ground for Salt in the area of cloud management with the introduction of Salt providing basic cloud controller functionality.

Major Features

Salt - As a Cloud Controller

This is the first primitive inroad to using Salt as a cloud controller is available in 0.14.0. Be advised that this is alpha, only tested in a few very small environments.

The cloud controller is built using kvm and libvirt for the hypervisors. Hypervisors are autodetected as minions and only need to have libvirt running and kvm installed to function. The features of the Salt cloud controller are as follows:

- Basic vm discovery and reporting
- Creation of new virtual machines
- Seeding virtual machines with Salt via qemu-nbd or libguestfs
- Live migration (shared and non shared storage)
- Delete existing VMs

It is noteworthy that this feature is still Alpha, meaning that all rights are reserved to change the interface if needs be in future releases!

Libvirt State

One of the problems with libvirt is management of certificates needed for live migration and cross communication between hypervisors. The new `libvirt` state makes the Salt Master hold a CA and manage the signing and distribution of keys onto hypervisors, just add a call to the libvirt state in the sls formulas used to set up a hypervisor:

```
libvirt_keys:
  libvirt.keys
```

New get Functions

An easier way to manage data has been introduced. The pillar, grains, and config execution modules have been extended with the new `get` function. This function works much in the same way as the get method in a python dict, but with an enhancement, nested dict components can be extracted using a `:` delimiter.

If a structure like this is in pillar:

```
foo:
  bar:
    baz: quo
```

Extracting it from the raw pillar in an sls formula or file template is done this way:

```
{{ pillar['foo']['bar']['baz'] }}
```

Now with the new get function the data can be safely gathered and a default can be set allowing the template to fall back if the value is not available:

```
{{ salt['pillar.get']('foo:bar:baz', 'qux') }}
```

This makes handling nested structures much easier, and defaults can be cleanly set. This new function is being used extensively in the new formulae repository of salt sls formulas.

26.2.31 Salt 0.14.1 Release Notes

release 2013-04-13

26.2.32 Salt 0.15.0 Release Notes

release 2013-05-03

The many new features of Salt 0.15.0 have arrived! Salt 0.15.0 comes with many smaller features and a few larger ones.

These features range from better debugging tools to the new Salt Mine system.

Major Features

The Salt Mine

First there was the peer system, allowing for commands to be executed from a minion to other minions to gather data live. Then there was the external job cache for storing and accessing long term data. Now the middle ground is being filled in with the Salt Mine. The Salt Mine is a system used to execute functions on a regular basis on minions and then store only the most recent data from the functions on the master, then the data is looked up via targets.

The mine caches data that is public to all minions, so when a minion posts data to the mine all other minions can see it.

IPV6 Support

0.13.0 saw the addition of initial IPV6 support but errors were encountered and it needed to be stripped out. This time the code covers more cases and must be explicitly enabled. But the support is much more extensive than before.

Copy Files From Minions to the Master

Minions have long been able to copy files down from the master file server, but until now files could not be easily copied from the minion up to the master.

A new function called `cp.push` can push files from the minions up to the master server. The uploaded files are then cached on the master in the master cachedir for each minion.

Better Template Debugging

Template errors have long been a burden when writing states and pillar. 0.15.0 will now send the compiled template data to the debug log, this makes tracking down the intermittent stage templates much easier. So running state.sls or state.highstate with *-l debug* will now print out the rendered templates in the debug information.

State Event Firing

The state system is now more closely tied to the master's event bus. Now when a state fails the failure will be fired on the master event bus so that the reactor can respond to it.

Major Syndic Updates

The Syndic system has been basically re-written. Now it runs in a completely asynchronous way and functions primarily as an event broker. This means that the events fired on the syndic are now pushed up to the higher level master instead of the old method used which waited for the client libraries to return.

This makes the syndic much more accurate and powerful, it also means that all events fired on the syndic master make it up the pipe as well making a reactor on the higher level master able to react to minions further downstream.

Peer System Updates

The Peer System has been updated to run using the client libraries instead of firing directly over the publish bus. This makes the peer system much more consistent and reliable.

Minion Key Revocation

In the past when a minion was decommissioned the key needed to be manually deleted on the master, but now a function on the minion can be used to revoke the calling minion's key:

```
$ salt-call saltutil.revoke_auth
```

Function Return Codes

Functions can now be assigned numeric return codes to determine if the function executed successfully. While not all functions have been given return codes, many have and it is an ongoing effort to fill out all functions that might return a non-zero return code.

Functions in Overstate

The overstate system was originally created to just manage the execution of states, but with the addition of return codes to functions, requisite logic can now be used with respect to the overstate. This means that an overstate stage can now run single functions instead of just state executions.

Pillar Error Reporting

Previously if errors surfaced in pillar, then the pillar would consist of only an empty dict. Now all data that was successfully rendered stays in pillar and the render error is also made available. If errors are found in the pillar, states will refuse to run.

Using Cached State Data

Sometimes states are executed purely to maintain a specific state rather than to update states with new configs. This is grounds for the new cached state system. By adding *cache=True* to a state call the state will not be generated fresh from the master but the last state data to be generated will be used. If no previous state data is available then fresh data will be generated.

Monitoring States

The new monitoring states system has been started. This is very young but allows for states to be used to configure monitoring routines. So far only one monitoring state is available, the `disk.status` state. As more capabilities are added to Salt UI the monitoring capabilities of Salt will continue to be expanded.

26.2.33 Salt 0.15.1 Release Notes

release 2013-05-08

The 0.15.1 release has been posted, this release includes fixes to a number of bugs in 0.15.1 and a three security patches.

Security Updates

A number of security issues have been resolved via the 0.15.1 release.

Path Injection in Minion IDs

Salt masters did not properly validate the id of a connecting minion. This can lead to an attacker uploading files to the master in arbitrary locations. In particular this can be used to bypass the manual validation of new unknown minions. Exploiting this vulnerability does not require authentication.

This issue affects all known versions of Salt.

This issue was reported by Ronald Volgers.

Patch The issue is fixed in Salt 0.15.1. Updated packages are available in the usual locations.

Specific commits:

https://github.com/saltstack/salt/commit/5427b9438e452a5a8910d9128c6aafb45d8fd5d3

https://github.com/saltstack/salt/commit/7560908ee62351769c3cd43b03d74c1ca772cc52

https://github.com/saltstack/salt/commit/e200b8a7ff53780124e08d2bdefde7587e52bfca

RSA Key Generation Fault

RSA key generation was done incorrectly, leading to very insecure keys. It is recommended to regenerate all RSA keys.

This issue can be used to impersonate Salt masters or minions, or decrypt any transferred data.

This issue can only be exploited by attackers who are able to observe or modify traffic between Salt minions and the legitimate Salt master.

A tool was included in 0.15.1 to assist in mass key regeneration, the manage.regen_keys runner.

This issue affects all known versions of Salt.

This issue was reported by Ronald Volgers.

Patch The issue is fixed in Salt 0.15.1. Updated packages are available in the usual locations.

Specific commits:

https://github.com/saltstack/salt/commit/5dd304276ba5745ec21fc1e6686a0b28da29e6fc

Command Injection Via ext_pillar

Arbitrary shell commands could be executed on the master by an authenticated minion through options passed when requesting a pillar.

Ext pillar options have been restricted to only allow safe external pillars to be called when prompted by the minion.

This issue affects Salt versions from 0.14.0 to 0.15.0.

This issue was reported by Ronald Volgers.

Patch The issue is fixed in Salt 0.15.1. Updated packages are available in the usual locations.

Specific commits:

https://github.com/saltstack/salt/commit/43d8c16bd26159d827d1a945c83ac28159ec5865

26.2.34 Salt 0.15.2 Release Notes

release 2013-05-29

26.2.35 Salt 0.15.3 Release Notes

release 2013-06-01

26.2.36 Salt 0.16.0 Release Notes

release 2013-07-01

The 0.16.0 release is an exciting one, with new features in master redundancy, and a new, powerful requisite.

Major Features

Multi-Master

This new capability allows for a minion to be actively connected to multiple salt masters at the same time. This allows for multiple masters to send out commands to minions and for minions to automatically reconnect to masters that have gone down. A tutorial is available to help get started here:

Multi Master Tutorial

Prereq, the New Requisite

The new *prereq* requisite is very powerful! It allows for states to execute based on a state that is expected to make changes in the future. This allows for a change on the system to be preempted by another execution. A good example is needing to shut down a service before modifying files associated with it, allowing, for instance, a webserver to be shut down allowing a load balancer to stop sending requests while server side code is updated. In this case, the prereq will only run if changes are expected to happen in the prerequired state, and the prerequired state will always run after the prereq state and only if the prereq state succeeds.

Peer System Improvements

The peer system has been revamped to make it more reliable, faster, and like the rest of Salt, async. The peer calls when an updated minion and master are used together will be much faster!

Relative Includes

The ability to include an sls relative to the defined sls has been added, the new syntax id documented here:

Includes

More State Output Options

The `state_output` option in the past only supported *full* and *terse*, 0.16.0 add the *mixed* and *changes* modes further refining how states are sent to users' eyes.

Improved Windows Support

Support for Salt on Windows continues to improve. Software management on Windows has become more seamless with Linux/UNIX/BSD software management. Installed software is now recognized by the short names defined in the *repository SLS*. This makes it possible to run `salt '*' pkg.version firefox` and get back results from Windows and non-Windows minions alike.

When templating files on Windows, Salt will now correctly use Windows appropriate line endings. This makes it much easier to edit and consume files on Windows.

When using the cmd state the `shell` option now allows for specifying Windows Powershell as an alternate shell to execute cmd.run and cmd.script. This opens up Salt to all the power of Windows Powershell and its advanced Windows management capabilities.

Several fixes and optimizations were added for the Windows networking modules, especially when working with IPv6.

A system module was added that makes it easy to restart and shutdown Windows minions.

The Salt Minion will now look for its config file in `c:\salt\conf` by default. This means that it's no longer necessary to specify the `-c` option to specify the location of the config file when starting the Salt Minion on Windows in a terminal.

Muliple Targets for pkg.removed, pkg.purged States

Both `pkg.removed` and `pkg.purged` now support the `pkgs` argument, which allow for multiple packages to be targeted in a single state. This, as in `pkg.installed`, helps speed up these states by reducing the number of times that the package management tools (apt, yum, etc.) need to be run.

Random Times in Cron States

The temporal parameters in `cron.present` states (minute, hour, etc.) can now be randomized by using `random` instead of a specific value. For example, by using the `random` keyword in the `minute` parameter of a cron state, the same cron job can be pushed to hundreds or thousands of hosts, and they would each use a randomly-generated minute. This can be helpful when the cron job accesses a network resource, and it is not desirable for all hosts to run the job concurrently.

```
/path/to/cron/script:
  cron.present:
    - user: root
    - minute: random
    - hour: 2
```

Since Salt assumes a value of `*` for unspecified temporal parameters, adding a parameter to the state and setting it to `random` will change that value from `*` to a randomized numeric value. However, if that field in the cron entry on the minion already contains a numeric value, then using the `random` keyword will not modify it.

Confirmation Prompt on Key Acceptance

When accepting new keys with `salt-key -a minion-id` or `salt-key -A`, there is now a prompt that will show the affected keys and ask for confirmation before proceeding. This prompt can be bypassed using the `-y` or `--yes` command line argument, as with other `salt-key` commands.

Support for Setting Password Hashes on BSD Minions

FreeBSD, NetBSD, and OpenBSD all now support setting passwords in `user.present` states.

26.2.37 Salt 0.16.1 Release Notes

release 2013-07-29

26.2.38 Salt 0.16.2 Release Notes

release 2013-08-01

Version 0.16.2 is a bugfix release for *0.16.0*, and contains a number of fixes.

Windows

- Only allow Administrator's group and SYSTEM user access to C:\salt. This eliminates a race condition where a non-admin user could modify a template or managed file before it is executed by the minion (which is running as an elevated user), thus avoiding a potential escalation of privileges. (issue 6361)

Grains

- Fixed detection of `virtual` grain on OpenVZ hardware nodes
- Gracefully handle lsb_release data when it is enclosed in quotes
- LSB grains are now prefixed with `lsb_distrib_` instead of simply `lsb_`. The old naming is not preserved, so SLS may be affected.
- Improved grains detection on MacOS

Pillar

- Don't try to load *git_pillar* if not enabled in master config (issue 6052)
- Functions `pillar.item` and `pillar.items` added for parity with `grains.item`/`grains.items`. The old function `pillar.data` is preserved for backwards compatibility.
- Fixed minion traceback when Pillar SLS is malformed (issue 5910)

Peer Publishing

- More gracefully handle improperly quoted publish commands (issue 5958)
- Fixed traceback when timeout specified via the CLI fo `publish.publish`, `publish.full_data` (issue 5959)
- Fixed unintended change in output of `publish.publish` (issue 5928)

Minion

- Fixed salt-key usage in minionswarm script
- Quieted warning about **SALT_MINION_CONFIG** environment variable on minion startup and for CLI commands run via `salt-call` (issue 5956)
- Added minion config parameter `random_reauth_delay` to stagger re-auth attempts when the minion is waiting for the master to approve its public key. This helps prevent SYN flooding in larger environments.

User/Group Management

- Implement previously-ignored `unique` option for `user.present` states in FreeBSD
- Report in state output when a `group.present` state attempts to use a gid in use by another group
- Fixed regression that prevents a `user.present` state to set the password hash to the system default (i.e. an unset password)
- Fixed multiple `group.present` states with the same group (issue 6439)

File Management

- Fixed file.mkdir setting incorrect permissions (issue 6033)
- Fixed cleanup of source files for templates when `/tmp` is in file_roots (issue 6118)
- Fixed caching of zero-byte files when a non-empty file was previously cached at the same path
- Added HTTP authentication support to the cp module (issue 5641)
- Diffs are now suppressed when binary files are changed

Package/Repository Management

- Fixed traceback when there is only one target for `pkg.latest` states
- Fixed regression in detection of virtual packages (apt)
- Limit number of pkg database refreshes to once per `state.sls`/`state.highstate`
- YUM: Allow 32-bit packages with arches other than i686 to be managed on 64-bit systems (issue 6299)
- Fixed incorrect reporting in pkgrepo.managed states (issue 5517)
- Fixed 32-bit binary package installs on 64-bit RHEL-based distros, and added proper support for 32-bit packages on 64-bit Debian-based distros (issue 6303)
- Fixed issue where requisites were inadvertently being put into YUM repo files (issue 6471)

Service Management

- Fixed inaccurate reporting of results in `service.running` states when the service fails to start (issue 5894)
- Fixed handling of custom initscripts in RHEL-based distros so that they are immediately available, negating the need for a second state run to manage the service that the initscript controls

Networking

- Function network.hwaddr renamed to `network.hw_addr` to match `network.ip_addrs` and `network.ip_addrs6`. All three functions also now work without the underscore in the name, as well.
- Fixed traceback in `bridge.show` when interface is not present (issue 6326)

SSH

- Fixed incorrect result reporting for some `ssh_known_hosts.present` states
- Fixed inaccurate reporting when `ssh_auth.present` states are run with `test=True`, when rsa/dss is used for the `enc` param instead of ssh-rsa/ssh-dss (issue 5374)

pip

- Properly handle `-f` lines in pip freeze output
- Fixed regression in pip.installed states with specifying a requirements file (issue 6003)
- Fixed use of `editable` argument in `pip.installed` states (issue 6025)
- Deprecated `runas` parameter in execution function calls, in favor of `user`

MySQL

- Allow specification of *MySQL* connection arguments via the CLI, overriding/bypassing minion config params
- Allow `mysql_user.present` states to set a passwordless login (issue 5550)
- Fixed endless loop when `mysql.processlist` is run (issue 6297)

PostgreSQL

- Fixed traceback in `postgres.user_list` (issue 6352)

Miscellaneous

- Don't allow *npm states* to be used if *npm module* is not available
- Fixed `alternatives.install` states for which the target is a symlink (issue 6162)
- Fixed traceback in *sysbench module* (issue 6175)
- Fixed traceback in job cache
- Fixed tempfile cleanup for windows
- Fixed issue where SLS files using the *pydsl renderer* were not being run
- Fixed issue where returners were being passed incorrect information (issue 5518)
- Fixed traceback when numeric args are passed to `cmd.script` states
- Fixed bug causing `cp.get_dir` to return more directories than expected (issue 6048)
- Fixed traceback when `supervisord.running` states are run with `test=True` (issue 6053)
- Fixed tracebacks when Salt encounters problems running rbenv (issue 5888)

- Only make the *monit module* available if monit binary is present (issue 5871)
- Fixed incorrect behavior of `img.mount_image`
- Fixed traceback in `tomcat.deploy_war` in Windows
- Don't re-write /etc/fstab if mount fails
- Fixed tracebacks when Salt encounters problems running gem (issue 5886)
- Fixed incorrect behavior of `selinux.boolean` states (issue 5912)
- *RabbitMQ*: Quote passwords to avoid symbols being interpolated by the shell (issue 6338)
- Fixed tracebacks in `extfs.mkfs` and `extfs.tune` (issue 6462)
- Fixed a regression with the `module.run` state where the `m_name` and `m_fun` arguments were being ignored (issue 6464)

26.2.39 Salt 0.16.3 Release Notes

release 2013-08-09

Version 0.16.3 is another bugfix release for *0.16.0*. The changes include:

- Various documentation fixes
- Fix proc directory regression (issue 6502)
- Properly detect Linaro Linux (issue 6496)
- Fix regressions in `mount.mounted` (issue 6522, issue 6545)
- Skip malformed state requisites (issue 6521)
- Fix regression in gitfs from bad import
- Fix for watching prereq states (including recursive requisite error) (issue 6057)
- Fix mod_watch not overriding prereq (issue 6520)
- Don't allow functions which compile states to be called within states (issue 5623)
- Return error for malformed top.sls (issue 6544)
- Fix traceback in `mysql.query`
- Fix regression in binary package installation for 64-bit packages on Debian-based Linux distros (issue 6563)
- Fix traceback caused by running `cp.push` without having set `file_recv` in the master config file
- Fix scheduler configuration in pillar (issue 6201)

26.2.40 Salt 0.16.4 Release Notes

release 2013-09-07

Version 0.16.4 is another bugfix release for *0.16.0*, likely to be the last before 0.17.0 is released. The changes include:

- Multiple documentation improvements/additions
- Added the `osfinger` and `osarch` grains
- Properly handle 32-bit packages for debian32 on x86_64 (issue 6607)
- Fix regression in yum package installation in CentOS 5 (issue 6677)

- Fix bug in `hg.latest` state that would erroneously delete directories (issue 6661)
- Fix bug related to pid not existing for `ps.top` (issue 6679)
- Fix regression in `MySQL returner` (issue 6695)
- Fix IP addresses grains (`ipv4` and `ipv6`) to include all addresses (issue 6656)
- Fix regression preventing authenticated FTP (issue 6733)
- Fix setting password for windows users (issue 6824)
- Fix `file.contains` on values YAML parses as non-string (issue 6817)
- Fix `file.get_gid`, `file.get_uid`, and `file.chown` for broken symlinks (issue 6826)
- Fix comment for service reloads in service state (issue 6851)

26.2.41 Salt 0.17.0 Release Notes

release 2013-09-26

The 0.17.0 release is a very exciting release of Salt, this brings to Salt some very powerful new features and advances. The advances range from the state system to the test suite, covering new transport capabilities and making states easier and more powerful, to extending Salt Virt and much more!

The 0.17.0 release will also be the last release of Salt to follow the old 0.XX.X numbering system, the next release of Salt will change the numbering to be date based following this format:

<Year>.<Month>.<Minor>

So if the release happens in November of 2013 the number will be 13.11.0, the first bugfix release will be 13.11.1 and so forth.

Major Features

Halite

The new Halite web GUI is now available on PyPI. A great deal of work has been put into Halite to make it fully event driven and amazingly fast. The Halite UI can be started from within the Salt Master (after being installed from PyPI), or standalone, and does not require an external database to run. It is very lightweight!

This initial release of Halite is primarily the framework for the UI and the communication systems, making it easy to extend and build the UI up. It presently supports watching the event bus and firing commands over Salt.

At this time, Halite is not available as a package, but installation documentation is available at: http://docs.saltstack.com/topics/tutorials/halite.html

Halite is, like the rest of Salt, Open Source!

Much more will be coming in the future of Halite!

Salt SSH

The new `salt-ssh` command has been added to Salt. This system allows for remote execution and states to be run over ssh. The benefit here being, that salt can run relying only on the ssh agent, rather than requiring a minion to be deployed.

The `salt-ssh` system runs states in a compatible way as Salt and states created and run with salt-ssh can be moved over to a standard salt deployment without modification.

Since this is the initial release of salt-ssh, there is plenty of room for improvement, but it is fully operational, not just a bootstrap tool.

Rosters

Salt is designed to have the minions be aware of the master and the master does not need to be aware of the location of the minions. The new salt roster system was created and designed to facilitate listing the targets for salt-ssh.

The roster system, like most of Salt, is a plugin system, allowing for the list of systems to target to be derived from any pluggable backend. The rosters shipping with 0.17.0 are flat and scan. Flat is a file which is read in via the salt render system and the scan roster does simple network scanning to discover ssh servers.

State Auto Order

This is a major change in how states are evaluated in Salt. State Auto Order is a new feature that makes states get evaluated and executed in the order in which they are defined in the sls file. This feature makes it very easy to see the finite order in which things will be executed, making Salt now, fully imperative AND fully declarative.

The requisite system still takes precedence over the order in which states are defined, so no existing states should break with this change. But this new feature can be turned off by setting `state_auto_order: False` in the master config, thus reverting to the old lexicographical order.

state.sls Runner

The `state.sls` runner has been created to allow for a more powerful system for orchestrating state runs and function calls across the salt minions. This new system uses the state system for organizing executions.

This allows for states to be defined that are executed on the master to call states on minions via `salt-run state.sls`.

Salt Thin

Salt Thin is an exciting new component of Salt, this is the ability to execute Salt routines without any transport mechanisms installed, it is a pure python subset of Salt.

Salt Thin does not have any networking capability, but can be dropped into any system with Python installed and then `salt-call` can be called directly. The Salt Thin system, is used by the `salt-ssh` command, but can still be used to just drop salt somewhere for easy use.

Event Namespacing

Events have been updated to be much more flexible. The tags in events have all been namespaced allowing easier tracking of event names.

Mercurial Fileserver Backend

The popular git fileserver backend has been joined by the mercurial fileserver backend, allowing the state tree to be managed entirely via mercurial.

External Logging Handlers

The external logging handler system allows for Salt to directly hook into any external logging system. Currently supported are sentry and logstash.

Jenkins Testing

The testing systems in Salt have been greatly enhanced, tests for salt are now executed, via jenkins.saltstack.com, across many supported platforms. Jenkins calls out to salt-cloud to create virtual machines on Rackspace, then the minion on the virtual machine checks into the master running on Jenkins where a state run is executed that sets up the minion to run tests and executes the test suite.

This now automates the sequence of running platform tests and allows for continuous destructive tests to be run.

Salt Testing Project

The testing libraries for salt have been moved out of the main salt code base and into a standalone codebase. This has been done to ease the use of the testing systems being used in salt based projects other than Salt itself.

StormPath External Authentication

The external auth system now supports the fantastic Stormpath cloud based authentication system.

LXC Support

Extensive additions have been added to Salt for LXC support. This included the backend libs for managing LXC containers. Addition into the salt-virt system is still in the works.

Mac OS X User/Group Support

Salt is now able to manage users and groups on Minions running Mac OS X. However, at this time user passwords cannot be managed.

Django ORM External Pillar

Pillar data can now be derived from Django managed databases.

Fixes from RC to release

- Multiple documentation fixes
- Add multiple source files + templating for `file.append` (issue 6905)
- Support sysctl configuration files in systemd>=207 (issue 7351)
- Add `file.search` and `file.replace`
- Fix cross-calling execution functions in provider overrides
- Fix locale override for postgres (issue 4543)

- Fix Raspbian identification for service/pkg support (issue 7371)
- Fix `cp.push` file corruption (issue 6495)
- Fix ALT Linux password hash specification (issue 3474)
- Multiple salt-ssh-related fixes and improvements

26.2.42 Salt 0.17.1 Release Notes

release 2013-10-17

Note: THIS RELEASE IS NOT COMPATIBLE WITH PREVIOUS VERSIONS. If you update your master to 0.17.1, you must update your minions as well. Sorry for the inconvenience -- this is a result of one of the security fixes listed below.

The 0.17.1 release comes with a number of improvements to salt-ssh, many bugfixes, and a number of security updates.

Salt SSH has been improved to be faster, more featureful and more secure. Since the original release of Salt SSH was primarily a proof of concept, it has been very exciting to see its rapid adoption. We appreciate the willingness of security experts to review Salt SSH and help discover oversights and ensure that security issues only exist for such a tiny window of time.

SSH Enhancements

Shell Improvements

Improvements to Salt SSH's communication have been added that improve routine execution regardless of the target system's login shell.

Performance

Deployment of routines is now faster and takes fewer commands to execute.

Security Updates

Be advised that these security issues all apply to a small subset of Salt users and mostly apply to Salt SSH.

Insufficient Argument Validation

This issue allowed for a user with limited privileges to embed executions inside of routines to execute routines that should be restricted. This applies to users using external auth or client ACL and opening up specific routines.

Be advised that these patches address the direct issue. Additional commits have been applied to help mitigate this issue from resurfacing.

CVE CVE-2013-4435

Affected Versions

0.15.0 - 0.17.0

Patches https://github.com/saltstack/salt/commit/6d8ef68b605fd63c36bb8ed96122a75ad2e80269
https://github.com/saltstack/salt/commit/ebdef37b7e5d2b95a01d34b211c61c61da67e46a
https://github.com/saltstack/salt/commit/7f190ff890e47cdd591d9d7cefa5126574660824
https://github.com/saltstack/salt/commit/8e5afe59cef6743fe5dbd510dcf463dbdfca1ced
https://github.com/saltstack/salt/commit/aca78f314481082862e96d4f0c1b75fa382bb885
https://github.com/saltstack/salt/commit/6a9752cdb1e8df2c9505ea910434c79d132eb1e2
https://github.com/saltstack/salt/commit/b73677435ba54ecfc93c1c2d840a7f9ba6f53410
https://github.com/saltstack/salt/commit/07972eb0a6f985749a55d8d4a2e471596591c80d
https://github.com/saltstack/salt/commit/1e3f197726aa13ac5c3f2416000089f477f489b5

Found By Feth Arezki, of Majerti

MITM SSH attack in salt-ssh

SSH host keys were being accepted by default and not enforced on future SSH connections. These patches set SSH host key checking by default and can be overridden by passing the -i flag to *salt-ssh*.

CVE CVE-2013-4436

Affected Versions 0.17.0

Found By Michael Scherer, Red Hat

Insecure Usage of /tmp in salt-ssh

The initial release of salt-ssh used the /tmp directory in an insecure way. These patches not only secure usage of files under /tmp in salt-ssh, but also add checksum validation for all packages sent into the now secure locations on target systems.

CVE CVE-2013-4438

Affected Versions 0.17.0

Patches https://github.com/saltstack/salt/commit/aa4bb77ef230758cad84381dde0ec660d2dc340a
https://github.com/saltstack/salt/commit/8f92b6b2cb2e4ec3af8783eb6bf4ff06f5a352cf
https://github.com/saltstack/salt/commit/c58e56811d5a50c908df0597a0ba0b643b45ebfd
https://github.com/saltstack/salt/commit/0359db9b46e47614cff35a66ea6a6a76846885d2
https://github.com/saltstack/salt/commit/4348392860e0fd43701c331ac3e681cf1a8c17b0
https://github.com/saltstack/salt/commit/664d1a1cac05602fad2693f6f97092d98a72bf61
https://github.com/saltstack/salt/commit/bab92775a576e28ff9db262f32db9cf2375bba87
https://github.com/saltstack/salt/commit/c6d34f1acf64900a3c87a2d37618ff414e5a704e

Found By Michael Scherer, Red Hat

YAML Calling Unsafe Loading Routine

It has been argued that this is not a valid security issue, as the YAML loading that was happening was only being called after an initial gateway filter in Salt has already safely loaded the YAML and would fail if non-safe routines were embedded. Nonetheless, the CVE was filed and patches applied.

CVE CVE-2013-4438

Patches

https://github.com/saltstack/salt/commit/339b0a51befae6b6b218ebcb55daa9cd3329a1c5

Found By Michael Scherer, Red Hat

Failure to Drop Supplementary Group on Salt Master

If a salt master was started as a non-root user by the root user, root's groups would still be applied to the running process. This fix changes the process to have only the groups of the running user.

CVE CVE not considered necessary by submitter.

Affected Versions 0.11.0 - 0.17.0

Patches https://github.com/saltstack/salt/commit/b89fa9135822d029795ab1eecd68cce2d1ced715

Found By Michael Scherer, Red Hat

Failure to Validate Minions Posting Data

This issue allowed a minion to pose as another authorized minion when posting data such as the mine data. All minions now pass through the id challenge before posting such data.

CVE CVE-2013-4439

Affected Versions 0.15.0 - 0.17.0

Patches

https://github.com/saltstack/salt/commit/7b850ff3d07ef6782888914ac4556c01e8a1c482
https://github.com/saltstack/salt/commit/151759b2a1e1c6ce29277aa81b054219147f80fd

Found By David Anderson

Fix Reference

Version 0.17.1 is the first bugfix release for *0.17.0*. The changes include:

- Fix symbolic links in thin.tgz (issue 7482)
- Pass env through to file.patch state (issue 7452)
- Service provider fixes and reporting improvements (issue 7361)
- Add `--priv` option for specifying salt-ssh private key
- Fix salt-thin's salt-call on setuptools installations (issue 7516)
- Fix salt-ssh to support passwords with spaces (issue 7480)
- Fix regression in wildcard includes (issue 7455)
- Fix salt-call outputter regression (issue 7456)
- Fix custom returner support for startup states (issue 7540)
- Fix value handling in augeas (issue 7605)
- Fix regression in apt (issue 7624)
- Fix minion ID guessing to use `socket.getfqdn()` first (issue 7558)
- Add minion ID caching (issue 7558)
- Fix salt-key race condition (issue 7304)
- Add `--include-all` flag to salt-key (issue 7399)
- Fix custom grains in pillar (part of issue 5716, issue 6083)
- Fix race condition in salt-key (issue 7304)
- Fix regression in minion ID guessing, prioritize `socket.getfqdn()` (issue 7558)
- Cache minion ID on first guess (issue 7558)
- Allow trailing slash in `file.directory` state
- Fix reporting of file_roots in pillar return (issue 5449 and issue 5951)
- Remove pillar matching for mine.get (issue 7197)
- Sanitize args for multiple execution modules
- Fix yumpkg mod_repo functions to filter hidden args (issue 7656)
- Fix conflicting IDs in state includes (issue 7526)
- Fix mysql_grants.absent string formatting issue (issue 7827)
- Fix postgres.version so it won't return None (issue 7695)
- Fix for trailing slashes in mount.mounted state
- Fix rogue AttributErrors in the outputter system (issue 7845)
- Fix for incorrect ssh key encodings resulting in incorrect key added (issue 7718)
- Fix for pillar/grains naming regression in python renderer (issue 7693)
- Fix args/kwargs handling in the scheduler (issue 7422)
- Fix logfile handling for *file://*, *tcp://*, and *udp://* (issue 7754)
- Fix error handling in config file parsing (issue 6714)

- Fix RVM using sudo when running as non-root user (issue 2193)
- Fix client ACL and underlying logging bugs (issue 7706)
- Fix scheduler bug with returner (issue 7367)
- Fix user management bug related to default groups (issue 7690)
- Fix various salt-ssh bugs (issue 7528)
- Many various documentation fixes

26.2.43 Salt 0.17.2 Release Notes

release 2013-11-14

Version 0.17.2 is another bugfix release for *0.17.0*. The changes include:

- Add ability to delete key with grains.delval (issue 7872)
- Fix possible state compiler stack trace (issue 5767)
- Fix architecture regression in yumpkg (issue 7813)
- Use correct `ps` on Debian to prevent truncating (issue 5646)
- Fix grains targeting for new grains (issue 5737)
- Fix bug with merging in git_pillar (issue 6992)
- Fix print_jobs duplicate results
- Fix apt version specification for pkg.install
- Fix possible KeyError from ext_job_cache missing option
- Fix auto_order for `- names` states (issue 7649)
- Fix regression in new gitfs installs (directory not found error)
- Fix escape pipe issue on Windows for file.recurse (issue 7967)
- Fix fileclient in case of master restart (issue 7987)
- Try to output warning if CLI command malformed (issue 6538)
- Fix `--out=quiet` to actually be quiet (issue 8000)
- Fix for state.sls in salt-ssh (issue 7991)
- Fix for MySQL grants ordering issue (issue 5817)
- Fix traceback for certain missing CLI args (issue 8016)
- Add ability to disable lspci queries on master (issue 4906)
- Fail if sls defined in topfile does not exist (issue 5998)
- Add ability to downgrade MySQL grants (issue 6606)
- Fix ssh_auth.absent traceback (issue 8043)
- Add upstart detection for Debian/Raspbian (issue 8039)
- Fix ID-related issues (issue 8052, issue 8050, and others)
- Fix for jinja rendering issues (issue 8066 and issue 8079)
- Fix argument parsing in salt-ssh (issue 7928)

- Fix some GPU detection instances (issue 6945)
- Fix bug preventing includes from other environments in SLS files
- Fix for kwargs with dashes (issue 8102)
- Fix salt.utils.which for windows `.exe' (issue 7904)
- Fix apache.adduser without apachectl (issue 8123)
- Fix issue with evaluating `test` kwarg in states (issue 7788)
- Fix regression in `salt.client.Caller()` (issue 8078)
- Fix apt-key silent failure
- Fix bug where cmd.script would try to run even if caching failed (issue 7601)
- Fix apt `pkg.latest` regression (issue 8067)
- Fix for mine data not being updated (issue 8144)
- Fix for noarch packages in yum
- Fix a Xen detection edge case (issue 7839)
- Fix windows `__opts__` dictionary persistence (issue 7714)
- Fix version generation for when it's part of another git repo (issue 8090)
- Fix _handle_iorder stacktrace so that the real syntax error is shown (issue 8114 and issue 7905)
- Fix `git.latest` state when a commit SHA is used (issue 8163)
- Fix various small bugs in yumpkg.py (issue 8201)
- Fix for specifying identify file in git.latest (issue 8094)
- Fix for `--output-file` CLI arg (issue 8205)
- Add ability to specify shutdown time for system.shutdown (issue 7833)
- Fix for salt version using non-salt git repo info (issue 8266)
- Add additional hints at impact of `pkgrepo` states when `test=True` (issue 8247)
- Fix for salt-ssh files not being owned by root (issue 8216)
- Fix retry logic and error handling in fileserver (related to issue 7755)
- Fix file.replace with `test=True` (issue 8279)
- Add flag for limiting file traversal in fileserver (issue 6928)
- Fix for extra mine processes (issue 5729)
- Fix for unloading custom modules (issue 7691)
- Fix for salt-ssh opts (issue 8005 and issue 8271)
- Fix compound matcher for grains (issue 7944)
- Improve error reporting in ebuild module (related to issue 5393)
- Add `dir_mode` to `file.managed` (issue 7860)
- Improve traceroute support for FreeBSD and OS X (issue 4927)
- Fix for matching minions under syndics (issue 7671)
- Improve exception handling for missing ID (issue 8259)

- Fix grain mismatch for ScientificLinux (issue 8338)
- Add configuration option for minion_id_caching
- Fix open mode auth errors (issue 8402)

26.2.44 Salt 0.17.3 Release Notes

release 2013-12-08

Note: 0.17.3 had some regressions which were promptly fixed in the 0.17.4 release. Please use 0.17.4 instead.

Version 0.17.3 is another bugfix release for *0.17.0*. The changes include:

- Fix some jinja render errors (issue 8418)
- Fix `file.replace` state changing file ownership (issue 8399)
- Fix state ordering with the PyDSL renderer (issue 8446)
- Fix for new npm version (issue 8517)
- Fix for pip state requiring `name` even with requirements file (issue 8519)
- Fix yum logging to open terminals (issue 3855)
- Add sane maxrunning defaults for scheduler (issue 8563)
- Fix states duplicate key detection (issue 8053)
- Fix SUSE patch level reporting (issue 8428)
- Fix managed file creation umask (issue 8590)
- Fix logstash exception (issue 8635)
- Improve argument exception handling for salt command (issue 8016)
- Fix pecl success reporting (issue 8750)
- Fix launchctl module exceptions (issue 8759)
- Fix argument order in pw_user module
- Add warnings for failing grains (issue 8690)
- Fix hgfs problems caused by connections left open (issue 8811 and issue 8810)
- Add Debian iptables default for iptables-persistent package (issue 8889)
- Fix installation of packages with dots in pkg name (issue 8614)
- Fix noarch package installation on CentOS 6 (issue 8945)
- Fix portage_config.enforce_nice_config (issue 8252)
- Fix salt.util.copyfile umask usage (issue 8590)
- Fix rescheduling of failed jobs (issue 8941)
- Fix pkg on Amazon Linux (uses yumpkg5 now) (issue 8226)
- Fix conflicting options in postgres module (issue 8717)
- Fix ps modules for psutil >= 0.3.0 (issue 7432)
- Fix postgres module to return False on failure (issue 8778)

- Fix argument passing for args with pound signs (issue 8585)
- Fix pid of salt CLi command showing in status.pid output (issue 8720)
- Fix rvm to run gem as the correct user (issue 8951)
- Fix namespace issue in win_file module (issue 9060)
- Fix masterless state paths on windows (issue 9021)
- Fix timeout option in master config (issue 9040)

26.2.45 Salt 0.17.4 Release Notes

release 2013-12-10

Version 0.17.4 is another bugfix release for *0.17.0*. The changes include:

- Fix file.replace bug when replacement str is numeric (issue 9101)
- Fix regression in file.managed (issue 9131)
- Prevent traceback when job is None. (issue 9145)

26.2.46 Salt 0.17.5 Release Notes

release 2014-01-27

Version 0.17.5 is another bugfix release for *0.17.0*. The changes include:

- Fix `user.present` states with non-string fullname (issue 9085)
- Fix `virt.init` return value on failure (issue 6870)
- Fix reporting of `file.blockreplace` state when `test=True`
- Fix `network.interfaces` when used in cron (issue 7990)
- Fix bug in pkgrepo when switching to/from mirrorlist-based repo def (issue 9121)
- Fix infinite recursion when cache file is corrupted
- Add checking for rev and mirror/bare args in `git.latest` (issue 9107)
- Add `cmd.watch` alias (points to `cmd.wait`) (issue 8612)
- Fix stacktrace when prereq is not formed as a list (issue 8235)
- Fix stdin issue with lvdisplay command (issue 9128)
- Add pre-check function for range matcher (issue 9236)
- Add exception handling for psutil for processes that go missing (issue 9274)
- Allow `_in` requisites to match both on ID and name (issue 9061)
- Fix multiple client timeout issues (issue 7157 and issue 9302, probably others)
- Fix `ZMQError: Operation cannot be accomplished in current state` errors (issue 6306)
- Multiple optimization in minion auth routines
- Clarify logs for minion ID caching

26.2.47 Salt 0.6.0 release notes

The Salt remote execution manager has reached initial functionality! Salt is a management application which can be used to execute commands on remote sets of servers.

The whole idea behind Salt is to create a system where a group of servers can be remotely controlled from a single master, not only can commands be executed on remote systems, but salt can also be used to gather information about your server environment.

Unlike similar systems, like Func and MCollective, Salt is extremely simple to setup and use, the entire application is contained in a single package, and the master and minion daemons require no running dependencies in the way that Func requires Certmaster and MCollective requires activeMQ.

Salt also manages authentication and encryption. Rather than using SSL for encryption, salt manages encryption on a payload level, so the data sent across the network is encrypted with fast AES encryption, and authentication uses RSA keys. This means that Salt is fast, secure, and very efficient.

Messaging in Salt is executed with ZeroMQ, so the message passing interface is built into salt and does not require an external ZeroMQ server. This also adds speed to Salt since there is no additional bloat on the networking layer, and ZeroMQ has already proven itself as a very fast networking system.

The remote execution in Salt is ``Lazy Execution'', in that once the command is sent the requesting network connection is closed. This makes it easier to detach the execution from the calling process on the master, it also means that replies are cached, so that information gathered from historic commands can be queried in the future.

Salt also allows users to make execution modules in Python. Writers of these modules should also be pleased to know that they have access to the impressive information gathered from PuppetLabs' Facter application, making Salt module more flexible. In the future I hope to also allow Salt to group servers based on Facter information as well.

All in all Salt is fast, efficient, and clean, can be used from a simple command line client or through an API, uses message queue technology to make network execution extremely fast, and encryption is handled in a very fast and efficient manner. Salt is also VERY easy to use and VERY easy to extend.

You can find the source code for Salt on my GitHub page, I have also set up a few wiki pages explaining how to use and set up Salt. If you are using Arch Linux there is a package available in the Arch Linux AUR.

Salt 0.6.0 Source: https://cloud.github.com/downloads/saltstack/salt/salt-0.6.0.tar.gz

GitHub page: https://github.com/saltstack/salt

Wiki: https://github.com/saltstack/salt/wiki

Arch Linux Package: https://aur.archlinux.org/packages/salt-git/

I am very open to contributions, for instance I need packages for more Linux distributions as well as BSD packages and testers.

Give Salt a try, this is the initial release and is not a 1.0 quality release, but it has been working well for me! I am eager to get your feedback!

26.2.48 Salt 0.7.0 release notes

I am pleased to announce the release of Salt 0.7.0!

This release marks what is the first stable release of salt, 0.7.0 should be suitable for general use.

0.7.0 Brings the following new features to Salt:

- Integration with Facter data from puppet labs
- Allow for matching minions from the salt client via Facter information

- Minion job threading, many jobs can be executed from the master at once
- Preview of master clustering support - Still experimental
- Introduce new minion modules for stats, virtualization, service management and more
- Add extensive logging to the master and minion daemons
- Add sys.reload_functions for dynamic function reloading
- Greatly improve authentication
- Introduce the saltkey command for managing public keys
- Begin backend development preparatory to introducing butter
- Addition of man pages for the core commands
- Extended and cleaned configuration

0.7.0 Fixes the following major bugs:

- Fix crash in minions when matching failed
- Fix configuration file lookups for the local client
- Repair communication bugs in encryption
- Numerous fixes in the minion modules

The next release of Salt should see the following features:

- Stabilize the cluster support
- Introduce a remote client for salt command tiers
- salt-ftp system for distributed file copies
- Initial support for ``butter''

Coming up next is a higher level management framework for salt called Butter. I want salt to stay as a simple and effective communication framework, and allow for more complicated executions to be managed via Butter.

Right now Butter is being developed to act as a cloud controller using salt as the communication layer, but features like system monitoring and advanced configuration control (a puppet manager) are also in the pipe.

Special thanks to Joseph Hall for the status and network modules, and thanks to Matthias Teege for tracking down some configuration bugs!

Salt can be downloaded from the following locations;

Source Tarball:

https://cloud.github.com/downloads/saltstack/salt/salt-0.7.0.tar.gz

Arch Linux Package:

https://aur.archlinux.org/packages/salt-git/

Please enjoy the latest Salt release!

26.2.49 Salt 0.8.0 release notes

Salt 0.8.0 is ready for general consumption! The source tarball is available on GitHub for download:

https://cloud.github.com/downloads/saltstack/salt/salt-0.8.0.tar.gz

A lot of work has gone into salt since the last release just 2 weeks ago, and salt has improved a great deal. A swath of new features are here along with performance and threading improvements!

The main new features of salt 0.8.0 are:

Salt-cp

Cython minion modules

Dynamic returners

Faster return handling

Lowered required Python version to 2.6

Advanced minion threading

Configurable minion modules

Salt-cp

The salt-cp command introduces the ability to copy simple files via salt to targeted servers. Using salt-cp is very simple, just call salt-cp with a target specification, the source file(s) and where to copy the files on the minions. For instance:

salt-cp '*' /etc/hosts /etc/hosts

Will copy the local /etc/hosts file to all of the minions.

Salt-cp is very young, in the future more advanced features will be added, and the functionality will much more closely resemble the cp command.

Cython minion modules

Cython is an amazing tool used to compile Python modules down to c. This is arguably the fastest way to run Python code, and since pyzmq requires cython, adding support to salt for cython adds no new dependencies.

Cython minion modules allow minion modules to be written in cython and therefore executed in compiled c. Simply write the salt module in cython and use the file extension ".pyx" and the minion module will be compiled when the minion is started. An example cython module is included in the main distribution called cytest.pyx:

https://github.com/saltstack/salt/blob/develop/salt/modules/cytest.pyx

Dynamic Returners

By default salt returns command data back to the salt master, but now salt can return command data to any system. This is enabled via the new returners modules feature for salt. The returners modules take the return data and sends it to a specific module. The returner modules work like minion modules, so any returner can be added to the minions.

This means that a custom data returner can be added to communicate the return data so anything from MySQL, Redis, MongoDB, and more!

There are 2 simple stock returners in the returners directory:

https://github.com/saltstack/salt/blob/develop/salt/returners

The documentation on writing returners will be added to the wiki shortly, and returners can be written in pure Python, or in cython.

Configurable Minion Modules

Minion modules may need to be configured, now the options passed to the minion configuration file can be accessed inside of the minion modules via the __opt__ dict.

Information on how to use this simple addition has been added to the wiki: *Writing modules*

The test module has an example of using the __opts__ dict, and how to set default options:

https://github.com/saltstack/salt/blob/develop/salt/modules/test.py

Advanced Minion Threading

In 0.7.0 the minion would block after receiving a command from the master, now the minion will spawn a thread or multiprocess. By default Python threads are used because for general use they have proved to be faster, but the minion can now be configured to use the Python multiprocessing module instead. Using multiprocessing will cause executions that are CPU bound or would otherwise exploit the negative aspects of the Python GIL to run faster and more reliably, but simple calls will still be faster with Python threading. The configuration option can be found in the minion configuration file:

https://github.com/saltstack/salt/blob/develop/conf/minion

Lowered Supported Python to 2.6

The requirement for Python 2.7 has been removed to support Python 2.6. I have received requests to take the minimum Python version back to 2.4, but unfortunately this will not be possible, since the ZeroMQ Python bindings do not support Python 2.4.

Salt 0.8.0 is a very major update, it also changes the network protocol slightly which makes communication with older salt daemons impossible, your master and minions need to be upgraded together!

I could use some help bringing salt to the people! Right now I only have packages for Arch Linux, Fedora 14 and Gentoo. We need packages for Debian and people willing to help test on more platforms. We also need help writing more minion modules and returner modules. If you want to contribute to salt please hop on the mailing list and send in patches, make a fork on GitHub and send in pull requests! If you want to help but are not sure where you can, please email me directly or post tot he mailing list!

I hope you enjoy salt, while it is not yet 1.0 salt is completely viable and usable!

-Thomas S. Hatch

26.2.50 Salt 0.8.7 release notes

It has been a month since salt 0.8.0, and it has been a long month! But Salt is still coming along strong. 0.8.7 has a lot of changes and a lot of updates. This update makes Salt's ZeroMQ back end better, strips Facter from the dependencies, and introduces interfaces to handle more capabilities.

Many of the major updates are in the background, but the changes should shine through to the surface. A number of the new features are still a little thin, but the back end to support expansion is in place.

I also recently gave a presentation to the Utah Python users group in Salt Lake City, the slides from this presentation are available here: https://cloud.github.com/downloads/saltstack/salt/Salt.pdf

The video from this presentation will be available shortly.

The major new features and changes in Salt 0.8.7 are:

- Revamp ZeroMQ topology on the master for better scalability

- State enforcement
- Dynamic state enforcement managers
- Extract the module loader into salt.loader
- Make Job ids more granular
- Replace Facter functionality with the new salt grains interface
- Support for "virtual" salt modules
- Introduce the salt-call command
- Better debugging for minion modules

The new ZeroMQ topology allows for better scalability, this will be required by the need to execute massive file transfers to multiple machines in parallel and state management. The new ZeroMQ topology is available in the aforementioned presentation.

0.8.7 introduces the capability to declare states, this is similar to the capabilities of Puppet. States in salt are declared via state data structures. This system is very young, but the core feature set is available. Salt states work around rendering files which represent Salt high data. More on the Salt state system will be documented in the near future.

The system for loading salt modules has been pulled out of the minion class to be a standalone module, this has enabled more dynamic loading of Salt modules and enables many of the updates in 0.8.7 –

https://github.com/saltstack/salt/blob/develop/salt/loader.py

Salt Job ids are now microsecond precise, this was needed to repair a race condition unveiled by the speed improvements in the new ZeroMQ topology.

The new grains interface replaces the functionality of Facter, the idea behind grains differs from Facter in that the grains are only used for static system data, dynamic data needs to be derived from a call to a salt module. This makes grains much faster to use, since the grains data is generated when the minion starts.

Virtual salt modules allows for a salt module to be presented as something other than its module name. The idea here is that based on information from the minion decisions about which module should be presented can be made. The best example is the pacman module. The pacman module will only load on Arch Linux minions, and will be called pkg. Similarly the yum module will be presented as pkg when the minion starts on a Fedora/RedHat system.

The new salt-call command allows for minion modules to be executed from the minion. This means that on the minion a salt module can be executed, this is a great tool for testing Salt modules. The salt-call command can also be used to view the grains data.

In previous releases when a minion module threw an exception very little data was returned to the master. Now the stack trace from the failure is returned making debugging of minion modules MUCH easier.

Salt is nearing the goal of 1.0, where the core feature set and capability is complete!

Salt 0.8.7 can be downloaded from GitHub here: https://cloud.github.com/downloads/saltstack/salt/salt-0.8.7.tar.gz

-Thomas S Hatch

26.2.51 Salt 0.8.8 release notes

Salt 0.8.8 is here! This release adds a great deal of code and some serious new features. The latest release can be downloaded here: https://cloud.github.com/downloads/saltstack/salt/salt-0.8.8.tar.gz

Improved Documentation has been set up for salt using sphinx thanks to the efforts of Seth House. This new documentation system will act as the back end to the salt website which is still under heavy development. The new sphinx documentation system has also been used to greatly clean up the salt manpages. The salt 7 manpage in particular now contains extensive information which was previously only in the wiki. The new documentation can be

found at: http://docs.saltstack.com/ We still have a lot to add, and when the domain is set up I will post another announcement.

More additions have been made to the ZeroMQ setup, particularly in the realm of file transfers. Salt 0.8.8 introduces a built in, stateless, encrypted file server which allows salt minions to download files from the salt master using the same encryption system used for all other salt communications. The main motivation for the salt file server has been to facilitate the new salt state system.

Much of the salt code has been cleaned up and a new cleaner logging system has been introduced thanks to the efforts of Pedro Algarvio. These additions will allow for much more flexible logging to be executed by salt, and fixed a great deal of my poor spelling in the salt docstrings! Pedro Algarvio has also cleaned up the API, making it easier to embed salt into another application.

The biggest addition to salt found in 0.8.8 is the new state system. The salt module system has received a new front end which allows salt to be used as a configuration management system. The configuration management system allows for system configuration to be defined in data structures. The configuration management system, or as it is called in salt, the "salt state system" supports many of the features found in other configuration managers, but allows for system states to be written in a far simpler format, executes at blazing speeds, and operates via the salt minion matching system. The state system also operates within the normal scope of salt, and requires no additional configuration to use.

The salt state system can enforce the following states with many more to come: Packages Files Services Executing commands Hosts

The system used to define the salt states is based on a data structure, the data structure used to define the salt states has been made to be as easy to use as possible. The data structure is defined by default using a YAML file rendered via a Jinja template. This means that the state definition language supports all of the data structures that YAML supports, and all of the programming constructs and logic that Jinja supports. If the user does not like YAML or Jinja the states can be defined in yaml-mako, json-jinja, or json-mako. The system used to render the states is completely dynamic, and any rendering system can be added to the capabilities of Salt, this means that a rendering system that renders XML data in a cheetah template, or whatever you can imagine, can be easily added to the capabilities of salt.

The salt state system also supports isolated environments, as well as matching code from several environments to a single salt minion.

The feature base for Salt has grown quite a bit since my last serious documentation push. As we approach 0.9.0 the goals are becoming very clear, and the documentation needs a lot of work. The main goals for 0.9.0 are to further refine the state system, fix any bugs we find, get Salt running on as many platforms as we can, and get the documentation filled out. There is a lot more to come as Salt moves forward to encapsulate a much larger scope, while maintaining supreme usability and simplicity.

If you would like a more complete overview of Salt please watch the Salt presentation: Slides: https://cloud.github.com/downloads/saltstack/salt/Salt.pdf

-Thomas S Hatch

26.2.52 Salt 0.8.9 Release Notes

Salt 0.8.9 has finally arrived! Unfortunately this is much later than I had hoped to release 0.8.9, life has been very crazy over the last month. But despite challenges, Salt has moved forward!

This release, as expected, adds few new features and many refinements. One of the most exciting aspect of this release is that the development community for salt has grown a great deal and much of the code is from contributors.

Also, I have filled out the documentation a great deal. So information on States is properly documented, and much of the documentation that was out of date has been filled in.

Download!

The Salt source can be downloaded from the salt GitHub site:

https://cloud.github.com/downloads/saltstack/salt/salt-0.8.9.tar.gz

Or from PyPI:

https://pypi.python.org/packages/source/s/salt/salt-0.8.9.tar.gz

Here s the md5sum:

7d5aca4633bc22f59045f59e82f43b56

For instructions on how to set up Salt please see the *Installation* instructions.

New Features

Salt Run

A big feature is the addition of Salt run, the `salt-run` command allows for master side execution modules to be made that gather specific information or execute custom routines from the master.

Documentation for salt-run can be found *here*

Refined Outputters

One problem often complained about in salt was the fact that the output was so messy. Thanks to help from Jeff Schroeder a cleaner interface for the command output for the Salt CLI has been made. This new interface makes adding new printout formats easy and additions to the capabilities of minion modules makes it possible to set the printout mode or `outputter` for functions in minion modules.

Cross Calling Salt Modules

Salt modules can now call each other, the `__salt__` dict has been added to the predefined references in minion modules. This new feature is documented in the *modules documentation*.

Watch Option Added to Salt State System

Now in Salt states you can set the watch option, this will allow watch enabled states to change based on a change in the other defined states. This is similar to subscribe and notify statements in puppet.

Root Dir Option

Travis Cline has added the ability to define the option `root_dir` which allows the salt minion to operate in a subdir. This is a strong move in supporting the minion running as an unprivileged user

Config Files Defined in Variables

Thanks again to Travis Cline, the master and minion configuration file locations can be defined in environment variables now.

New Modules

Quite a few new modules, states, returners, and runners have been made.

New Minion Modules

apt Support for apt-get has been added, this adds greatly improved Debian and Ubuntu support to Salt!

useradd and groupadd Support for manipulating users and groups on Unix-like systems.

moosefs Initial support for reporting on aspects of the distributed file system, MooseFS. For more information on MooseFS please see: http://www.moosefs.org

Thanks to Joseph Hall for his work on MooseFS support.

mount Manage mounts and the fstab.

puppet Execute puppet on remote systems.

shadow Manipulate and manage the user password file.

ssh Interact with ssh keys.

New States

user and group Support for managing users and groups in Salt States.

mount Enforce mounts and the fstab.

New Returners

mongo_return Send the return information to a MongoDB server.

New Runners

manage Display minions that are up or down.

26.2.53 Salt 0.9.0 Release Notes

release 2011-08-27

Salt 0.9.0 is here. This is an exciting release, 0.9.0 includes the new network topology features allowing peer salt commands and masters of masters via the syndic interface.

0.9.0 also introduces many more modules, improvements to the API and improvements to the ZeroMQ systems.

Download!

The Salt source can be downloaded from the salt GitHub site:

https://cloud.github.com/downloads/saltstack/salt/salt-0.9.0.tar.gz

Or from PyPI:

https://pypi.python.org/packages/source/s/salt/salt-0.9.0.tar.gz

Here is the md5sum:

9a925da04981e65a0f237f2e77ddab37

For instructions on how to set up Salt please see the *Installation* instructions.

New Features

Salt Syndic

The new *Syndic interface* allows a master to be commanded via another higher level salt master. This is a powerful solution allowing a master control structure to exist, allowing salt to scale to much larger levels then before.

Peer Communication

0.9.0 introduces the capability for a minion to call a publication on the master and receive the return from another set of minions. This allows salt to act as a communication channel between minions and as a general infrastructure message bus.

Peer communication is turned off by default but can be enabled via the `peer` option in the master configuration file. Documentation on the new *Peer interface*.

Easily Extensible API

The minion and master classes have been redesigned to allow for specialized minion and master servers to be easily created. An example on how this is done for the master can be found in the `master.py` salt module:

https://github.com/saltstack/salt/blob/develop/salt/master.py

The `Master` class extends the `SMaster` class and set up the main master server.

The minion functions can now also be easily added to another application via the `SMinion` class, this class can be found in the `minion.py` module:

https://github.com/saltstack/salt/blob/develop/salt/minion.py

Cleaner Key Management

This release changes some of the key naming to allow for multiple master keys to be held based on the type of minion gathering the master key.

The -d option has also been added to the salt-key command allowing for easy removal of accepted public keys.

The --gen-keys option is now available as well for salt-key, this allows for a salt specific RSA key pair to be easily generated from the command line.

Improved 0MQ Master Workers

The 0MQ worker system has been further refined to be faster and more robust. This new system has been able to handle a much larger load than the previous setup. The new system uses the IPC protocol in 0MQ instead of TCP.

New Modules

Quite a few new modules have been made.

New Minion Modules

apache Work directly with apache servers, great for managing balanced web servers

cron Read out the contents of a systems crontabs

mdadm Module to manage raid devices in Linux, appears as the `raid` module

mysql Gather simple data from MySQL databases

ps Extensive utilities for managing processes

publish Used by the peer interface to allow minions to make publications

26.2.54 Salt 0.9.1 Release Notes

release 2011-08-29

26.2.55 Salt 0.9.2 Release Notes

release 2011-09-17

Salt 0.9.2 has arrived! 0.9.2 is primarily a bugfix release, the exciting component in 0.9.2 is greatly improved support for salt states. All of the salt states interfaces have been more thoroughly tested and the new salt-states git repo is growing with example of how to use states.

This release introduces salt states for early developers and testers to start helping us clean up the states interface and make it ready for the world!

0.9.2 also fixes a number of bugs found on Python 2.6.

Download!

The Salt source can be downloaded from the salt GitHub site:

https://cloud.github.com/downloads/saltstack/salt/salt-0.9.2.tar.gz

Or from PyPI:

https://pypi.python.org/packages/source/s/salt/salt-0.9.2.tar.gz

For instructions on how to set up Salt please see the *Installation* instructions.

New Features

Salt-Call Additions

The salt-call command has received an overhaul, it now hooks into the outputter system so command output looks clean, and the logging system has been hooked into salt-call, so the -l option allows the logging output from salt minion functions to be displayed.

The end result is that the salt-call command can execute the state system and return clean output:

```
# salt-call state.highstate
```

State System Fixes

The state system has been tested and better refined. As of this release the state system is ready for early testers to start playing with. If you are interested in working with the state system please check out the (still very small) salt-states GitHub repo:

https://github.com/saltstack/salt-states

This git repo is the active development branch for determining how a clean salt-state database should look and act. Since the salt state system is still very young a lot of help is still needed here. Please fork the salt-states repo and help us develop a truly large and scalable system for configuration management!

Notable Bug Fixes

Python 2.6 String Formatting

Python 2.6 does not support format strings without an index identifier, all of them have been repaired.

Cython Loading Disabled by Default

Cython loading requires a development tool chain to be installed on the minion, requiring this by default can cause problems for most Salt deployments. If Cython auto loading is desired it will need to be turned on in the minion config.

26.2.56 Salt 0.9.3 Release Notes

release 2011-11-05

Salt 0.9.3 is finally arrived. This is another big step forward for Salt, new features range from proper FreeBSD support to fixing issues seen when attaching a minion to a master over the Internet.

The biggest improvements in 0.9.3 though can be found in the state system, it has progressed from something ready for early testers to a system ready to compete with platforms such as Puppet and Chef. The backbone of the state system has been greatly refined and many new features are available.

Download!

The Salt source can be downloaded from the salt GitHub site:

https://cloud.github.com/downloads/saltstack/salt/salt-0.9.3.tar.gz

Or from PyPI:

https://pypi.python.org/packages/source/s/salt/salt-0.9.3.tar.gz

For instructions on how to set up Salt please see the *Installation* instructions.

New Features

WAN Support

Recently more people have been testing Salt minions connecting to Salt Masters over the Internet. It was found that Minions would commonly loose their connection to the master when working over the internet. The minions can now detect if the connection has been lost and reconnect to the master, making WAN connections much more reliable.

State System Fixes

Substantial testing has gone into the state system and it is ready for real world usage. A great deal has been added to the documentation for states and the modules and functions available to states have been cleanly documented.

A number of State System bugs have also been founds and repaired, the output from the state system has also been refined to be extremely clear and concise.

Error reporting has also been introduced, issues found in sls files will now be clearly reported when executing Salt States.

Extend Declaration

The Salt States have also gained the `extend` declaration. This declaration allows for states to be cleanly modified in a post environment. Simply said, if there is an apache.sls file that declares the apache service, then another sls can include apache and then extend it:

```
include:
  - apache

extend:
  apache:
    service:
      - require:
        - pkg: mod_python

mod_python:
  pkg:
    - installed
```

The notable behavior with the extend functionality is that it literally extends or overwrites a declaration set up in another sls module. This means that Salt will behave as though the modifications were made directly to the apache sls. This ensures that the apache service in this example is directly tied to all requirements.

Highstate Structure Specification

This release comes with a clear specification of the Highstate data structure that is used to declare Salt States. This specification explains everything that can be declared in the Salt SLS modules.

The specification is extremely simple, and illustrates how Salt has been able to fulfill the requirements of a central configuration manager within a simple and easy to understand format and specification.

SheBang Renderer Switch

It came to our attention that having many renderers means that there may be a situation where more than one State Renderer should be available within a single State Tree.

The method chosen to accomplish this was something already familiar to developers and systems administrators, a SheBang. The Python State Renderer displays this new capability.

Python State Renderer

Until now Salt States could only be declared in yaml or json using Jinja or Mako. A new, very powerful, renderer has been added, making it possible to write Salt States in pure Python:

```
#!py

def run():
    '''
    Install the python-mako package
    '''
    return {'include': ['python'],
            'python-mako': {'pkg': ['installed']}}
```

This renderer is used by making a run function that returns the Highstate data structure. Any capabilities of Python can be used in pure Python sls modules.

This example of a pure Python sls module is the same as this example in yaml:

```
include:
  - python

python-mako:
  pkg:
    - installed
```

FreeBSD Support

Additional support has been added for FreeBSD, this is Salt's first branch out of the Linux world and proves the viability of Salt on non-Linux platforms.

Salt remote execution already worked on FreeBSD, and should work without issue on any Unix-like platform. But this support comes in the form of package management and user support, so Salt States also work on FreeBSD now.

The new freebsdpkg module provides package management support for FreeBSD and the new pw_user and pw_group provide user and group management.

Module and State Additions

Cron Support

Support for managing the system crontab has been added, declaring a cron state can be done easily:

```
date > /tmp/datestamp:
  cron:
    - present
    - user: fred
    - minute: 5
    - hour: 3
```

File State Additions

The file state has been given a number of new features, primarily the directory, recurse, symlink, and absent functions.

file.directory Make sure that a directory exists and has the right permissions.

```
    /srv/foo:
      file:
        - directory
        - user: root
        - group: root
        - mode: 1755
```

file.symlink Make a symlink.

```
    /var/lib/www:
      file:
        - symlink
        - target: /srv/www
        - force: True
```

file.recurse The recurse state function will recursively download a directory on the master file server and place it on the minion. Any change in the files on the master will be pushed to the minion. The recurse function is very powerful and has been tested by pushing out the full Linux kernel source.

```
    /opt/code:
      file:
        - recurse
        - source: salt://linux
```

file.absent Make sure that the file is not on the system, recursively deletes directories, files, and symlinks.

```
    /etc/httpd/conf.d/somebogusfile.conf:
      file:
        - absent
```

Sysctl Module and State

The sysctl module and state allows for sysctl components in the kernel to be managed easily. the sysctl module contains the following functions:

sysctl.show Return a list of sysctl parameters for this minion

sysctl.get Return a single sysctl parameter for this minion

sysctl.assign Assign a single sysctl parameter for this minion

sysctl.persist Assign and persist a simple sysctl parameter for this minion

The sysctl state allows for sysctl parameters to be assigned:

```
vm.swappiness:
  sysctl:
    - present
    - value: 20
```

Kernel Module Management

A module for managing Linux kernel modules has been added. The new functions are as follows:

kmod.available Return a list of all available kernel modules

kmod.check_available Check to see if the specified kernel module is available

kmod.lsmod Return a dict containing information about currently loaded modules

kmod.load Load the specified kernel module

kmod.remove Unload the specified kernel module

The kmod state can enforce modules be either present or absent:

```
kvm_intel:
  kmod:
    - present
```

Ssh Authorized Keys

The ssh_auth state can distribute ssh authorized keys out to minions. Ssh authorized keys can be present or absent.

```
AAAAB3NzaC1kc3MAAACBAL0sQ9fJ5bYTEyYvlRBsJdDOo49CNfhlWHWXQRqul6rwL4KIuPrhY7hBw0tV7UNC7J9IZRNO4iGod9C+C
  ssh_auth:
    - present
    - user: frank
    - enc: dsa
    - comment: 'Frank's key'
```

26.2.57 Salt 0.9.4 Release Notes

release 2011-11-27

Salt 0.9.4 has arrived. This is a critical update that repairs a number of key bugs found in 0.9.3. But this update is not without feature additions as well! 0.9.4 adds support for Gentoo portage to the pkg module and state system. Also there are 2 major new state additions, the failhard option and the ability to set up finite state ordering with the `order` option.

This release also sees our largest increase in community contributions. These contributors have and continue to be the life blood of the Salt project, and the team continues to grow. I want to put out a big thanks to our new and existing contributors.

Download!

The Salt source can be downloaded from the salt GitHub site:

https://cloud.github.com/downloads/saltstack/salt/salt-0.9.4.tar.gz

Or from PyPI:

https://pypi.python.org/packages/source/s/salt/salt-0.9.4.tar.gz

For instructions on how to set up Salt please see the *Installation* instructions.

New Features

Failhard State Option

Normally, when a state fails Salt continues to execute the remainder of the defined states and will only refuse to execute states that require the failed state.

But the situation may exist, where you would want all state execution to stop if a single state execution fails. The capability to do this is called `failing hard`.

State Level Failhard A single state can have a failhard set, this means that if this individual state fails that all state execution will immediately stop. This is a great thing to do if there is a state that sets up a critical config file and setting a require for each state that reads the config would be cumbersome. A good example of this would be setting up a package manager early on:

```
/etc/yum.repos.d/company.repo:
  file:
    - managed
    - source: salt://company/yumrepo.conf
    - user: root
    - group: root
    - mode: 644
    - order: 1
    - failhard: True
```

In this situation, the yum repo is going to be configured before other states, and if it fails to lay down the config file, than no other states will be executed.

Global Failhard It may be desired to have failhard be applied to every state that is executed, if this is the case, then failhard can be set in the master configuration file. Setting failhard in the master configuration file will result in failing hard when any minion gathering states from the master have a state fail.

This is NOT the default behavior, normally Salt will only fail states that require a failed state.

Using the global failhard is generally not recommended, since it can result in states not being executed or even checked. It can also be confusing to see states failhard if an admin is not actively aware that the failhard has been set.

To use the global failhard set failhard: True in the master configuration

Finite Ordering of State Execution

When creating salt sls files, it is often important to ensure that they run in a specific order. While states will always execute in the same order, that order is not necessarily defined the way you want it.

A few tools exist in Salt to set up the correct state ordering, these tools consist of requisite declarations and order options.

The Order Option Before using the order option, remember that the majority of state ordering should be done with requisite statements, and that a requisite statement will override an order option.

The order option is used by adding an order number to a state declaration with the option *order*:

```
vim:
  pkg:
    - installed
    - order: 1
```

By adding the order option to *1* this ensures that the vim package will be installed in tandem with any other state declaration set to the order *1*.

Any state declared without an order option will be executed after all states with order options are executed.

But this construct can only handle ordering states from the beginning. Sometimes you may want to send a state to the end of the line, to do this set the order to last:

```
vim:
  pkg:
    - installed
    - order: last
```

Substantial testing has gone into the state system and it is ready for real world usage. A great deal has been added to the documentation for states and the modules and functions available to states have been cleanly documented.

A number of State System bugs have also been founds and repaired, the output from the state system has also been refined to be extremely clear and concise.

Error reporting has also been introduced, issues found in sls files will now be clearly reported when executing Salt States.

Gentoo Support

Additional experimental support has been added for Gentoo. This is found in the contribution from Doug Renn, aka nestegg.

26.2.58 Salt 0.9.5 Release Notes

release 2012-01-15

Salt 0.9.5 is one of the largest steps forward in the development of Salt.

0.9.5 comes with many milestones, this release has seen the community of developers grow out to an international team of 46 code contributors and has many feature additions, feature enhancements, bug fixes and speed improvements.

Warning: Be sure to *read the upgrade instructions* about the switch to msgpack before upgrading!

Community

Nothing has proven to have more value to the development of Salt that the outstanding community that has been growing at such a great pace around Salt. This has proven not only that Salt has great value, but also the expandability of Salt is as exponential as I originally intended.

0.9.5 has received over 600 additional commits since 0.9.4 with a swath of new committers. The following individuals have contributed to the development of 0.9.5:

- Aaron Bull Schaefer
- Antti Kaihola
- Bas Tichelaar
- Brad Barden
- Brian Wagner
- Byron Clark
- Chris Scheller
- Christer Edwards
- Clint Savage
- Corey Quinn
- David Boucha
- Eivind Uggedal
- Eric Poelke
- Evan Borgstrom
- Jed Glazner
- Jeff Schroeder
- Jeffrey C. Ollie
- Jonas Buckner
- Kent Tenney
- Martin Schnabel
- Maxim Burgerhout
- Mitch Anderson
- Nathaniel Whiteinge
- Seth House
- Thomas S Hatch
- Thomas Schreiber
- Tor Hveem
- lzyeval

- syphernl

This makes 21 new developers since 0.9.4 was released!

To keep up with the growing community follow Salt on Ohloh (http://www.ohloh.net/p/salt), to join the Salt development community, fork Salt on Github, and get coding (https://github.com/saltstack/salt)!

Major Features

SPEED! Pickle to msgpack

For a few months now we have been talking about moving away from Python pickles for network serialization, but a preferred serialization format had not yet been found. After an extensive performance testing period involving everything from JSON to protocol buffers, a clear winner emerged. Message Pack (http://msgpack.org/) proved to not only be the fastest and most compact, but also the most ``salt like''. Message Pack is simple, and the code involved is very small. The msgpack library for Python has been added directly to Salt.

This move introduces a few changes to Salt. First off, Salt is no longer a ``noarch'' package, since the msgpack lib is written in C. Salt 0.9.5 will also have compatibility issues with 0.9.4 with the default configuration.

We have gone through great lengths to avoid backwards compatibility issues with Salt, but changing the serialization medium was going to create issues regardless. Salt 0.9.5 is somewhat backwards compatible with earlier minions. A 0.9.5 master can command older minions, but only if the `serial` config value in the master is set to `pickle`. This will tell the master to publish messages in pickle format and will allow the master to receive messages in both msgpack and pickle formats.

Therefore **the suggested methods for upgrading** are either to just upgrade everything at once, or:

1. Upgrade the master to 0.9.5
2. Set `serial` to `pickle` in the master config
3. Upgrade the minions
4. Remove the `serial` option from the master config

Since pickles can be used as a security exploit the ability for a master to accept pickles from minions at all will be removed in a future release.

C Bindings for YAML

All of the YAML rendering is now done with the YAML C bindings. This speeds up all of the sls files when running states.

Experimental Windows Support

David Boucha has worked tirelessly to bring initial support to Salt for Microsoft Windows operating systems. Right now the Salt Minion can run as a native Windows service and accept commands.

In the weeks and months to come Windows will receive the full treatment and will have support for Salt States and more robust support for managing Windows systems. This is a big step forward for Salt to move entirely outside of the Unix world, and proves Salt is a viable cross platform solution. Big Thanks to Dave for his contribution here!

Dynamic Module Distribution

Many Salt users have expressed the desire to have Salt distribute in-house modules, states, renderers, returners, and grains. This support has been added in a number of ways:

Modules via States Now when salt modules are deployed to a minion via the state system as a file, then the modules will be automatically loaded into the active running minion - no restart required - and into the active running state. So custom state modules can be deployed and used in the same state run.

Modules via Module Environment Directories Under the file_roots each environment can now have directories that are used to deploy large groups of modules. These directories sync modules at the beginning of a state run on the minion, or can be manually synced via the Salt module `salt.modules.saltutil.sync_all`.

The directories are named:

- `_modules`
- `_states`
- `_grains`
- `_renderers`
- `_returners`

The modules are pushed to their respective scopes on the minions.

Module Reloading

Modules can now be reloaded without restarting the minion, this is done by calling the `salt.modules.sys.reload_modules` function.

But wait, there's more! Now when a salt module of any type is added via states the modules will be automatically reloaded, allowing for modules to be laid down with states and then immediately used.

Finally, all modules are reloaded when modules are dynamically distributed from the salt master.

Enable / Disable Added to Service

A great deal of demand has existed for adding the capability to set services to be started at boot in the service module. This feature also comes with an overhaul of the service modules and initial systemd support.

This means that the `service state` can now accept `- enable: True` to make sure a service is enabled at boot, and `- enable: False` to make sure it is disabled.

Compound Target

A new target type has been added to the lineup, the compound target. In previous versions the desired minions could only be targeted via a single specific target type, but now many target specifications can be declared.

These targets can also be separated by and/or operators, so certain properties can be used to omit a node:

```
salt -C 'webserv* and G@os:Debian or E@db.*' test.ping
```

will match all minions with ids starting with webserv via a glob and minions matching the `os:Debian` grain. Or minions that match the `db.*` regular expression.

Node Groups

Often the convenience of having a predefined group of minions to execute targets on is desired. This can be accomplished with the new nodegroups feature. Nodegroups allow for predefined compound targets to be declared in the master configuration file:

```
nodegroups:
  group1: 'L@foo.domain.com,bar.domain.com,baz.domain.com and bl*.domain.com'
  group2: 'G@os:Debian and foo.domain.com'
```

And then used via the `-N` option:

```
salt -N group1 test.ping
```

Minion Side Data Store

The data module introduces the initial approach into storing persistent data on the minions, specific to the minions. This allows for data to be stored on minions that can be accessed from the master or from the minion.

The Minion datastore is young, and will eventually provide an interface similar to a more mature key/value pair server.

Major Grains Improvement

The Salt grains have been overhauled to include a massive amount of extra data. this includes hardware data, os data and salt specific data.

Salt -Q is Useful Now

In the past the salt query system, which would display the data from recent executions would be displayed in pure Python, and it was unreadable.

0.9.5 has added the outputter system to the `-Q` option, thus enabling the salt query system to return readable output.

Packaging Updates

Huge strides have been made in packaging Salt for distributions. These additions are thanks to our wonderful community where the work to set up packages has proceeded tirelessly.

FreeBSD

Salt on FreeBSD? There a port for that:

http://svnweb.freebsd.org/ports/head/sysutils/py-salt/

This port was developed and added by Christer Edwards. This also marks the first time Salt has been included in an upstream packaging system!

Fedora and Red Hat Enterprise

Salt packages have been prepared for inclusion in the Fedora Project and in EPEL for Red Hat Enterprise 5 and 6. These packages are the result of the efforts made by Clint Savage (herlo).

Debian/Ubuntu

A team of many contributors have assisted in developing packages for Debian and Ubuntu. Salt is still actively seeking inclusion in upstream Debian and Ubuntu and the package data that has been prepared is being pushed through the needed channels for inclusion.

These packages have been prepared with the help of:

- Corey
- Aaron Toponce
- and`

More to Come

We are actively seeking inclusion in more distributions. Primarily getting Salt into Gentoo, SUSE, OpenBSD, and preparing Solaris support are all turning into higher priorities.

Refinement

Salt continues to be refined into a faster, more stable and more usable application. 0.9.5 comes with more debug logging, more bug fixes and more complete support.

More Testing, More BugFixes

0.9.5 comes with more bugfixes due to more testing than any previous release. The growing community and the introduction a a dedicated QA environment have unearthed many issues that were hiding under the covers. This has further refined and cleaned the state interface, taking care of things from minor visual issues to repairing misleading data.

Custom Exceptions

A custom exception module has been added to throw salt specific exceptions. This allows Salt to give much more granular error information.

New Modules

`data` The new data module manages a persistent datastore on the minion. Big thanks to bastichelaar for his help refining this module

`freebsdkmod` FreeBSD kernel modules can now be managed in the same way Salt handles Linux kernel modules.

This module was contributed thanks to the efforts of Christer Edwards

gentoo_service Support has been added for managing services in Gentoo. Now Gentoo services can be started, stopped, restarted, enabled, disabled, and viewed.

pip The pip module introduces management for pip installed applications. Thanks goes to whitinge for the addition of the pip module

rh_service The rh_service module enables Red Hat and Fedora specific service management. Now Red Hat like systems come with extensive management of the classic init system used by Red Hat

saltutil The saltutil module has been added as a place to hold functions used in the maintenance and management of salt itself. Saltutil is used to salt the salt minion. The saltutil module is presently used only to sync extension modules from the master server.

systemd Systemd support has been added to Salt, now systems using this next generation init system are supported on systems running systemd.

virtualenv The virtualenv module has been added to allow salt to create virtual Python environments. Thanks goes to whitinge for the addition of the virtualenv module

win_disk Support for gathering disk information on Microsoft Windows minions The windows modules come courtesy of Utah_Dave

win_service The win_service module adds service support to Salt for Microsoft Windows services

win_useradd Salt can now manage local users on Microsoft Windows Systems

yumpkg5 The yumpkg module introduces in 0.9.4 uses the yum API to interact with the yum package manager. Unfortunately, on Red Hat 5 systems salt does not have access to the yum API because the yum API is running under Python 2.4 and Salt needs to run under Python 2.6.

The yumpkg5 module bypasses this issue by shelling out to yum on systems where the yum API is not available.

New States

mysql_database The new mysql_database state adds the ability to systems running a mysql server to manage the existence of mysql databases.

The mysql states are thanks to syphernl

mysql_user The mysql_user state enables mysql user management.

virtualenv The virtualenv state can manage the state of Python virtual environments. Thanks to Whitinge for the virtualenv state

New Returners

cassandra_returner A returner allowing Salt to send data to a cassandra server. Thanks to Byron Clark for contributing this returner

26.2.59 Salt 0.9.6 Release Notes

release 2012-01-21

Salt 0.9.6 is a release targeting a few bugs and changes. This is primarily targeting an issue found in the names declaration in the state system. But a few other bugs were also repaired, like missing support for grains in extmods.

Due to a conflict in distribution packaging msgpack will no longer be bundled with Salt, and is required as a dependency.

New Features

HTTP and ftp support in files.managed

Now under the source option in the file.managed state a HTTP or ftp address can be used instead of a file located on the salt master.

Allow Multiple Returners

Now the returner interface can define multiple returners, and will also return data back to the master, making the process less ambiguous.

Minion Memory Improvements

A number of modules have been taken out of the minion if the underlying systems required by said modules are not present on the minion system. A number of other modules need to be stripped out in this same way which should continue to make the minion more efficient.

Minions Can Locally Cache Return Data

A new option, cache_jobs, has been added to the minion to allow for all of the historically run jobs to cache on the minion, allowing for looking up historic returns. By default cache_jobs is set to False.

Pure Python Template Support For file.managed

Templates in the file.managed state can now be defined in a Python script. This script needs to have a run function that returns the string that needs to be in the named file.

26.2.60 Salt 0.9.7 Release Notes

release 2012-02-15

Salt 0.9.7 is here! The latest iteration of Salt brings more features and many fixes. This release is a great refinement over 0.9.6, adding many conveniences under the hood, as well as some features that make working with Salt much better.

A few highlights include the new Job system, refinements to the requisite system in states, the `mod_init` interface for states, external node classification, search path to managed files in the file state, and refinements and additions to dynamic module loading.

0.9.7 also introduces the long developed (and oft changed) unit test framework and the initial unit tests.

Major Features

Salt Jobs Interface

The new jobs interface makes the management of running executions much cleaner and more transparent. Building on the existing execution framework the jobs system allows clear introspection into the active running state of the running Salt interface.

The Jobs interface is centered in the new minion side proc system. The minions now store msgpack serialized files under `/var/cache/salt/proc`. These files keep track of the active state of processes on the minion.

Functions in the saltutil Module A number of functions have been added to the saltutil module to manage and view the jobs:

`running` - Returns the data of all running jobs that are found in the proc directory.

`find_job` - Returns specific data about a certain job based on job id.

`signal_job` - Allows for a given jid to be sent a signal.

`term_job` - Sends a termination signal (`SIGTERM, 15`) to the process controlling the specified job.

`kill_job` Sends a kill signal (`SIGKILL, 9`) to the process controlling the specified job.

The jobs Runner

A convenience runner front end and reporting system has been added as well. The jobs runner contains functions to make viewing data easier and cleaner.

The jobs runner contains a number of functions...

active The active function runs `saltutil.running` on all minions and formats the return data about all running jobs in a much more usable and compact format. The active function will also compare jobs that have returned and jobs that are still running, making it easier to see what systems have completed a job and what systems are still being waited on.

lookup_jid When jobs are executed the return data is sent back to the master and cached. By default is is cached for 24 hours, but this can be configured via the `keep_jobs` option in the master configuration.

Using the `lookup_jid` runner will display the same return data that the initial job invocation with the salt command would display.

list_jobs Before finding a historic job, it may be required to find the job id. `list_jobs` will parse the cached execution data and display all of the job data for jobs that have already, or partially returned.

External Node Classification

Salt can now use external node classifiers like Cobbler's `cobbler-ext-nodes`.

Salt uses specific data from the external node classifier. In particular the classes value denotes which sls modules to run, and the environment value sets to another environment.

An external node classification can be set in the master configuration file via the `external_nodes` option: http://salt.readthedocs.org/en/latest/ref/configuration/master.html#external-nodes

External nodes are loaded in addition to the top files. If it is intended to only use external nodes, do not deploy any top files.

State Mod Init System

An issue arose with the pkg state. Every time a package was run Salt would need to refresh the package database. This made systems with slower package metadata refresh speeds much slower to work with. To alleviate this issue the `mod_init` interface has been added to salt states.

The `mod_init` interface is a function that can be added to a state file. This function is called with the first state called. In the case of the pkg state, the `mod_init` function sets up a tag which makes the package database only refresh on the first attempt to install a package.

In a nutshell, the `mod_init` interface allows a state to run any command that only needs to be run once, or can be used to set up an environment for working with the state.

Source File Search Path

The file state continues to be refined, adding speed and capabilities. This release adds the ability to pass a list to the source option. This list is then iterated over until the source file is found, and the first found file is used.

The new syntax looks like this:

```
/etc/httpd/conf/httpd.conf:
  file:
    - managed
    - source:
      - salt://httpd/httpd.conf
      - http://myserver/httpd.conf: md5=8c1fe119e6f1fd96bc06614473509bf1
```

The source option can take sources in the list from the salt file server as well as an arbitrary web source. If using an arbitrary web source the checksum needs to be passed as well for file verification.

Refinements to the Requisite System

A few discrepancies were still lingering in the requisite system, in particular, it was not possible to have a `require` and a `watch` requisite declared in the same state declaration.

This issue has been alleviated, as well as making the requisite system run more quickly.

Initial Unit Testing Framework

Because of the module system, and the need to test real scenarios, the development of a viable unit testing system has been difficult, but unit testing has finally arrived. Only a small amount of unit testing coverage has been developed, much more coverage will be in place soon.

A huge thanks goes out to those who have helped with unit testing, and the contributions that have been made to get us where we are. Without these contributions unit tests would still be in the dark.

Compound Targets Expanded

Originally only support for `and` and `or` were available in the compound target. 0.9.7 adds the capability to negate compound targets with `not`.

Nodegroups in the Top File

Previously the nodegroups defined in the master configuration file could not be used to match nodes for states. The nodegroups support has been expanded and the nodegroups defined in the master configuration can now be used to match minions in the top file.

26.2.61 Salt 0.9.8 Release Notes

release 2012-03-21

Salt 0.9.8 is a big step forward, with many additions and enhancements, as well as a number of precursors to advanced future developments.

This version of Salt adds much more power to the command line, making the old hard timeout issues a thing of the past and adds keyword argument support. These additions are also available in the salt client API, making the available API tools much more powerful.

The new pillar system allows for data to be stored on the master and assigned to minions in a granular way similar to the state system. It also allows flexibility for users who want to keep data out of their state tree similar to `external lookup' functionality in other tools.

A new way to extend requisites was added, the ``requisite in'' statement. This makes adding requires or watch statements to external state decs much easier.

Additions to requisites making them much more powerful have been added as well as improved error checking for sls files in the state system. A new provider system has been added to allow for redirecting what modules run in the background for individual states.

Support for OpenSUSE has been added and support for Solaris has begun serious development. Windows support has been significantly enhanced as well.

The matcher and target systems have received a great deal of attention. The default behavior of grain matching has changed slightly to reflect the rest of salt and the compound matcher system has been refined.

A number of impressive features with keyword arguments have been added to both the CLI and to the state system. This makes states much more powerful and flexible while maintaining the simple configuration everyone loves.

The new batch size capability allows for executions to be rolled through a group of targeted minions a percentage or specific number at a time. This was added to prevent the ``thundering herd'' problem when targeting large numbers of minions for things like service restarts or file downloads.

Upgrade Considerations

Upgrade Issues

There was a previously missed oversight which could cause a newer minion to crash an older master. That oversight has been resolved so the version incompatibility issue will no longer occur. When upgrading to 0.9.8 make sure to upgrade the master first, followed by the minions.

Debian/Ubuntu Packages

The original Debian/Ubuntu packages were called salt and included all salt applications. New packages in the ppa are split by function. If an old salt package is installed then it should be manually removed and the new split packages need to be freshly installed.

On the master:

```
# apt-get purge salt
# apt-get install salt-{master,minion}
```

On the minions:

```
# apt-get purge salt
# apt-get install salt-minion
```

And on any Syndics:

```
# apt-get install salt-syndic
```

The official Salt PPA for Ubuntu is located at: https://launchpad.net/~saltstack/+archive/salt

Major Features

Pillar

Pillar offers an interface to declare variable data on the master that is then assigned to the minions. The pillar data is made available to all modules, states, sls files etc. It is compiled on the master and is declared using the existing renderer system. This means that learning pillar should be fairly trivial to those already familiar with salt states.

CLI Additions

The `salt` command has received a serious overhaul and is more powerful than ever. Data is returned to the terminal as it is received, and the salt command will now wait for all running minions to return data before stopping. This makes adding very large *--timeout* arguments completely unnecessary and gets rid of long running operations returning empty `{}` when the timeout is exceeded.

When calling salt via sudo, the user originally running salt is saved to the log for auditing purposes. This makes it easy to see who ran what by just looking through the minion logs.

The *salt-key* command gained the *-D* and *--delete-all* arguments for removing all keys. Be careful with this one!

Running States Without a Master

The addition of running states without a salt-master has been added to 0.9.8. This feature allows for the unmodified salt state tree to be read locally from a minion. The result is that the UNMODIFIED state tree has just become portable, allowing minions to have a local copy of states or to manage states without a master entirely.

This is accomplished via the new file client interface in Salt that allows for the `salt://` URI to be redirected to custom interfaces. This means that there are now two interfaces for the salt file server, calling the master or looking in a local, minion defined `file_roots`.

This new feature can be used by modifying the minion config to point to a local `file_roots` and setting the `file_client` option to `local`.

Keyword Arguments and States

State modules now accept the `**kwargs` argument. This results in all data in a sls file assigned to a state being made available to the state function.

This passes data in a transparent way back to the modules executing the logic. In particular, this allows adding arguments to the `pkg.install` module that enable more advanced and granular controls with respect to what the state is capable of.

An example of this along with the new debconf module for installing ldap client packages on Debian:

```
ldap-client-packages:
  pkg:
    - debconf: salt://debconf/ldap-client.ans
    - installed
    - names:
      - nslcd
      - libpam-ldapd
      - libnss-ldapd
```

Keyword Arguments and the CLI

In the past it was required that all arguments be passed in the proper order to the *salt* and *salt-call* commands. As of 0.9.8, keyword arguments can be passed in the form of `kwarg=argument`.

```
# salt -G 'type:dev' git.clone \
    repository=https://github.com/saltstack/salt.git cwd=/tmp/salt user=jeff
```

Matcher Refinements and Changes

A number of fixes and changes have been applied to the Matcher system. The most noteworthy is the change in the grain matcher. The grain matcher used to use a regular expression to match the passed data to a grain, but now defaults to a shell glob like the majority of match interfaces in Salt. A new option is available that still uses the old style regex matching to grain data called `grain-pcre`. To use regex matching in compound matches use the letter *P*.

For example, this would match any ArchLinux or Fedora minions:

```
# salt --grain-pcre 'os:(Arch:Fed).*' test.ping
```

And the associated compound matcher suitable for `top.sls` is *P*:

```
P@os:(Arch|Fed).*
```

NOTE: Changing the grains matcher from pcre to glob is backwards incompatible.

Support has been added for matching minions with Yahoo's range library. This is handled by passing range syntax with *-R* or *--range* arguments to salt.

More information at: https://github.com/ytoolshed/range/wiki/%22yamlfile%22-module-file-spec

Requisite ``in''

A new means to updating requisite statements has been added to make adding watchers and requires to external states easier. Before 0.9.8 the only way to extend the states that were watched by a state outside of the sls was to use an extend statement:

```
include:
  - http
extend:
  apache:
    service:
      - watch:
        - pkg: tomcat

tomcat:
  pkg:
    - installed
```

But the new `Requisite in` statement allows for easier extends for requisites:

```
include:
  - http

tomcat:
  pkg:
    - installed
    - watch_in:
      - service: apache
```

Requisite in is part of the extend system, so still remember to always include the sls that is being extended!

Providers

Salt predetermines what modules should be mapped to what uses based on the properties of a system. These determinations are generally made for modules that provide things like package and service management. The apt module maps to pkg on Debian and the yum module maps to pkg on Fedora for instance.

Sometimes in states, it may be necessary for a non-default module to be used for the desired functionality. For instance, an Arch Linux system may have been set up with systemd support. Instead of using the default service module detected for Arch Linux, the systemd module can be used:

```
http:
  service:
    - running
    - enable: True
    - provider: systemd
```

Default providers can also be defined in the minion config file:

```
providers:
  service: systemd
```

When default providers are passed in the minion config, then those providers will be applied to all functionality in Salt, this means that the functions called by the minion will use these modules, as well as states.

Requisite Glob Matching

Requisites can now be defined with glob expansion. This means that if there are many requisites, they can be defined on a single line.

To watch all files in a directory:

```
http:
  service:
    - running
    - enable: True
    - watch:
      - file: /etc/http/conf.d/*
```

This example will watch all defined files that match the glob `/etc/http/conf.d/*`

Batch Size

The new batch size option allows commands to be executed while maintaining that only so many hosts are executing the command at one time. This option can take a percentage or a finite number:

```
salt '*' -b 10 test.ping

salt -G 'os:RedHat' --batch-size 25% apache.signal restart
```

This will only run test.ping on 10 of the targeted minions at a time and then restart apache on 25% of the minions matching `os:RedHat` at a time and work through them all until the task is complete. This makes jobs like rolling web server restarts behind a load balancer or doing maintenance on BSD firewalls using carp much easier with salt.

Module Updates

This is a list of notable, but non-exhaustive updates with new and existing modules.

Windows support has seen a flurry of support this release cycle. We've gained all new *file*, *network*, and *shadow* modules. Please note that these are still a work in progress.

For our ruby users, new *rvm* and *gem* modules have been added along with the *associated states*

The *virt* module gained basic Xen support.

The *yum* module gained Scientific Linux support.

The *pkg* module on Debian, Ubuntu, and derivatives force apt to run in a non-interactive mode. This prevents issues when package installation waits for confirmation.

A *pkg* module for OpenSUSE's zypper was added.

The *service* module on Ubuntu natively supports upstart.

A new *debconf* module was contributed by our community for more advanced control over deb package deployments on Debian based distributions.

The *mysql.user* state and *mysql* module gained a *password_hash* argument.

The *cmd* module and state gained a *shell* keyword argument for specifying a shell other than `/bin/sh` on Linux / Unix systems.

New *git* and *mercurial* modules have been added for fans of distributed version control.

In Progress Development

Master Side State Compiling

While we feel strongly that the advantages gained with minion side state compiling are very critical, it does prevent certain features that may be desired. 0.9.8 has support for initial master side state compiling, but many more components still need to be developed, it is hoped that these can be finished for 0.9.9.

The goal is that states can be compiled on both the master and the minion allowing for compilation to be split between master and minion. Why will this be great? It will allow storing sensitive data on the master and sending it to some minions without all minions having access to it. This will be good for handling ssl certificates on front-end web servers for instance.

Solaris Support

Salt 0.9.8 sees the introduction of basic Solaris support. The daemon runs well, but grains and more of the modules need updating and testing.

Windows Support

Salt states on windows are now much more viable thanks to contributions from our community! States for file, service, local user, and local group management are more fully fleshed out along with network and disk modules. Windows users can also now manage registry entries using the new ``reg'' module.

26.2.62 Salt 0.9.9 Release Notes

release 2012-04-27

0.9.9 is out and comes with some serious bug fixes and even more serious features. This release is the last major feature release before 1.0.0 and could be considered the 1.0.0 release candidate.

A few updates include more advanced kwargs support, the ability for salt states to more safely configure a running salt minion, better job directory management and the new state test interface.

Many new tests have been added as well, including the new minion swarm test that allows for easier testing of Salt working with large groups of minions. This means that if you have experienced stability issues with Salt before, particularly in larger deployments, that these bugs have been tested for, found, and killed.

Major Features

State Test Interface

Until 0.9.9 the only option when running states to see what was going to be changed was to print out the highstate with state.show_highstate and manually look it over. But now states can be run to discover what is going to be changed.

Passing the option `test=True` to many of the state functions will now cause the salt state system to only check for what is going to be changed and report on those changes.

```
salt '*' state.highstate test=True
```

Now states that would have made changes report them back in yellow.

State Syntax Update

A shorthand syntax has been added to sls files, and it will be the default syntax in documentation going forward. The old syntax is still fully supported and will not be deprecated, but it is recommended to move to the new syntax in the future. This change moves the state function up into the state name using a dot notation. This is in-line with how state functions are generally referred to as well:

The new way:

```
/etc/sudoers:
  file.present:
    - source: salt://sudo/sudoers
    - user: root
    - mode: 400
```

Use and Use_in Requisites

Two new requisite statements are available in 0.9.9. The use and use_in requisite and requisite-in allow for the transparent duplication of data between states. When a state ``uses'' another state it copies the other state's arguments as defaults. This was created in direct response to the new network state, and allows for many network interfaces to be configured in the same way easily. A simple example:

```
root_file:
  file.absent:
    - name: /tmp/nothing
    - user: root
    - mode: 644
    - group: root
    - use_in:
      - file: /etc/vimrc

fred_file:
  file.absent:
    - name: /tmp/nothing
    - user: fred
    - group: marketing
    - mode: 660
```

```
/files/marketing/district7.rst:
  file.present:
    - source: salt://marketing/district7.rst
    - template: jinja
    - use:
      - file: fred_file

/etc/vimrc:
  file.present:
    - source: salt://edit/vimrc
```

This makes the 2 lower state decs inherit the options from their respectively ``used'' state decs.

Network State

The new network state allows for the configuration of network devices via salt states and the ip salt module. This addition has been given to the project by Jeff Hutchins and Bret Palsson from Jive Communications.

Currently the only network configuration backend available is for Red Hat based systems, like Red Hat Enterprise, CentOS, and Fedora.

Exponential Jobs

Originally the jobs executed were stored on the master in the format: `<cachedir>/jobs/jid/{minion ids}` But this format restricted the number of jobs in the cache to the number of subdirectories allowed on the filesystem. Ext3 for instance limits subdirectories to 32000. To combat this the new format for 0.9.9 is: `<cachedir>/jobs/jid_hash[:2]/jid_hash[2:]/{minion ids}` So that now the number of maximum jobs that can be run before the cleanup cycle hits the job directory is substantially higher.

ssh_auth Additions

The original ssh_auth state was limited to accepting only arguments to apply to a public key, and the key itself. This was restrictive due to the way the we learned that many people were using the state, so the key section has been expanded to accept options and arguments to the key that over ride arguments passed in the state. This gives substantial power to using ssh_auth with names:

```
sshkeys:
  ssh_auth:
    - present
    - user: backup
    - enc: ssh-dss
    - options:
      - option1="value1"
      - option2="value2 flag2"
    - comment: backup
    - names:
      - AAAAB3NzaC1yc2EAAAABIwAAAQEAlyE26SMFFVY5YJvnL7AF5CRTPtAigSW1U887ASfBt6FDa7Qr1YdO5ochiLoz8aSi
      - AAAAB3NzaC1yc2EAAAABIwAAAQEAlyE26SMFFVY5YJvnL7AF5CRTPtAigSW1U887ASfBt6FDa7Qr1YdO5ochiLoz8aSi
      - ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAQEAlyE26SMFFVY5YJvnL7AF5CRTPtAigSW1U887ASfBt6FDa7Qr1YdO5och
      - ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAQEAlyE26SMFFVY5YJvnL7AF5CRTPtAigSW1U887ASfBt6FDa7Qr1YdO5och
      - option3="value3",option4="value4 flag4" ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAQEAlyE26SMFFVY5YJvn
      - option3="value3" ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAQEAlyE26SMFFVY5YJvnL7AF5CRTPtAigSW1U887ASf
```

LocalClient Additions

To follow up the recent additions in 0.9.8 of additional kwargs support, 0.9.9 also adds the capability to send kwargs into commands via a dict. This addition to the LocalClient api can be used like so:

```
import salt.client

client = salt.client.LocalClient('/etc/salt/master')
ret = client.cmd('*', 'cmd.run', ['ls -l'], kwarg={'cwd': '/etc'})
```

This update has been added to all cmd methods in the LocalClient class.

Better Self Salting

One problem faced with running Salt states, is that it has been difficult to manage the Salt minion via states, this is due to the fact that if the minion is called to restart while a state run is happening then the state run would be killed. 0.9.9 slightly changes the process scope of the state runs, so now when salt is executing states it can safely restart the salt-minion daemon.

In addition to daemonizing the state run, the apt module also daemonizes. This update makes it possible to cleanly update the salt-minion package on Debian/Ubuntu systems without leaving apt in an inconsistent state or killing the active minion process mid-execution.

Wildcards for SLS Modules

Now, when including sls modules in include statements or in the top file, shell globs can be used. This can greatly simplify listing matched sls modules in the top file and include statements:

```
base:
  '*':
    - files*
    - core*
```

```
include:
  - users.dev.*
  - apache.ser*
```

External Pillar

Since the pillar data is just, data, it does not need to come expressly from the pillar interface. The external pillar system allows for hooks to be added making it possible to extract pillar data from any arbitrary external interface. The external pillar interface is configured via the `ext_pillar` option. Currently interfaces exist to gather external pillar data via hiera or via a shell command that sends yaml data to the terminal:

```
ext_pillar:
  - cmd_yaml: cat /etc/salt/ext.yaml
  - hiera: /etc/hirea.yaml
```

The initial external pillar interfaces and extra interfaces can be added to the file salt/pillar.py, it is planned to add more external pillar interfaces. If the need arises a new module loader interface will be created in the future to manage external pillar interfaces.

Single State Executions

The new state.single function allows for single states to be cleanly executed. This is a great tool for setting up a small group of states on a system or for testing out the behavior of single states:

```
salt '*' state.single user.present name=wade uid=2000
```

The test interface functions here as well, so changes can also be tested against as:

```
salt '*' state.single user.present name=wade uid=2000 test=True
```

New Tests

A few exciting new test interfaces have been added, the minion swarm allows not only testing of larger loads, but also allows users to see how Salt behaves with large groups of minions without having to create a large deployment.

Minion Swarm

The minion swarm test system allows for large groups of minions to be tested against easily without requiring large numbers of servers or virtual machines. The minion swarm creates as many minions as a system can handle and roots them in the /tmp directory and connects them to a master.

The benefit here is that we were able to replicate issues that happen only when there are large numbers of minions. A number of elusive bugs which were causing stability issues in masters and minions have since been hunted down. Bugs that used to take careful watch by users over several days can now be reliably replicated in minutes, and fixed in minutes.

Using the swarm is easy, make sure a master is up for the swarm to connect to, and then use the minionswarm.py script in the tests directory to spin up as many minions as you want. Remember, this is a fork bomb, don't spin up more than your hardware can handle!

```
python minionswarm.py -m 20 --master salt-master
```

Shell Tests

The new Shell testing system allows us to test the behavior of commands executed from a high level. This allows for the high level testing of salt runners and commands like salt-key.

Client Tests

Tests have been added to test the aspects of the client APIs and ensure that the client calls work, and that they manage passed data, in a desirable way.

See also:

`Legacy salt-cloud release docs`

See also:

Legacy salt-api release docs

CHAPTER 27

Salt Based Projects

A number of unofficial open source projects, based on Salt, or written to enhance Salt have been created.

27.1 Salt Sandbox

Created by Aaron Bull Schaefer, aka ``elasticdog''.

https://github.com/elasticdog/salt-sandbox

Salt Sandbox is a multi-VM Vagrant-based Salt development environment used for creating and testing new Salt state modules outside of your production environment. It's also a great way to learn firsthand about Salt and its remote execution capabilities.

Salt Sandbox will set up three separate virtual machines:

- salt.example.com - the Salt master server
- minion1.example.com - the first Salt minion machine
- minion2.example.com - the second Salt minion machine

These VMs can be used in conjunction to segregate and test your modules based on node groups, top file environments, grain values, etc. You can even test modules on different Linux distributions or release versions to better match your production infrastructure.

Security disclosure policy

email security@saltstack.com

gpg key ID 4EA0793D

gpg key fingerprint 8ABE 4EFC F0F4 B24B FF2A AF90 D570 F2D3 4EA0 793D

gpg public key:

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG/MacGPG2 v2.0.22 (Darwin)

mQINBFO15mMBEADa3CfQwk5ED9wAQ8fFDku277CegG3U1hVGdcxqKNvucblwoKCb
hRK6u9ihgaO9V9duV2glwgjytiBI/z6lyWqdaD37YXG/gTL+9Md+qdSDeaOa/9eg
7y+g4P+FvU9HWUlujRVlofUn5Dj/IZgUywbxwEybutuzvvFVTzsn+DFVwTH34Qoh
QIuNzQCSEz3Lhh8zq9LqkNy91ZZQO1ZIUrypafspH6GBHHcE8msBFgYiNBnVcUFH
u0r4j1Rav+621EtD5GZsOt05+NJI8pkaC/dDKjURcuiV6bhmeSpNzLaXUhwx6f29
Vhag5JhVGGNQxlRTxNEM86HEFp+4zJQ8m/wRDrGX5IAHsdESdhP+ljDVlAAX/ttP
/Ucl2fgpTnDKVHOA00E515Q87ZHv6awJ3GL1veqi8zfsLaag7rw1TuuHyGLOPkDt
t5PAjsS9R3KI7pGnhqI6bTOi591odUdgzUhZChWUUX1VStiIDi2jCvyoOOLMOGS5
AEYXuWYP7KgujZCDRaTNqRDdgPd93Mh9JI8UmkzXDUgijdzVpzPjYgFaWtyK8lsc
Fizqe3/Yzf9RCVX/lmRbiEH+ql/zSxcWlBQd17PKaL+TisQFXcmQzccYgAxFbj2r
QHp5ABEu9YjFme2Jzun7Mv9V4qo3JF5dmnUk31yupZeAOGZkirIsaWC3hwARAQAB
tDBTYWx0U3RhY2sgU2VjdXJpdHkgVGVhbSA8c2VjdXJpdHlAc2FsdHN0YWNrLmNv
bT6JAj4EEwECACgFAlO15mMCGwMFCQeGH4AGCwkIBwMCBhUIAgkKCwQWAgMBAh4B
AheAAAoJENVw8tNOoHk9z/MP/2vzY27fmVxU5X8joiiturjlgEqQw41IYEmWv1Bw
4WVXYCHP1yu/1MC1uuvOmOd5BlI8YO2C2oyW7d1B0NorguPtz55b7jabCElekVCh
h/H4ZVThiwqgPpthRv/2npXjIm7SLSs/kuaXo6Qy2JpszwDVFw+xCRVL0tH9KJxz
HuNBeVq7abWD5fzIWkmGM9hicG/R2D0RIlco1Q0VNKy8klG+pOFOW886KnwkSPc7
JUYp1oUlHsSlhTmkLEG54cyVzrTP/XuZuyMTdtyTc3mfgW0adneAL6MARtC5UB/h
q+v9dqMf4iD3wY6ctu8KWE8Vo5MUEsNNO9EA2dUR88LwFZ3ZnnXdQkizgR/Aa515
dm17vlNkSoomYCo84eN7GOTfxWcq+iXYSWcKWT4X+h/ra+LmNndQWQBRebVUtbKE
ZDwKmiQz/5LY5EhlWcuU4lVmMSFpWXt5FR/PtzgTdZAo9QKkBjcv97LYbXvsPI69
El1BLAg+m+1UpE1L7zJT1il6PqVyEFAWBxW46wXCCkGssFsvz2yRp0PDX8A6u4yq
rTkt09uYht1is61joLDJ/kq3+6k8gJWkDOW+2NMrmf+/qcdYCMYXmrtOpg/wF27W
GMNAkbdyzgeX/MbUBCGCMdzhevRuivOI5bu4vT5s3KdshG+yhzV45bapKRd5VN+1
mZRquQINBFO15mMBEAC5UuLii9ZLz6qHfIJp35IOW9U8SOf7QFhzXR7NZ3DmJsd3
f6Nb/habQFIHjm3K9wbpj+FvaW2oWRlFVvYdzjUq6c82GUUjW1dnqgUvFwdmM835
1n0YQ2TonmyaF882RvsRZrbJ65uvy7SQxlouXaAYOdqwLsPxBEOyOnMPSktW5V2U
IWyxsNP3sADchWIGq9p5D3Y/loyIMsS1dj+TjoQZOKSj7CuRT98+8yhGAY8YBEXu
9r3I9o6mDkuPpAljuMc8r09Im6az2egtK/szKt4Hy1bpSSBZU4W/XR7XwQNywmb3
wxjmYT6Od3Mwj0jtzc3gQiH8hcEy3+BO+NNmyzFVyIwOLziwjmEcw62S57wYKUVn
HD2nglMsQa8Ve0e6ABBMEY7zGEGStva59rfgeh0jUMJiccGiUDTMs0tdkC6knYKb
u/fdRqNYFoNuDcSeLEw4DdCuP01l2W4yY+fiK6hAcL25amjzc+yYo9eaaqTn6RAT
```

```
bzdhHQZdpAMxY+vNT0+NhP1Zo5gYBMR65Zp/VhFsf67ijb03FUtdw9N8dHwiR2m8
vVA8kO/gCD6wS2p9RdXqrJ9JhnHYWjiVuXR+f755ZAndyQfRtowMdQIoiXuJEXYw
6XN+/BX81gJaynJYc0uw0MnxWQX+A5m8HqEsbIFUXBYXPgbwXTm7c4IHGgXXdwAR
AQABiQIlBBgBAgAPBQJTteZjAhsMBQkHhh+AAAoJENVw8tNOoHk91rcQAIhxLv4g
duF/J1Cyf6Wixz4rqslBQ7DgNztdIUMjCThg3eB6pvIzY5d3DNROmwU5JvGP1rEw
hNiJhgBDFaB0J/y28uSci+orhKDTHb/cn30IxfuAuqrv9dujvmlgM7JUswOtLZhs
5FYGa6v1RORRWhUx2PQsF6ORg22QAaagc7OlaO3BXBoiE/FWsnEQCUsc7GnnPqi7
um45OJl/pJntsBUKvivEU20fj7j1UpjmeWz56NcjXoKtEvGh99gM5W2nSMLE3aPw
vcKhS4yRyLjOe19NfYbtID8m8oshUDji0XjQ1z5NdGcf2V1YNGHU5xyK6zwyGxgV
xZqaWnbhDTu1UnYBna8BiUobkuqclb4T9k2WjbrUSmTwKixokCOirFDZvqISkgmN
r6/g3w2TRi11/LtbUciF0FN2pd7rj5mWrOBPEFYJmrB6SQeswWNhr5RIsXrQd/Ho
zvNm0HnUNEe6w5YBfA6sXQy8B0Zs6pcgLogkFB15TuHIIIpxIsVRv5z8SlEnB7HQ
Io9hZT58yjhekJuzVQB9loU0C/W0lzci/pXTt6fd9puYQe1DG37pSifRG6kfHxrR
if6nRyrfdTlawqbqdkoqFDmEybAM9/hv3BqriGahGGH/hgplNQbYoXfNwYMYaHuB
aSkJvrOQW8bpuAzgVyd7TyNFv+t1kLlfaRYJ
=wBTJ
-----END PGP PUBLIC KEY BLOCK-----
```

The SaltStack Security Team is available at security@saltstack.com for security-related bug reports or questions.

We request the disclosure of any security-related bugs or issues be reported non-publicly until such time as the issue can be resolved and a security-fix release can be prepared. At that time we will release the fix and make a public announcement with upgrade instructions and download locations.

28.1 Security response procedure

SaltStack takes security and the trust of our customers and users very seriously. Our disclosure policy is intended to resolve security issues as quickly and safely as is possible.

1. A security report sent to security@saltstack.com is assigned to a team member. This person is the primary contact for questions and will coordinate the fix, release, and announcement.
2. The reported issue is reproduced and confirmed. A list of affected projects and releases is made.
3. Fixes are implemented for all affected projects and releases that are actively supported. Back-ports of the fix are made to any old releases that are actively supported.
4. Packagers are notified via the salt-packagers mailing list that an issue was reported and resolved, and that an announcement is incoming.
5. A new release is created and pushed to all affected repositories. The release documentation provides a full description of the issue, plus any upgrade instructions or other relevant details.
6. An announcement is made to the salt-users and salt-announce mailing lists. The announcement contains a description of the issue and a link to the full release documentation and download locations.

28.2 Receiving security announcemnts

The fastest place to receive security announcements is via the salt-announce mailing list. This list is low-traffic.

CHAPTER 29

Frequently Asked Questions

FAQ

- Frequently Asked Questions
  - Is Salt open-core?
  - What ports should I open on my firewall?
  - I'm seeing weird behavior (including but not limited to packages not installing their users properly)
  - My script runs every time I run a *state.highstate*. Why?
  - When I run *test.ping*, why don't the Minions that aren't responding return anything? Returning `False` would be helpful.
  - How does Salt determine the Minion's id?
  - I'm trying to manage packages/services but I get an error saying that the state is not available. Why?
  - I'm using gitfs and my custom modules/states/etc are not syncing. Why?
  - Why aren't my custom modules/states/etc. available on my Minions?
  - Module `X` isn't available, even though the shell command it uses is installed. Why?
  - Can I run different versions of Salt on my Master and Minion?
  - Does Salt support backing up managed files?
  - What is the best way to restart a Salt daemon using Salt?
    - Linux/Unix
    - Windows
  - Salting the Salt Master

29.1 Is Salt open-core?

No. Salt is 100% committed to being open-source, including all of our APIs and the `Halite' web interface which was introduced in version 0.17.0. It is developed under the Apache 2.0 license, allowing it to be used in both open and proprietary projects.

29.2 What ports should I open on my firewall?

Minions need to be able to connect to the Master on TCP ports 4505 and 4506. Minions do not need any inbound ports open. More detailed information on firewall settings can be found *here*.

29.3 I'm seeing weird behavior (including but not limited to packages not installing their users properly)

This is often caused by SELinux. Try disabling SELinux or putting it in permissive mode and see if the weird behavior goes away.

29.4 My script runs every time I run a *state.highstate*. Why?

You are probably using `cmd.run` rather than `cmd.wait`. A `cmd.wait` state will only run when there has been a change in a state that it is watching.

A `cmd.run` state will run the corresponding command *every time* (unless it is prevented from running by the `unless` or `onlyif` arguments).

More details can be found in the documentation for the `cmd` states.

29.5 When I run *test.ping*, why don't the Minions that aren't responding return anything? Returning `False` would be helpful.

When you run *test.ping* the Master tells Minions to run commands/functions, and listens for the return data, printing it to the screen when it is received. If it doesn't receive anything back, it doesn't have anything to display for that Minion.

There are a couple options for getting information on Minions that are not responding. One is to use the verbose (`-v`) option when you run salt commands, as it will display ``Minion did not return'' for any Minions which time out.

```
salt -v '*' pkg.install zsh
```

Another option is to use the `manage.down` runner:

```
salt-run manage.down
```

Also, if the Master is under heavy load, it is possible that the CLI will exit without displaying return data for all targeted Minions. However, this doesn't mean that the Minions did not return; this only means that the Salt CLI timed out waiting for a response. Minions will still send their return data back to the Master once the job completes. If any expected Minions are missing from the CLI output, the `jobs.list_jobs` runner can be used to show the job IDs of the jobs that have been run, and the `jobs.lookup_jid` runner can be used to get the return data for that job.

```
salt-run jobs.list_jobs
salt-run jobs.lookup_jid 20130916125524463507
```

If you find that you are often missing Minion return data on the CLI, only to find it with the jobs runners, then this may be a sign that the `worker_threads` value may need to be increased in the master config file. Additionally, running your Salt CLI commands with the `-t` option will make Salt wait longer for the return data before the CLI command exits. For instance, the below command will wait up to 60 seconds for the Minions to return:

```
salt -t 60 '*' test.ping
```

29.6 How does Salt determine the Minion's id?

If the Minion id is not configured explicitly (using the `id` parameter), Salt will determine the id based on the hostname. Exactly how this is determined varies a little between operating systems and is described in detail *here*.

29.7 I'm trying to manage packages/services but I get an error saying that the state is not available. Why?

Salt detects the Minion's operating system and assigns the correct package or service management module based on what is detected. However, for certain custom spins and OS derivatives this detection fails. In cases like this, an issue should be opened on our tracker, with the following information:

1. The output of the following command:

```
salt <minion_id> grains.items | grep os
```

2. The contents of `/etc/lsb-release`, if present on the Minion.

29.8 I'm using gitfs and my custom modules/states/etc are not syncing. Why?

In versions of Salt 0.16.3 or older, there is a bug in *gitfs* which can affect the syncing of custom types. Upgrading to 0.16.4 or newer will fix this.

29.9 Why aren't my custom modules/states/etc. available on my Minions?

Custom modules are only synced to Minions when `state.highstate`, `saltutil.sync_modules`, or `saltutil.sync_all` is run. Similarly, custom states are only synced to Minions when `state.highstate`, `saltutil.sync_states`, or `saltutil.sync_all` is run.

Other custom types (renderers, outputters, etc.) have similar behavior, see the documentation for the `saltutil` module for more information.

29.10 Module X isn't available, even though the shell command it uses is installed. Why?

This is most likely a PATH issue. Did you custom-compile the software which the module requires? RHEL/CentOS/etc. in particular override the root user's path in `/etc/init.d/functions`, setting it to `/sbin:/usr/sbin:/bin:/usr/bin`, making software installed into `/usr/local/bin` unavailable to Salt when the Minion is started using the initscript. In version 2014.1.0, Salt will have a better solution for these sort of PATH-related issues, but recompiling the software to install it into a location within the PATH should resolve the issue in the meantime. Alternatively, you can create a symbolic link within the PATH using a `file.symlink` state.

```
/usr/bin/foo:
  file.symlink:
    - target: /usr/local/bin/foo
```

29.11 Can I run different versions of Salt on my Master and Minion?

This depends on the versions. In general, it is recommended that Master and Minion versions match.

When upgrading Salt, the master(s) should always be upgraded first. Backwards compatibility for minions running newer versions of salt than their masters is not guaranteed.

Whenever possible, backwards compatibility between new masters and old minions will be preserved. Generally, the only exception to this policy is in case of a security vulnerability.

Recent examples of backwards compatibility breakage include the 0.17.1 release (where all backwards compatibility was broken due to a security fix), and the 2014.1.0 release (which retained compatibility between 2014.1.0 masters and 0.17 minions, but broke compatibility for 2014.1.0 minions and older masters).

29.12 Does Salt support backing up managed files?

Yes. Salt provides an easy to use addition to your file.managed states that allow you to back up files via *backup_mode*, backup_mode can be configured on a per state basis, or in the minion config (note that if set in the minion config this would simply be the default method to use, you still need to specify that the file should be backed up!).

29.13 What is the best way to restart a Salt daemon using Salt?

Updating the salt-minion package requires a restart of the salt-minion service. But restarting the service while in the middle of a state run interrupts the process of the minion running states and sending results back to the master. It's a tricky problem to solve, and we're working on it, but in the meantime one way of handling this (on Linux and UNIX-based operating systems) is to use **at** (a job scheduler which predates cron) to schedule a restart of the service. **at** is not installed by default on most distros, and requires a service to be running (usually called **atd**) in order to schedule jobs. Here's an example of how to upgrade the salt-minion package at the end of a Salt run, and schedule a service restart for one minute after the package update completes.

29.13.1 Linux/Unix

```
salt-minion:
  pkg.installed:
    - name: salt-minion
    - version: 2014.1.7-3.el6
    - order: last
  service.running:
    - name: salt-minion
    - require:
      - pkg: salt-minion
  cmd.wait:
    - name: echo service salt-minion restart | at now + 1 minute
```

```
    - watch:
      - pkg: salt-minion
```

To ensure that **at** is installed and **atd** is running, the following states can be used (be sure to double-check the package name and service name for the distro the minion is running, in case they differ from the example below.

```
at:
  pkg.installed:
    - name: at
  service.running:
    - name: atd
    - enable: True
```

An alternatvie to using the **atd** daemon is to fork and disown the process.

```
restart_minion:
  cmd.run:
    - name: |
        nohup /bin/sh -c 'sleep 10 && salt-call --local service.restart salt-minion'
    - python_shell: True
    - order: last
```

29.13.2 Windows

For Windows machines, restarting the minion at can be accomplished by adding the following state:

```
schedule-start:
  cmd.run:
    - name: 'start powershell "Restart-Service -Name salt-minion"'
    - order: last
```

or running immediately from the command line:

```
salt -G kernel:Windows cmd.run 'start powershell "Restart-Service -Name salt-minion"'
```

29.14 Salting the Salt Master

In order to configure a master server via states, the Salt master can also be ``salted'' in order to enforce state on the Salt master as well as the Salt minions. Salting the Salt master requires a Salt minion to be installed on the same machine as the Salt master. Once the Salt minion is installed, the minion configuration file must be pointed to the local Salt master:

```
master: 127.0.0.1
```

Once the Salt master has been ``salted'' with a Salt minion, it can be targeted just like any other minion. If the minion on the salted master is running, the minion can be targeted via any usual `salt` command. Additionally, the `salt-call` command can execute operations to enforce state on the salted master without requiring the minion to be running.

More information about salting the Salt master can be found in the salt-formula for salt itself:

https://github.com/saltstack-formulas/salt-formula

CHAPTER 30

Glossary

Auto-Order The evaluation of states in the order that they are defined in a SLS file. *See also*: *ordering*.

Bootstrap A stand-alone Salt project which can download and install a Salt master and/or a Salt minion onto a host. *See also*: salt-bootstrap.

Compound Matcher A combination of many target definitions that can be combined with boolean operators. *See also*: *targeting*.

EAuth Shorthand for `external authentication'. A system for calling to a system outside of Salt in order to authenticate users and determine if they are allowed to issue particular commands to Salt. *See also*: *external auth*.

Environment A directory tree containing state files which can be applied to minions. *See also*: *top file*.

Execution Module A Python module that contains execution functions which directly perform various system-management tasks on a server. Salt ships with a number of execution modules but users can also write their own execution modules to perform specialized tasks. *See also*: *the list of execution modules*.

Execution Function A Python function inside an Execution Module that may take arguments and performs specific system-management tasks. *See also*: *the list of execution modules*.

External Job Cache An external data-store that can archive information about jobs that have been run. A default returner. *See also*: `ext_job_cache`, *the list of returners*.

External Pillar A module that accepts arbitrary arguments and returns a dictionary. The dictionary is automatically added to a pillar for a minion.

Event A notice emitted onto an event bus. Events are often driven by requests for actions to occur on a minion or master and the results of those actions. *See also*: *Salt Reactor*.

File Server A local or remote location for storing both Salt-specific files such as top files or SLS files as well as files that can be distributed to minions, such as system configuration files. *See also*: *Salt's file server*.

Grain A key-value pair which contains a fact about a system, such as its hostname, network addresses. *See also*: *targeting with grains*.

Halite The Salt GUI. *See also*: Halite.

Jinja A templating language which allows variables and simple logic to be dynamically inserted into static text files when they are rendered. *See also*: `Salt's Jinja documentation`.

Job The complete set of tasks to be performed by the execution of a Salt command are a single job. *See also*: `jobs runner`.

Job ID A unique identifier to represent a given *job*.

Highdata The data structure in a SLS file the represents a set of state declarations. *See also*: *state layers*.

Highstate The collection of states to be applied to a system. *See also*: *state layers*.

Low State The collection of processed states after requisites and order are evaluated. *See also*: *state layers*.

Master A central Salt daemon which from which commands can be issued to listening minions.

Masterless A minion which does not require a Salt master to operate. All configuration is local. *See also*: `file_client`.

Master Tops A system for the master that allows hooks into external systems to generate top file data.

Mine A facility to collect arbitrary data from minions and store that data on the master. This data is then available to all other minions. [Sometimes referred to as Salt Mine.] *See also*: *Salt Mine*.

Minion A server running a Salt minion daemon which can listen to commands from a master and perform the requested tasks. Generally, minions are servers which are to be controlled using Salt.

Minion ID A globally unique identifier for a minion. *See also*: `id`.

Multi-Master The ability for a minion to be actively connected to multiple Salt masters at the same time in high-availability environments.

Node Group A pre-defined group of minions declared in the master configuration file. *See also*: *targeting*.

Outputter A formatter for defining the characteristics of output data from a Salt command. *See also*: *list of outputters*.

Overstate A system by which a Master can issue function calls to minions in a deterministic order. *See also*: *overstate*.

Peer Communication The ability for minions to communicate directly with other minions instead of brokering commands through the Salt master. *See also*: *peer communication*.

Pillar A simple key-value store for user-defined data to be made available to a minion. Often used to store and distribute sensitive data to minions. *See also*: *Pillar*, *list of Pillar modules*.

Proxy Minion A minion which can control devices that are unable to run a Salt minion locally, such as routers and switches.

PyDSL A Pythonic domain-specific-language used as a Salt renderer. PyDSL can be used in cases where adding pure Python into SLS files is beneficial. *See also*: `PyDSL`.

Reactor An interface for listening to events and defining actions that Salt should taken upon receipt of given events. *See also*: *Reactor*.

Render Pipe Allows SLS files to be rendered by multiple renderers, with each renderer receiving the output of the previous. *See also*: *composing renderers*.

Renderer Responsible for translating a given data serialization format such as YAML or JSON into a Python data structure that can be consumed by Salt. *See also*: *list of renderers*.

Returner Allows for the results of a Salt command to be sent to a given data-store such as a database or log file for archival. *See also*: *list of returners*.

Roster A flat-file list of target hosts. (Currently only used by salt-ssh.)

Runner Module A module containing a set of runner functions. *See also*: *list of runner modules*.

Runner Function A function which is is called by the **`salt-run`** command and executes on the master instead of on a minion. *See also*: *Runner Module*.

Salt Cloud A suite of tools used to create and deploy systems on many hosted cloud providers. *See also*: *salt-cloud*.

Salt SSH A configuration management and remote orchestration system that does not require that any software besides SSH be installed on systems to be controlled.

Salt Thin A subset of the normal Salt distribution that does not include any transport routines. A Salt Thin bundle can be dropped onto a host and used directly without any requirement that the host be connected to a network. Used by Salt SSH. *See also*: `thin runner`.

Salt Virt Used to manage the creation and deployment of virtual machines onto a set of host machines. Often used to create and deploy private clouds. *See also*: `virt runner`.

SLS Module Contains a set of *state declarations*.

State Declaration A data structure which contains a unique ID and describes one or more states of a system such as ensuring that a package is installed or a user is defined. *See also*: *highstate structure*.

State Module A module which contains a set of state functions. *See also*: *list of state modules*.

State Function A function contained inside a *state module* which can manages the application of a particular state to a system. State functions frequently call out to one or more *execution modules* to perform a given task.

State Run The application of a set of states on a set of systems.

State Compiler Translates *highdata* into lowdata.

Syndic A forwarder which can relay messages between tiered masters. **See also**: *Syndic*.

Target Minion(s) to which a given salt command will apply. *See also*: *targeting*.

Top File Determines which SLS files should be applied to various systems and organizes those groups of systems into environments. *See also*: *top file*, *list of master top modules*.

Worker A master process which can send notices and receive replies from minions. *See also*: `worker_threads`.

__virtual__ A function in a module that is called on module load to determine whether or not the module should be available to a minion. This function commonly contains logic to determine if all requirements for a module are available, such as external libraries.

a

salt.auth.auto, 305
salt.auth.django, 305
salt.auth.keystone, 307
salt.auth.ldap, 307
salt.auth.mysql, 307
salt.auth.pam, 308
salt.auth.pki, 309
salt.auth.stormpath_mod, 309
salt.auth.yubico, 310

b

salt.beacons.btmp, 1446
salt.beacons.diskusage, 1446
salt.beacons.inotify, 1447
salt.beacons.journald, 1447
salt.beacons.load, 1447
salt.beacons.network_info, 1448
salt.beacons.service, 1449
salt.beacons.sh, 1449
salt.beacons.twilio_txt_msg, 1450
salt.beacons.wtmp, 1450

c

salt.cloud.clouds.aliyun, 345
salt.cloud.clouds.botocore_aws, 348
salt.cloud.clouds.cloudstack, 348
salt.cloud.clouds.digital_ocean, 350
salt.cloud.clouds.digital_ocean_v2, 351
salt.cloud.clouds.ec2, 352
salt.cloud.clouds.gce, 359
salt.cloud.clouds.gogrid, 364
salt.cloud.clouds.libcloud_aws, 365
salt.cloud.clouds.linode, 366
salt.cloud.clouds.lxc, 369
salt.cloud.clouds.msazure, 369
salt.cloud.clouds.nova, 371
salt.cloud.clouds.opennebula, 374
salt.cloud.clouds.openstack, 375
salt.cloud.clouds.parallels, 377
salt.cloud.clouds.proxmox, 379
salt.cloud.clouds.pyrax, 381
salt.cloud.clouds.rackspace, 381
salt.cloud.clouds.saltify, 383
salt.cloud.clouds.softlayer, 383
salt.cloud.clouds.softlayer_hw, 384
salt.cloud.clouds.vsphere, 386

e

salt.exceptions, 476

f

salt.fileserver.gitfs, 466
salt.fileserver.hgfs, 467
salt.fileserver.minionfs, 468
salt.fileserver.roots, 469
salt.fileserver.s3fs, 470
salt.fileserver.svnfs, 471

l

salt.log.handlers.logstash_mod, 458
salt.log.handlers.sentry_mod, 460

m

salt.modules.aliases, 485
salt.modules.alternatives, 486
salt.modules.apache, 487
salt.modules.aptpkg, 489
salt.modules.archive, 495
salt.modules.artifactory, 499
salt.modules.at, 500
salt.modules.augeas_cfg, 501
salt.modules.aws_sqs, 502
salt.modules.blockdev, 504
salt.modules.bluez, 504
salt.modules.boto_asg, 506
salt.modules.boto_cfn, 508
salt.modules.boto_cloudwatch, 509
salt.modules.boto_dynamodb, 511
salt.modules.boto_elasticache, 512

salt.modules.boto_elb, 514
salt.modules.boto_iam, 517
salt.modules.boto_route53, 520
salt.modules.boto_secgroup, 521
salt.modules.boto_sns, 522
salt.modules.boto_sqs, 523
salt.modules.boto_vpc, 525
salt.modules.brew, 532
salt.modules.bridge, 535
salt.modules.bsd_shadow, 536
salt.modules.btrfs, 537
salt.modules.cassandra, 539
salt.modules.chef, 540
salt.modules.chocolatey, 542
salt.modules.cloud, 545
salt.modules.cmdmod, 547
salt.modules.composer, 553
salt.modules.config, 554
salt.modules.container_resource, 557
salt.modules.cp, 557
salt.modules.cpan, 560
salt.modules.cron, 561
salt.modules.cyg, 562
salt.modules.daemontools, 563
salt.modules.darwin_sysctl, 565
salt.modules.data, 565
salt.modules.ddns, 566
salt.modules.deb_apache, 567
salt.modules.debconfmod, 568
salt.modules.debian_ip, 569
salt.modules.debian_service, 570
salt.modules.defaults, 572
salt.modules.devmap, 573
salt.modules.dig, 573
salt.modules.disk, 574
salt.modules.djangomod, 575
salt.modules.dnsmasq, 575
salt.modules.dnsutil, 576
salt.modules.dockerio, 578
salt.modules.dpkg, 589
salt.modules.drac, 590
salt.modules.drbd, 593
salt.modules.ebuild, 593
salt.modules.eix, 598
salt.modules.elasticsearch, 598
salt.modules.environ, 599
salt.modules.eselect, 600
salt.modules.etcd_mod, 601
salt.modules.event, 602
salt.modules.extfs, 603
salt.modules.file, 606
salt.modules.firewalld, 626
salt.modules.freebsd_sysctl, 628
salt.modules.freebsdjail, 629
salt.modules.freebsdkmod, 630
salt.modules.freebsdpkg, 631
salt.modules.freebsdports, 634
salt.modules.freebsdservice, 636
salt.modules.fsutils, 637
salt.modules.gem, 637
salt.modules.genesis, 640
salt.modules.gentoo_service, 641
salt.modules.gentoolkitmod, 642
salt.modules.git, 644
salt.modules.glance, 650
salt.modules.glusterfs, 651
salt.modules.gnomedesktop, 653
salt.modules.gpg, 655
salt.modules.grains, 660
salt.modules.groupadd, 663
salt.modules.grub_legacy, 664
salt.modules.guestfs, 665
salt.modules.hadoop, 665
salt.modules.haproxyconn, 666
salt.modules.hashutil, 667
salt.modules.hg, 668
salt.modules.hosts, 669
salt.modules.htpasswd, 670
salt.modules.http, 671
salt.modules.ilo, 672
salt.modules.img, 674
salt.modules.incron, 675
salt.modules.influx, 676
salt.modules.ini_manage, 680
salt.modules.introspect, 681
salt.modules.ipset, 682
salt.modules.iptables, 683
salt.modules.jboss7, 687
salt.modules.jboss7_cli, 690
salt.modules.junos, 692
salt.modules.key, 692
salt.modules.keyboard, 692
salt.modules.keystone, 693
salt.modules.kmod, 698
salt.modules.launchctl, 699
salt.modules.layman, 700
salt.modules.ldapmod, 701
salt.modules.linux_acl, 702
salt.modules.linux_lvm, 703
salt.modules.linux_sysctl, 704
salt.modules.localemod, 705
salt.modules.locate, 706
salt.modules.logadm, 707
salt.modules.logrotate, 707
salt.modules.lvs, 708
salt.modules.lxc, 710
salt.modules.mac_group, 723
salt.modules.mac_user, 724

salt.modules.macports, 726
salt.modules.makeconf, 728
salt.modules.match, 736
salt.modules.mdadm, 738
salt.modules.memcached, 740
salt.modules.mine, 741
salt.modules.mod_random, 742
salt.modules.modjk, 743
salt.modules.mongodb, 746
salt.modules.monit, 748
salt.modules.moosefs, 748
salt.modules.mount, 749
salt.modules.munin, 751
salt.modules.mysql, 751
salt.modules.nagios, 758
salt.modules.netbsd_sysctl, 760
salt.modules.netbsdservice, 760
salt.modules.netscaler, 762
salt.modules.network, 765
salt.modules.neutron, 769
salt.modules.nfs3, 787
salt.modules.nftables, 787
salt.modules.nginx, 792
salt.modules.nova, 792
salt.modules.npm, 797
salt.modules.omapi, 798
salt.modules.openbsd_sysctl, 799
salt.modules.openbsdpkg, 799
salt.modules.openbsdrcctl, 801
salt.modules.openbsdservice, 803
salt.modules.openstack_config, 804
salt.modules.oracle, 805
salt.modules.osxdesktop, 806
salt.modules.pacman, 807
salt.modules.pagerduty, 810
salt.modules.pam, 811
salt.modules.parted, 811
salt.modules.pecl, 815
salt.modules.pillar, 816
salt.modules.pip, 817
salt.modules.pkg, 479
salt.modules.pkg_resource, 822
salt.modules.pkgin, 823
salt.modules.pkgng, 825
salt.modules.pkgutil, 837
salt.modules.portage_config, 839
salt.modules.postfix, 840
salt.modules.postgres, 841
salt.modules.poudriere, 847
salt.modules.powerpath, 849
salt.modules.ps, 849
salt.modules.publish, 853
salt.modules.puppet, 854
salt.modules.pw_group, 855
salt.modules.pw_user, 856
salt.modules.pyenv, 858
salt.modules.qemu_img, 859
salt.modules.qemu_nbd, 860
salt.modules.quota, 860
salt.modules.rabbitmq, 861
salt.modules.raet_publish, 865
salt.modules.rbenv, 866
salt.modules.rdp, 867
salt.modules.redismod, 868
salt.modules.reg, 871
salt.modules.rest_package, 872
salt.modules.rest_sample, 873
salt.modules.rest_service, 873
salt.modules.ret, 873
salt.modules.rh_ip, 874
salt.modules.rh_service, 875
salt.modules.riak, 877
salt.modules.rpm, 878
salt.modules.rsync, 879
salt.modules.runit, 880
salt.modules.rvm, 881
salt.modules.s3, 884
salt.modules.saltcloudmod, 886
salt.modules.saltutil, 886
salt.modules.schedule, 890
salt.modules.scsi, 892
salt.modules.sdb, 892
salt.modules.seed, 893
salt.modules.selinux, 894
salt.modules.sensors, 895
salt.modules.serverdensity_device, 895
salt.modules.service, 896
salt.modules.shadow, 897
salt.modules.slack_notify, 899
salt.modules.smartos_imgadm, 900
salt.modules.smartos_vmadm, 901
salt.modules.smf, 902
salt.modules.smtp, 904
salt.modules.softwareupdate, 905
salt.modules.solaris_group, 907
salt.modules.solaris_shadow, 907
salt.modules.solaris_user, 908
salt.modules.solarisips, 910
salt.modules.solarispkg, 913
salt.modules.solr, 916
salt.modules.splunk_search, 922
salt.modules.sqlite3, 923
salt.modules.ssh, 924
salt.modules.state, 927
salt.modules.status, 931
salt.modules.sudo, 934
salt.modules.supervisord, 934
salt.modules.svn, 936

salt.modules.swift, 939
salt.modules.sysbench, 941
salt.modules.syslog_ng, 942
salt.modules.sysmod, 946
salt.modules.sysrc, 951
salt.modules.system, 952
salt.modules.systemd, 952
salt.modules.test, 955
salt.modules.test_virtual, 959
salt.modules.timezone, 959
salt.modules.tls, 960
salt.modules.tomcat, 964
salt.modules.twilio_notify, 968
salt.modules.upstart, 968
salt.modules.useradd, 971
salt.modules.uwsgi, 973
salt.modules.varnish, 973
salt.modules.vbox_guest, 974
salt.modules.virt, 976
salt.modules.virtualenv_mod, 983
salt.modules.win_autoruns, 984
salt.modules.win_disk, 984
salt.modules.win_dns_client, 984
salt.modules.win_file, 985
salt.modules.win_firewall, 990
salt.modules.win_groupadd, 991
salt.modules.win_ip, 991
salt.modules.win_network, 993
salt.modules.win_ntp, 995
salt.modules.win_path, 995
salt.modules.win_pkg, 996
salt.modules.win_repo, 998
salt.modules.win_servermanager, 999
salt.modules.win_service, 1000
salt.modules.win_shadow, 1002
salt.modules.win_status, 1002
salt.modules.win_system, 1003
salt.modules.win_timezone, 1005
salt.modules.win_update, 1006
salt.modules.win_useradd, 1008
salt.modules.xapi, 1010
salt.modules.xfs, 1014
salt.modules.xmpp, 1016
salt.modules.yumpkg, 1017
salt.modules.zcbuildout, 1026
salt.modules.zfs, 1028
salt.modules.zk_concurrency, 1030
salt.modules.znc, 1031
salt.modules.zpool, 1031
salt.modules.zypper, 1034

n

salt.netapi.rest_cherrypy.app, 1040
salt.netapi.rest_cherrypy.wsgi, 1043
salt.netapi.rest_tornado.saltnado, 1058
salt.netapi.rest_tornado.saltnado_websockets, 1059
salt.netapi.rest_wsgi, 1065

o

salt.output.compact, 1067
salt.output.highstate, 1068
salt.output.json_out, 1069
salt.output.key, 1069
salt.output.nested, 1069
salt.output.newline_values_only, 1070
salt.output.no_out, 1071
salt.output.no_return, 1071
salt.output.overstatestage, 1072
salt.output.pprint_out, 1072
salt.output.progress, 1072
salt.output.raw, 1072
salt.output.txt, 1073
salt.output.virt_query, 1073
salt.output.yaml_out, 1073

p

salt.pillar.cmd_json, 1076
salt.pillar.cmd_yaml, 1076
salt.pillar.cmd_yamlex, 1076
salt.pillar.cobbler, 1076
salt.pillar.django_orm, 1077
salt.pillar.ec2_pillar, 1079
salt.pillar.etcd_pillar, 1079
salt.pillar.file_tree, 1080
salt.pillar.foreman, 1082
salt.pillar.git_pillar, 1082
salt.pillar.hiera, 1084
salt.pillar.libvirt, 1084
salt.pillar.mongo, 1084
salt.pillar.mysql, 1085
salt.pillar.pepa, 1089
salt.pillar.pillar_ldap, 1093
salt.pillar.puppet, 1093
salt.pillar.reclass_adapter, 1094
salt.pillar.redismod, 1094
salt.pillar.s3, 1095
salt.pillar.svn_pillar, 1096
salt.pillar.varstack_pillar, 1097
salt.pillar.virtkey, 1097

r

salt.renderers.cheetah, 1100
salt.renderers.genshi, 1100
salt.renderers.gpg, 1100
salt.renderers.jinja, 1101
salt.renderers.json, 1106
salt.renderers.mako, 1107

salt.renderers.msgpack, 1107
salt.renderers.py, 1107
salt.renderers.pydsl, 1108
salt.renderers.pyobjects, 1113
salt.renderers.stateconf, 1116
salt.renderers.wempy, 1120
salt.renderers.yaml, 1121
salt.renderers.yamlex, 1122
salt.returners.carbon_return, 1128
salt.returners.cassandra_return, 1129
salt.returners.couchbase_return, 1130
salt.returners.couchdb_return, 1130
salt.returners.django_return, 1131
salt.returners.elasticsearch_return, 1132
salt.returners.etcd_return, 1133
salt.returners.kafka_return, 1134
salt.returners.local, 1134
salt.returners.local_cache, 1134
salt.returners.memcache_return, 1135
salt.returners.mongo_future_return, 1136
salt.returners.mongo_return, 1137
salt.returners.multi_returner, 1138
salt.returners.mysql, 1138
salt.returners.odbc, 1140
salt.returners.postgres, 1143
salt.returners.postgres_local_cache, 1145
salt.returners.redis_return, 1146
salt.returners.sentry_return, 1147
salt.returners.sms_return, 1148
salt.returners.smtp_return, 1148
salt.returners.sqlite3_return, 1149
salt.returners.syslog_return, 1151
salt.returners.xmpp_return, 1151
salt.roster.ansible, 1152
salt.roster.cache, 1154
salt.roster.cloud, 1155
salt.roster.clustershell, 1155
salt.roster.flat, 1155
salt.roster.scan, 1155
salt.runners.cache, 1157
salt.runners.cloud, 1158
salt.runners.doc, 1158
salt.runners.drac, 1159
salt.runners.error, 1159
salt.runners.f5, 1160
salt.runners.fileserver, 1161
salt.runners.git_pillar, 1164
salt.runners.http, 1164
salt.runners.jobs, 1165
salt.runners.launchd, 1166
salt.runners.lxc, 1166
salt.runners.manage, 1168
salt.runners.mine, 1170
salt.runners.nacl, 1170
salt.runners.network, 1171
salt.runners.pagerduty, 1171
salt.runners.pillar, 1173
salt.runners.queue, 1173
salt.runners.sdb, 1175
salt.runners.search, 1175
salt.runners.state, 1175
salt.runners.survey, 1179
salt.runners.test, 1179
salt.runners.thin, 1180
salt.runners.virt, 1180
salt.runners.winrepo, 1181

S

salt.sdb.etcd_db, 1450
salt.sdb.keyring_db, 1451
salt.sdb.memcached, 1452
salt.sdb.sqlite3, 1452
salt.states.alias, 1235
salt.states.alternatives, 1235
salt.states.apache, 1236
salt.states.apache_module, 1237
salt.states.apt, 1238
salt.states.archive, 1238
salt.states.artifactory, 1239
salt.states.at, 1240
salt.states.augeas, 1241
salt.states.aws_sqs, 1242
salt.states.blockdev, 1243
salt.states.boto_asg, 1244
salt.states.boto_cloudwatch_alarm, 1248
salt.states.boto_dynamodb, 1249
salt.states.boto_elasticache, 1252
salt.states.boto_elb, 1254
salt.states.boto_iam_role, 1258
salt.states.boto_lc, 1260
salt.states.boto_route53, 1262
salt.states.boto_secgroup, 1264
salt.states.boto_sns, 1266
salt.states.boto_sqs, 1267
salt.states.chef, 1268
salt.states.cloud, 1270
salt.states.cmd, 1271
salt.states.composer, 1278
salt.states.cron, 1279
salt.states.cyg, 1282
salt.states.ddns, 1283
salt.states.debconfmod, 1284
salt.states.disk, 1286
salt.states.dockerio, 1286
salt.states.drac, 1292
salt.states.environ, 1292

salt.states.eselect, 1293
salt.states.event, 1293
salt.states.file, 1294
salt.states.gem, 1314
salt.states.git, 1314
salt.states.glusterfs, 1317
salt.states.gnomedesktop, 1318
salt.states.grafana, 1320
salt.states.grains, 1323
salt.states.group, 1325
salt.states.hg, 1326
salt.states.hipchat, 1326
salt.states.host, 1327
salt.states.htpasswd, 1328
salt.states.http, 1328
salt.states.incron, 1329
salt.states.influxdb_database, 1330
salt.states.influxdb_user, 1330
salt.states.ini_manage, 1331
salt.states.ipset, 1332
salt.states.iptables, 1334
salt.states.jboss7, 1338
salt.states.keyboard, 1343
salt.states.keystone, 1344
salt.states.kmod, 1346
salt.states.layman, 1347
salt.states.libvirt, 1347
salt.states.linux_acl, 1347
salt.states.locale, 1348
salt.states.lvm, 1348
salt.states.lvs_server, 1349
salt.states.lvs_service, 1350
salt.states.lxc, 1350
salt.states.makeconf, 1353
salt.states.mdadm, 1354
salt.states.memcached, 1355
salt.states.modjk, 1355
salt.states.modjk_worker, 1356
salt.states.module, 1357
salt.states.mongodb_database, 1358
salt.states.mongodb_user, 1359
salt.states.mount, 1360
salt.states.mysql_database, 1361
salt.states.mysql_grants, 1361
salt.states.mysql_query, 1363
salt.states.mysql_user, 1364
salt.states.network, 1365
salt.states.nftables, 1368
salt.states.npm, 1371
salt.states.ntp, 1372
salt.states.openstack_config, 1372
salt.states.pagerduty, 1373
salt.states.pecl, 1374
salt.states.pip_state, 1374
salt.states.pkg, 1378
salt.states.pkgng, 1384
salt.states.pkgrepo, 1384
salt.states.portage_config, 1386
salt.states.ports, 1387
salt.states.postgres_database, 1388
salt.states.postgres_extension, 1389
salt.states.postgres_group, 1390
salt.states.postgres_schema, 1391
salt.states.postgres_user, 1392
salt.states.powerpath, 1393
salt.states.process, 1393
salt.states.pyenv, 1394
salt.states.pyrax_queues, 1395
salt.states.quota, 1395
salt.states.rabbitmq_cluster, 1396
salt.states.rabbitmq_plugin, 1396
salt.states.rabbitmq_policy, 1397
salt.states.rabbitmq_user, 1398
salt.states.rabbitmq_vhost, 1398
salt.states.rbenv, 1399
salt.states.rdp, 1400
salt.states.redismod, 1401
salt.states.reg, 1401
salt.states.rvm, 1402
salt.states.saltmod, 1404
salt.states.schedule, 1406
salt.states.selinux, 1408
salt.states.serverdensity_device, 1409
salt.states.service, 1410
salt.states.slack, 1411
salt.states.smtp, 1412
salt.states.splunk_search, 1412
salt.states.ssh_auth, 1413
salt.states.ssh_known_hosts, 1415
salt.states.stateconf, 1415
salt.states.status, 1416
salt.states.supervisord, 1416
salt.states.svn, 1417
salt.states.sysctl, 1418
salt.states.syslog_ng, 1418
salt.states.sysrc, 1419
salt.states.test, 1419
salt.states.timezone, 1421
salt.states.tomcat, 1421
salt.states.user, 1423
salt.states.vbox_guest, 1425
salt.states.virtualenv_mod, 1426
salt.states.win_dns_client, 1426
salt.states.win_firewall, 1427
salt.states.win_network, 1427
salt.states.win_path, 1428
salt.states.win_servermanager, 1429
salt.states.win_system, 1430

salt.states.win_update, 1430
salt.states.winrepo, 1432
salt.states.xmpp, 1433
salt.states.zcbuildout, 1433
salt.states.zk_concurrency, 1434

t

salt.tops.cobbler, 1441
salt.tops.ext_nodes, 1441
salt.tops.mongo, 1442
salt.tops.reclass_adapter, 1443

u

salt.utils.aggregation, 472
salt.utils.serializers, 478
salt.utils.serializers.json, 478
salt.utils.serializers.msgpack, 479
salt.utils.serializers.yaml, 478

w

salt.wheel.config, 1444
salt.wheel.error, 1444
salt.wheel.file_roots, 1444
salt.wheel.key, 1445
salt.wheel.minions, 1445
salt.wheel.pillar_roots, 1446

Symbols

--args-separator=ARGS_SEPARATOR
    salt command line option, 317
--async
    salt command line option, 316
--auto-create
    salt-key command line option, 326
--file-root=FILE_ROOT
    salt-call command line option, 314
--force-color
    salt command line option, 318, 332
    salt-call command line option, 315
    salt-cloud command line option, 321
    salt-key command line option, 325
--gen-keys-dir=GEN_KEYS_DIR
    salt-key command line option, 326
--gen-keys=GEN_KEYS
    salt-key command line option, 326
--gen-signature
    salt-key command line option, 326
--grain-pcre
    salt command line option, 317, 331
    salt-cp command line option, 323
--hard-crash
    salt-call command line option, 314
    salt-key command line option, 324
    salt-run command line option, 329
--hide-timeout
    salt command line option, 316
--id=ID
    salt-call command line option, 314
--include-all
    salt-key command line option, 326
--key-deploy
    command line option, 330
--keysize=KEYSIZE
    salt-key command line option, 326
--list-images=LIST_IMAGES
    salt-cloud command line option, 321
--list-locations=LIST_LOCATIONS
    salt-cloud command line option, 321
--list-profiles
    salt-cloud command line option, 321
--list-providers
    salt-cloud command line option, 321
--list-sizes=LIST_SIZES
    salt-cloud command line option, 321
--local
    salt-call command line option, 314
--log-file-level=LOG_LEVEL_LOGFILE
    salt command line option, 317, 331
    salt-api command line option, 334
    salt-call command line option, 315
    salt-cp command line option, 323
    salt-key command line option, 325
    salt-master command line option, 327
    salt-minion command line option, 328
    salt-run command line option, 329
    salt-syndic command line option, 333
--log-file=LOG_FILE
    salt command line option, 317, 331
    salt-api command line option, 334
    salt-call command line option, 315
    salt-cp command line option, 323
    salt-key command line option, 325
    salt-master command line option, 327
    salt-minion command line option, 328
    salt-run command line option, 329
    salt-syndic command line option, 333
--master=MASTER
    salt-call command line option, 314
--max-procs
    command line option, 330
--metadata
    salt-call command line option, 314
--no-color
    salt command line option, 318, 332
    salt-call command line option, 315
    salt-cloud command line option, 321
    salt-key command line option, 325
--out

salt command line option, 318, 331
salt-call command line option, 315
salt-cloud command line option, 321
salt-key command line option, 325
--out-file=OUTPUT_FILE, --output-file=OUTPUT_FILE
salt command line option, 318, 332
salt-call command line option, 315
salt-cloud command line option, 321
salt-key command line option, 325
--out-indent OUTPUT_INDENT, --output-indent OUTPUT_INDENT
salt command line option, 318, 332
salt-call command line option, 315
salt-cloud command line option, 321
salt-key command line option, 325
--passwd
command line option, 330
--pid-file PIDFILE
salt-master command line option, 327
salt-minion command line option, 328
salt-syndic command line option, 333
--pid-file=PIDFILE
salt-api command line option, 334
--pillar-root=PILLAR_ROOT
salt-call command line option, 314
--priv
command line option, 330
--priv=PRIV
salt-key command line option, 326
--pub=PUB
salt-key command line option, 326
--refresh, --refresh-cache
command line option, 330
--refresh-grains-cache
salt-call command line option, 314
--retcode-passthrough
salt-call command line option, 314
--return RETURNER
salt-call command line option, 314
--return=RETURNER
salt command line option, 317
--roster
command line option, 330
--roster-file
command line option, 330
--rotate-aes-key=ROTATE_AES_KEY
salt-key command line option, 324
--script-args=SCRIPT_ARGS
salt-cloud command line option, 320
--set-password=<USERNAME> <PROVIDER>
salt-cloud command line option, 321
--show-deploy-args
salt-cloud command line option, 320
--signature-path=SIGNATURE_PATH
salt-key command line option, 326
--skip-grains
salt-call command line option, 314
--state-output=STATE_OUTPUT
salt command line option, 316
--subset=SUBSET
salt command line option, 316
--version
salt command line option, 316, 330
salt-api command line option, 334
salt-call command line option, 314
salt-cloud command line option, 319
salt-cp command line option, 323
salt-key command line option, 324
salt-master command line option, 327
salt-minion command line option, 328
salt-run command line option, 329
salt-syndic command line option, 332
--versions-report
salt command line option, 316, 331
salt-api command line option, 334
salt-call command line option, 314
salt-cloud command line option, 319
salt-cp command line option, 323
salt-key command line option, 324
salt-master command line option, 327
salt-minion command line option, 328
salt-run command line option, 329
salt-syndic command line option, 333
-A, --accept-all
salt-key command line option, 325
-C, --compound
salt command line option, 318
-D, --delete-all
salt-key command line option, 326
-E, --pcre
salt command line option, 317, 331
salt-cp command line option, 323
-F, --finger-all
salt-key command line option, 326
-F, --full-query
salt-cloud command line option, 320
-G, --grain
salt command line option, 317, 331
salt-cp command line option, 323
-H, --hard
salt-cloud command line option, 320
-I, --pillar
salt command line option, 318
-L LOCATION, --location=LOCATION
salt-cloud command line option, 319
-L, --list
salt command line option, 317, 331
salt-cp command line option, 323

-L, --list-all
salt-key command line option, 325
-N, --nodegroup
salt command line option, 318, 331
salt-cp command line option, 323
-P, --parallel
salt-cloud command line option, 320
-P, --print-all
salt-key command line option, 326
-Q, --query
salt-cloud command line option, 320
-R, --range
salt command line option, 318, 331
salt-cp command line option, 323
-R, --reject-all
salt-key command line option, 326
-S, --ipcidr
salt command line option, 318
-S, --select-query
salt-cloud command line option, 320
-T, --make-token
salt command line option, 317
-a ACCEPT, --accept=ACCEPT
salt-key command line option, 325
-a ACTION, --action=ACTION
salt-cloud command line option, 320
-a EAUTH, --auth=EAUTH
salt command line option, 317
-b BATCH, --batch-size=BATCH
salt command line option, 317
-c CONFIG_DIR, --config-dir=CONFIG_dir
salt command line option, 316, 331
salt-api command line option, 334
salt-call command line option, 314
salt-cloud command line option, 319
salt-cp command line option, 323
salt-key command line option, 324
salt-master command line option, 327
salt-minion command line option, 328
salt-run command line option, 329
salt-syndic command line option, 333
-d DELETE, --delete=DELETE
salt-key command line option, 326
-d, --daemon
salt-api command line option, 334
salt-master command line option, 327
salt-minion command line option, 328
salt-syndic command line option, 333
-d, --destroy
salt-cloud command line option, 320
-d, --doc, --documentation
salt command line option, 317
salt-call command line option, 314
salt-run command line option, 329
-f FINGER, --finger=FINGER
salt-key command line option, 326
-f <FUNC-NAME> <PROVIDER>, --function=<FUNC-NAME> <PROVIDER>
salt-cloud command line option, 320
-g, --grains
salt-call command line option, 314
-h, --help
salt command line option, 316, 331
salt-api command line option, 334
salt-call command line option, 314
salt-cloud command line option, 319
salt-cp command line option, 323
salt-key command line option, 324
salt-master command line option, 327
salt-minion command line option, 328
salt-run command line option, 329
salt-syndic command line option, 333
-i, --ignore-host-keys
command line option, 330
-k, --keep-tmp
salt-cloud command line option, 320
-l ARG, --list=ARG
salt-key command line option, 325
-l LOG_LEVEL, --log-level=LOG_LEVEL
salt command line option, 317, 331
salt-api command line option, 334
salt-call command line option, 315
salt-cp command line option, 323
salt-master command line option, 327
salt-minion command line option, 328
salt-run command line option, 329
salt-syndic command line option, 333
-m MAP, --map=MAP
salt-cloud command line option, 320
-m MODULE_DIRS, --module-dirs=MODULE_DIRS
salt-call command line option, 314
-p PRINT, --print=PRINT
salt-key command line option, 326
-p PROFILE, --profile=PROFILE
salt-cloud command line option, 320
-q, --quiet
salt-key command line option, 324
-r REJECT, --reject=REJECT
salt-key command line option, 325
-r, --raw, --raw-shell
command line option, 330
-s, --static
salt command line option, 316
-t TIMEOUT, --timeout=TIMEOUT
salt command line option, 316
salt-cp command line option, 323
salt-run command line option, 329
-u USER, --user=USER

salt-key command line option, 324
salt-master command line option, 327
salt-minion command line option, 328
salt-syndic command line option, 333
-u, --update-bootstrap
salt-cloud command line option, 320
-v VERBOSE, --verbose
salt command line option, 316
-y, --assume-yes
salt-cloud command line option, 320
-y, --yes
salt-key command line option, 324
__virtual__, **1679**

A

A() (in module salt.modules.dig), 573
A() (in module salt.modules.dnsutil), 576
a2dismod() (in module salt.modules.deb_apache), 567
a2dissite() (in module salt.modules.deb_apache), 567
a2enmod() (in module salt.modules.deb_apache), 567
a2ensite() (in module salt.modules.deb_apache), 568
AAAA() (in module salt.modules.dig), 573
AAAA() (in module salt.modules.dnsutil), 576
abort_import() (in module salt.modules.solr), 917
absent() (in module salt.states.alias), 1235
absent() (in module salt.states.at), 1240
absent() (in module salt.states.aws_sqs), 1242
absent() (in module salt.states.boto_asg), 1247
absent() (in module salt.states.boto_cloudwatch_alarm), 1249
absent() (in module salt.states.boto_dynamodb), 1251
absent() (in module salt.states.boto_elasticache), 1253
absent() (in module salt.states.boto_elb), 1257
absent() (in module salt.states.boto_iam_role), 1259
absent() (in module salt.states.boto_lc), 1261
absent() (in module salt.states.boto_route53), 1263
absent() (in module salt.states.boto_secgroup), 1265
absent() (in module salt.states.boto_sns), 1267
absent() (in module salt.states.boto_sqs), 1268
absent() (in module salt.states.cloud), 1270
absent() (in module salt.states.cron), 1281
absent() (in module salt.states.ddns), 1284
absent() (in module salt.states.dockerio), 1287
absent() (in module salt.states.drac), 1292
absent() (in module salt.states.file), 1297
absent() (in module salt.states.grains), 1323
absent() (in module salt.states.group), 1325
absent() (in module salt.states.host), 1328
absent() (in module salt.states.incron), 1329
absent() (in module salt.states.influxdb_database), 1330
absent() (in module salt.states.influxdb_user), 1330
absent() (in module salt.states.ipset), 1333
absent() (in module salt.states.kmod), 1346
absent() (in module salt.states.layman), 1347
absent() (in module salt.states.linux_acl), 1347
absent() (in module salt.states.lvs_server), 1349
absent() (in module salt.states.lvs_service), 1350
absent() (in module salt.states.lxc), 1350
absent() (in module salt.states.makeconf), 1353
absent() (in module salt.states.mdadm), 1354
absent() (in module salt.states.memcached), 1355
absent() (in module salt.states.mongodb_database), 1358
absent() (in module salt.states.mongodb_user), 1359
absent() (in module salt.states.mysql_database), 1361
absent() (in module salt.states.mysql_grants), 1362
absent() (in module salt.states.mysql_user), 1364
absent() (in module salt.states.openstack_config), 1373
absent() (in module salt.states.pkgrepo), 1385
absent() (in module salt.states.postgres_database), 1388
absent() (in module salt.states.postgres_extension), 1389
absent() (in module salt.states.postgres_group), 1390
absent() (in module salt.states.postgres_schema), 1391
absent() (in module salt.states.postgres_user), 1392
absent() (in module salt.states.process), 1393
absent() (in module salt.states.pyenv), 1394
absent() (in module salt.states.pyrax_queues), 1395
absent() (in module salt.states.rabbitmq_policy), 1397
absent() (in module salt.states.rabbitmq_user), 1398
absent() (in module salt.states.rabbitmq_vhost), 1398
absent() (in module salt.states.rbenv), 1400
absent() (in module salt.states.redismod), 1401
absent() (in module salt.states.reg), 1401
absent() (in module salt.states.schedule), 1408
absent() (in module salt.states.splunk_search), 1413
absent() (in module salt.states.ssh_auth), 1414
absent() (in module salt.states.ssh_known_hosts), 1415
absent() (in module salt.states.sysrc), 1419
absent() (in module salt.states.user), 1424
absent() (in module salt.states.win_path), 1428
accept() (in module salt.wheel.key), 1445
accept_dict() (in module salt.wheel.key), 1445
acceptance_wait_time
conf/minion, 446
acceptance_wait_time_max
conf/minion, 446
access() (in module salt.modules.file), 606
accumulated() (in module salt.states.file), 1297
action() (in module salt.modules.cloud), 545
action() (in module salt.runners.cloud), 1158
action() (salt.cloud.CloudClient method), 343
activate() (in module salt.states.modjk_worker), 1356
active() (in module salt.modules.mount), 749
active() (in module salt.runners.jobs), 1165
active_tcp() (in module salt.modules.network), 765
add() (in module salt.modules.bridge), 535
add() (in module salt.modules.btrfs), 537
add() (in module salt.modules.git), 644
add() (in module salt.modules.groupadd), 663

add() (in module salt.modules.ipset), 682
add() (in module salt.modules.layman), 700
add() (in module salt.modules.mac_group), 723
add() (in module salt.modules.mac_user), 724
add() (in module salt.modules.memcached), 740
add() (in module salt.modules.pw_group), 855
add() (in module salt.modules.pw_user), 856
add() (in module salt.modules.schedule), 890
add() (in module salt.modules.solaris_group), 907
add() (in module salt.modules.solaris_user), 908
add() (in module salt.modules.supervisord), 934
add() (in module salt.modules.svn), 936
add() (in module salt.modules.useradd), 971
add() (in module salt.modules.win_groupadd), 991
add() (in module salt.modules.win_path), 996
add() (in module salt.modules.win_useradd), 1008
add() (in module salt.modules.zpool), 1031
add_argument() (salt.modules.syslog_ng.TypedParameterValue method), 944
add_child() (salt.modules.syslog_ng.Statement method), 943
add_dns() (in module salt.modules.win_dns_client), 984
add_gateway_router() (in module salt.modules.neutron), 770
add_host() (in module salt.modules.ddns), 567
add_host() (in module salt.modules.hosts), 669
add_host() (in module salt.modules.omapi), 798
add_interface_router() (in module salt.modules.neutron), 770
add_license() (in module salt.modules.powerpath), 849
add_lock() (in module salt.modules.zypper), 1034
add_parameter() (salt.modules.syslog_ng.Option method), 943
add_pkg() (in module salt.modules.pkg_resource), 822
add_pool_member() (in module salt.runners.f5), 1160
add_pool_member() (salt.runners.f5.F5Mgmt method), 1160
add_record() (in module salt.modules.boto_route53), 520
add_rule() (in module salt.modules.win_firewall), 990
add_rule() (in module salt.states.win_firewall), 1427
add_server() (in module salt.modules.lvs), 708
add_service() (in module salt.modules.firewalld), 626
add_service() (in module salt.modules.lvs), 708
add_user() (in module salt.modules.rabbitmq), 861
add_value() (salt.modules.syslog_ng.TypedParameter method), 944
add_vhost() (in module salt.modules.rabbitmq), 861
add_volume_bricks() (in module salt.modules.glusterfs), 651
add_volume_bricks() (in module salt.states.glusterfs), 1317
added() (salt.states.cyg.DictDiffer method), 1282
addgroup() (in module salt.modules.win_useradd), 1009
addif() (in module salt.modules.bridge), 535
additions_install() (in module salt.modules.vbox_guest), 974
additions_installed() (in module salt.states.vbox_guest), 1425
additions_mount() (in module salt.modules.vbox_guest), 975
additions_remove() (in module salt.modules.vbox_guest), 975
additions_removed() (in module salt.states.vbox_guest), 1425
additions_umount() (in module salt.modules.vbox_guest), 975
additions_version() (in module salt.modules.vbox_guest), 975
address_() (in module salt.modules.bluez), 504
adduser() (in module salt.modules.groupadd), 663
adduser() (in module salt.modules.win_groupadd), 991
Aggregate (class in salt.utils.aggregation), 474
aggregate() (in module salt.utils.aggregation), 474
align_check() (in module salt.modules.parted), 812
all_status() (in module salt.modules.status), 931
always_verify_signature
    conf/minion, 453
appdata_ptr (salt.auth.pam.PamConv attribute), 308
append() (in module salt.modules.file), 606
append() (in module salt.modules.grains), 660
append() (in module salt.modules.iptables), 683
append() (in module salt.modules.nftables), 787
append() (in module salt.states.file), 1298
append() (in module salt.states.grains), 1323
append() (in module salt.states.iptables), 1337
append() (in module salt.states.nftables), 1370
append_cflags() (in module salt.modules.makeconf), 728
append_cxxflags() (in module salt.modules.makeconf), 729
append_domain
    conf/minion, 445
append_emerge_default_opts() (in module salt.modules.makeconf), 729
append_features() (in module salt.modules.makeconf), 729
append_gentoo_mirrors() (in module salt.modules.makeconf), 729
append_makeopts() (in module salt.modules.makeconf), 729
append_to_package_conf() (in module salt.modules.portage_config), 839
append_use_flags() (in module salt.modules.portage_config), 839
append_var() (in module salt.modules.makeconf), 730
apply() (in module salt.wheel.config), 1444
apply_() (in module salt.modules.seed), 893
apply_() (in module salt.modules.state), 927

apply_network_profile() (in module salt.modules.lxc), 710
apply_network_settings() (in module salt.modules.debian_ip), 569
apply_network_settings() (in module salt.modules.rh_ip), 874
archive() (in module salt.modules.git), 644
archive() (in module salt.modules.hg), 668
arg() (in module salt.modules.test), 955
arg() (in module salt.runners.test), 1179
arg_repr() (in module salt.modules.test), 955
arg_type() (in module salt.modules.test), 955
argspec() (in module salt.modules.sysmod), 946
Argument (class in salt.modules.syslog_ng), 942
arp() (in module salt.modules.network), 766
ArtifactoryError, 499
as_list (salt.pillar.mysql.merger attribute), 1088
assemble() (in module salt.modules.mdadm), 738
assertion() (in module salt.modules.test), 955
assign() (in module salt.modules.darwin_sysctl), 565
assign() (in module salt.modules.freebsd_sysctl), 628
assign() (in module salt.modules.linux_sysctl), 704
assign() (in module salt.modules.netbsd_sysctl), 760
assign() (in module salt.modules.openbsd_sysctl), 799
associate_dhcp_options_to_vpc() (in module salt.modules.boto_vpc), 525
associate_network_acl_to_subnet() (in module salt.modules.boto_vpc), 525
associate_new_dhcp_options_to_vpc() (in module salt.modules.boto_vpc), 525
associate_new_network_acl_to_subnet() (in module salt.modules.boto_vpc), 526
associate_profile_to_role() (in module salt.modules.boto_iam), 518
associate_route_table() (in module salt.modules.boto_vpc), 526
async() (salt.runner.RunnerClient method), 341
async() (salt.wheel.WheelClient method), 342
at() (in module salt.modules.at), 500
atc() (in module salt.modules.at), 500
atq() (in module salt.modules.at), 500
atrm() (in module salt.modules.at), 500
attach_disk() (in module salt.cloud.clouds.gce), 360
attach_lb() (in module salt.cloud.clouds.gce), 360
attach_subnets() (in module salt.modules.boto_elb), 515
attach_volume() (in module salt.cloud.clouds.ec2), 353
attachable() (in module salt.modules.lxc), 710
attr_call() (in module salt.modules.test), 955
attributes() (in module salt.modules.extfs), 603
audit() (in module salt.modules.pkgng), 826
auth() (in module salt.auth.auto), 305
auth() (in module salt.auth.django), 306
auth() (in module salt.auth.keystone), 307
auth() (in module salt.auth.ldap), 307
auth() (in module salt.auth.mysql), 308
auth() (in module salt.auth.pam), 308
auth() (in module salt.auth.pki), 309
auth() (in module salt.auth.stormpath_mod), 310
auth() (in module salt.auth.yubico), 310
auth() (in module salt.modules.keystone), 693
auth_keys() (in module salt.modules.ssh), 924
authenticate() (in module salt.auth.pam), 309
AuthenticationError, 474, 476
AuthorizationError, 474, 476
authorize() (in module salt.modules.boto_secgroup), 521
authorize_cache_security_group_ingress() (in module salt.modules.boto_elasticache), 513
auto() (in module salt.modules.alternatives), 486
auto() (in module salt.states.alternatives), 1236
Auto-Order, **1677**
auto_accept
    conf/master, 420
autoload_dynamic_modules
    conf/minion, 450
automaster() (in module salt.modules.mount), 749
autoreject_file
    conf/master, 420
autoremove() (in module salt.modules.aptpkg), 489
autoremove() (in module salt.modules.pkgng), 826
AutoSearch() (salt.modules.win_update.PyWinUpdater method), 1006
AutoSearch() (salt.states.win_update.PyWinUpdater method), 1431
autosign_file
    conf/master, 420
autosign_timeout
    conf/master, 420
avail() (in module salt.modules.localemod), 705
avail() (in module salt.modules.smartos_imgadm), 900
avail_images() (in module salt.cloud.clouds.aliyun), 346
avail_images() (in module salt.cloud.clouds.cloudstack), 348
avail_images() (in module salt.cloud.clouds.digital_ocean), 350
avail_images() (in module salt.cloud.clouds.digital_ocean_v2), 351
avail_images() (in module salt.cloud.clouds.ec2), 353
avail_images() (in module salt.cloud.clouds.gce), 360
avail_images() (in module salt.cloud.clouds.gogrid), 364
avail_images() (in module salt.cloud.clouds.linode), 367
avail_images() (in module salt.cloud.clouds.lxc), 369
avail_images() (in module salt.cloud.clouds.msazure), 370
avail_images() (in module salt.cloud.clouds.nova), 372
avail_images() (in module salt.cloud.clouds.opennebula), 374
avail_images() (in module salt.cloud.clouds.openstack), 376

avail_images() (in module salt.cloud.clouds.parallels), 378
avail_images() (in module salt.cloud.clouds.proxmox), 379
avail_images() (in module salt.cloud.clouds.rackspace), 382
avail_images() (in module salt.cloud.clouds.softlayer), 383
avail_images() (in module salt.cloud.clouds.softlayer_hw), 385
avail_images() (in module salt.cloud.clouds.vsphere), 386
avail_locations() (in module salt.cloud.clouds.aliyun), 346
avail_locations() (in module salt.cloud.clouds.cloudstack), 349
avail_locations() (in module salt.cloud.clouds.digital_ocean), 350
avail_locations() (in module salt.cloud.clouds.digital_ocean_v2), 351
avail_locations() (in module salt.cloud.clouds.ec2), 354
avail_locations() (in module salt.cloud.clouds.gce), 360
avail_locations() (in module salt.cloud.clouds.linode), 367
avail_locations() (in module salt.cloud.clouds.msazure), 370
avail_locations() (in module salt.cloud.clouds.nova), 372
avail_locations() (in module salt.cloud.clouds.opennebula), 374
avail_locations() (in module salt.cloud.clouds.openstack), 376
avail_locations() (in module salt.cloud.clouds.proxmox), 379
avail_locations() (in module salt.cloud.clouds.rackspace), 382
avail_locations() (in module salt.cloud.clouds.softlayer), 383
avail_locations() (in module salt.cloud.clouds.softlayer_hw), 385
avail_locations() (in module salt.cloud.clouds.vsphere), 387
avail_platforms() (in module salt.modules.genesis), 640
avail_sizes() (in module salt.cloud.clouds.aliyun), 346
avail_sizes() (in module salt.cloud.clouds.cloudstack), 349
avail_sizes() (in module salt.cloud.clouds.digital_ocean), 350
avail_sizes() (in module salt.cloud.clouds.digital_ocean_v2), 351
avail_sizes() (in module salt.cloud.clouds.ec2), 354
avail_sizes() (in module salt.cloud.clouds.gce), 360
avail_sizes() (in module salt.cloud.clouds.gogrid), 364
avail_sizes() (in module salt.cloud.clouds.linode), 367
avail_sizes() (in module salt.cloud.clouds.msazure), 370
avail_sizes() (in module salt.cloud.clouds.nova), 372
avail_sizes() (in module salt.cloud.clouds.opennebula), 374
avail_sizes() (in module salt.cloud.clouds.openstack), 376
avail_sizes() (in module salt.cloud.clouds.rackspace), 382
avail_sizes() (in module salt.cloud.clouds.softlayer), 383
avail_sizes() (in module salt.cloud.clouds.softlayer_hw), 385
available() (in module salt.modules.daemontools), 563
available() (in module salt.modules.debian_service), 570
available() (in module salt.modules.freebsdkmod), 630
available() (in module salt.modules.freebsdservice), 636
available() (in module salt.modules.gentoo_service), 641
available() (in module salt.modules.kmod), 698
available() (in module salt.modules.launchctl), 699
available() (in module salt.modules.netbsdservice), 760
available() (in module salt.modules.openbsdrcctl), 801
available() (in module salt.modules.openbsdservice), 803
available() (in module salt.modules.rh_service), 875
available() (in module salt.modules.runit), 880
available() (in module salt.modules.service), 896
available() (in module salt.modules.smf), 902
available() (in module salt.modules.systemd), 952
available() (in module salt.modules.upstart), 969
available() (in module salt.modules.win_service), 1000
available_extensions() (in module salt.modules.postgres), 842
available_version() (in module salt.modules.macports), 726
available_version() (in module salt.modules.pkgin), 823
available_version() (in module salt.modules.solarisips), 910

B

backup() (in module salt.modules.pkgng), 826
backup() (in module salt.modules.solr), 917
backup_mode
    conf/minion, 446
backup_mode() (in module salt.modules.config), 554
ban() (in module salt.modules.varnish), 973
ban_list() (in module salt.modules.varnish), 974
base64_decodestring() (in module salt.modules.hashutil), 667
base64_encodestring() (in module salt.modules.hashutil), 667
basename() (in module salt.modules.file), 606
beacon() (in module salt.beacons.btmp), 1446
beacon() (in module salt.beacons.diskusage), 1446
beacon() (in module salt.beacons.inotify), 1447
beacon() (in module salt.beacons.journald), 1447
beacon() (in module salt.beacons.load), 1447
beacon() (in module salt.beacons.network_info), 1448
beacon() (in module salt.beacons.service), 1449
beacon() (in module salt.beacons.sh), 1449
beacon() (in module salt.beacons.twilio_txt_msg), 1450
beacon() (in module salt.beacons.wtmp), 1450
Best Practices, 1455

bgrewriteaof() (in module salt.modules.redismod), 868
bgsave() (in module salt.modules.redismod), 868
bindings_exist() (in module salt.states.jboss7), 1339
blkid() (in module salt.modules.disk), 574
block() (in module salt.modules.bluez), 504
block_device_mappings() (in module salt.cloud.clouds.cloudstack), 349
block_device_mappings() (in module salt.cloud.clouds.ec2), 354
block_device_mappings() (in module salt.cloud.clouds.libcloud_aws), 365
blockreplace() (in module salt.modules.file), 606
blockreplace() (in module salt.states.file), 1300
blocks() (in module salt.modules.extfs), 604
boolean() (in module salt.states.selinux), 1409
boot() (in module salt.cloud.clouds.linode), 367
boot() (in module salt.modules.nova), 793
boot_time() (in module salt.modules.ps), 849
Bootstrap, **1677**
bootstrap() (in module salt.modules.chocolatey), 542
bootstrap() (in module salt.modules.genesis), 640
bootstrap() (in module salt.modules.img), 674
bootstrap() (in module salt.modules.lxc), 710
bootstrap() (in module salt.modules.zcbuildout), 1026
bootstrap() (in module salt.runners.manage), 1168
bootstrap() (in module salt.states.npm), 1371
bootstrap_psexec() (in module salt.runners.manage), 1168
branch() (in module salt.modules.git), 644
build() (in module salt.modules.dockerio), 581
build() (salt.modules.syslog_ng.Argument method), 942
build() (salt.modules.syslog_ng.Buildable method), 942
build() (salt.modules.syslog_ng.GivenStatement method), 942
build() (salt.modules.syslog_ng.Option method), 943
build() (salt.modules.syslog_ng.SimpleParameter method), 943
build() (salt.modules.syslog_ng.SimpleParameterValue method), 943
build() (salt.modules.syslog_ng.TypedParameter method), 944
build() (salt.modules.syslog_ng.TypedParameterValue method), 944
build_body() (salt.modules.syslog_ng.Buildable method), 942
build_bond() (in module salt.modules.debian_ip), 569
build_bond() (in module salt.modules.rh_ip), 874
build_header() (salt.modules.syslog_ng.Buildable method), 942
build_header() (salt.modules.syslog_ng.Statement method), 943
build_interface() (in module salt.modules.debian_ip), 569
build_interface() (in module salt.modules.rh_ip), 874
build_network_settings() (in module salt.modules.debian_ip), 569
build_network_settings() (in module salt.modules.rh_ip), 874
build_routes() (in module salt.modules.debian_ip), 570
build_routes() (in module salt.modules.rh_ip), 874
build_rule() (in module salt.modules.iptables), 684
build_rule() (in module salt.modules.nftables), 788
build_schedule_item() (in module salt.modules.schedule), 890
build_tail() (salt.modules.syslog_ng.Buildable method), 942
build_tail() (salt.modules.syslog_ng.Statement method), 943
Buildable (class in salt.modules.syslog_ng), 942
buildmod() (in module salt.modules.znc), 1031
buildout() (in module salt.modules.zcbuildout), 1027
built() (in module salt.states.dockerio), 1287
bulk_activate() (in module salt.modules.modjk), 743
bulk_build() (in module salt.modules.poudriere), 847
bulk_disable() (in module salt.modules.modjk), 744
bulk_recover() (in module salt.modules.modjk), 744
bulk_stop() (in module salt.modules.modjk), 744

C

ca_exists() (in module salt.modules.tls), 960
cache_dir() (in module salt.modules.cp), 557
cache_file() (in module salt.modules.cp), 557
cache_files() (in module salt.modules.cp), 558
cache_jobs
  conf/minion, 445
cache_local_file() (in module salt.modules.cp), 558
cache_master() (in module salt.modules.cp), 558
cache_sreqs
  conf/minion, 447
cachedir
  conf/master, 416
  conf/minion, 445
call() (in module salt.states.cmd), 1273
Caller (class in salt.client), 340
cas() (in module salt.modules.data), 565
cert_base_path() (in module salt.modules.tls), 960
cert_info() (in module salt.modules.tls), 960
cflags_contains() (in module salt.modules.makeconf), 730
chain_absent() (in module salt.states.iptables), 1337
chain_absent() (in module salt.states.nftables), 1370
chain_present() (in module salt.states.iptables), 1337
chain_present() (in module salt.states.nftables), 1370
change() (in module salt.states.augeas), 1241
change_password() (in module salt.modules.drac), 590
change_password() (in module salt.modules.ilo), 672
change_password() (in module salt.modules.rabbitmq), 861
change_username() (in module salt.modules.ilo), 672

changed() (salt.states.cyg.DictDiffer method), 1282
check() (in module salt.modules.ipset), 682
check() (in module salt.modules.iptables), 684
check() (in module salt.modules.nftables), 788
check() (in module salt.modules.parted), 812
check() (in module salt.modules.pkgng), 827
check_available() (in module salt.modules.freebsdkmod), 630
check_available() (in module salt.modules.kmod), 698
check_chain() (in module salt.modules.iptables), 685
check_chain() (in module salt.modules.nftables), 788
check_db() (in module salt.modules.ebuild), 593
check_db() (in module salt.modules.yumpkg), 1017
check_extra_requirements() (in module salt.modules.ebuild), 593
check_extra_requirements() (in module salt.modules.pkg_resource), 822
check_file_meta() (in module salt.modules.file), 607
check_hash() (in module salt.modules.file), 607
check_installed() (in module salt.modules.alternatives), 486
check_ip() (in module salt.modules.dig), 574
check_ip() (in module salt.modules.dnsutil), 577
check_key() (in module salt.modules.ssh), 925
check_key_file() (in module salt.modules.ssh), 925
check_known_host() (in module salt.modules.ssh), 925
check_managed() (in module salt.modules.file), 607
check_managed_changes() (in module salt.modules.file), 608
check_member_pool() (in module salt.runners.f5), 1160
check_member_pool() (salt.runners.f5.F5Mgmt method), 1160
check_mod_enabled() (in module salt.modules.deb_apache), 568
check_perms() (in module salt.modules.file), 608
check_pool() (in module salt.runners.f5), 1161
check_pool() (salt.runners.f5.F5Mgmt method), 1160
check_request() (in module salt.modules.state), 927
check_server() (in module salt.modules.lvs), 708
check_service() (in module salt.modules.lvs), 708
check_set() (in module salt.modules.ipset), 682
check_site_enabled() (in module salt.modules.deb_apache), 568
check_table() (in module salt.modules.nftables), 789
check_valid_package() (in module salt.modules.cyg), 562
check_virtualserver() (in module salt.runners.f5), 1161
check_virtualserver() (salt.runners.f5.F5Mgmt method), 1160
checkout() (in module salt.modules.git), 644
checkout() (in module salt.modules.svn), 937
chfullname() (in module salt.modules.mac_user), 724
chfullname() (in module salt.modules.pw_user), 856
chfullname() (in module salt.modules.solaris_user), 908
chfullname() (in module salt.modules.useradd), 971
chfullname() (in module salt.modules.win_useradd), 1009
chgid() (in module salt.modules.groupadd), 663
chgid() (in module salt.modules.mac_group), 724
chgid() (in module salt.modules.mac_user), 724
chgid() (in module salt.modules.pw_group), 855
chgid() (in module salt.modules.pw_user), 856
chgid() (in module salt.modules.solaris_group), 907
chgid() (in module salt.modules.solaris_user), 908
chgid() (in module salt.modules.useradd), 971
chgroups() (in module salt.modules.mac_user), 724
chgroups() (in module salt.modules.pw_user), 856
chgroups() (in module salt.modules.solaris_user), 909
chgroups() (in module salt.modules.useradd), 971
chgroups() (in module salt.modules.win_useradd), 1009
chgrp() (in module salt.modules.file), 608
chgrp() (in module salt.modules.win_file), 985
chhome() (in module salt.modules.mac_user), 725
chhome() (in module salt.modules.pw_user), 856
chhome() (in module salt.modules.solaris_user), 909
chhome() (in module salt.modules.useradd), 971
chhome() (in module salt.modules.win_useradd), 1009
chhomephone() (in module salt.modules.pw_user), 856
chhomephone() (in module salt.modules.solaris_user), 909
chhomephone() (in module salt.modules.useradd), 971
chloginclass() (in module salt.modules.useradd), 972
chocolatey_version() (in module salt.modules.chocolatey), 542
chost_contains() (in module salt.modules.makeconf), 730
chown() (in module salt.modules.file), 608
chown() (in module salt.modules.win_file), 985
chpgrp() (in module salt.modules.win_file), 986
chprofile() (in module salt.modules.win_useradd), 1009
chroomnumber() (in module salt.modules.pw_user), 857
chroomnumber() (in module salt.modules.solaris_user), 909
chroomnumber() (in module salt.modules.useradd), 972
chshell() (in module salt.modules.mac_user), 725
chshell() (in module salt.modules.pw_user), 857
chshell() (in module salt.modules.solaris_user), 909
chshell() (in module salt.modules.useradd), 972
chuid() (in module salt.modules.mac_user), 725
chuid() (in module salt.modules.pw_user), 857
chuid() (in module salt.modules.solaris_user), 909
chuid() (in module salt.modules.useradd), 972
chworkphone() (in module salt.modules.pw_user), 857
chworkphone() (in module salt.modules.solaris_user), 909
chworkphone() (in module salt.modules.useradd), 972
clean() (in module salt.modules.pkgng), 827
clean_dynamic_modules
    conf/minion, 450
clean_locks() (in module salt.modules.zypper), 1034
clean_metadata() (in module salt.modules.yumpkg), 1017

clean_old_jobs() (in module salt.returners.local_cache), 1134
clean_old_jobs() (in module salt.returners.multi_returner), 1138
clean_old_jobs() (in module salt.returners.postgres_local_cache), 1146
clear() (in module salt.modules.data), 566
clear() (in module salt.modules.lvs), 708
clear() (in module salt.modules.qemu_nbd), 860
clear_all() (in module salt.runners.cache), 1157
clear_cache() (in module salt.fileserver.gitfs), 467
clear_cache() (in module salt.fileserver.hgfs), 468
clear_cache() (in module salt.fileserver.svnfs), 472
clear_cache() (in module salt.modules.saltutil), 886
clear_cache() (in module salt.modules.state), 927
clear_cache() (in module salt.runners.fileserver), 1161
clear_grains() (in module salt.runners.cache), 1157
clear_lock() (in module salt.fileserver.gitfs), 467
clear_lock() (in module salt.fileserver.hgfs), 468
clear_lock() (in module salt.fileserver.svnfs), 472
clear_lock() (in module salt.runners.fileserver), 1161
clear_mine() (in module salt.runners.cache), 1157
clear_mine_func() (in module salt.runners.cache), 1157
clear_password() (in module salt.modules.rabbitmq), 861
clear_pillar() (in module salt.runners.cache), 1157
clear_request() (in module salt.modules.state), 927
client() (in module salt.modules.chef), 540
client() (in module salt.states.chef), 1269
client_acl
    conf/master, 420
client_acl_blacklist
    conf/master, 421
client_config() (in module salt.config), 335
client_version() (in module salt.modules.oracle), 805
clone() (in module salt.modules.git), 645
clone() (in module salt.modules.hg), 668
clone() (in module salt.modules.lxc), 711
cloned() (in module salt.states.lxc), 1350
cloud_init() (in module salt.modules.lxc), 711
cloud_init() (in module salt.runners.lxc), 1166
cloud_init_interface() (in module salt.modules.lxc), 711
CloudClient (class in salt.cloud), 343
cluster_commit() (in module salt.modules.riak), 877
cluster_join() (in module salt.modules.riak), 877
cluster_plan() (in module salt.modules.riak), 877
cluster_status() (in module salt.modules.rabbitmq), 862
cmd() (in module salt.modules.saltutil), 886
cmd() (salt.client.LocalClient method), 337
cmd() (salt.client.ssh.client.SSHClient method), 344
cmd() (salt.runner.RunnerClient method), 341
cmd() (salt.wheel.WheelClient method), 342
cmd_async() (salt.client.LocalClient method), 338
cmd_async() (salt.runner.RunnerClient method), 341
cmd_async() (salt.wheel.WheelClient method), 342
cmd_batch() (salt.client.LocalClient method), 339
cmd_iter() (in module salt.modules.saltutil), 886
cmd_iter() (salt.client.LocalClient method), 339
cmd_iter() (salt.client.ssh.client.SSHClient method), 345
cmd_iter_no_block() (salt.client.LocalClient method), 339
cmd_subset() (salt.client.LocalClient method), 339
cmd_sync() (salt.runner.RunnerClient method), 341
cmd_sync() (salt.wheel.WheelClient method), 343
cmd_unzip() (in module salt.modules.archive), 495
cmd_zip() (in module salt.modules.archive), 496
collatz() (in module salt.modules.test), 955
collectstatic() (in module salt.modules.djangomod), 575
color
    conf/master, 417
column_families() (in module salt.modules.cassandra), 539
column_family_definition() (in module salt.modules.cassandra), 539
command line option
    --key-deploy, 330
    --max-procs, 330
    --passwd, 330
    --priv, 330
    --refresh, --refresh-cache, 330
    --roster, 330
    --roster-file, 330
    -i, --ignore-host-keys, 330
    -r, --raw, --raw-shell, 330
command() (in module salt.modules.djangomod), 575
CommandExecutionError, 474, 476
CommandNotFoundError, 474, 476
comment() (in module salt.modules.file), 608
comment() (in module salt.states.file), 1301
commit() (in module salt.modules.dockerio), 581
commit() (in module salt.modules.git), 645
commit() (in module salt.modules.junos), 692
commit() (in module salt.modules.svn), 937
compactionstats() (in module salt.modules.cassandra), 540
Compound Matcher, **1677**
compound() (in module salt.modules.match), 736
computer_desc() (in module salt.states.win_system), 1430
computer_name() (in module salt.states.win_system), 1430
conf() (in module salt.modules.grub_legacy), 664
conf/logging
    external-logging-handlers, 458
    log_datefmt, 457
    log_datefmt_logfile, 457
    log_file, 456
    log_fmt_console, 457
    log_fmt_logfile, 457

log_granular_levels, 458
log_level, 457
log_level_logfile, 457
conf/master
auto_accept, 420
autoreject_file, 420
autosign_file, 420
autosign_timeout, 420
cachedir, 416
client_acl, 420
client_acl_blacklist, 421
color, 417
cython_enable, 423
default_include, 441
enable_gpu_grains, 417
enforce_mine_cache, 418
event_return, 418
ext_job_cache, 418
ext_pillar, 435
ext_pillar_first, 436
extension_modules, 416
external_auth, 421
external_nodes, 423
failhard, 424
file_buffer_size, 425
file_ignore_glob, 425
file_ignore_regex, 425
file_recv, 421
file_roots, 426
fileserver_backend, 425
gitfs_base, 427
gitfs_env_blacklist, 428
gitfs_env_whitelist, 428
gitfs_insecure_auth, 428
gitfs_mountpoint, 427
gitfs_passphrase, 429
gitfs_password, 428
gitfs_privkey, 429
gitfs_provider, 426
gitfs_pubkey, 429
gitfs_remotes, 426
gitfs_root, 427
gitfs_ssl_verify, 427
gitfs_user, 428
hash_type, 425
hgfs_base, 431
hgfs_branch_method, 430
hgfs_env_blacklist, 431
hgfs_env_whitelist, 431
hgfs_mountpoint, 430
hgfs_remotes, 429
hgfs_root, 430
include, 441
interface, 414
ipv6, 414
job_cache, 417
keep_jobs, 416
log_datefmt, 440
log_datefmt_logfile, 440
log_file, 439
log_fmt_console, 440
log_fmt_logfile, 440
log_granular_levels, 441
log_level, 440
log_level_logfile, 440
loop_interval, 417
master_id, 414
master_job_cache, 418
master_pubkey_signature, 422
master_sign_key_name, 422
master_sign_pubkey, 422
master_tops, 423
master_use_pubkey_signature, 422
max_minions, 418
max_open_files, 414
minion_data_cache, 417
minionfs_blacklist, 435
minionfs_env, 434
minionfs_mountpoint, 434
minionfs_whitelist, 434
nodegroups, 441
open_mode, 420
order_masters, 438
output, 417
peer, 439
peer_run, 439
pidfile, 415
pillar_roots, 435
pillar_source_merging_strategy, 436
pki_dir, 416
presence_events, 419
publish_port, 414
range_server, 441
renderer, 423
ret_port, 415
root_dir, 415
roster_file, 419
rotate_aes_key, 422
runner_dirs, 423
sock_dir, 417
ssh_minion_opts, 419
state_output, 424
state_top, 423
state_verbose, 424
svnfs_branches, 433
svnfs_env_blacklist, 434
svnfs_env_whitelist, 433
svnfs_mountpoint, 432

svnfs_remotes, 432
svnfs_root, 432
svnfs_tags, 433
svnfs_trunk, 433
syndic_log_file, 438
syndic_master, 438
syndic_master_log_file, 438
syndic_master_port, 438
test, 424
timeout, 416
token_expire, 421
user, 414
verify_env, 416
win_gitrepos, 442
win_repo, 442
win_repo_mastercachefile, 442
worker_threads, 415
yaml_utf8, 424
conf/minion
acceptance_wait_time, 446
acceptance_wait_time_max, 446
always_verify_signature, 453
append_domain, 445
autoload_dynamic_modules, 450
backup_mode, 446
cache_jobs, 445
cache_sreqs, 447
cachedir, 445
clean_dynamic_modules, 450
cython_enable, 449
disable_modules, 448
disable_returners, 448
dns_check, 447
environment, 450
failhard, 454
file_client, 451
file_roots, 451
grains_dirs, 449
hash_type, 451
id, 445
include, 455
ipc_mode, 448
log_datefmt, 454
log_datefmt_logfile, 454
log_file, 453
log_fmt_console, 454
log_fmt_logfile, 454
log_granular_levels, 454
log_level, 453
log_level_logfile, 453
master, 443
master_port, 444
master_sign_key_name, 452
master_type, 443
module_dirs, 416, 448
multiprocessing, 453
open_mode, 452
pidfile, 444
pillar_roots, 452
pki_dir, 444
providers, 449
random_reauth_delay, 446
recon_default, 446
recon_max, 447
recon_randomize, 447
render_dirs, 449
renderer, 450
retry_dns, 444
returner_dirs, 449
root_dir, 444
sock_dir, 446
state_output, 450
state_verbose, 450
states_dirs, 449
tcp_pub_port, 448
tcp_pull_port, 448
update_restart_services, 455
update_url, 455
use_master_when_local, 451
user, 444
verify_env, 445
verify_master_pubkey_sign, 452
conf_test() (in module salt.modules.test), 956
config() (in module salt.modules.apache), 487
config() (in module salt.modules.freebsdports), 634
config() (in module salt.modules.rsync), 879
config() (in module salt.modules.syslog_ng), 944
config() (in module salt.states.git), 1315
config() (in module salt.states.syslog_ng), 1419
config_get() (in module salt.modules.git), 645
config_get() (in module salt.modules.redismod), 868
config_set() (in module salt.modules.git), 646
config_set() (in module salt.modules.redismod), 868
config_test() (in module salt.modules.syslog_ng), 944
configfile() (in module salt.states.apache), 1237
configtest() (in module salt.modules.nginx), 792
configurable_test_state() (in module salt.states.test), 1420
configure_network() (in module salt.modules.ilo), 672
configure_snmp() (in module salt.modules.ilo), 672
connect() (in module salt.modules.network), 766
connect() (in module salt.modules.qemu_nbd), 860
connected() (in module salt.wheel.minions), 1445
contains() (in module salt.modules.file), 609
contains_glob() (in module salt.modules.file), 609
contains_regex() (in module salt.modules.file), 609
contains_regex_multiline() (in module salt.modules.file), 609
context() (in module salt.states.stateconf), 1415

conv (salt.auth.pam.PamConv attribute), 308
convert() (in module salt.modules.btrfs), 537
convert_to_arn() (in module salt.modules.boto_cloudwatch), 510
convert_to_group_ids() (in module salt.modules.boto_secgroup), 521
copy() (in module salt.modules.file), 609
copy() (in module salt.modules.schedule), 890
copy() (in module salt.states.file), 1302
copy_snapshot() (in module salt.cloud.clouds.ec2), 354
core_status() (in module salt.modules.solr), 917
cp() (in module salt.modules.lxc), 712
cp() (in module salt.modules.parted), 812
cpu() (in module salt.modules.sysbench), 941
cpu_percent() (in module salt.modules.ps), 849
cpu_times() (in module salt.modules.ps), 849
cpuinfo() (in module salt.modules.status), 931
cpustats() (in module salt.modules.status), 931
create() (in module salt.cloud.clouds.aliyun), 346
create() (in module salt.cloud.clouds.cloudstack), 349
create() (in module salt.cloud.clouds.digital_ocean), 350
create() (in module salt.cloud.clouds.digital_ocean_v2), 351
create() (in module salt.cloud.clouds.ec2), 354
create() (in module salt.cloud.clouds.gce), 360
create() (in module salt.cloud.clouds.gogrid), 364
create() (in module salt.cloud.clouds.libcloud_aws), 365
create() (in module salt.cloud.clouds.linode), 367
create() (in module salt.cloud.clouds.lxc), 369
create() (in module salt.cloud.clouds.msazure), 370
create() (in module salt.cloud.clouds.nova), 372
create() (in module salt.cloud.clouds.opennebula), 374
create() (in module salt.cloud.clouds.openstack), 376
create() (in module salt.cloud.clouds.parallels), 378
create() (in module salt.cloud.clouds.proxmox), 379
create() (in module salt.cloud.clouds.rackspace), 382
create() (in module salt.cloud.clouds.saltify), 383
create() (in module salt.cloud.clouds.softlayer), 383
create() (in module salt.cloud.clouds.softlayer_hw), 385
create() (in module salt.cloud.clouds.vsphere), 387
create() (in module salt.modules.boto_asg), 506
create() (in module salt.modules.boto_cfn), 509
create() (in module salt.modules.boto_elasticache), 513
create() (in module salt.modules.boto_elb), 515
create() (in module salt.modules.boto_secgroup), 522
create() (in module salt.modules.boto_sns), 523
create() (in module salt.modules.boto_sqs), 524
create() (in module salt.modules.boto_vpc), 526
create() (in module salt.modules.cloud), 545
create() (in module salt.modules.glusterfs), 651
create() (in module salt.modules.lxc), 713
create() (in module salt.modules.mdadm), 738
create() (in module salt.modules.saltcloudmod), 886
create() (in module salt.modules.serverdensity_device), 895
create() (in module salt.modules.splunk_search), 922
create() (in module salt.modules.virt), 976
create() (in module salt.modules.virtualenv_mod), 983
create() (in module salt.modules.xapi), 1010
create() (in module salt.modules.zfs), 1028
create() (in module salt.modules.zpool), 1032
create() (in module salt.runners.cloud), 1158
create() (salt.cloud.CloudClient method), 343
create_address() (in module salt.cloud.clouds.gce), 360
create_attach_volumes() (in module salt.cloud.clouds.ec2), 354
create_attach_volumes() (in module salt.cloud.clouds.libcloud_aws), 365
create_attach_volumes() (in module salt.cloud.clouds.msazure), 370
create_ca() (in module salt.modules.tls), 960
create_ca_signed_cert() (in module salt.modules.tls), 961
create_cache_security_group() (in module salt.modules.boto_elasticache), 513
create_config() (in module salt.cloud.clouds.linode), 367
create_container() (in module salt.modules.dockerio), 581
create_csr() (in module salt.modules.tls), 962
create_customer_gateway() (in module salt.modules.boto_vpc), 527
create_datasource() (in module salt.modules.jboss7), 688
create_dhcp_options() (in module salt.modules.boto_vpc), 527
create_disk() (in module salt.cloud.clouds.gce), 360
create_disk_from_distro() (in module salt.cloud.clouds.linode), 367
create_dns_record() (in module salt.cloud.clouds.digital_ocean_v2), 351
create_event() (in module salt.modules.pagerduty), 810
create_event() (in module salt.runners.pagerduty), 1171
create_event() (in module salt.states.pagerduty), 1373
create_extension() (in module salt.modules.postgres), 842
create_file_vdev() (in module salt.modules.zpool), 1032
create_floatingip() (in module salt.modules.neutron), 771
create_fwrule() (in module salt.cloud.clouds.gce), 361
create_hc() (in module salt.cloud.clouds.gce), 361
create_ikepolicy() (in module salt.modules.neutron), 771
create_instance_profile() (in module salt.modules.boto_iam), 518
create_ipsec_site_connection() (in module salt.modules.neutron), 771
create_ipsecpolicy() (in module salt.modules.neutron), 772
create_jail() (in module salt.modules.poudriere), 847
create_key() (in module salt.modules.gpg), 655
create_key() (in module salt.modules.reg), 872
create_keypair() (in module salt.cloud.clouds.ec2), 354

create_launch_configuration() (in module salt.modules.boto_asg), 507
create_lb() (in module salt.cloud.clouds.gce), 361
create_listeners() (in module salt.modules.boto_elb), 515
create_metadata() (in module salt.modules.postgres), 842
create_network() (in module salt.cloud.clouds.gce), 361
create_network() (in module salt.modules.neutron), 773
create_network_acl() (in module salt.modules.boto_vpc), 527
create_network_acl_entry() (in module salt.modules.boto_vpc), 527
create_node() (in module salt.cloud.clouds.aliyun), 346
create_node() (in module salt.cloud.clouds.digital_ocean), 350
create_node() (in module salt.cloud.clouds.digital_ocean_v2), 351
create_node() (in module salt.cloud.clouds.parallels), 378
create_node() (in module salt.cloud.clouds.proxmox), 379
create_or_update_alarm() (in module salt.modules.boto_cloudwatch), 510
create_pkcs12() (in module salt.modules.tls), 962
create_pool() (in module salt.runners.f5), 1161
create_pool() (salt.runners.f5.F5Mgmt method), 1160
create_port() (in module salt.modules.neutron), 773
create_ports_tree() (in module salt.modules.poudriere), 848
create_queue() (in module salt.modules.aws_sqs), 502
create_role() (in module salt.modules.boto_iam), 518
create_role_policy() (in module salt.modules.boto_iam), 518
create_route() (in module salt.modules.boto_vpc), 528
create_route_table() (in module salt.modules.boto_vpc), 528
create_router() (in module salt.modules.neutron), 773
create_security_group() (in module salt.modules.neutron), 774
create_security_group_rule() (in module salt.modules.neutron), 774
create_self_signed_cert() (in module salt.modules.tls), 963
create_simple_binding() (in module salt.modules.jboss7), 688
create_snapshot() (in module salt.cloud.clouds.ec2), 354
create_snapshot() (in module salt.cloud.clouds.gce), 361
create_subnet() (in module salt.modules.boto_vpc), 528
create_subnet() (in module salt.modules.neutron), 774
create_swap_disk() (in module salt.cloud.clouds.linode), 367
create_table() (in module salt.modules.boto_dynamodb), 512
create_user() (in module salt.modules.drac), 590
create_user() (in module salt.modules.ilo), 672
create_volume() (in module salt.cloud.clouds.ec2), 354
create_vpnservice() (in module salt.modules.neutron), 775
create_vs() (in module salt.runners.f5), 1161
create_vs() (salt.runners.f5.F5Mgmt method), 1160
create_win_salt_restart_task() (in module salt.modules.win_service), 1000
create_xml_path() (in module salt.modules.virt), 976
create_xml_str() (in module salt.modules.virt), 976
created() (in module salt.states.glusterfs), 1317
created() (in module salt.states.lxc), 1350
createsuperuser() (in module salt.modules.djangomod), 575
cross_test() (in module salt.modules.test), 956
ctrl_alt_del() (in module salt.modules.virt), 976
current_branch() (in module salt.modules.git), 646
custom() (in module salt.modules.status), 932
custom() (in module salt.modules.supervisord), 934
customer_gateway_exists() (in module salt.modules.boto_vpc), 528
cxxflags_contains() (in module salt.modules.makeconf), 730
cython_enable
    conf/master, 423
    conf/minion, 449

D

dashboard_absent() (in module salt.states.grafana), 1323
dashboard_present() (in module salt.states.grafana), 1323
data() (in module salt.modules.match), 736
datasource_exists() (in module salt.states.jboss7), 1339
db_alter() (in module salt.modules.postgres), 842
db_check() (in module salt.modules.mysql), 752
db_create() (in module salt.modules.influx), 676
db_create() (in module salt.modules.mysql), 752
db_create() (in module salt.modules.postgres), 842
db_exists() (in module salt.modules.influx), 676
db_exists() (in module salt.modules.mongodb), 746
db_exists() (in module salt.modules.mysql), 752
db_exists() (in module salt.modules.postgres), 842
db_list() (in module salt.modules.influx), 677
db_list() (in module salt.modules.mongodb), 746
db_list() (in module salt.modules.mysql), 752
db_list() (in module salt.modules.postgres), 843
db_optimize() (in module salt.modules.mysql), 752
db_remove() (in module salt.modules.influx), 677
db_remove() (in module salt.modules.mongodb), 747
db_remove() (in module salt.modules.mysql), 752
db_remove() (in module salt.modules.postgres), 843
db_repair() (in module salt.modules.mysql), 753
db_tables() (in module salt.modules.mysql), 753
dbsize() (in module salt.modules.redismod), 868
dead() (in module salt.states.service), 1410
dead() (in module salt.states.supervisord), 1416
dec() (in module salt.runners.nacl), 1170
decrement() (in module salt.modules.memcached), 740

decrypt() (in module salt.modules.gpg), 655
decrypt_ciphertext() (in module salt.renderers.gpg), 1101
decrypt_object() (in module salt.renderers.gpg), 1101
default() (in module salt.modules.pyenv), 858
default() (in module salt.modules.rbenv), 866
default_config() (in module salt.modules.linux_sysctl), 704
default_hash() (in module salt.modules.bsd_shadow), 536
default_hash() (in module salt.modules.shadow), 897
default_hash() (in module salt.modules.solaris_shadow), 907
default_include
    conf/master, 441
default_route() (in module salt.modules.network), 766
default_zone() (in module salt.modules.firewalld), 626
define_vol_xml_path() (in module salt.modules.virt), 976
define_vol_xml_str() (in module salt.modules.virt), 977
define_xml_path() (in module salt.modules.virt), 977
define_xml_str() (in module salt.modules.virt), 977
defragment() (in module salt.modules.btrfs), 537
defragment() (in module salt.modules.xfs), 1014
deinstall() (in module salt.modules.freebsdports), 634
del_export() (in module salt.modules.nfs3), 787
del_password() (in module salt.modules.shadow), 897
del_repo() (in module salt.modules.aptpkg), 489
del_repo() (in module salt.modules.yumpkg), 1017
del_repo() (in module salt.modules.zypper), 1034
del_tags() (in module salt.cloud.clouds.ec2), 354
del_tags() (in module salt.cloud.clouds.libcloud_aws), 365
delete() (in module salt.modules.boto_asg), 507
delete() (in module salt.modules.boto_cfn), 509
delete() (in module salt.modules.boto_dynamodb), 512
delete() (in module salt.modules.boto_elasticache), 513
delete() (in module salt.modules.boto_elb), 515
delete() (in module salt.modules.boto_secgroup), 522
delete() (in module salt.modules.boto_sns), 523
delete() (in module salt.modules.boto_sqs), 524
delete() (in module salt.modules.boto_vpc), 528
delete() (in module salt.modules.bridge), 535
delete() (in module salt.modules.btrfs), 537
delete() (in module salt.modules.ddns), 567
delete() (in module salt.modules.elasticsearch), 598
delete() (in module salt.modules.glusterfs), 652
delete() (in module salt.modules.groupadd), 664
delete() (in module salt.modules.ipset), 682
delete() (in module salt.modules.iptables), 685
delete() (in module salt.modules.layman), 700
delete() (in module salt.modules.mac_group), 724
delete() (in module salt.modules.mac_user), 725
delete() (in module salt.modules.memcached), 740
delete() (in module salt.modules.mine), 741
delete() (in module salt.modules.nftables), 789
delete() (in module salt.modules.nova), 793
delete() (in module salt.modules.openstack_config), 804
delete() (in module salt.modules.postfix), 840
delete() (in module salt.modules.pw_group), 855
delete() (in module salt.modules.pw_user), 857
delete() (in module salt.modules.redismod), 869
delete() (in module salt.modules.s3), 885
delete() (in module salt.modules.schedule), 890
delete() (in module salt.modules.serverdensity_device), 895
delete() (in module salt.modules.smartos_imgadm), 900
delete() (in module salt.modules.solaris_group), 907
delete() (in module salt.modules.solaris_user), 909
delete() (in module salt.modules.splunk_search), 922
delete() (in module salt.modules.swift), 940
delete() (in module salt.modules.useradd), 972
delete() (in module salt.modules.win_groupadd), 991
delete() (in module salt.modules.win_useradd), 1009
delete() (in module salt.runners.queue), 1174
delete() (in module salt.states.iptables), 1337
delete() (in module salt.states.nftables), 1370
delete() (in module salt.wheel.key), 1445
delete_address() (in module salt.cloud.clouds.gce), 361
delete_alarm() (in module salt.modules.boto_cloudwatch), 510
delete_backup() (in module salt.modules.file), 609
delete_cache_security_group() (in module salt.modules.boto_elasticache), 514
delete_chain() (in module salt.modules.iptables), 685
delete_chain() (in module salt.modules.nftables), 789
delete_customer_gateway() (in module salt.modules.boto_vpc), 529
delete_dict() (in module salt.wheel.key), 1445
delete_disk() (in module salt.cloud.clouds.gce), 361
delete_dns_record() (in module salt.cloud.clouds.digital_ocean_v2), 351
delete_floatingip() (in module salt.modules.neutron), 775
delete_fwrule() (in module salt.cloud.clouds.gce), 361
delete_hc() (in module salt.cloud.clouds.gce), 362
delete_host() (in module salt.modules.ddns), 567
delete_host() (in module salt.modules.omapi), 799
delete_ikepolicy() (in module salt.modules.neutron), 775
delete_instance_profile() (in module salt.modules.boto_iam), 518
delete_ipsec_site_connection() (in module salt.modules.neutron), 775
delete_ipsecpolicy() (in module salt.modules.neutron), 776
delete_jail() (in module salt.modules.poudriere), 848
delete_key() (in module salt.modules.gpg), 656
delete_key() (in module salt.modules.reg), 872
delete_keypair() (in module salt.cloud.clouds.ec2), 354
delete_launch_configuration() (in module salt.modules.boto_asg), 507
delete_lb() (in module salt.cloud.clouds.gce), 362

delete_listeners() (in module salt.modules.boto_elb), 515
delete_message() (in module salt.modules.aws_sqs), 502
delete_network() (in module salt.cloud.clouds.gce), 362
delete_network() (in module salt.modules.neutron), 776
delete_network_acl() (in module salt.modules.boto_vpc), 529
delete_network_acl_entry() (in module salt.modules.boto_vpc), 529
delete_policy() (in module salt.modules.rabbitmq), 862
delete_port() (in module salt.modules.neutron), 776
delete_queue() (in module salt.modules.aws_sqs), 503
delete_quota() (in module salt.modules.neutron), 776
delete_record() (in module salt.modules.boto_route53), 520
delete_role() (in module salt.modules.boto_iam), 518
delete_role_policy() (in module salt.modules.boto_iam), 519
delete_route() (in module salt.modules.boto_vpc), 529
delete_route_table() (in module salt.modules.boto_vpc), 529
delete_router() (in module salt.modules.neutron), 777
delete_security_group() (in module salt.modules.neutron), 777
delete_security_group_rule() (in module salt.modules.neutron), 777
delete_server() (in module salt.modules.lvs), 708
delete_service() (in module salt.modules.firewalld), 626
delete_service() (in module salt.modules.lvs), 709
delete_set() (in module salt.modules.ipset), 682
delete_snapshot() (in module salt.cloud.clouds.ec2), 354
delete_snapshot() (in module salt.cloud.clouds.gce), 362
delete_ssh_key() (in module salt.modules.ilo), 672
delete_subnet() (in module salt.modules.boto_vpc), 530
delete_subnet() (in module salt.modules.neutron), 777
delete_table() (in module salt.modules.nftables), 789
delete_user() (in module salt.modules.drac), 591
delete_user() (in module salt.modules.ilo), 672
delete_user() (in module salt.modules.rabbitmq), 862
delete_vhost() (in module salt.modules.rabbitmq), 862
delete_volume() (in module salt.cloud.clouds.ec2), 354
delete_vpnservice() (in module salt.modules.neutron), 778
delete_zone() (in module salt.modules.firewalld), 626
delfacl() (in module salt.modules.linux_acl), 702
delif() (in module salt.modules.bridge), 535
delta_import() (in module salt.modules.solr), 918
deluser() (in module salt.modules.groupadd), 664
deluser() (in module salt.modules.win_groupadd), 991
delval() (in module salt.modules.grains), 660
delvol_on_destroy() (in module salt.cloud.clouds.ec2), 354
depclean() (in module salt.modules.ebuild), 594
deploy() (in module salt.modules.jboss7), 688
deploy_war() (in module salt.modules.tomcat), 965
deployed() (in module salt.states.jboss7), 1340
depth (salt.pillar.mysql.merger attribute), 1088
deregister_instances() (in module salt.modules.boto_elb), 515
describe() (in module salt.modules.boto_vpc), 530
describe() (in module salt.modules.git), 646
describe() (in module salt.modules.hg), 668
describe_snapshots() (in module salt.cloud.clouds.ec2), 355
describe_volumes() (in module salt.cloud.clouds.ec2), 355
deserialize() (in module salt.utils.serializers.json), 478
deserialize() (in module salt.utils.serializers.msgpack), 479
deserialize() (in module salt.utils.serializers.yaml), 478
desktop_interface() (in module salt.states.gnomedesktop), 1318
desktop_lockdown() (in module salt.states.gnomedesktop), 1319
destroy() (in module salt.cloud.clouds.aliyun), 346
destroy() (in module salt.cloud.clouds.cloudstack), 349
destroy() (in module salt.cloud.clouds.digital_ocean), 350
destroy() (in module salt.cloud.clouds.digital_ocean_v2), 352
destroy() (in module salt.cloud.clouds.ec2), 355
destroy() (in module salt.cloud.clouds.gce), 362
destroy() (in module salt.cloud.clouds.gogrid), 364
destroy() (in module salt.cloud.clouds.libcloud_aws), 365
destroy() (in module salt.cloud.clouds.linode), 367
destroy() (in module salt.cloud.clouds.lxc), 369
destroy() (in module salt.cloud.clouds.msazure), 370
destroy() (in module salt.cloud.clouds.nova), 372
destroy() (in module salt.cloud.clouds.opennebula), 374
destroy() (in module salt.cloud.clouds.openstack), 376
destroy() (in module salt.cloud.clouds.parallels), 378
destroy() (in module salt.cloud.clouds.proxmox), 379
destroy() (in module salt.cloud.clouds.rackspace), 382
destroy() (in module salt.cloud.clouds.softlayer), 384
destroy() (in module salt.cloud.clouds.softlayer_hw), 385
destroy() (in module salt.cloud.clouds.vsphere), 387
destroy() (in module salt.modules.cloud), 545
destroy() (in module salt.modules.lxc), 713
destroy() (in module salt.modules.mdadm), 739
destroy() (in module salt.modules.smartos_vmadm), 901
destroy() (in module salt.modules.virt), 977
destroy() (in module salt.modules.xapi), 1010
destroy() (in module salt.modules.zfs), 1029
destroy() (in module salt.modules.zpool), 1032
destroy() (in module salt.runners.cloud), 1158
destroy() (salt.cloud.CloudClient method), 343
detach_disk() (in module salt.cloud.clouds.gce), 362
detach_lb() (in module salt.cloud.clouds.gce), 362
detach_subnets() (in module salt.modules.boto_elb), 516
detach_volume() (in module salt.cloud.clouds.ec2), 355
detail() (in module salt.modules.mdadm), 739

devices() (in module salt.modules.btrfs), 538
devices() (in module salt.modules.xfs), 1014
dfs() (in module salt.modules.hadoop), 665
dfs_absent() (in module salt.modules.hadoop), 665
dfs_present() (in module salt.modules.hadoop), 665
dhcp_options_exists() (in module salt.modules.boto_vpc), 530
DictDiffer (class in salt.states.cyg), 1282
did_composer_install() (in module salt.modules.composer), 553
diff() (in module salt.modules.dockerio), 582
diff() (in module salt.modules.junos), 692
diff() (in module salt.modules.rpm), 878
diff() (in module salt.modules.svn), 937
diff() (in module salt.modules.yumpkg), 1017
diff() (in module salt.modules.zypper), 1034
diff() (in module salt.runners.survey), 1179
dig() (in module salt.modules.network), 766
dig() (in module salt.modules.win_network), 993
dir_list() (in module salt.fileserver.gitfs), 467
dir_list() (in module salt.fileserver.hgfs), 468
dir_list() (in module salt.fileserver.minionfs), 469
dir_list() (in module salt.fileserver.roots), 470
dir_list() (in module salt.fileserver.s3fs), 471
dir_list() (in module salt.fileserver.svnfs), 472
dir_list() (in module salt.runners.fileserver), 1162
directives() (in module salt.modules.apache), 487
directory() (in module salt.states.file), 1302
directory_exists() (in module salt.modules.file), 610
dirinfo() (in module salt.modules.moosefs), 748
dirname() (in module salt.modules.file), 610
dirty() (in module salt.states.svn), 1417
disable() (in module salt.modules.debian_service), 571
disable() (in module salt.modules.freebsdservice), 636
disable() (in module salt.modules.gentoo_service), 641
disable() (in module salt.modules.netbsdservice), 760
disable() (in module salt.modules.openbsdrcctl), 801
disable() (in module salt.modules.puppet), 854
disable() (in module salt.modules.rdp), 867
disable() (in module salt.modules.rh_service), 875
disable() (in module salt.modules.schedule), 891
disable() (in module salt.modules.smf), 902
disable() (in module salt.modules.state), 927
disable() (in module salt.modules.systemd), 953
disable() (in module salt.modules.upstart), 969
disable() (in module salt.modules.win_firewall), 990
disable() (in module salt.modules.win_ip), 991
disable() (in module salt.modules.win_service), 1000
disable() (in module salt.states.apache_module), 1237
disable() (in module salt.states.modjk_worker), 1356
disable_availability_zones() (in module salt.modules.boto_elb), 516
disable_dhcp() (in module salt.modules.ilo), 673
disable_job() (in module salt.modules.schedule), 891
disable_modules
    conf/minion, 448
disable_plugin() (in module salt.modules.rabbitmq), 862
disable_returners
    conf/minion, 448
disable_server() (in module salt.modules.haproxyconn), 666
disable_ssh() (in module salt.modules.ilo), 673
disable_term_protect() (in module salt.cloud.clouds.botocore_aws), 348
disable_term_protect() (in module salt.cloud.clouds.ec2), 355
disabled() (in module salt.modules.debian_service), 571
disabled() (in module salt.modules.freebsdservice), 636
disabled() (in module salt.modules.gentoo_service), 641
disabled() (in module salt.modules.netbsdservice), 760
disabled() (in module salt.modules.openbsdrcctl), 801
disabled() (in module salt.modules.openbsdservice), 803
disabled() (in module salt.modules.rh_service), 876
disabled() (in module salt.modules.smf), 902
disabled() (in module salt.modules.systemd), 953
disabled() (in module salt.modules.upstart), 969
disabled() (in module salt.modules.win_service), 1000
disabled() (in module salt.states.rabbitmq_plugin), 1397
disabled() (in module salt.states.rdp), 1400
disabled() (in module salt.states.service), 1411
disabled() (in module salt.states.win_firewall), 1427
disassociate_network_acl() (in module salt.modules.boto_vpc), 530
disassociate_profile_from_role() (in module salt.modules.boto_iam), 519
disassociate_route_table() (in module salt.modules.boto_vpc), 530
discoverable() (in module salt.modules.bluez), 504
disk_io_counters() (in module salt.modules.ps), 850
disk_partition_usage() (in module salt.modules.ps), 850
disk_partitions() (in module salt.modules.ps), 850
disk_usage() (in module salt.modules.ps), 850
diskstats() (in module salt.modules.status), 932
diskusage() (in module salt.modules.file), 610
diskusage() (in module salt.modules.status), 932
display() (in module salt.modules.alternatives), 486
display() (salt.output.nested.NestDisplay method), 1070
display() (salt.output.no_return.NestDisplay method), 1071
django_auth_setup() (in module salt.auth.django), 306
dns_check
    conf/minion, 447
dns_dhcp() (in module salt.modules.win_dns_client), 984
dns_dhcp() (in module salt.states.win_dns_client), 1426
dns_exists() (in module salt.states.win_dns_client), 1426
do() (in module salt.modules.pyenv), 858
do() (in module salt.modules.rbenv), 866
do() (in module salt.modules.rvm), 881

do_with_python() (in module salt.modules.pyenv), 858
do_with_ruby() (in module salt.modules.rbenv), 866
doc() (in module salt.modules.sysmod), 947
dot_vals() (in module salt.modules.config), 555
down() (in module salt.modules.debian_ip), 570
down() (in module salt.modules.rh_ip), 875
down() (in module salt.runners.manage), 1168
download() (in module salt.modules.softwareupdate), 905
download() (in module salt.modules.yumpkg), 1018
download() (in module salt.modules.zypper), 1034
Download() (salt.modules.win_update.PyWinUpdater method), 1006
Download() (salt.states.win_update.PyWinUpdater method), 1431
download_all() (in module salt.modules.softwareupdate), 905
download_updates() (in module salt.modules.win_update), 1007
downloaded() (in module salt.states.artifactory), 1239
downloaded() (in module salt.states.win_update), 1431
drop_extension() (in module salt.modules.postgres), 843
dump() (in module salt.modules.blockdev), 504
dump() (in module salt.modules.data), 566
dump() (in module salt.modules.extfs), 604
dump() (in module salt.modules.xfs), 1014
dump_config() (in module salt.modules.modjk), 744
dumpconf() (in module salt.modules.znc), 1031

E

EAuth, **1677**
EauthAuthenticationError, 474, 476
ec2_credentials_create() (in module salt.modules.keystone), 693
ec2_credentials_delete() (in module salt.modules.keystone), 694
ec2_credentials_get() (in module salt.modules.keystone), 694
ec2_credentials_list() (in module salt.modules.keystone), 694
echo() (in module salt.modules.test), 956
eclean_dist() (in module salt.modules.gentoolkitmod), 642
eclean_pkg() (in module salt.modules.gentoolkitmod), 643
edit_conf() (in module salt.modules.lxc), 714
edit_server() (in module salt.modules.lvs), 709
edit_service() (in module salt.modules.lvs), 709
edited_conf() (in module salt.states.lxc), 1350
email_alerts() (in module salt.modules.drac), 591
emerge_default_opts_contains() (in module salt.modules.makeconf), 730
empty_dir_list() (in module salt.runners.fileserver), 1162
enable() (in module salt.modules.debian_service), 571
enable() (in module salt.modules.freebsdservice), 636
enable() (in module salt.modules.gentoo_service), 641
enable() (in module salt.modules.netbsdservice), 761
enable() (in module salt.modules.openbsdrcctl), 801
enable() (in module salt.modules.puppet), 854
enable() (in module salt.modules.rdp), 868
enable() (in module salt.modules.rh_service), 876
enable() (in module salt.modules.schedule), 891
enable() (in module salt.modules.smf), 903
enable() (in module salt.modules.state), 927
enable() (in module salt.modules.systemd), 953
enable() (in module salt.modules.upstart), 969
enable() (in module salt.modules.win_firewall), 990
enable() (in module salt.modules.win_ip), 992
enable() (in module salt.modules.win_service), 1000
enable() (in module salt.states.apache_module), 1237
enable_availability_zones() (in module salt.modules.boto_elb), 516
enable_dhcp() (in module salt.modules.ilo), 673
enable_gpu_grains
    conf/master, 417
enable_job() (in module salt.modules.schedule), 891
enable_plugin() (in module salt.modules.rabbitmq), 862
enable_server() (in module salt.modules.haproxyconn), 666
enable_ssh() (in module salt.modules.ilo), 673
enable_term_protect() (in module salt.cloud.clouds.botocore_aws), 348
enable_term_protect() (in module salt.cloud.clouds.ec2), 355
enabled() (in module salt.modules.debian_service), 571
enabled() (in module salt.modules.freebsdservice), 636
enabled() (in module salt.modules.gentoo_service), 641
enabled() (in module salt.modules.netbsdservice), 761
enabled() (in module salt.modules.openbsdrcctl), 801
enabled() (in module salt.modules.openbsdservice), 803
enabled() (in module salt.modules.rh_service), 876
enabled() (in module salt.modules.smf), 903
enabled() (in module salt.modules.systemd), 953
enabled() (in module salt.modules.upstart), 969
enabled() (in module salt.modules.win_service), 1000
enabled() (in module salt.states.rabbitmq_plugin), 1397
enabled() (in module salt.states.rdp), 1400
enabled() (in module salt.states.service), 1411
enabled_service_owners() (in module salt.modules.introspect), 681
enc() (in module salt.runners.nacl), 1170
encrypt() (in module salt.modules.gpg), 656
endpoint_absent() (in module salt.states.keystone), 1345
endpoint_create() (in module salt.modules.keystone), 694
endpoint_delete() (in module salt.modules.keystone), 694
endpoint_get() (in module salt.modules.keystone), 694
endpoint_list() (in module salt.modules.keystone), 695
endpoint_present() (in module salt.states.keystone), 1345
enforce_mine_cache

conf/master, 418
enforce_nice_config() (in module salt.modules.portage_config), 839
ensure_views() (in module salt.returners.couchdb_return), 1131
enter_root() (salt.pillar.mysql.merger method), 1088
env_absent() (in module salt.states.cron), 1281
env_present() (in module salt.states.cron), 1281
Environment, **1677**
environment
conf/minion, 450
envs() (in module salt.fileserver.gitfs), 467
envs() (in module salt.fileserver.hgfs), 468
envs() (in module salt.fileserver.minionfs), 469
envs() (in module salt.fileserver.roots), 470
envs() (in module salt.fileserver.s3fs), 471
envs() (in module salt.fileserver.svnfs), 472
envs() (in module salt.pillar.git_pillar), 1083
envs() (in module salt.runners.fileserver), 1162
envs() (salt.pillar.git_pillar.GitPillar method), 1083
error() (in module salt.runners.error), 1159
error() (in module salt.wheel.error), 1444
estimate() (in module salt.modules.xfs), 1015
Event, *see* Reactor, **160**, **1677**
event bus, 160
event system, 160
event() (in module salt.runners.state), 1175
event_return
conf/master, 418
event_return() (in module salt.returners.local), 1134
event_return() (in module salt.returners.mysql), 1140
Events (class in salt.netapi.rest_cherrypy.app), 1050
EventsSaltAPIHandler (in module salt.netapi.rest_tornado.saltnado), 1065
ex_mod_init() (in module salt.modules.ebuild), 594
exception() (in module salt.modules.test), 956
exec_action() (in module salt.modules.eselect), 600
exec_code() (in module salt.modules.cmdmod), 547
exec_code_all() (in module salt.modules.cmdmod), 548
execs() (in module salt.modules.systemd), 953
execute() (in module salt.modules.augeas_cfg), 501
execute_salt_restart_task() (in module salt.modules.win_service), 1000
Execution Function, **1677**
Execution Module, **1677**
execution() (in module salt.runners.doc), 1158
exists() (in module salt.modules.boto_asg), 507
exists() (in module salt.modules.boto_cfn), 509
exists() (in module salt.modules.boto_dynamodb), 512
exists() (in module salt.modules.boto_elasticache), 514
exists() (in module salt.modules.boto_elb), 516
exists() (in module salt.modules.boto_secgroup), 522
exists() (in module salt.modules.boto_sns), 523
exists() (in module salt.modules.boto_sqs), 524
exists() (in module salt.modules.boto_vpc), 530
exists() (in module salt.modules.dockerio), 582
exists() (in module salt.modules.elasticsearch), 598
exists() (in module salt.modules.lxc), 714
exists() (in module salt.modules.parted), 812
exists() (in module salt.modules.redismod), 869
exists() (in module salt.modules.win_path), 996
exists() (in module salt.modules.zfs), 1029
exists() (in module salt.modules.zpool), 1032
exists() (in module salt.states.aws_sqs), 1243
exists() (in module salt.states.file), 1304
exists() (in module salt.states.win_path), 1429
expand_repo_def() (in module salt.modules.aptpkg), 489
expand_repo_def() (in module salt.modules.yumpkg), 1018
expire() (in module salt.modules.redismod), 869
expireat() (in module salt.modules.redismod), 869
export() (in module salt.modules.dockerio), 582
export() (in module salt.modules.svn), 937
export() (in module salt.modules.zpool), 1032
export() (in module salt.states.svn), 1417
export_key() (in module salt.modules.gpg), 656
ext() (in module salt.modules.pillar), 816
ext_job_cache
conf/master, 418
ext_pillar
conf/master, 435
ext_pillar() (in module salt.pillar.cmd_json), 1076
ext_pillar() (in module salt.pillar.cmd_yaml), 1076
ext_pillar() (in module salt.pillar.cmd_yamlex), 1076
ext_pillar() (in module salt.pillar.cobbler), 1077
ext_pillar() (in module salt.pillar.django_orm), 1078
ext_pillar() (in module salt.pillar.ec2_pillar), 1079
ext_pillar() (in module salt.pillar.etcd_pillar), 1080
ext_pillar() (in module salt.pillar.file_tree), 1081
ext_pillar() (in module salt.pillar.foreman), 1082
ext_pillar() (in module salt.pillar.git_pillar), 1083
ext_pillar() (in module salt.pillar.hiera), 1084
ext_pillar() (in module salt.pillar.libvirt), 1084
ext_pillar() (in module salt.pillar.mongo), 1085
ext_pillar() (in module salt.pillar.mysql), 1088
ext_pillar() (in module salt.pillar.pepa), 1093
ext_pillar() (in module salt.pillar.pillar_ldap), 1093
ext_pillar() (in module salt.pillar.puppet), 1093
ext_pillar() (in module salt.pillar.reclass_adapter), 1094
ext_pillar() (in module salt.pillar.redismod), 1095
ext_pillar() (in module salt.pillar.s3), 1096
ext_pillar() (in module salt.pillar.svn_pillar), 1097
ext_pillar() (in module salt.pillar.varstack_pillar), 1097
ext_pillar() (in module salt.pillar.virtkey), 1097
ext_pillar_first
conf/master, 436
extension_modules
conf/master, 416

External Job Cache, 1677
External Pillar, 1677
external-logging-handlers
    conf/logging, 458
external_auth
    conf/master, 421
external_nodes
    conf/master, 423
extra_action() (salt.cloud.CloudClient method), 343
extract_hash() (in module salt.modules.file), 610
extract_ipv4() (in module salt.roster.cache), 1154
extract_ipv4() (in module salt.roster.cloud), 1155
extract_queries() (salt.pillar.mysql.merger method), 1088
extracted() (in module salt.states.archive), 1238

F

F5Mgmt (class in salt.runners.f5), 1160
fact() (in module salt.modules.puppet), 854
facts() (in module salt.modules.puppet), 854
facts_refresh() (in module salt.modules.junos), 692
fail_with_changes() (in module salt.states.test), 1420
fail_without_changes() (in module salt.states.test), 1420
failhard
    conf/master, 424
    conf/minion, 454
features() (in module salt.modules.btrfs), 538
features_contains() (in module salt.modules.makeconf), 730
fetch() (in module salt.modules.git), 646
fetch() (in module salt.modules.pkgng), 827
fetch() (in module salt.modules.sqlite3), 923
fib() (in module salt.modules.test), 956
field_names (salt.pillar.mysql.merger attribute), 1088
File Server, 1677
file() (in module salt.states.cron), 1281
file_buffer_size
    conf/master, 425
file_client
    conf/minion, 451
file_dict() (in module salt.modules.aptpkg), 490
file_dict() (in module salt.modules.dpkg), 589
file_dict() (in module salt.modules.freebsdpkg), 632
file_dict() (in module salt.modules.pacman), 807
file_dict() (in module salt.modules.pkgin), 823
file_dict() (in module salt.modules.rpm), 878
file_dict() (in module salt.modules.yumpkg), 1018
file_dict() (in module salt.modules.zypper), 1035
file_exists() (in module salt.modules.file), 610
file_hash() (in module salt.fileserver.gitfs), 467
file_hash() (in module salt.fileserver.hgfs), 468
file_hash() (in module salt.fileserver.minionfs), 469
file_hash() (in module salt.fileserver.roots), 470
file_hash() (in module salt.fileserver.s3fs), 471
file_hash() (in module salt.fileserver.svnfs), 472
file_ignore_glob
    conf/master, 425
file_ignore_regex
    conf/master, 425
file_list() (in module salt.fileserver.gitfs), 467
file_list() (in module salt.fileserver.hgfs), 468
file_list() (in module salt.fileserver.minionfs), 469
file_list() (in module salt.fileserver.roots), 470
file_list() (in module salt.fileserver.s3fs), 471
file_list() (in module salt.fileserver.svnfs), 472
file_list() (in module salt.modules.aptpkg), 490
file_list() (in module salt.modules.dpkg), 589
file_list() (in module salt.modules.freebsdpkg), 632
file_list() (in module salt.modules.pacman), 807
file_list() (in module salt.modules.pkgin), 823
file_list() (in module salt.modules.rpm), 878
file_list() (in module salt.modules.yumpkg), 1018
file_list() (in module salt.modules.zypper), 1035
file_list() (in module salt.runners.fileserver), 1163
file_list_emptydirs() (in module salt.fileserver.gitfs), 467
file_list_emptydirs() (in module salt.fileserver.hgfs), 468
file_list_emptydirs() (in module salt.fileserver.roots), 470
file_list_emptydirs() (in module salt.fileserver.s3fs), 471
file_list_emptydirs() (in module salt.fileserver.svnfs), 472
file_recv
    conf/master, 421
file_roots
    conf/master, 426
    conf/minion, 451
fileinfo() (in module salt.modules.moosefs), 749
fileio() (in module salt.modules.sysbench), 941
fileserver_backend
    conf/master, 425
FileserverConfigError, 475, 476
filter_by() (in module salt.modules.grains), 660
filter_by() (in module salt.modules.match), 736
find() (in module salt.modules.file), 611
find() (in module salt.modules.mongodb), 747
find() (in module salt.wheel.file_roots), 1444
find() (in module salt.wheel.pillar_roots), 1446
find_cached_job() (in module salt.modules.saltutil), 887
find_file() (in module salt.fileserver.gitfs), 467
find_file() (in module salt.fileserver.hgfs), 468
find_file() (in module salt.fileserver.minionfs), 469
find_file() (in module salt.fileserver.roots), 470
find_file() (in module salt.fileserver.s3fs), 471
find_file() (in module salt.fileserver.svnfs), 472
find_guest() (in module salt.runners.lxc), 1166
find_guests() (in module salt.runners.lxc), 1166
find_interfaces() (in module salt.modules.bridge), 535
find_job() (in module salt.modules.saltutil), 887
find_room() (in module salt.modules.slack_notify), 899
find_user() (in module salt.modules.slack_notify), 899
finger() (in module salt.modules.key), 692

finger() (in module salt.wheel.key), 1445
finger_master() (in module salt.modules.key), 692
fire() (in module salt.modules.event), 602
fire_master() (in module salt.modules.event), 602
flags() (in module salt.states.portage_config), 1387
flavor_create() (in module salt.modules.nova), 793
flavor_delete() (in module salt.modules.nova), 794
flavor_list() (in module salt.modules.nova), 794
flush() (in module salt.modules.ipset), 682
flush() (in module salt.modules.iptables), 685
flush() (in module salt.modules.mine), 741
flush() (in module salt.modules.nftables), 790
flush() (in module salt.states.ipset), 1333
flush() (in module salt.states.iptables), 1337
flush() (in module salt.states.nftables), 1371
flushall() (in module salt.modules.redismod), 869
flushdb() (in module salt.modules.redismod), 869
focus (salt.pillar.mysql.merger attribute), 1088
force_off() (in module salt.runners.virt), 1180
force_reload() (in module salt.modules.debian_service), 571
force_reload() (in module salt.modules.netbsdservice), 761
force_reload() (in module salt.modules.systemd), 953
force_reload() (in module salt.modules.upstart), 970
force_reset() (in module salt.modules.rabbitmq), 862
formatted() (in module salt.states.blockdev), 1243
free_slave() (in module salt.modules.mysql), 753
freecpu() (in module salt.modules.virt), 977
freecpu() (in module salt.modules.xapi), 1010
freemem() (in module salt.modules.virt), 977
freemem() (in module salt.modules.xapi), 1011
freeze() (in module salt.modules.lxc), 714
freeze() (in module salt.modules.pip), 818
freeze() (in module salt.runners.lxc), 1166
frozen() (in module salt.states.lxc), 1351
fstab() (in module salt.modules.freebsdjail), 629
fstab() (in module salt.modules.mount), 749
full_data() (in module salt.modules.publish), 853
full_data() (in module salt.modules.raet_publish), 865
full_import() (in module salt.modules.solr), 918
full_info() (in module salt.modules.virt), 977
full_info() (in module salt.modules.xapi), 1011
full_query() (in module salt.modules.cloud), 545
full_query() (in module salt.runners.cloud), 1158
full_query() (salt.cloud.CloudClient method), 343
full_restart() (in module salt.modules.daemontools), 564
full_restart() (in module salt.modules.runit), 880
full_restart() (in module salt.modules.upstart), 970
fullversion() (in module salt.modules.apache), 487
fullversion() (in module salt.modules.dnsmasq), 575
fullversion() (in module salt.modules.linux_lvm), 703
fullversion() (in module salt.modules.tomcat), 965
function() (in module salt.states.saltmod), 1404
function() (salt.client.Caller method), 340

G

gather_bootstrap_script() (in module salt.modules.config), 555
gemset_copy() (in module salt.modules.rvm), 881
gemset_create() (in module salt.modules.rvm), 882
gemset_delete() (in module salt.modules.rvm), 882
gemset_empty() (in module salt.modules.rvm), 882
gemset_list() (in module salt.modules.rvm), 882
gemset_list_all() (in module salt.modules.rvm), 882
gemset_present() (in module salt.states.rvm), 1403
gen() (in module salt.wheel.key), 1445
gen_accept() (in module salt.wheel.key), 1445
gen_hyper_keys() (in module salt.pillar.libvirt), 1084
gen_locale() (in module salt.modules.localemod), 705
gen_password() (in module salt.modules.shadow), 897
generate() (in module salt.runners.thin), 1180
genrepo() (in module salt.modules.win_repo), 999
genrepo() (in module salt.runners.winrepo), 1181
genrepo() (in module salt.states.winrepo), 1432
gentoo_mirrors_contains() (in module salt.modules.makeconf), 731
get() (in module salt.modules.augeas_cfg), 501
get() (in module salt.modules.config), 555
get() (in module salt.modules.darwin_sysctl), 565
get() (in module salt.modules.defaults), 572
get() (in module salt.modules.elasticsearch), 598
get() (in module salt.modules.environ), 599
get() (in module salt.modules.freebsd_sysctl), 628
get() (in module salt.modules.gnomedesktop), 653
get() (in module salt.modules.grains), 661
get() (in module salt.modules.linux_sysctl), 705
get() (in module salt.modules.memcached), 740
get() (in module salt.modules.mine), 741
get() (in module salt.modules.netbsd_sysctl), 760
get() (in module salt.modules.openbsd_sysctl), 799
get() (in module salt.modules.openstack_config), 805
get() (in module salt.modules.pillar), 816
get() (in module salt.modules.rvm), 883
get() (in module salt.modules.s3), 885
get() (in module salt.modules.sdb), 892
get() (in module salt.modules.smartos_imgadm), 900
get() (in module salt.modules.splunk_search), 923
get() (in module salt.modules.swift), 940
get() (in module salt.modules.sysrc), 951
get() (in module salt.runners.mine), 1170
get() (in module salt.runners.sdb), 1175
get() (in module salt.sdb.etcd_db), 1451
get() (in module salt.sdb.keyring_db), 1452
get() (in module salt.sdb.memcached), 1452
get() (in module salt.sdb.sqlite3), 1453
GET() (salt.netapi.rest_cherrypy.app.Events method), 1050

GET() (salt.netapi.rest_cherrypy.app.Jobs method), 1048
GET() (salt.netapi.rest_cherrypy.app.Keys method), 1054
GET() (salt.netapi.rest_cherrypy.app.Login method), 1044
GET() (salt.netapi.rest_cherrypy.app.LowDataAdapter method), 1044
GET() (salt.netapi.rest_cherrypy.app.Minions method), 1046
GET() (salt.netapi.rest_cherrypy.app.Stats method), 1057
GET() (salt.netapi.rest_cherrypy.app.WebsocketEndpoint method), 1056
get_() (in module salt.modules.etcd_mod), 601
get_account_id() (in module salt.modules.boto_iam), 519
get_alarm() (in module salt.modules.boto_cloudwatch), 511
get_alias() (in module salt.modules.hosts), 669
get_all() (in module salt.modules.daemontools), 564
get_all() (in module salt.modules.debian_service), 571
get_all() (in module salt.modules.freebsdservice), 636
get_all() (in module salt.modules.gentoo_service), 641
get_all() (in module salt.modules.launchctl), 699
get_all() (in module salt.modules.netbsdservice), 761
get_all() (in module salt.modules.openbsdrcctl), 801
get_all() (in module salt.modules.openbsdservice), 803
get_all() (in module salt.modules.rh_service), 876
get_all() (in module salt.modules.runit), 880
get_all() (in module salt.modules.service), 896
get_all() (in module salt.modules.smf), 903
get_all() (in module salt.modules.systemd), 953
get_all() (in module salt.modules.upstart), 970
get_all() (in module salt.modules.win_service), 1000
get_all_alarms() (in module salt.modules.boto_cloudwatch), 511
get_all_interfaces() (in module salt.modules.win_ip), 992
get_all_topics() (in module salt.modules.boto_sns), 523
get_arn() (in module salt.modules.boto_sns), 523
get_attributes() (in module salt.modules.boto_elb), 516
get_attributes() (in module salt.modules.boto_sqs), 524
get_attributes() (in module salt.modules.win_file), 986
get_auth() (in module salt.cloud.clouds.linode), 367
get_auth_url() (in module salt.auth.keystone), 307
get_availability_zone() (in module salt.cloud.clouds.ec2), 355
get_availability_zone() (in module salt.cloud.clouds.libcloud_aws), 365
get_available_extension() (in module salt.modules.postgres), 843
get_block_device() (in module salt.modules.parted), 812
get_bond() (in module salt.modules.debian_ip), 570
get_bond() (in module salt.modules.rh_ip), 875
get_bufsize() (in module salt.modules.network), 766
get_ca() (in module salt.modules.tls), 964
get_cache_subnet_group() (in module salt.modules.boto_elasticache), 514
get_cflags() (in module salt.modules.makeconf), 731
get_chost() (in module salt.modules.makeconf), 731
get_cli_returns() (salt.client.LocalClient method), 339
get_cloud_init_mime() (in module salt.modules.boto_asg), 507
get_computer_desc() (in module salt.modules.win_system), 1003
get_computer_name() (in module salt.modules.win_system), 1003
get_config() (in module salt.modules.boto_asg), 507
get_config() (in module salt.modules.boto_elasticache), 514
get_config() (in module salt.modules.boto_secgroup), 522
get_config() (in module salt.modules.dnsmasq), 576
get_config() (in module salt.modules.win_firewall), 990
get_config_file() (in module salt.modules.syslog_ng), 944
get_configured_provider() (in module salt.cloud.clouds.aliyun), 346
get_configured_provider() (in module salt.cloud.clouds.botocore_aws), 348
get_configured_provider() (in module salt.cloud.clouds.cloudstack), 349
get_configured_provider() (in module salt.cloud.clouds.digital_ocean), 350
get_configured_provider() (in module salt.cloud.clouds.digital_ocean_v2), 352
get_configured_provider() (in module salt.cloud.clouds.ec2), 355
get_configured_provider() (in module salt.cloud.clouds.gce), 362
get_configured_provider() (in module salt.cloud.clouds.gogrid), 364
get_configured_provider() (in module salt.cloud.clouds.libcloud_aws), 365
get_configured_provider() (in module salt.cloud.clouds.linode), 367
get_configured_provider() (in module salt.cloud.clouds.lxc), 369
get_configured_provider() (in module salt.cloud.clouds.msazure), 370
get_configured_provider() (in module salt.cloud.clouds.nova), 372
get_configured_provider() (in module salt.cloud.clouds.opennebula), 374
get_configured_provider() (in module salt.cloud.clouds.openstack), 376
get_configured_provider() (in module salt.cloud.clouds.parallels), 378
get_configured_provider() (in module salt.cloud.clouds.proxmox), 379
get_configured_provider() (in module salt.cloud.clouds.pyrax), 381
get_configured_provider() (in module salt.cloud.clouds.rackspace), 382

get_configured_provider() (in module salt.cloud.clouds.saltify), 383
get_configured_provider() (in module salt.cloud.clouds.softlayer), 384
get_configured_provider() (in module salt.cloud.clouds.softlayer_hw), 385
get_configured_provider() (in module salt.cloud.clouds.vsphere), 387
get_conn() (in module salt.cloud.clouds.cloudstack), 349
get_conn() (in module salt.cloud.clouds.gce), 362
get_conn() (in module salt.cloud.clouds.gogrid), 364
get_conn() (in module salt.cloud.clouds.libcloud_aws), 366
get_conn() (in module salt.cloud.clouds.linode), 367
get_conn() (in module salt.cloud.clouds.msazure), 370
get_conn() (in module salt.cloud.clouds.nova), 372
get_conn() (in module salt.cloud.clouds.openstack), 376
get_conn() (in module salt.cloud.clouds.pyrax), 381
get_conn() (in module salt.cloud.clouds.rackspace), 382
get_conn() (in module salt.cloud.clouds.softlayer), 384
get_conn() (in module salt.cloud.clouds.softlayer_hw), 385
get_conn() (in module salt.cloud.clouds.vsphere), 387
get_console_output() (in module salt.cloud.clouds.ec2), 355
get_container_profile() (in module salt.modules.lxc), 715
get_container_root() (in module salt.modules.dockerio), 582
get_containers() (in module salt.modules.dockerio), 582
get_current_target() (in module salt.modules.eselect), 600
get_cxxflags() (in module salt.modules.makeconf), 731
get_data() (salt.roster.flat.RosterMatcher method), 1155
get_default_gateway() (in module salt.modules.win_ip), 992
get_devmm() (in module salt.modules.file), 612
get_diff() (in module salt.modules.file), 612
get_dir() (in module salt.modules.cp), 558
get_disabled() (in module salt.modules.debian_service), 571
get_disabled() (in module salt.modules.freebsdservice), 636
get_disabled() (in module salt.modules.gentoo_service), 641
get_disabled() (in module salt.modules.netbsdservice), 761
get_disabled() (in module salt.modules.openbsdrcctl), 802
get_disabled() (in module salt.modules.openbsdservice), 803
get_disabled() (in module salt.modules.rh_service), 876
get_disabled() (in module salt.modules.smf), 903
get_disabled() (in module salt.modules.systemd), 953
get_disabled() (in module salt.modules.upstart), 970
get_disabled() (in module salt.modules.win_service), 1001
get_disk_size() (in module salt.cloud.clouds.linode), 367
get_disks() (in module salt.modules.virt), 977
get_disks() (in module salt.modules.xapi), 1011
get_dns_config() (in module salt.modules.win_dns_client), 985
get_dns_servers() (in module salt.modules.win_dns_client), 985
get_docker() (in module salt.modules.mine), 741
get_elb_config() (in module salt.modules.boto_elb), 516
get_emerge_default_opts() (in module salt.modules.makeconf), 731
get_enabled() (in module salt.modules.debian_service), 571
get_enabled() (in module salt.modules.freebsdjail), 629
get_enabled() (in module salt.modules.freebsdservice), 636
get_enabled() (in module salt.modules.gentoo_service), 641
get_enabled() (in module salt.modules.netbsdservice), 761
get_enabled() (in module salt.modules.openbsdrcctl), 802
get_enabled() (in module salt.modules.openbsdservice), 803
get_enabled() (in module salt.modules.rh_service), 876
get_enabled() (in module salt.modules.smf), 903
get_enabled() (in module salt.modules.systemd), 953
get_enabled() (in module salt.modules.upstart), 970
get_enabled() (in module salt.modules.win_service), 1001
get_event_iter_returns() (salt.client.LocalClient method), 340
get_features() (in module salt.modules.makeconf), 731
get_file() (in module salt.modules.cp), 558
get_file_str() (in module salt.modules.cp), 558
get_flags_from_package_conf() (in module salt.modules.portage_config), 839
get_fmri() (in module salt.modules.solarisips), 910
get_fun() (in module salt.modules.ret), 873
get_fun() (in module salt.returners.couchdb_return), 1131
get_fun() (in module salt.returners.etcd_return), 1133
get_fun() (in module salt.returners.memcache_return), 1135
get_fun() (in module salt.returners.mongo_future_return), 1136
get_fun() (in module salt.returners.mongo_return), 1137
get_fun() (in module salt.returners.mysql), 1140
get_fun() (in module salt.returners.odbc), 1142
get_fun() (in module salt.returners.postgres), 1144
get_fun() (in module salt.returners.redis_return), 1147
get_fun() (in module salt.returners.sqlite3_return), 1150
get_gentoo_mirrors() (in module salt.modules.makeconf), 731
get_gid() (in module salt.modules.file), 612

get_gid() (in module salt.modules.win_file), 986
get_glob() (salt.roster.ansible.Target method), 1154
get_graphics() (in module salt.modules.virt), 977
get_group() (in module salt.modules.file), 612
get_group() (in module salt.modules.win_file), 986
get_group_id() (in module salt.modules.boto_secgroup), 522
get_hash() (in module salt.modules.file), 613
get_health_check() (in module salt.modules.boto_elb), 517
get_hostname() (in module salt.modules.network), 766
get_hwclock() (in module salt.modules.timezone), 959
get_hwclock() (in module salt.modules.win_timezone), 1005
get_icmp_types() (in module salt.modules.firewalld), 627
get_id() (in module salt.modules.boto_vpc), 531
get_id() (in module salt.modules.parted), 812
get_image() (in module salt.cloud.clouds.aliyun), 346
get_image() (in module salt.cloud.clouds.cloudstack), 349
get_image() (in module salt.cloud.clouds.digital_ocean), 350
get_image() (in module salt.cloud.clouds.digital_ocean_v2), 352
get_image() (in module salt.cloud.clouds.gogrid), 364
get_image() (in module salt.cloud.clouds.linode), 368
get_image() (in module salt.cloud.clouds.nova), 372
get_image() (in module salt.cloud.clouds.opennebula), 374
get_image() (in module salt.cloud.clouds.openstack), 376
get_image() (in module salt.cloud.clouds.parallels), 378
get_image() (in module salt.cloud.clouds.rackspace), 382
get_images() (in module salt.modules.dockerio), 583
get_installed_extension() (in module salt.modules.postgres), 843
get_instance() (in module salt.modules.cloud), 545
get_instance_health() (in module salt.modules.boto_elb), 517
get_instances() (in module salt.modules.boto_asg), 508
get_interface() (in module salt.modules.debian_ip), 570
get_interface() (in module salt.modules.rh_ip), 875
get_interface() (in module salt.modules.win_ip), 992
get_ip() (in module salt.cloud.clouds.cloudstack), 349
get_ip() (in module salt.modules.hosts), 670
get_jid() (in module salt.modules.ret), 874
get_jid() (in module salt.returners.couchbase_return), 1130
get_jid() (in module salt.returners.couchdb_return), 1131
get_jid() (in module salt.returners.etcd_return), 1133
get_jid() (in module salt.returners.local_cache), 1134
get_jid() (in module salt.returners.memcache_return), 1135
get_jid() (in module salt.returners.mongo_future_return), 1136
get_jid() (in module salt.returners.mongo_return), 1137
get_jid() (in module salt.returners.multi_returner), 1138
get_jid() (in module salt.returners.mysql), 1140
get_jid() (in module salt.returners.odbc), 1142
get_jid() (in module salt.returners.postgres), 1144
get_jid() (in module salt.returners.postgres_local_cache), 1146
get_jid() (in module salt.returners.redis_return), 1147
get_jid() (in module salt.returners.sqlite3_return), 1150
get_jids() (in module salt.modules.ret), 874
get_jids() (in module salt.returners.couchbase_return), 1130
get_jids() (in module salt.returners.couchdb_return), 1131
get_jids() (in module salt.returners.etcd_return), 1133
get_jids() (in module salt.returners.local_cache), 1135
get_jids() (in module salt.returners.memcache_return), 1135
get_jids() (in module salt.returners.mongo_future_return), 1136
get_jids() (in module salt.returners.multi_returner), 1138
get_jids() (in module salt.returners.mysql), 1140
get_jids() (in module salt.returners.odbc), 1142
get_jids() (in module salt.returners.postgres), 1144
get_jids() (in module salt.returners.postgres_local_cache), 1146
get_jids() (in module salt.returners.redis_return), 1147
get_jids() (in module salt.returners.sqlite3_return), 1150
get_kernels() (in module salt.cloud.clouds.linode), 368
get_key() (in module salt.cloud.clouds.cloudstack), 349
get_key() (in module salt.modules.gpg), 657
get_key() (in module salt.modules.redismod), 869
get_keyid() (in module salt.cloud.clouds.digital_ocean), 350
get_keyid() (in module salt.cloud.clouds.digital_ocean_v2), 352
get_keypair() (in module salt.cloud.clouds.cloudstack), 349
get_known_host() (in module salt.modules.ssh), 925
get_latest_snapshot() (in module salt.modules.artifactory), 499
get_lb_conn() (in module salt.cloud.clouds.gce), 362
get_load() (in module salt.returners.couchbase_return), 1130
get_load() (in module salt.returners.etcd_return), 1133
get_load() (in module salt.returners.local_cache), 1135
get_load() (in module salt.returners.memcache_return), 1135
get_load() (in module salt.returners.mongo_future_return), 1136
get_load() (in module salt.returners.multi_returner), 1138
get_load() (in module salt.returners.mysql), 1140
get_load() (in module salt.returners.odbc), 1142
get_load() (in module salt.returners.postgres), 1144

get_load() (in module salt.returners.postgres_local_cache), 1146
get_load() (in module salt.returners.redis_return), 1147
get_load() (in module salt.returners.sqlite3_return), 1151
get_locale() (in module salt.modules.localemod), 705
get_location() (in module salt.cloud.clouds.aliyun), 346
get_location() (in module salt.cloud.clouds.cloudstack), 349
get_location() (in module salt.cloud.clouds.digital_ocean), 350
get_location() (in module salt.cloud.clouds.digital_ocean_v2), 352
get_location() (in module salt.cloud.clouds.ec2), 355
get_location() (in module salt.cloud.clouds.libcloud_aws), 366
get_location() (in module salt.cloud.clouds.linode), 368
get_location() (in module salt.cloud.clouds.opennebula), 374
get_location() (in module salt.cloud.clouds.softlayer), 384
get_location() (in module salt.cloud.clouds.softlayer_hw), 385
get_locked_packages() (in module salt.modules.yumpkg), 1018
get_loginclass() (in module salt.modules.useradd), 972
get_macs() (in module salt.modules.smartos_vmadm), 901
get_macs() (in module salt.modules.virt), 978
get_macs() (in module salt.modules.xapi), 1011
get_makeopts() (in module salt.modules.makeconf), 732
get_managed() (in module salt.modules.file), 613
get_master_status() (in module salt.modules.mysql), 753
get_minions() (in module salt.modules.ret), 874
get_minions() (in module salt.returners.couchdb_return), 1131
get_minions() (in module salt.returners.etcd_return), 1133
get_minions() (in module salt.returners.memcache_return), 1136
get_minions() (in module salt.returners.mongo_future_return), 1137
get_minions() (in module salt.returners.mysql), 1140
get_minions() (in module salt.returners.odbc), 1142
get_minions() (in module salt.returners.postgres), 1144
get_minions() (in module salt.returners.redis_return), 1147
get_minions() (in module salt.returners.sqlite3_return), 1151
get_missing_flags() (in module salt.modules.portage_config), 839
get_mode() (in module salt.modules.file), 613
get_mode() (in module salt.modules.quota), 860
get_mode() (in module salt.modules.win_file), 987
get_modules() (in module salt.modules.eselect), 601
get_network_profile() (in module salt.modules.lxc), 715
get_network_settings() (in module salt.modules.debian_ip), 570
get_network_settings() (in module salt.modules.rh_ip), 875
get_networkid() (in module salt.cloud.clouds.cloudstack), 349
get_nics() (in module salt.modules.virt), 978
get_nics() (in module salt.modules.xapi), 1011
get_node() (in module salt.cloud.clouds.cloudstack), 349
get_node() (in module salt.cloud.clouds.linode), 368
get_node() (in module salt.cloud.clouds.openstack), 376
get_offset() (in module salt.modules.timezone), 959
get_offset() (in module salt.modules.win_timezone), 1005
get_one_kernel() (in module salt.cloud.clouds.linode), 368
get_option() (in module salt.modules.ini_manage), 680
get_opts() (in module salt.modules.test), 956
get_or_set_hash() (in module salt.modules.grains), 662
get_output_volume() (in module salt.modules.osxdesktop), 806
get_parameter() (in module salt.modules.lxc), 716
get_password() (in module salt.cloud.clouds.cloudstack), 349
get_password() (in module salt.cloud.clouds.linode), 368
get_password_data() (in module salt.cloud.clouds.ec2), 356
get_path() (in module salt.modules.win_path), 996
get_pending_computer_name() (in module salt.modules.win_system), 1003
get_pgid() (in module salt.modules.win_file), 987
get_pgroup() (in module salt.modules.win_file), 987
get_pid_list() (in module salt.modules.ps), 850
get_placementgroup() (in module salt.cloud.clouds.ec2), 356
get_policy() (in module salt.modules.iptables), 686
get_private_ip() (in module salt.cloud.clouds.linode), 368
get_profiles() (in module salt.modules.virt), 978
get_project() (in module salt.cloud.clouds.cloudstack), 349
get_provider() (in module salt.cloud.clouds.ec2), 356
get_provider() (in module salt.cloud.clouds.lxc), 369
get_pubkey() (in module salt.cloud.clouds.linode), 368
get_quotas_tenant() (in module salt.modules.neutron), 778
get_record() (in module salt.modules.boto_route53), 520
get_release() (in module salt.modules.artifactory), 499
get_repo() (in module salt.modules.aptpkg), 490
get_repo() (in module salt.modules.yumpkg), 1018
get_repo() (in module salt.modules.zypper), 1035
get_repo_data() (in module salt.modules.win_pkg), 996
get_resource_content() (in module salt.modules.virtualenv_mod), 983
get_resource_path() (in module salt.modules.virtualenv_mod), 983

get_resources_nodes() (in module salt.cloud.clouds.proxmox), 380
get_resources_vms() (in module salt.cloud.clouds.proxmox), 380
get_role_policy() (in module salt.modules.boto_iam), 519
get_routes() (in module salt.modules.debian_ip), 570
get_routes() (in module salt.modules.rh_ip), 875
get_rule() (in module salt.modules.win_firewall), 990
get_rule_handle() (in module salt.modules.nftables), 790
get_rules() (in module salt.modules.iptables), 686
get_rules() (in module salt.modules.lvs), 709
get_rules() (in module salt.modules.nftables), 790
get_running() (in module salt.modules.modjk), 744
get_running() (in module salt.modules.smf), 903
get_salt_interface() (in module salt.cloud.clouds.ec2), 356
get_salt_interface() (in module salt.cloud.clouds.libcloud_aws), 366
get_salt_interface() (in module salt.cloud.clouds.nova), 372
get_salt_interface() (in module salt.cloud.clouds.openstack), 377
get_salt_interface() (in module salt.cloud.clouds.rackspace), 382
get_saved_policy() (in module salt.modules.iptables), 686
get_saved_rules() (in module salt.modules.iptables), 686
get_saved_rules() (in module salt.modules.nftables), 790
get_scaling_policy_arn() (in module salt.modules.boto_asg), 508
get_sd_auth() (in module salt.modules.serverdensity_device), 895
get_secret_key() (in module salt.modules.gpg), 657
get_section() (in module salt.modules.ini_manage), 680
get_securitygroup() (in module salt.cloud.clouds.aliyun), 346
get_selections() (in module salt.modules.aptpkg), 490
get_selections() (in module salt.modules.debconfmod), 568
get_selinux_context() (in module salt.modules.file), 614
get_servers() (in module salt.modules.win_ntp), 995
get_service_name() (in module salt.modules.win_service), 1001
get_services() (in module salt.modules.firewalld), 627
get_site_packages() (in module salt.modules.virtualenv_mod), 984
get_size() (in module salt.cloud.clouds.aliyun), 346
get_size() (in module salt.cloud.clouds.cloudstack), 349
get_size() (in module salt.cloud.clouds.digital_ocean), 350
get_size() (in module salt.cloud.clouds.digital_ocean_v2), 352
get_size() (in module salt.cloud.clouds.gogrid), 364
get_size() (in module salt.cloud.clouds.linode), 368
get_size() (in module salt.cloud.clouds.nova), 372
get_size() (in module salt.cloud.clouds.openstack), 377
get_size() (in module salt.cloud.clouds.rackspace), 382
get_slave_status() (in module salt.modules.mysql), 753
get_snapshot() (in module salt.modules.artifactory), 500
get_spot_config() (in module salt.cloud.clouds.ec2), 356
get_ssh_gateway_config() (in module salt.cloud.clouds.ec2), 356
get_ssh_key_filename() (in module salt.cloud.clouds.linode), 368
get_stopped() (in module salt.modules.smf), 903
get_str() (in module salt.modules.mod_random), 742
get_subnet_association() (in module salt.modules.boto_vpc), 531
get_subnet_length() (in module salt.modules.win_ip), 992
get_subnetid() (in module salt.cloud.clouds.ec2), 356
get_sum() (in module salt.modules.file), 614
get_swap() (in module salt.cloud.clouds.linode), 368
get_sync() (in module salt.modules.makeconf), 732
get_sys() (in module salt.modules.keyboard), 692
get_system_date() (in module salt.modules.win_system), 1003
get_system_time() (in module salt.modules.win_system), 1003
get_tags() (in module salt.cloud.clouds.ec2), 356
get_tags() (in module salt.cloud.clouds.libcloud_aws), 366
get_target() (in module salt.modules.aliases), 485
get_target_list() (in module salt.modules.eselect), 601
get_template() (in module salt.modules.boto_cfn), 509
get_template() (in module salt.modules.cp), 558
get_tenancy() (in module salt.cloud.clouds.ec2), 356
get_uid() (in module salt.modules.file), 614
get_uid() (in module salt.modules.win_file), 987
get_url() (in module salt.modules.cp), 559
get_user() (in module salt.modules.file), 614
get_user() (in module salt.modules.ilo), 673
get_user() (in module salt.modules.win_file), 987
get_users() (in module salt.modules.ps), 850
get_valid_salt_views() (in module salt.returners.couchdb_return), 1131
get_var() (in module salt.modules.makeconf), 732
get_vm_status() (in module salt.cloud.clouds.proxmox), 380
get_vmconfig() (in module salt.cloud.clouds.proxmox), 380
get_weight() (in module salt.modules.haproxyconn), 666
get_x() (in module salt.modules.keyboard), 692
get_xml() (in module salt.modules.virt), 978
get_yaml_loader() (in module salt.renderers.yaml), 1121
get_zone() (in module salt.modules.timezone), 959
get_zone() (in module salt.modules.win_timezone), 1005
get_zonecode() (in module salt.modules.timezone), 959
get_zonecode() (in module salt.modules.win_timezone), 1005

get_zones() (in module salt.modules.firewalld), 627
GetAvailableCategories() (salt.modules.win_update.PyWinUpdater method), 1006
GetAvailableCategories() (salt.states.win_update.PyWinUpdater method), 1431
GetCategories() (salt.modules.win_update.PyWinUpdater method), 1006
GetCategories() (salt.states.win_update.PyWinUpdater method), 1431
getClockFormat() (in module salt.modules.gnomedesktop), 653
getClockShowDate() (in module salt.modules.gnomedesktop), 653
GetDownloadResults() (salt.modules.win_update.PyWinUpdater method), 1006
GetDownloadResults() (salt.states.win_update.PyWinUpdater method), 1431
getenforce() (in module salt.modules.selinux), 894
getent() (in module salt.modules.groupadd), 664
getent() (in module salt.modules.mac_group), 724
getent() (in module salt.modules.mac_user), 725
getent() (in module salt.modules.pw_group), 856
getent() (in module salt.modules.pw_user), 857
getent() (in module salt.modules.solaris_group), 907
getent() (in module salt.modules.solaris_user), 909
getent() (in module salt.modules.useradd), 972
getent() (in module salt.modules.win_groupadd), 991
getent() (in module salt.modules.win_useradd), 1009
getfacl() (in module salt.modules.linux_acl), 702
getgoal() (in module salt.modules.moosefs), 749
getIdleActivation() (in module salt.modules.gnomedesktop), 654
getIdleDelay() (in module salt.modules.gnomedesktop), 654
GetInstallationResults() (salt.modules.win_update.PyWinUpdater method), 1006
GetInstallationResults() (salt.states.win_update.PyWinUpdater method), 1431
GetInstallationResultsPretty() (salt.modules.win_update.PyWinUpdater method), 1007
GetSearchResults() (salt.modules.win_update.PyWinUpdater method), 1007
GetSearchResultsPretty() (salt.modules.win_update.PyWinUpdater method), 1007
getsebool() (in module salt.modules.selinux), 894
getsid() (in module salt.modules.win_service), 1001
getval() (in module salt.modules.data), 566
getvals() (in module salt.modules.data), 566
gid_to_group() (in module salt.modules.file), 614
gid_to_group() (in module salt.modules.win_file), 988
gitfs_base
    conf/master, 427
gitfs_env_blacklist
    conf/master, 428
gitfs_env_whitelist
    conf/master, 428
gitfs_insecure_auth
    conf/master, 428
gitfs_mountpoint
    conf/master, 427
gitfs_passphrase
    conf/master, 429
gitfs_password
    conf/master, 428
gitfs_privkey
    conf/master, 429
gitfs_provider
    conf/master, 426
gitfs_pubkey
    conf/master, 429
gitfs_remotes
    conf/master, 426
gitfs_root
    conf/master, 427
gitfs_ssl_verify
    conf/master, 427
gitfs_user
    conf/master, 428
GitPillar (class in salt.pillar.git_pillar), 1083
GivenStatement (class in salt.modules.syslog_ng), 942
glob() (in module salt.modules.match), 736
global_settings() (in module salt.modules.ilo), 673
glsa_check_list() (in module salt.modules.gentoolkitmod), 643
Grain, 1677
grain() (in module salt.modules.match), 736
grain_pcre() (in module salt.modules.match), 737
grains() (in module salt.loader), 336
grains() (in module salt.runners.cache), 1157
grains_dirs
    conf/minion, 449
grains_refresh() (in module salt.modules.rest_sample), 873
grant_access_to_shared_folders_to() (in module salt.modules.vbox_guest), 975
grant_access_to_shared_folders_to() (in module salt.states.vbox_guest), 1426
grant_add() (in module salt.modules.mysql), 754
grant_exists() (in module salt.modules.mysql), 754
grant_revoke() (in module salt.modules.mysql), 754
grep() (in module salt.modules.file), 615
group_create() (in module salt.modules.postgres), 843
group_diff() (in module salt.modules.yumpkg), 1019
group_info() (in module salt.modules.yumpkg), 1019

group_install() (in module salt.modules.yumpkg), 1019
group_list() (in module salt.modules.yumpkg), 1020
group_remove() (in module salt.modules.postgres), 843
group_to_gid() (in module salt.modules.file), 615
group_to_gid() (in module salt.modules.win_file), 988
group_update() (in module salt.modules.postgres), 844
groups() (in module salt.auth.ldap), 307
groups() (in module salt.auth.pam), 309
gunzip() (in module salt.modules.archive), 496
gzip() (in module salt.modules.archive), 497

H

Halite, **1677**
halt() (in module salt.modules.system), 952
halt() (in module salt.modules.win_system), 1003
handle (salt.auth.pam.PamHandle attribute), 308
has_exec() (in module salt.modules.cmdmod), 548
has_flag() (in module salt.modules.portage_config), 839
has_instance() (in module salt.modules.cloud), 545
has_pair() (in module salt.modules.hosts), 670
has_powerpath() (in module salt.modules.powerpath), 849
has_powershell() (in module salt.modules.win_service), 1001
has_target() (in module salt.modules.aliases), 485
has_use() (in module salt.modules.portage_config), 839
has_value() (in module salt.modules.environ), 599
has_value() (in module salt.modules.grains), 662
hash() (in module salt.modules.mod_random), 742
hash() (in module salt.runners.survey), 1179
hash_file() (in module salt.modules.cp), 559
hash_known_hosts() (in module salt.modules.ssh), 925
hash_type
    conf/master, 425
    conf/minion, 451
head() (in module salt.modules.s3), 885
head() (in module salt.modules.swift), 940
held() (in module salt.states.apt), 1238
hget() (in module salt.modules.redismod), 869
hgetall() (in module salt.modules.redismod), 870
hgfs_base
    conf/master, 431
hgfs_branch_method
    conf/master, 430
hgfs_env_blacklist
    conf/master, 431
hgfs_env_whitelist
    conf/master, 431
hgfs_mountpoint
    conf/master, 430
hgfs_remotes
    conf/master, 429
hgfs_root
    conf/master, 430
high() (in module salt.modules.state), 928
Highdata, **1677**
Highstate, **1678**
highstate() (in module salt.modules.state), 928
hmac_signature() (in module salt.modules.hashutil), 667
hold() (in module salt.modules.aptpkg), 490
hold() (in module salt.modules.postfix), 840
hold() (in module salt.modules.yumpkg), 1020
host_keys() (in module salt.modules.ssh), 925
hosts_append() (in module salt.modules.dnsutil), 577
hosts_remove() (in module salt.modules.dnsutil), 577
hw_addr() (in module salt.modules.network), 766
hw_addr() (in module salt.modules.win_network), 993
hwaddr() (in module salt.modules.network), 766
hwaddr() (in module salt.modules.win_network), 994
hyper_info() (in module salt.runners.virt), 1180

I

iam_profile() (in module salt.cloud.clouds.ec2), 356
iam_profile() (in module salt.cloud.clouds.libcloud_aws), 366
id
    conf/minion, 445
ignore() (in module salt.modules.softwareupdate), 905
ignore_cidr() (in module salt.cloud.clouds.nova), 372
ignore_cidr() (in module salt.cloud.clouds.openstack), 377
ignore_null (salt.pillar.mysql.merger attribute), 1088
image_create() (in module salt.modules.glance), 651
image_delete() (in module salt.modules.glance), 651
image_list() (in module salt.modules.glance), 651
image_list() (in module salt.modules.nova), 794
image_meta_delete() (in module salt.modules.nova), 794
image_meta_set() (in module salt.modules.nova), 794
image_show() (in module salt.modules.glance), 651
images() (in module salt.modules.lxc), 716
import_() (in module salt.modules.zpool), 1033
import_image() (in module salt.modules.dockerio), 583
import_image() (in module salt.modules.smartos_imgadm), 900
import_key() (in module salt.modules.gpg), 657
import_status() (in module salt.modules.solr), 918
in_subnet() (in module salt.modules.network), 767
in_subnet() (in module salt.modules.win_network), 994
include
    conf/master, 441
    conf/minion, 455
increment() (in module salt.modules.memcached), 740
index() (in module salt.modules.elasticsearch), 598
indexes() (in module salt.modules.sqlite3), 924
indices() (in module salt.modules.sqlite3), 924
info() (in module salt.modules.bsd_shadow), 536
info() (in module salt.modules.btrfs), 538
info() (in module salt.modules.cassandra), 540

info() (in module salt.modules.dockerio), 583
info() (in module salt.modules.groupadd), 664
info() (in module salt.modules.lxc), 716
info() (in module salt.modules.mac_group), 724
info() (in module salt.modules.mac_user), 725
info() (in module salt.modules.pw_group), 856
info() (in module salt.modules.pw_user), 857
info() (in module salt.modules.redismod), 870
info() (in module salt.modules.shadow), 897
info() (in module salt.modules.solaris_group), 907
info() (in module salt.modules.solaris_shadow), 908
info() (in module salt.modules.solaris_user), 910
info() (in module salt.modules.svn), 938
info() (in module salt.modules.useradd), 972
info() (in module salt.modules.win_groupadd), 991
info() (in module salt.modules.win_shadow), 1002
info() (in module salt.modules.win_useradd), 1009
info() (in module salt.modules.xfs), 1015
info() (in module salt.runners.lxc), 1166
init() (in module salt.fileserver.gitfs), 467
init() (in module salt.fileserver.hgfs), 468
init() (in module salt.fileserver.svnfs), 472
init() (in module salt.modules.git), 646
init() (in module salt.modules.lxc), 716
init() (in module salt.modules.qemu_nbd), 860
init() (in module salt.modules.smartos_vmadm), 901
init() (in module salt.modules.system), 952
init() (in module salt.modules.virt), 978
init() (in module salt.modules.win_system), 1003
init() (in module salt.runners.lxc), 1167
init() (in module salt.runners.virt), 1180
inodeusage() (in module salt.modules.disk), 574
insert() (in module salt.modules.iptables), 686
insert() (in module salt.modules.mongodb), 747
insert() (in module salt.modules.nftables), 790
insert() (in module salt.runners.queue), 1174
insert() (in module salt.states.iptables), 1338
insert() (in module salt.states.nftables), 1371
inspect_container() (in module salt.modules.dockerio), 583
inspect_image() (in module salt.modules.dockerio), 583
install() (in module salt.modules.alternatives), 486
install() (in module salt.modules.aptpkg), 491
install() (in module salt.modules.brew), 532
install() (in module salt.modules.chocolatey), 542
install() (in module salt.modules.composer), 553
install() (in module salt.modules.cpan), 560
install() (in module salt.modules.cyg), 563
install() (in module salt.modules.ebuild), 594
install() (in module salt.modules.freebsdpkg), 632
install() (in module salt.modules.freebsdports), 634
install() (in module salt.modules.gem), 637
install() (in module salt.modules.macports), 726
install() (in module salt.modules.npm), 797
install() (in module salt.modules.openbsdpkg), 799
install() (in module salt.modules.pacman), 807
install() (in module salt.modules.pecl), 815
install() (in module salt.modules.pip), 818
install() (in module salt.modules.pkgin), 823
install() (in module salt.modules.pkgng), 828
install() (in module salt.modules.pkgutil), 837
install() (in module salt.modules.pyenv), 858
install() (in module salt.modules.rbenv), 866
install() (in module salt.modules.rest_package), 872
install() (in module salt.modules.rvm), 883
install() (in module salt.modules.softwareupdate), 905
install() (in module salt.modules.solarisips), 910
install() (in module salt.modules.solarispkg), 913
install() (in module salt.modules.win_pkg), 996
install() (in module salt.modules.win_servermanager), 999
install() (in module salt.modules.yumpkg), 1020
install() (in module salt.modules.zypper), 1035
install() (in module salt.states.alternatives), 1236
Install() (salt.modules.win_update.PyWinUpdater method), 1007
Install() (salt.states.win_update.PyWinUpdater method), 1431
install_agent() (in module salt.modules.serverdensity_device), 895
install_cygwin() (in module salt.modules.chocolatey), 542
install_gem() (in module salt.modules.chocolatey), 542
install_missing() (in module salt.modules.chocolatey), 543
install_pyenv() (in module salt.states.pyenv), 1394
install_python() (in module salt.modules.chocolatey), 543
install_python() (in module salt.modules.pyenv), 858
install_rbenv() (in module salt.states.rbenv), 1400
install_ruby() (in module salt.modules.rbenv), 866
install_ruby() (in module salt.modules.rvm), 883
install_updates() (in module salt.modules.win_update), 1007
install_webpi() (in module salt.modules.chocolatey), 543
install_windowsfeatures() (in module salt.modules.chocolatey), 543
installed() (in module salt.modules.rest_package), 872
installed() (in module salt.states.composer), 1278
installed() (in module salt.states.cyg), 1283
installed() (in module salt.states.dockerio), 1287
installed() (in module salt.states.gem), 1314
installed() (in module salt.states.npm), 1371
installed() (in module salt.states.pecl), 1374
installed() (in module salt.states.pip_state), 1374
installed() (in module salt.states.pkg), 1378
installed() (in module salt.states.ports), 1387
installed() (in module salt.states.pyenv), 1395
installed() (in module salt.states.rbenv), 1400

installed() (in module salt.states.rvm), 1404
installed() (in module salt.states.win_servermanager), 1429
installed() (in module salt.states.win_update), 1432
installed() (in module salt.states.zcbuildout), 1433
installed_extensions() (in module salt.modules.postgres), 844
instance_profile_exists() (in module salt.modules.boto_iam), 519
interface
  conf/master, 414
interface() (in module salt.modules.network), 767
interface_ip() (in module salt.modules.network), 767
interfaces() (in module salt.modules.bridge), 535
interfaces() (in module salt.modules.network), 767
interfaces() (in module salt.modules.win_network), 994
interfaces_names() (in module salt.modules.win_network), 994
Inventory (class in salt.roster.ansible), 1154
inventory() (in module salt.modules.xfs), 1015
iostat() (in module salt.modules.zpool), 1033
ip_addrs() (in module salt.modules.network), 767
ip_addrs() (in module salt.modules.win_network), 994
ip_addrs6() (in module salt.modules.network), 767
ip_addrs6() (in module salt.modules.win_network), 994
ip_in_subnet() (in module salt.modules.network), 767
ipaddrs() (in module salt.modules.network), 767
ipaddrs() (in module salt.modules.win_network), 994
ipaddrs6() (in module salt.modules.network), 768
ipaddrs6() (in module salt.modules.win_network), 994
ipc_mode
  conf/minion, 448
ipcidr() (in module salt.modules.match), 737
ipv6
  conf/master, 414
is_available_extension() (in module salt.modules.postgres), 844
is_blkdev() (in module salt.modules.file), 615
is_cached() (in module salt.modules.cp), 559
is_chrdev() (in module salt.modules.file), 615
is_disabled() (in module salt.modules.win_ip), 992
is_enabled() (in module salt.modules.freebsdjail), 629
is_enabled() (in module salt.modules.win_ip), 992
is_fifo() (in module salt.modules.file), 615
is_fuse_exec() (in module salt.modules.mount), 749
is_hyper() (in module salt.modules.virt), 978
is_hyper() (in module salt.modules.xapi), 1011
is_installed() (in module salt.modules.pyenv), 858
is_installed() (in module salt.modules.rbenv), 867
is_installed() (in module salt.modules.rvm), 883
is_installed() (in module salt.modules.solarisips), 911
is_installed_extension() (in module salt.modules.postgres), 844
is_jail() (in module salt.modules.poudriere), 848
is_kvm_hyper() (in module salt.modules.virt), 978
is_link() (in module salt.modules.file), 615
is_link() (in module salt.modules.win_file), 988
is_loaded() (in module salt.modules.freebsdkmod), 630
is_loaded() (in module salt.modules.kmod), 698
is_loopback() (in module salt.modules.network), 768
is_mounted() (in module salt.modules.mount), 749
is_present() (in module salt.modules.portage_config), 840
is_private() (in module salt.modules.network), 768
is_replication_enabled() (in module salt.modules.solr), 919
is_running() (in module salt.modules.dockerio), 584
is_running() (in module salt.modules.saltutil), 887
is_xen_hyper() (in module salt.modules.virt), 978
item() (in module salt.modules.environ), 599
item() (in module salt.modules.grains), 662
item() (in module salt.modules.pillar), 816
items() (in module salt.modules.environ), 599
items() (in module salt.modules.grains), 662
items() (in module salt.modules.pillar), 817

J

Jinja, **1677**
Job, **1677**
Job ID, **1677**
Job Management, 153
job_cache
  conf/master, 417
jobcheck() (in module salt.modules.at), 501
Jobs (class in salt.netapi.rest_cherrypy.app), 1048
JobsSaltAPIHandler (in module salt.netapi.rest_tornado.saltnado), 1064
join() (in module salt.modules.file), 615
join() (in module salt.states.rabbitmq_cluster), 1396
join_cluster() (in module salt.modules.rabbitmq), 862
join_domain() (in module salt.modules.win_system), 1004
joined() (in module salt.states.rabbitmq_cluster), 1396

K

keep_jobs
  conf/master, 416
keepvol_on_destroy() (in module salt.cloud.clouds.ec2), 357
key_json() (in module salt.pillar.redismod), 1095
key_regen() (in module salt.runners.manage), 1169
key_str() (in module salt.wheel.key), 1445
key_type() (in module salt.modules.redismod), 870
key_value() (in module salt.pillar.redismod), 1095
key_value_to_tree() (in module salt.pillar.pepa), 1093
keygen() (in module salt.runners.nacl), 1171
keyname() (in module salt.cloud.clouds.ec2), 357
keyname() (in module salt.cloud.clouds.libcloud_aws), 366

keypair_add() (in module salt.modules.nova), 794
keypair_delete() (in module salt.modules.nova), 794
keypair_list() (in module salt.modules.nova), 794
Keys (class in salt.netapi.rest_cherrypy.app), 1054
keys() (in module salt.modules.redismod), 870
keys() (in module salt.states.libvirt), 1347
keyspaces() (in module salt.modules.cassandra), 540
kill() (in module salt.modules.dockerio), 584
kill_job() (in module salt.modules.saltutil), 887
kill_pid() (in module salt.modules.ps), 850
kwarg() (in module salt.modules.test), 956

L

lastsave() (in module salt.modules.redismod), 870
latest() (in module salt.states.git), 1315
latest() (in module salt.states.hg), 1326
latest() (in module salt.states.pkg), 1382
latest() (in module salt.states.svn), 1417
latest_version() (in module salt.modules.aptpkg), 492
latest_version() (in module salt.modules.brew), 533
latest_version() (in module salt.modules.ebuild), 595
latest_version() (in module salt.modules.freebsdpkg), 633
latest_version() (in module salt.modules.macports), 727
latest_version() (in module salt.modules.openbsdpkg), 800
latest_version() (in module salt.modules.pacman), 808
latest_version() (in module salt.modules.pkgin), 824
latest_version() (in module salt.modules.pkgng), 830
latest_version() (in module salt.modules.pkgutil), 837
latest_version() (in module salt.modules.solarisips), 911
latest_version() (in module salt.modules.solarispkg), 914
latest_version() (in module salt.modules.win_pkg), 997
latest_version() (in module salt.modules.yumpkg), 1021
latest_version() (in module salt.modules.zypper), 1036
launch_configuration_exists() (in module salt.modules.boto_asg), 508
lb_edit() (in module salt.modules.modjk), 744
lbmethods() (salt.runners.f5.F5Mgmt method), 1160
lchown() (in module salt.modules.file), 616
lchown() (in module salt.modules.win_file), 988
leaks() (in module salt.modules.tomcat), 966
license_absent() (in module salt.states.powerpath), 1393
license_present() (in module salt.states.powerpath), 1393
link() (in module salt.modules.file), 616
list() (in module salt.modules.splunk_search), 923
list() (in module salt.runners.virt), 1181
list_() (in module salt.modules.bridge), 535
list_() (in module salt.modules.chocolatey), 543
list_() (in module salt.modules.cpan), 560
list_() (in module salt.modules.cyg), 563
list_() (in module salt.modules.gem), 638
list_() (in module salt.modules.lvs), 709
list_() (in module salt.modules.lxc), 718
list_() (in module salt.modules.match), 737
list_() (in module salt.modules.mdadm), 739
list_() (in module salt.modules.nova), 795
list_() (in module salt.modules.npm), 798
list_() (in module salt.modules.parted), 813
list_() (in module salt.modules.pecl), 815
list_() (in module salt.modules.pip), 820
list_() (in module salt.modules.pyenv), 859
list_() (in module salt.modules.rbenv), 867
list_() (in module salt.modules.rest_service), 873
list_() (in module salt.modules.rvm), 883
list_() (in module salt.modules.schedule), 891
list_() (in module salt.modules.win_autoruns), 984
list_() (in module salt.modules.zfs), 1029
list_() (in module salt.modules.zpool), 1033
list_() (in module salt.runners.lxc), 1167
list_() (in module salt.wheel.key), 1445
list_absent() (in module salt.states.grains), 1324
list_active_vms() (in module salt.modules.smartos_vmadm), 901
list_active_vms() (in module salt.modules.virt), 979
list_aliases() (in module salt.modules.aliases), 486
list_all() (in module salt.modules.firewalld), 627
list_all() (in module salt.modules.freebsdports), 635
list_all() (in module salt.modules.splunk_search), 923
list_all() (in module salt.wheel.key), 1445
list_avail() (in module salt.modules.localemod), 706
list_availability_zones() (in module salt.cloud.clouds.aliyun), 346
list_available() (in module salt.modules.win_pkg), 997
list_available() (in module salt.modules.win_servermanager), 999
list_backups() (in module salt.modules.file), 616
list_backups_dir() (in module salt.modules.file), 616
list_clusters() (in module salt.cloud.clouds.vsphere), 387
list_configured_members() (in module salt.modules.modjk), 745
list_custom_images() (in module salt.cloud.clouds.softlayer), 384
list_datacenters() (in module salt.cloud.clouds.vsphere), 387
list_datastores() (in module salt.cloud.clouds.vsphere), 387
list_deployments() (in module salt.modules.jboss7), 688
list_disabled() (in module salt.modules.state), 928
list_disks() (in module salt.cloud.clouds.msazure), 370
list_downloads() (in module salt.modules.softwareupdate), 906
list_env() (in module salt.wheel.file_roots), 1444
list_env() (in module salt.wheel.pillar_roots), 1446
list_escalation_policies() (in module salt.modules.pagerduty), 810
list_escalation_policies() (in module salt.runners.pagerduty), 1172
list_exports() (in module salt.modules.nfs3), 787

list_extensions() (in module salt.modules.neutron), 778
list_floatingips() (in module salt.modules.neutron), 778
list_folders() (in module salt.cloud.clouds.vsphere), 387
list_functions() (in module salt.modules.sysmod), 947
list_groups() (in module salt.modules.mac_user), 725
list_groups() (in module salt.modules.pw_user), 857
list_groups() (in module salt.modules.solaris_user), 910
list_groups() (in module salt.modules.useradd), 973
list_groups() (in module salt.modules.win_useradd), 1009
list_hosted_services() (in module salt.cloud.clouds.msazure), 370
list_hosts() (in module salt.cloud.clouds.vsphere), 387
list_hosts() (in module salt.modules.hosts), 670
list_ignored() (in module salt.modules.softwareupdate), 906
list_ikepolicies() (in module salt.modules.neutron), 779
list_images() (in module salt.modules.cloud), 546
list_images() (in module salt.runners.cloud), 1158
list_images() (salt.cloud.CloudClient method), 344
list_inactive_vms() (in module salt.modules.smartos_vmadm), 901
list_inactive_vms() (in module salt.modules.virt), 979
list_incidents() (in module salt.modules.pagerduty), 810
list_incidents() (in module salt.runners.pagerduty), 1172
list_installed() (in module salt.modules.smartos_imgadm), 900
list_installed() (in module salt.modules.win_servermanager), 999
list_installed_patterns() (in module salt.modules.zypper), 1036
list_ipsec_site_connections() (in module salt.modules.neutron), 779
list_ipsecpolicies() (in module salt.modules.neutron), 779
list_items() (in module salt.runners.queue), 1174
list_jails() (in module salt.modules.poudriere), 848
list_job() (in module salt.runners.jobs), 1165
list_jobs() (in module salt.runners.jobs), 1165
list_keypairs() (in module salt.cloud.clouds.digital_ocean), 351
list_keypairs() (in module salt.cloud.clouds.digital_ocean_v2), 352
list_keys() (in module salt.modules.gpg), 658
list_length() (in module salt.runners.queue), 1174
list_licenses() (in module salt.modules.powerpath), 849
list_local() (in module salt.modules.layman), 700
list_locations() (in module salt.modules.cloud), 546
list_locations() (in module salt.runners.cloud), 1158
list_locations() (salt.cloud.CloudClient method), 344
list_locks() (in module salt.modules.zypper), 1036
list_maintenance_windows() (in module salt.modules.pagerduty), 810
list_maintenance_windows() (in module salt.runners.pagerduty), 1172
list_master() (in module salt.modules.cp), 559
list_master_dirs() (in module salt.modules.cp), 559
list_master_symlinks() (in module salt.modules.cp), 559
list_minion() (in module salt.modules.cp), 559
list_modules() (in module salt.modules.sysmod), 947
list_monitor_data() (in module salt.cloud.clouds.aliyun), 346
list_networks() (in module salt.modules.neutron), 779
list_nodes() (in module salt.cloud.clouds.aliyun), 346
list_nodes() (in module salt.cloud.clouds.cloudstack), 349
list_nodes() (in module salt.cloud.clouds.digital_ocean), 351
list_nodes() (in module salt.cloud.clouds.digital_ocean_v2), 352
list_nodes() (in module salt.cloud.clouds.ec2), 357
list_nodes() (in module salt.cloud.clouds.gce), 362
list_nodes() (in module salt.cloud.clouds.gogrid), 364
list_nodes() (in module salt.cloud.clouds.linode), 368
list_nodes() (in module salt.cloud.clouds.lxc), 369
list_nodes() (in module salt.cloud.clouds.msazure), 370
list_nodes() (in module salt.cloud.clouds.nova), 372
list_nodes() (in module salt.cloud.clouds.opennebula), 374
list_nodes() (in module salt.cloud.clouds.openstack), 377
list_nodes() (in module salt.cloud.clouds.parallels), 378
list_nodes() (in module salt.cloud.clouds.proxmox), 380
list_nodes() (in module salt.cloud.clouds.rackspace), 382
list_nodes() (in module salt.cloud.clouds.saltify), 383
list_nodes() (in module salt.cloud.clouds.softlayer), 384
list_nodes() (in module salt.cloud.clouds.softlayer_hw), 385
list_nodes() (in module salt.cloud.clouds.vsphere), 387
list_nodes_full() (in module salt.cloud.clouds.aliyun), 347
list_nodes_full() (in module salt.cloud.clouds.cloudstack), 349
list_nodes_full() (in module salt.cloud.clouds.digital_ocean), 351
list_nodes_full() (in module salt.cloud.clouds.digital_ocean_v2), 352
list_nodes_full() (in module salt.cloud.clouds.ec2), 357
list_nodes_full() (in module salt.cloud.clouds.gce), 363
list_nodes_full() (in module salt.cloud.clouds.gogrid), 364
list_nodes_full() (in module salt.cloud.clouds.linode), 368
list_nodes_full() (in module salt.cloud.clouds.lxc), 369
list_nodes_full() (in module salt.cloud.clouds.msazure), 370
list_nodes_full() (in module salt.cloud.clouds.nova), 372
list_nodes_full() (in module salt.cloud.clouds.opennebula), 374
list_nodes_full() (in module salt.cloud.clouds.openstack), 377
list_nodes_full() (in module salt.cloud.clouds.parallels), 378
list_nodes_full() (in module salt.cloud.clouds.proxmox), 380

list_nodes_full() (in module salt.cloud.clouds.rackspace), 382
list_nodes_full() (in module salt.cloud.clouds.saltify), 383
list_nodes_full() (in module salt.cloud.clouds.softlayer), 384
list_nodes_full() (in module salt.cloud.clouds.softlayer_hw), 385
list_nodes_full() (in module salt.cloud.clouds.vsphere), 387
list_nodes_min() (in module salt.cloud.clouds.aliyun), 347
list_nodes_min() (in module salt.cloud.clouds.ec2), 357
list_nodes_min() (in module salt.cloud.clouds.vsphere), 387
list_nodes_select() (in module salt.cloud.clouds.aliyun), 347
list_nodes_select() (in module salt.cloud.clouds.cloudstack), 349
list_nodes_select() (in module salt.cloud.clouds.digital_ocean), 351
list_nodes_select() (in module salt.cloud.clouds.digital_ocean_v2), 352
list_nodes_select() (in module salt.cloud.clouds.ec2), 357
list_nodes_select() (in module salt.cloud.clouds.gce), 363
list_nodes_select() (in module salt.cloud.clouds.gogrid), 364
list_nodes_select() (in module salt.cloud.clouds.linode), 368
list_nodes_select() (in module salt.cloud.clouds.lxc), 369
list_nodes_select() (in module salt.cloud.clouds.msazure), 370
list_nodes_select() (in module salt.cloud.clouds.nova), 372
list_nodes_select() (in module salt.cloud.clouds.opennebula), 374
list_nodes_select() (in module salt.cloud.clouds.openstack), 377
list_nodes_select() (in module salt.cloud.clouds.parallels), 378
list_nodes_select() (in module salt.cloud.clouds.proxmox), 380
list_nodes_select() (in module salt.cloud.clouds.rackspace), 382
list_nodes_select() (in module salt.cloud.clouds.saltify), 383
list_nodes_select() (in module salt.cloud.clouds.softlayer), 384
list_nodes_select() (in module salt.cloud.clouds.softlayer_hw), 385
list_nodes_select() (in module salt.cloud.clouds.vsphere), 387
list_patterns() (in module salt.modules.zypper), 1036
list_peers() (in module salt.modules.glusterfs), 652
list_permissions() (in module salt.modules.rabbitmq), 862
list_pkgs() (in module salt.modules.aptpkg), 492
list_pkgs() (in module salt.modules.brew), 533
list_pkgs() (in module salt.modules.dpkg), 590
list_pkgs() (in module salt.modules.ebuild), 595
list_pkgs() (in module salt.modules.freebsdpkg), 633
list_pkgs() (in module salt.modules.macports), 727
list_pkgs() (in module salt.modules.openbsdpkg), 800
list_pkgs() (in module salt.modules.pacman), 808
list_pkgs() (in module salt.modules.pkgin), 824
list_pkgs() (in module salt.modules.pkgng), 830
list_pkgs() (in module salt.modules.pkgutil), 837
list_pkgs() (in module salt.modules.rest_package), 872
list_pkgs() (in module salt.modules.rpm), 878
list_pkgs() (in module salt.modules.solarisips), 911
list_pkgs() (in module salt.modules.solarispkg), 915
list_pkgs() (in module salt.modules.win_pkg), 997
list_pkgs() (in module salt.modules.yumpkg), 1022
list_pkgs() (in module salt.modules.zypper), 1036
list_plugins() (in module salt.modules.munin), 751
list_plugins() (in module salt.modules.nagios), 758
list_policies() (in module salt.modules.pagerduty), 811
list_policies() (in module salt.modules.rabbitmq), 863
list_policies() (in module salt.runners.pagerduty), 1172
list_ports() (in module salt.modules.neutron), 779
list_ports() (in module salt.modules.poudriere), 848
list_present() (in module salt.states.grains), 1324
list_products() (in module salt.modules.zypper), 1037
list_queues() (in module salt.modules.aws_sqs), 503
list_queues() (in module salt.modules.rabbitmq), 863
list_queues() (in module salt.runners.queue), 1174
list_queues_vhost() (in module salt.modules.rabbitmq), 863
list_quotas() (in module salt.modules.neutron), 780
list_renderers() (in module salt.modules.sysmod), 947
list_repo_pkgs() (in module salt.modules.yumpkg), 1022
list_repos() (in module salt.modules.aptpkg), 492
list_repos() (in module salt.modules.yumpkg), 1022
list_repos() (in module salt.modules.zypper), 1037
list_resourcepools() (in module salt.cloud.clouds.vsphere), 387
list_returner_functions() (in module salt.modules.sysmod), 948
list_returners() (in module salt.modules.sysmod), 948
list_role_policies() (in module salt.modules.boto_iam), 519
list_rooms() (in module salt.modules.slack_notify), 899
list_roots() (in module salt.wheel.file_roots), 1444
list_roots() (in module salt.wheel.pillar_roots), 1446
list_routers() (in module salt.modules.neutron), 780
list_runner_functions() (in module salt.modules.sysmod), 948
list_runners() (in module salt.modules.sysmod), 948
list_schedules() (in module salt.modules.pagerduty), 811
list_schedules() (in module salt.runners.pagerduty), 1172

list_sebool() (in module salt.modules.selinux), 894
list_secret_keys() (in module salt.modules.gpg), 658
list_security_group_rules() (in module salt.modules.neutron), 780
list_security_groups() (in module salt.modules.neutron), 780
list_securitygroup() (in module salt.cloud.clouds.aliyun), 347
list_servers() (in module salt.modules.haproxyconn), 666
list_services() (in module salt.modules.firewalld), 627
list_services() (in module salt.modules.pagerduty), 811
list_services() (in module salt.runners.pagerduty), 1172
list_sets() (in module salt.modules.ipset), 682
list_shared_folders_users() (in module salt.modules.vbox_guest), 976
list_sizes() (in module salt.modules.cloud), 546
list_sizes() (in module salt.runners.cloud), 1158
list_sizes() (salt.cloud.CloudClient method), 344
list_state_functions() (in module salt.modules.sysmod), 949
list_state_modules() (in module salt.modules.sysmod), 949
list_states() (in module salt.modules.cp), 560
list_storage_services() (in module salt.cloud.clouds.msazure), 370
list_subnets() (in module salt.modules.neutron), 780
list_tab() (in module salt.modules.cron), 561
list_tab() (in module salt.modules.incron), 675
list_updates() (in module salt.modules.win_update), 1008
list_upgrades() (in module salt.modules.aptpkg), 492
list_upgrades() (in module salt.modules.brew), 533
list_upgrades() (in module salt.modules.ebuild), 595
list_upgrades() (in module salt.modules.macports), 727
list_upgrades() (in module salt.modules.pacman), 808
list_upgrades() (in module salt.modules.pip), 820
list_upgrades() (in module salt.modules.pkgutil), 837
list_upgrades() (in module salt.modules.softwareupdate), 906
list_upgrades() (in module salt.modules.solarisips), 911
list_upgrades() (in module salt.modules.win_pkg), 997
list_upgrades() (in module salt.modules.yumpkg), 1023
list_upgrades() (in module salt.modules.zypper), 1037
list_user_permissions() (in module salt.modules.rabbitmq), 863
list_users() (in module salt.modules.drac), 591
list_users() (in module salt.modules.ilo), 673
list_users() (in module salt.modules.mac_user), 725
list_users() (in module salt.modules.pagerduty), 811
list_users() (in module salt.modules.pw_user), 857
list_users() (in module salt.modules.rabbitmq), 863
list_users() (in module salt.modules.slack_notify), 899
list_users() (in module salt.modules.useradd), 973
list_users() (in module salt.modules.win_useradd), 1010
list_users() (in module salt.runners.pagerduty), 1172
list_users_info() (in module salt.modules.ilo), 673
list_vhosts() (in module salt.modules.rabbitmq), 863
list_vlans() (in module salt.cloud.clouds.softlayer), 384
list_vlans() (in module salt.cloud.clouds.softlayer_hw), 385
list_vms() (in module salt.modules.smartos_vmadm), 901
list_vms() (in module salt.modules.virt), 979
list_vms() (in module salt.modules.xapi), 1011
list_volumes() (in module salt.modules.glusterfs), 652
list_vpnservices() (in module salt.modules.neutron), 781
list_webpi() (in module salt.modules.chocolatey), 544
list_windows() (in module salt.modules.pagerduty), 811
list_windows() (in module salt.runners.pagerduty), 1172
list_windowsfeatures() (in module salt.modules.chocolatey), 544
list_zones() (in module salt.modules.firewalld), 627
llen() (in module salt.modules.redismod), 870
load() (in module salt.modules.data), 566
load() (in module salt.modules.dockerio), 584
load() (in module salt.modules.freebsdkmod), 630
load() (in module salt.modules.kmod), 698
load_states() (in module salt.renderers.pyobjects), 1116
loadavg() (in module salt.modules.status), 932
loadavg() (in module salt.states.status), 1416
loaddata() (in module salt.modules.djangomod), 575
loaded() (in module salt.states.dockerio), 1288
LoaderError, 475, 476
local() (salt.netapi.NetapiClient method), 288
local_async() (salt.netapi.NetapiClient method), 288
local_batch() (salt.netapi.NetapiClient method), 289
LocalClient (class in salt.client), 337
locate() (in module salt.modules.locate), 706
lock() (in module salt.fileserver.gitfs), 467
lock() (in module salt.fileserver.hgfs), 468
lock() (in module salt.fileserver.svnfs), 472
lock() (in module salt.modules.nova), 795
lock() (in module salt.modules.osxdesktop), 806
lock() (in module salt.modules.zk_concurrency), 1030
lock() (in module salt.runners.fileserver), 1163
lock() (in module salt.states.zk_concurrency), 1435
lock_holders() (in module salt.modules.zk_concurrency), 1030
log_datefmt
  conf/logging, 457
  conf/master, 440
  conf/minion, 454
log_datefmt_logfile
  conf/logging, 457
  conf/master, 440
  conf/minion, 454
log_file
  conf/logging, 456
  conf/master, 439
  conf/minion, 453

log_fmt_console
conf/logging, 457
conf/master, 440
conf/minion, 454
log_fmt_logfile
conf/logging, 457
conf/master, 440
conf/minion, 454
log_granular_levels
conf/logging, 458
conf/master, 441
conf/minion, 454
log_level
conf/logging, 457
conf/master, 440
conf/minion, 453
log_level_logfile
conf/logging, 457
conf/master, 440
conf/minion, 453
Login (class in salt.netapi.rest_cherrypy.app), 1044
login() (in module salt.modules.dockerio), 584
Logout (class in salt.netapi.rest_cherrypy.app), 1046
logs() (in module salt.modules.dockerio), 584
lookup_jid() (in module salt.runners.jobs), 1165
loop_interval
conf/master, 417
Low State, **1678**
low() (in module salt.modules.state), 928
low() (salt.cloud.CloudClient method), 344
LowDataAdapter (class in salt.netapi.rest_cherrypy.app), 1044
lowstate, **1059**
lrange() (in module salt.modules.redismod), 870
ls() (in module salt.modules.augeas_cfg), 501
ls() (in module salt.modules.cron), 561
ls() (in module salt.modules.grains), 662
ls() (in module salt.modules.incron), 675
ls() (in module salt.modules.serverdensity_device), 895
ls() (in module salt.modules.tomcat), 966
ls_() (in module salt.modules.etcd_mod), 602
ls_() (in module salt.modules.lxc), 718
ls_() (in module salt.modules.scsi), 892
ls_remote() (in module salt.modules.git), 647
lsmod() (in module salt.modules.freebsdkmod), 631
lsmod() (in module salt.modules.kmod), 699
lstat() (in module salt.modules.file), 616
lucene_version() (in module salt.modules.solr), 919
lv_absent() (in module salt.states.lvm), 1348
lv_present() (in module salt.states.lvm), 1348
lvcreate() (in module salt.modules.linux_lvm), 703
lvdisplay() (in module salt.modules.linux_lvm), 703
lvremove() (in module salt.modules.linux_lvm), 703
lvresize() (in module salt.modules.linux_lvm), 703

M

make_image() (in module salt.modules.qemu_img), 859
make_pkgng_aware() (in module salt.modules.poudriere), 848
makedirs_() (in module salt.modules.file), 617
makedirs_perms() (in module salt.modules.file), 617
makeopts_contains() (in module salt.modules.makeconf), 732
manage_file() (in module salt.modules.file), 617
manage_mode() (in module salt.modules.config), 556
managed() (in module salt.states.file), 1304
managed() (in module salt.states.memcached), 1355
managed() (in module salt.states.network), 1368
managed() (in module salt.states.ntp), 1372
managed() (in module salt.states.pkgrepo), 1385
managed() (in module salt.states.sysrc), 1419
managed() (in module salt.states.virtualenv_mod), 1426
managed() (in module salt.states.win_network), 1428
managedcloud() (in module salt.cloud.clouds.nova), 373
managedcloud() (in module salt.cloud.clouds.openstack), 377
Map (class in salt.utils.aggregation), 474
map_branch() (salt.pillar.git_pillar.GitPillar method), 1083
map_run() (in module salt.runners.cloud), 1158
map_run() (salt.cloud.CloudClient method), 344
mask() (in module salt.modules.systemd), 954
Master, **1678**
master
conf/minion, 443
Master Tops, **1678**
master() (in module salt.modules.status), 932
master() (in module salt.modules.win_status), 1002
master_id
conf/master, 414
master_job_cache
conf/master, 418
master_port
conf/minion, 444
master_pubkey_signature
conf/master, 422
master_sign_key_name
conf/master, 422
conf/minion, 452
master_sign_pubkey
conf/master, 422
master_tops
conf/master, 423
master_type
conf/minion, 443
master_use_pubkey_signature
conf/master, 422
MasterExit, 475, 476
Masterless, **1678**

match() (in module salt.modules.augeas_cfg), 501
match_index_versions() (in module salt.modules.solr), 919
max_minions
    conf/master, 418
max_open_files
    conf/master, 414
maybe_fix_ssl_version() (in module salt.modules.tls), 964
md5_digest() (in module salt.modules.hashutil), 667
member_status() (in module salt.modules.riak), 877
members() (in module salt.modules.groupadd), 664
members() (in module salt.modules.win_groupadd), 991
meminfo() (in module salt.modules.status), 932
memory() (in module salt.modules.sysbench), 941
merge() (in module salt.modules.config), 556
merge() (in module salt.modules.git), 647
merger (class in salt.pillar.mysql), 1088
migrate() (in module salt.modules.virt), 979
migrate() (in module salt.modules.xapi), 1011
migrate() (in module salt.runners.virt), 1181
migrate_non_shared() (in module salt.modules.virt), 979
migrate_non_shared_inc() (in module salt.modules.virt), 979
min_query() (salt.cloud.CloudClient method), 344
Mine, **143**, **1678**
mine() (in module salt.runners.cache), 1157
Minion, **1678**
Minion ID, **1678**
Minion proc System, 153
minion_config() (in module salt.config), 335
minion_data_cache
    conf/master, 417
minion_mods() (in module salt.loader), 336
MinionError, 475, 476
minionfs_blacklist
    conf/master, 435
minionfs_env
    conf/master, 434
minionfs_mountpoint
    conf/master, 434
minionfs_whitelist
    conf/master, 434
Minions (class in salt.netapi.rest_cherrypy.app), 1046
MinionSaltAPIHandler (in module salt.netapi.rest_tornado.saltnado), 1064
missing() (in module salt.modules.daemontools), 564
missing() (in module salt.modules.debian_service), 571
missing() (in module salt.modules.freebsdservice), 637
missing() (in module salt.modules.gentoo_service), 642
missing() (in module salt.modules.launchctl), 699
missing() (in module salt.modules.netbsdservice), 761
missing() (in module salt.modules.openbsdrcctl), 802
missing() (in module salt.modules.openbsdservice), 803
missing() (in module salt.modules.rh_service), 876
missing() (in module salt.modules.runit), 880
missing() (in module salt.modules.service), 896
missing() (in module salt.modules.smf), 903
missing() (in module salt.modules.systemd), 954
missing() (in module salt.modules.upstart), 970
missing() (in module salt.modules.win_service), 1001
missing() (in module salt.states.file), 1307
missing_func() (in module salt.modules.test), 956
mkconfig() (in module salt.modules.seed), 893
mkdir() (in module salt.modules.file), 618
mkfs() (in module salt.modules.btrfs), 538
mkfs() (in module salt.modules.extfs), 604
mkfs() (in module salt.modules.parted), 813
mkfs() (in module salt.modules.xfs), 1015
mklabel() (in module salt.modules.parted), 813
mknod() (in module salt.modules.file), 618
mknod() (in module salt.states.file), 1307
mknod_blkdev() (in module salt.modules.file), 618
mknod_chrdev() (in module salt.modules.file), 618
mknod_fifo() (in module salt.modules.file), 618
mkpart() (in module salt.modules.parted), 813
mkpartfs() (in module salt.modules.parted), 813
mmodule() (in module salt.modules.saltutil), 887
mod_aggregate() (in module salt.states.iptables), 1338
mod_aggregate() (in module salt.states.pkg), 1383
mod_bufsize() (in module salt.modules.network), 768
mod_hostname() (in module salt.modules.network), 768
mod_list() (in module salt.modules.freebsdkmod), 631
mod_list() (in module salt.modules.kmod), 699
mod_repo() (in module salt.modules.aptpkg), 492
mod_repo() (in module salt.modules.yumpkg), 1023
mod_repo() (in module salt.modules.zypper), 1037
mod_run_check() (in module salt.states.cmd), 1274
mod_run_check() (in module salt.states.git), 1316
mod_run_check_cmd() (in module salt.states.file), 1308
mod_watch() (in module salt.states.cmd), 1274
mod_watch() (in module salt.states.dockerio), 1288
mod_watch() (in module salt.states.module), 1358
mod_watch() (in module salt.states.mount), 1360
mod_watch() (in module salt.states.service), 1411
mod_watch() (in module salt.states.supervisord), 1416
mod_watch() (in module salt.states.test), 1420
mod_watch() (in module salt.states.tomcat), 1422
mode() (in module salt.states.quota), 1396
mode() (in module salt.states.selinux), 1409
modfacl() (in module salt.modules.linux_acl), 702
modified() (in module salt.modules.rpm), 879
modified() (in module salt.modules.yumpkg), 1023
modified() (in module salt.modules.zypper), 1037
modify() (in module salt.modules.schedule), 891
modify() (in module salt.modules.sqlite3), 924
modify() (in module salt.modules.xfs), 1016
module_dirs
    conf/minion, 416, 448

module_report() (in module salt.modules.test), 956
modules() (in module salt.modules.apache), 488
modules() (in module salt.modules.syslog_ng), 945
monitor() (in module salt.modules.monit), 748
monitored() (in module salt.states.serverdensity_device), 1409
mount() (in module salt.modules.guestfs), 665
mount() (in module salt.modules.mount), 750
mount() (in module salt.modules.qemu_nbd), 860
mount_image() (in module salt.modules.img), 674
mounted() (in module salt.states.mount), 1360
mounts() (in module salt.modules.moosefs), 749
move() (in module salt.modules.file), 618
move() (in module salt.modules.schedule), 891
msg (salt.auth.pam.PamMessage attribute), 308
msg_style (salt.auth.pam.PamMessage attribute), 308
Multi-Master, **1678**
multipath_flush() (in module salt.modules.devmap), 573
multipath_list() (in module salt.modules.devmap), 573
multiprocessing
    conf/minion, 453
mutex() (in module salt.modules.sysbench), 941
MX() (in module salt.modules.dig), 573
MX() (in module salt.modules.dnsutil), 576

N

name() (in module salt.modules.parted), 813
NamedStatement (class in salt.modules.syslog_ng), 942
namenode_format() (in module salt.modules.hadoop), 665
nameservers() (in module salt.modules.drac), 591
NestDisplay (class in salt.output.nested), 1070
NestDisplay (class in salt.output.no_return), 1071
NetapiClient (class in salt.netapi), 288
netdev() (in module salt.modules.status), 933
netstat() (in module salt.modules.network), 768
netstat() (in module salt.modules.win_network), 994
netstats() (in module salt.modules.cassandra), 540
netstats() (in module salt.modules.status), 933
network() (in module salt.modules.ilo), 673
network() (in module salt.states.drac), 1292
network_acl_exists() (in module salt.modules.boto_vpc), 531
network_create() (in module salt.cloud.clouds.nova), 373
network_create() (in module salt.modules.cloud), 546
network_info() (in module salt.modules.drac), 591
network_io_counters() (in module salt.modules.ps), 851
network_list() (in module salt.cloud.clouds.nova), 373
network_list() (in module salt.modules.cloud), 546
networks() (in module salt.cloud.clouds.openstack), 377
new_chain() (in module salt.modules.iptables), 687
new_chain() (in module salt.modules.nftables), 791
new_service() (in module salt.modules.firewalld), 627
new_set() (in module salt.modules.ipset), 683
new_table() (in module salt.modules.nftables), 791
new_zone() (in module salt.modules.firewalld), 628
next_hyper() (in module salt.runners.virt), 1181
Node Group, **1678**
node_info() (in module salt.modules.virt), 979
node_info() (in module salt.modules.xapi), 1012
nodegroups
    conf/master, 441
noop() (in module salt.modules.puppet), 854
normalize_name() (in module salt.modules.solarisips), 911
normalize_name() (in module salt.modules.yumpkg), 1024
normpath() (in module salt.modules.file), 618
noscan() (in module salt.modules.bluez), 505
not_loaded() (in module salt.modules.test), 957
not_present() (in module salt.runners.manage), 1169
nproc() (in module salt.modules.status), 933
NS() (in module salt.modules.dig), 573
NS() (in module salt.modules.dnsutil), 577
nslookup() (in module salt.modules.win_network), 995
num_cpus() (in module salt.modules.ps), 851
num_fields (salt.pillar.mysql.merger attribute), 1088

O

off() (in module salt.modules.quota), 860
offline() (in module salt.modules.zpool), 1033
on() (in module salt.modules.quota), 860
online() (in module salt.modules.zpool), 1033
open_files() (in module salt.modules.file), 619
open_mode
    conf/master, 420
    conf/minion, 452
optimize() (in module salt.modules.solr), 920
optimize_providers() (in module salt.cloud.clouds.ec2), 357
Option (class in salt.modules.syslog_ng), 943
option() (in module salt.modules.config), 556
options() (in module salt.modules.supervisord), 934
options_absent() (in module salt.states.ini_manage), 1331
options_present() (in module salt.states.ini_manage), 1331
opts_pkg() (in module salt.modules.test), 957
orchestrate() (in module salt.runners.state), 1177
orchestrate_high() (in module salt.runners.state), 1178
orchestrate_single() (in module salt.runners.state), 1178
order_masters
    conf/master, 438
output
    conf/master, 417
output() (in module salt.output.compact), 1067
output() (in module salt.output.highstate), 1068
output() (in module salt.output.json_out), 1069
output() (in module salt.output.key), 1069

output() (in module salt.output.nested), 1070
output() (in module salt.output.newline_values_only), 1071
output() (in module salt.output.no_out), 1071
output() (in module salt.output.no_return), 1071
output() (in module salt.output.overstatestage), 1072
output() (in module salt.output.pprint_out), 1072
output() (in module salt.output.progress), 1072
output() (in module salt.output.raw), 1072
output() (in module salt.output.txt), 1073
output() (in module salt.output.virt_query), 1073
output() (in module salt.output.yaml_out), 1073
Outputter, **1678**
outputter() (in module salt.modules.test), 957
over() (in module salt.runners.state), 1178
Overstate, **1678**
overview() (in module salt.modules.drbd), 593
owner() (in module salt.modules.aptpkg), 493
owner() (in module salt.modules.pacman), 808
owner() (in module salt.modules.rpm), 879
owner() (in module salt.modules.yumpkg), 1024
owner() (in module salt.modules.zypper), 1038
owner_to() (in module salt.modules.postgres), 844

P

pack() (in module salt.modules.genesis), 640
pack() (salt.exceptions.SaltException method), 475, 477
pack_sources() (in module salt.modules.pkg_resource), 822
pair() (in module salt.modules.bluez), 505
PamConv (class in salt.auth.pam), 308
PamHandle (class in salt.auth.pam), 308
PamMessage (class in salt.auth.pam), 308
PamResponse (class in salt.auth.pam), 308
param_set() (in module salt.modules.varnish), 974
param_show() (in module salt.modules.varnish), 974
Parameter (class in salt.modules.syslog_ng), 943
ParameterValue (class in salt.modules.syslog_ng), 943
pardir() (in module salt.modules.file), 619
parse_config() (in module salt.modules.pkgng), 830
parse_config() (in module salt.modules.poudriere), 848
parse_hosts() (in module salt.modules.dnsutil), 577
parse_targets() (in module salt.modules.pkg_resource), 822
parse_zone() (in module salt.modules.dnsutil), 577
part_list() (in module salt.modules.parted), 814
passwd() (in module salt.modules.tomcat), 966
patch() (in module salt.modules.file), 619
patch() (in module salt.states.file), 1308
path_exists_glob() (in module salt.modules.file), 619
pause() (in module salt.modules.virt), 979
pause() (in module salt.modules.xapi), 1012
pause() (in module salt.runners.virt), 1181
pcre() (in module salt.modules.match), 737
peer
    conf/master, 439
Peer Communication, **1678**
peer() (in module salt.modules.glusterfs), 652
peer_run
    conf/master, 439
peered() (in module salt.states.glusterfs), 1318
percent() (in module salt.modules.disk), 574
persist() (in module salt.modules.darwin_sysctl), 565
persist() (in module salt.modules.freebsd_sysctl), 629
persist() (in module salt.modules.linux_sysctl), 705
persist() (in module salt.modules.netbsd_sysctl), 760
persist() (in module salt.modules.openbsd_sysctl), 799
pgrep() (in module salt.modules.ps), 851
pid() (in module salt.modules.status), 933
pidfile
    conf/master, 415
    conf/minion, 444
Pillar, **1678**
pillar() (in module salt.modules.match), 737
pillar() (in module salt.runners.cache), 1157
pillar_dir() (salt.pillar.svn_pillar.SvnPillar method), 1097
pillar_pcre() (in module salt.modules.match), 738
pillar_refresh() (in module salt.modules.saltutil), 887
pillar_roots
    conf/master, 435
    conf/minion, 452
pillar_source_merging_strategy
    conf/master, 436
ping() (in module salt.modules.gnomedesktop), 654
ping() (in module salt.modules.junos), 692
ping() (in module salt.modules.network), 768
ping() (in module salt.modules.redismod), 870
ping() (in module salt.modules.rest_sample), 873
ping() (in module salt.modules.solr), 920
ping() (in module salt.modules.sysbench), 941
ping() (in module salt.modules.test), 957
ping() (in module salt.modules.test_virtual), 959
ping() (in module salt.modules.win_network), 995
pkg() (in module salt.modules.state), 928
PkgParseError, 475, 476
pki_dir
    conf/master, 416
    conf/minion, 444
pkill() (in module salt.modules.ps), 851
plugin_is_enabled() (in module salt.modules.rabbitmq), 863
policy_exists() (in module salt.modules.rabbitmq), 863
pop() (in module salt.runners.queue), 1174
port() (in module salt.modules.dockerio), 584
porttree_matches() (in module salt.modules.ebuild), 596
POST (salt.netapi.rest_cherrypy.app.LowDataAdapter attribute), 1044

POST() (salt.netapi.rest_cherrypy.app.Keys method), 1055
POST() (salt.netapi.rest_cherrypy.app.Login method), 1045
POST() (salt.netapi.rest_cherrypy.app.Logout method), 1046
POST() (salt.netapi.rest_cherrypy.app.Minions method), 1047
POST() (salt.netapi.rest_cherrypy.app.Run method), 1049
POST() (salt.netapi.rest_cherrypy.app.Webhook method), 1052
post_message() (in module salt.modules.slack_notify), 899
post_message() (in module salt.states.slack), 1411
power() (in module salt.modules.bluez), 505
poweroff() (in module salt.modules.system), 952
poweroff() (in module salt.modules.win_system), 1004
poweroff() (in module salt.runners.drac), 1159
poweron() (in module salt.runners.drac), 1159
preferred_ip() (in module salt.cloud.clouds.nova), 373
preferred_ip() (in module salt.cloud.clouds.openstack), 377
preferred_ip() (in module salt.cloud.clouds.rackspace), 382
prep_bootstrap() (in module salt.modules.seed), 893
prep_jid() (in module salt.returners.carbon_return), 1129
prep_jid() (in module salt.returners.cassandra_return), 1129
prep_jid() (in module salt.returners.couchbase_return), 1130
prep_jid() (in module salt.returners.couchdb_return), 1131
prep_jid() (in module salt.returners.django_return), 1132
prep_jid() (in module salt.returners.elasticsearch_return), 1133
prep_jid() (in module salt.returners.etcd_return), 1133
prep_jid() (in module salt.returners.local_cache), 1135
prep_jid() (in module salt.returners.memcache_return), 1136
prep_jid() (in module salt.returners.mongo_future_return), 1137
prep_jid() (in module salt.returners.mongo_return), 1137
prep_jid() (in module salt.returners.multi_returner), 1138
prep_jid() (in module salt.returners.mysql), 1140
prep_jid() (in module salt.returners.odbc), 1143
prep_jid() (in module salt.returners.postgres), 1144
prep_jid() (in module salt.returners.postgres_local_cache), 1146
prep_jid() (in module salt.returners.redis_return), 1147
prep_jid() (in module salt.returners.sentry_return), 1147
prep_jid() (in module salt.returners.smtp_return), 1149
prep_jid() (in module salt.returners.sqlite3_return), 1151
prep_jid() (in module salt.returners.syslog_return), 1151
prepend() (in module salt.modules.file), 619
prepend() (in module salt.states.file), 1308
presence_events
    conf/master, 419
present() (in module salt.runners.manage), 1169
present() (in module salt.states.alias), 1235
present() (in module salt.states.at), 1240
present() (in module salt.states.boto_asg), 1247
present() (in module salt.states.boto_cloudwatch_alarm), 1249
present() (in module salt.states.boto_dynamodb), 1251
present() (in module salt.states.boto_elasticache), 1253
present() (in module salt.states.boto_elb), 1257
present() (in module salt.states.boto_iam_role), 1259
present() (in module salt.states.boto_lc), 1261
present() (in module salt.states.boto_route53), 1264
present() (in module salt.states.boto_secgroup), 1265
present() (in module salt.states.boto_sns), 1267
present() (in module salt.states.boto_sqs), 1268
present() (in module salt.states.cloud), 1270
present() (in module salt.states.cron), 1282
present() (in module salt.states.ddns), 1284
present() (in module salt.states.dockerio), 1288
present() (in module salt.states.drac), 1292
present() (in module salt.states.git), 1316
present() (in module salt.states.grains), 1324
present() (in module salt.states.group), 1325
present() (in module salt.states.host), 1328
present() (in module salt.states.incron), 1329
present() (in module salt.states.influxdb_database), 1330
present() (in module salt.states.influxdb_user), 1330
present() (in module salt.states.ipset), 1333
present() (in module salt.states.kmod), 1346
present() (in module salt.states.layman), 1347
present() (in module salt.states.linux_acl), 1347
present() (in module salt.states.locale), 1348
present() (in module salt.states.lvs_server), 1349
present() (in module salt.states.lvs_service), 1350
present() (in module salt.states.lxc), 1351
present() (in module salt.states.makeconf), 1353
present() (in module salt.states.mdadm), 1354
present() (in module salt.states.mongodb_user), 1359
present() (in module salt.states.mysql_database), 1361
present() (in module salt.states.mysql_grants), 1362
present() (in module salt.states.mysql_user), 1364
present() (in module salt.states.openstack_config), 1373
present() (in module salt.states.postgres_database), 1388
present() (in module salt.states.postgres_extension), 1389
present() (in module salt.states.postgres_group), 1390
present() (in module salt.states.postgres_schema), 1391
present() (in module salt.states.postgres_user), 1392
present() (in module salt.states.pyrax_queues), 1395
present() (in module salt.states.rabbitmq_policy), 1397
present() (in module salt.states.rabbitmq_user), 1398
present() (in module salt.states.rabbitmq_vhost), 1399

present() (in module salt.states.reg), 1401
present() (in module salt.states.schedule), 1408
present() (in module salt.states.splunk_search), 1413
present() (in module salt.states.ssh_auth), 1414
present() (in module salt.states.ssh_known_hosts), 1415
present() (in module salt.states.sysctl), 1418
present() (in module salt.states.user), 1424
primary_suffix() (in module salt.states.win_dns_client), 1427
print_job() (in module salt.runners.jobs), 1166
probe() (in module salt.modules.parted), 814
proc_info() (in module salt.modules.ps), 852
process() (in module salt.states.status), 1416
process_fields() (salt.pillar.mysql.merger method), 1088
process_queue() (in module salt.runners.queue), 1174
process_results() (salt.pillar.mysql.merger method), 1088
processlist() (in module salt.modules.mysql), 754
procs() (in module salt.modules.status), 933
procs() (in module salt.modules.win_status), 1002
profile() (in module salt.runners.cloud), 1158
profile() (in module salt.states.cloud), 1270
profile() (salt.cloud.CloudClient method), 344
profile_() (in module salt.modules.cloud), 546
profile_associated() (in module salt.modules.boto_iam), 519
progress_iter() (in module salt.output.progress), 1072
properties() (in module salt.modules.btrfs), 538
provider() (in module salt.modules.test), 957
providers
    conf/minion, 449
providers() (in module salt.modules.test), 957
Proxy Minion, **1678**
prune_dump() (in module salt.modules.xfs), 1016
psed() (in module salt.modules.file), 620
psql_query() (in module salt.modules.postgres), 844
publish() (in module salt.modules.publish), 853
publish() (in module salt.modules.raet_publish), 865
publish_port
    conf/master, 414
PublishError, 475, 477
pull() (in module salt.modules.dockerio), 585
pull() (in module salt.modules.git), 647
pull() (in module salt.modules.hg), 669
pulled() (in module salt.states.dockerio), 1289
purge() (in module salt.modules.aptpkg), 493
purge() (in module salt.modules.ebuild), 596
purge() (in module salt.modules.openbsdpkg), 800
purge() (in module salt.modules.pacman), 809
purge() (in module salt.modules.pkgin), 824
purge() (in module salt.modules.pkgutil), 837
purge() (in module salt.modules.schedule), 891
purge() (in module salt.modules.solarisips), 912
purge() (in module salt.modules.solarispkg), 915
purge() (in module salt.modules.varnish), 974
purge() (in module salt.modules.virt), 979
purge() (in module salt.modules.win_pkg), 997
purge() (in module salt.modules.yumpkg), 1024
purge() (in module salt.modules.zypper), 1038
purge() (in module salt.runners.lxc), 1167
purge() (in module salt.runners.virt), 1181
purged() (in module salt.states.pkg), 1383
push() (in module salt.modules.cp), 560
push() (in module salt.modules.dockerio), 585
push() (in module salt.modules.git), 647
push_dir() (in module salt.modules.cp), 560
pushed() (in module salt.states.dockerio), 1289
put() (in module salt.modules.s3), 886
put() (in module salt.modules.swift), 940
pv_absent() (in module salt.states.lvm), 1349
pv_present() (in module salt.states.lvm), 1349
pvcreate() (in module salt.modules.linux_lvm), 703
pvdisplay() (in module salt.modules.linux_lvm), 703
pvremove() (in module salt.modules.linux_lvm), 703
pxe() (in module salt.runners.drac), 1159
PyDSL, **1678**
PyWinUpdater (class in salt.modules.win_update), 1006
PyWinUpdater (class in salt.states.win_update), 1431

Q

query() (in module salt.cloud.clouds.aliyun), 347
query() (in module salt.cloud.clouds.digital_ocean), 351
query() (in module salt.cloud.clouds.digital_ocean_v2), 352
query() (in module salt.cloud.clouds.ec2), 357
query() (in module salt.cloud.clouds.parallels), 378
query() (in module salt.cloud.clouds.proxmox), 380
query() (in module salt.modules.cloud), 546
query() (in module salt.modules.http), 671
query() (in module salt.modules.influx), 677
query() (in module salt.modules.mysql), 755
query() (in module salt.runners.cloud), 1158
query() (in module salt.runners.http), 1164
query() (in module salt.runners.search), 1175
query() (in module salt.runners.virt), 1181
query() (in module salt.states.http), 1328
query() (salt.cloud.CloudClient method), 344
query_instance() (in module salt.cloud.clouds.ec2), 357
queue_exists() (in module salt.modules.aws_sqs), 503
queue_instances() (in module salt.cloud.clouds.ec2), 357
queues_create() (in module salt.cloud.clouds.pyrax), 381
queues_delete() (in module salt.cloud.clouds.pyrax), 381
queues_exists() (in module salt.cloud.clouds.pyrax), 381
queues_show() (in module salt.cloud.clouds.pyrax), 381
quote_identifier() (in module salt.modules.mysql), 755

R

rackconnect() (in module salt.cloud.clouds.nova), 373

rackconnect() (in module salt.cloud.clouds.openstack), 377
rand_sleep() (in module salt.modules.test), 957
rand_str() (in module salt.modules.test), 958
random_reauth_delay
    conf/minion, 446
range_server
    conf/master, 441
rar() (in module salt.modules.archive), 497
raw() (in module salt.modules.pillar), 817
raw_arg() (in module salt.runners.test), 1180
raw_cron() (in module salt.modules.cron), 561
raw_incron() (in module salt.modules.incron), 675
raw_interface_configs() (in module salt.modules.win_ip), 992
raw_mod() (in module salt.loader), 336
raw_system_incron() (in module salt.modules.incron), 675
Reactor, **136**, *see* Event, **1678**
read() (in module salt.wheel.file_roots), 1445
read() (in module salt.wheel.pillar_roots), 1446
read_conf() (in module salt.modules.lxc), 718
read_datasource() (in module salt.modules.jboss7), 689
read_file() (in module salt.modules.pam), 811
read_key() (in module salt.modules.reg), 872
read_simple_binding() (in module salt.modules.jboss7), 689
readdir() (in module salt.modules.file), 621
readlink() (in module salt.modules.file), 621
readlink() (in module salt.modules.win_file), 989
rebase() (in module salt.modules.git), 648
reboot() (in module salt.cloud.clouds.aliyun), 347
reboot() (in module salt.cloud.clouds.ec2), 357
reboot() (in module salt.cloud.clouds.gce), 363
reboot() (in module salt.cloud.clouds.nova), 373
reboot() (in module salt.cloud.clouds.openstack), 377
reboot() (in module salt.modules.smartos_vmadm), 901
reboot() (in module salt.modules.system), 952
reboot() (in module salt.modules.virt), 980
reboot() (in module salt.modules.win_system), 1004
reboot() (in module salt.modules.xapi), 1012
reboot() (in module salt.runners.drac), 1159
receive_keys() (in module salt.modules.gpg), 658
receive_message() (in module salt.modules.aws_sqs), 503
recon_default
    conf/minion, 446
recon_max
    conf/minion, 447
recon_randomize
    conf/minion, 447
recover_all() (in module salt.modules.modjk), 745
recurse() (in module salt.states.file), 1309
recv() (in module salt.modules.cp), 560
recv_known_host() (in module salt.modules.ssh), 925
refresh_db() (in module salt.modules.aptpkg), 493
refresh_db() (in module salt.modules.brew), 534
refresh_db() (in module salt.modules.ebuild), 596
refresh_db() (in module salt.modules.freebsdpkg), 633
refresh_db() (in module salt.modules.macports), 728
refresh_db() (in module salt.modules.pacman), 809
refresh_db() (in module salt.modules.pkgin), 824
refresh_db() (in module salt.modules.pkgng), 830
refresh_db() (in module salt.modules.pkgutil), 838
refresh_db() (in module salt.modules.solarisips), 912
refresh_db() (in module salt.modules.win_pkg), 997
refresh_db() (in module salt.modules.yumpkg), 1024
refresh_db() (in module salt.modules.zypper), 1038
refresh_modules() (in module salt.modules.saltutil), 887
refresh_pillar() (in module salt.modules.saltutil), 887
regen_keys() (in module salt.modules.saltutil), 888
register_instances() (in module salt.modules.boto_elb), 517
Registry (class in salt.modules.reg), 872
rehash() (in module salt.modules.pyenv), 859
rehash() (in module salt.modules.rbenv), 867
rehash() (in module salt.modules.win_path), 996
rehashconf() (in module salt.modules.znc), 1031
reinstall_ruby() (in module salt.modules.rvm), 883
reject() (in module salt.wheel.key), 1445
reject_dict() (in module salt.wheel.key), 1445
reload() (in module salt.modules.jboss7), 689
reload_() (in module salt.modules.daemontools), 564
reload_() (in module salt.modules.debian_service), 572
reload_() (in module salt.modules.freebsdservice), 637
reload_() (in module salt.modules.netbsdservice), 761
reload_() (in module salt.modules.openbsdrcctl), 802
reload_() (in module salt.modules.openbsdservice), 804
reload_() (in module salt.modules.rh_service), 876
reload_() (in module salt.modules.runit), 880
reload_() (in module salt.modules.schedule), 892
reload_() (in module salt.modules.service), 896
reload_() (in module salt.modules.smf), 903
reload_() (in module salt.modules.syslog_ng), 945
reload_() (in module salt.modules.systemd), 954
reload_() (in module salt.modules.tomcat), 966
reload_() (in module salt.modules.upstart), 970
reload_core() (in module salt.modules.solr), 920
reload_import_config() (in module salt.modules.solr), 920
reload_modules() (in module salt.modules.sysmod), 949
reloaded() (in module salt.states.jboss7), 1343
reloaded() (in module salt.states.syslog_ng), 1419
remote_get() (in module salt.modules.git), 648
remote_set() (in module salt.modules.git), 648
remotes() (in module salt.modules.git), 649
remount() (in module salt.modules.mount), 750
remove() (in module salt.modules.alternatives), 487
remove() (in module salt.modules.aptpkg), 494

remove() (in module salt.modules.augeas_cfg), 502
remove() (in module salt.modules.brew), 534
remove() (in module salt.modules.cpan), 561
remove() (in module salt.modules.ebuild), 596
remove() (in module salt.modules.file), 621
remove() (in module salt.modules.freebsdkmod), 631
remove() (in module salt.modules.freebsdpkg), 633
remove() (in module salt.modules.grains), 663
remove() (in module salt.modules.kmod), 699
remove() (in module salt.modules.logadm), 707
remove() (in module salt.modules.macports), 728
remove() (in module salt.modules.mongodb), 747
remove() (in module salt.modules.openbsdpkg), 800
remove() (in module salt.modules.pacman), 809
remove() (in module salt.modules.pkgin), 825
remove() (in module salt.modules.pkgng), 831
remove() (in module salt.modules.pkgutil), 838
remove() (in module salt.modules.rest_package), 872
remove() (in module salt.modules.solarisips), 912
remove() (in module salt.modules.solarispkg), 915
remove() (in module salt.modules.supervisord), 934
remove() (in module salt.modules.svn), 938
remove() (in module salt.modules.sysrc), 951
remove() (in module salt.modules.win_path), 996
remove() (in module salt.modules.win_pkg), 998
remove() (in module salt.modules.win_servermanager), 999
remove() (in module salt.modules.yumpkg), 1025
remove() (in module salt.modules.zypper), 1038
remove() (in module salt.states.alternatives), 1236
remove_complex_types() (in module salt.cloud.clouds.linode), 368
remove_container() (in module salt.modules.dockerio), 585
remove_datasource() (in module salt.modules.jboss7), 689
remove_gateway_router() (in module salt.modules.neutron), 781
remove_image() (in module salt.modules.dockerio), 585
remove_interface_router() (in module salt.modules.neutron), 781
remove_license() (in module salt.modules.powerpath), 849
remove_lock() (in module salt.modules.zypper), 1039
remove_option() (in module salt.modules.ini_manage), 680
remove_section() (in module salt.modules.ini_manage), 680
remove_service() (in module salt.modules.firewalld), 628
remove_var() (in module salt.modules.makeconf), 732
removed() (in module salt.states.cyg), 1283
removed() (in module salt.states.gem), 1314
removed() (in module salt.states.npm), 1372
removed() (in module salt.states.pecl), 1374
removed() (in module salt.states.pip_state), 1377
removed() (in module salt.states.pkg), 1383
removed() (in module salt.states.win_servermanager), 1429
removed() (salt.states.cyg.DictDiffer method), 1282
removegroup() (in module salt.modules.win_useradd), 1010
rename() (in module salt.cloud.clouds.ec2), 357
rename() (in module salt.cloud.clouds.libcloud_aws), 366
rename() (in module salt.modules.file), 621
rename() (in module salt.modules.mac_user), 726
rename() (in module salt.modules.pw_user), 858
rename() (in module salt.modules.solaris_user), 910
rename() (in module salt.modules.useradd), 973
rename() (in module salt.modules.win_useradd), 1010
rename() (in module salt.modules.zfs), 1029
rename() (in module salt.states.file), 1310
rename_set() (in module salt.modules.ipset), 683
Render Pipe, **1678**
render() (in module salt.renderers.cheetah), 1100
render() (in module salt.renderers.genshi), 1100
render() (in module salt.renderers.gpg), 1101
render() (in module salt.renderers.jinja), 1105
render() (in module salt.renderers.json), 1106
render() (in module salt.renderers.mako), 1107
render() (in module salt.renderers.msgpack), 1107
render() (in module salt.renderers.py), 1108
render() (in module salt.renderers.pydsl), 1113
render() (in module salt.renderers.pyobjects), 1116
render() (in module salt.renderers.wempy), 1120
render() (in module salt.renderers.yaml), 1121
render() (in module salt.renderers.yamlex), 1122
render_dirs
  conf/minion, 449
Renderer, **1678**
renderer
  conf/master, 423
  conf/minion, 450
renderer_doc() (in module salt.modules.sysmod), 949
replace() (in module salt.modules.file), 621
replace() (in module salt.modules.memcached), 740
replace() (in module salt.modules.zpool), 1033
replace() (in module salt.states.file), 1310
replace_network_acl_entry() (in module salt.modules.boto_vpc), 531
replace_route() (in module salt.modules.boto_vpc), 532
replace_route_table_association() (in module salt.modules.boto_vpc), 532
replication_details() (in module salt.modules.solr), 921
report() (in module salt.modules.quota), 860
request() (in module salt.modules.state), 929
request_instance() (in module salt.cloud.clouds.ec2), 358
request_instance() (in module salt.cloud.clouds.nova), 373

request_instance() (in module salt.cloud.clouds.openstack), 377
requeue() (in module salt.modules.postfix), 840
reread() (in module salt.modules.supervisord), 935
rescan_all() (in module salt.modules.scsi), 892
rescue() (in module salt.modules.parted), 814
reset() (in module salt.modules.git), 649
reset() (in module salt.modules.rabbitmq), 864
reset() (in module salt.modules.virt), 980
reset() (in module salt.modules.xapi), 1012
reset() (in module salt.runners.virt), 1181
reset_ignored() (in module salt.modules.softwareupdate), 906
reset_stats() (in module salt.modules.modjk), 745
resize() (in module salt.modules.btrfs), 539
resize() (in module salt.modules.parted), 814
resize2fs() (in module salt.modules.blockdev), 504
resp (salt.auth.pam.PamResponse attribute), 308
resp_retcode (salt.auth.pam.PamResponse attribute), 308
restart() (in module salt.modules.daemontools), 564
restart() (in module salt.modules.debian_service), 572
restart() (in module salt.modules.dockerio), 585
restart() (in module salt.modules.freebsdjail), 629
restart() (in module salt.modules.freebsdservice), 637
restart() (in module salt.modules.gentoo_service), 642
restart() (in module salt.modules.launchctl), 700
restart() (in module salt.modules.lxc), 718
restart() (in module salt.modules.monit), 748
restart() (in module salt.modules.netbsdservice), 761
restart() (in module salt.modules.openbsdrcctl), 802
restart() (in module salt.modules.openbsdservice), 804
restart() (in module salt.modules.rest_service), 873
restart() (in module salt.modules.rh_service), 877
restart() (in module salt.modules.runit), 881
restart() (in module salt.modules.service), 896
restart() (in module salt.modules.smf), 904
restart() (in module salt.modules.supervisord), 935
restart() (in module salt.modules.systemd), 954
restart() (in module salt.modules.upstart), 970
restart() (in module salt.modules.win_service), 1001
restore() (in module salt.modules.pkgng), 832
restore_backup() (in module salt.modules.file), 622
restorecon() (in module salt.modules.file), 622
result (salt.pillar.mysql.merger attribute), 1088
resume() (in module salt.modules.nova), 795
resume() (in module salt.modules.virt), 980
resume() (in module salt.modules.xapi), 1012
resume() (in module salt.runners.virt), 1181
ret_glob_minions() (salt.roster.flat.RosterMatcher method), 1155
ret_list_minions() (salt.roster.flat.RosterMatcher method), 1155
ret_pcre_minions() (salt.roster.flat.RosterMatcher method), 1155
ret_port
    conf/master, 415
retcode() (in module salt.modules.cmdmod), 548
retcode() (in module salt.modules.dockerio), 586
retcode() (in module salt.modules.lxc), 719
retcode() (in module salt.modules.nagios), 758
retcode() (in module salt.modules.test), 958
retcode_pillar() (in module salt.modules.nagios), 758
retrieve_auth_entries() (in module salt.auth.django), 306
retry_dns
    conf/minion, 444
Returner, **1678**
returner() (in module salt.returners.carbon_return), 1129
returner() (in module salt.returners.cassandra_return), 1130
returner() (in module salt.returners.couchbase_return), 1130
returner() (in module salt.returners.couchdb_return), 1131
returner() (in module salt.returners.django_return), 1132
returner() (in module salt.returners.elasticsearch_return), 1133
returner() (in module salt.returners.etcd_return), 1134
returner() (in module salt.returners.kafka_return), 1134
returner() (in module salt.returners.local), 1134
returner() (in module salt.returners.local_cache), 1135
returner() (in module salt.returners.memcache_return), 1136
returner() (in module salt.returners.mongo_future_return), 1137
returner() (in module salt.returners.mongo_return), 1137
returner() (in module salt.returners.multi_returner), 1138
returner() (in module salt.returners.mysql), 1140
returner() (in module salt.returners.odbc), 1143
returner() (in module salt.returners.postgres), 1144
returner() (in module salt.returners.postgres_local_cache), 1146
returner() (in module salt.returners.redis_return), 1147
returner() (in module salt.returners.sentry_return), 1147
returner() (in module salt.returners.sms_return), 1148
returner() (in module salt.returners.smtp_return), 1149
returner() (in module salt.returners.sqlite3_return), 1151
returner() (in module salt.returners.syslog_return), 1151
returner() (in module salt.returners.xmpp_return), 1152
returner_argspec() (in module salt.modules.sysmod), 950
returner_dirs
    conf/minion, 449
returner_doc() (in module salt.modules.sysmod), 950
revdep_rebuild() (in module salt.modules.gentoolkitmod), 643
reverse_ip() (in module salt.modules.network), 769
revision() (in module salt.modules.git), 649
revision() (in module salt.modules.hg), 669
revoke() (in module salt.modules.boto_secgroup), 522

revoke_auth() (in module salt.modules.saltutil), 888
revoke_cache_security_group_ingress() (in module salt.modules.boto_elasticache), 514
ring() (in module salt.modules.cassandra), 540
rm() (in module salt.modules.cron), 561
rm() (in module salt.modules.git), 649
rm() (in module salt.modules.incron), 675
rm() (in module salt.modules.parted), 814
rm_() (in module salt.modules.etcd_mod), 602
rm_alias() (in module salt.modules.aliases), 486
rm_auth_key() (in module salt.modules.ssh), 926
rm_automaster() (in module salt.modules.mount), 750
rm_dns() (in module salt.modules.win_dns_client), 985
rm_env() (in module salt.modules.cron), 562
rm_fstab() (in module salt.modules.mount), 750
rm_host() (in module salt.modules.hosts), 670
rm_job() (in module salt.modules.cron), 562
rm_job() (in module salt.modules.incron), 675
rm_known_host() (in module salt.modules.ssh), 926
rmconfig() (in module salt.modules.freebsdports), 635
rmdir() (in module salt.modules.file), 622
role_absent() (in module salt.states.keystone), 1345
role_create() (in module salt.modules.keystone), 695
role_delete() (in module salt.modules.keystone), 695
role_exists() (in module salt.modules.boto_iam), 519
role_get() (in module salt.modules.keystone), 695
role_get() (in module salt.modules.postgres), 845
role_list() (in module salt.modules.keystone), 695
role_present() (in module salt.states.keystone), 1345
rollback() (in module salt.modules.junos), 692
root_dir
    conf/master, 415
    conf/minion, 444
Roster, **1678**
roster_file
    conf/master, 419
RosterMatcher (class in salt.roster.flat), 1155
RosterMatcher (class in salt.roster.scan), 1155
rotate() (in module salt.modules.logadm), 707
rotate_aes_key
    conf/master, 422
route_table_exists() (in module salt.modules.boto_vpc), 532
routes() (in module salt.modules.network), 769
routes() (in module salt.states.network), 1368
rsync() (in module salt.modules.rsync), 880
rubygems() (in module salt.modules.rvm), 883
Run (class in salt.netapi.rest_cherrypy.app), 1049
run() (in module salt.modules.cmdmod), 548
run() (in module salt.modules.container_resource), 557
run() (in module salt.modules.dockerio), 586
run() (in module salt.modules.lxc), 719
run() (in module salt.modules.munin), 751
run() (in module salt.modules.nagios), 759
run() (in module salt.modules.puppet), 855
run() (in module salt.states.cmd), 1274
run() (in module salt.states.dockerio), 1289
run() (in module salt.states.module), 1358
run() (in module salt.states.mysql_query), 1363
run_all() (in module salt.modules.cmdmod), 549
run_all() (in module salt.modules.dockerio), 586
run_all() (in module salt.modules.lxc), 720
run_all() (in module salt.modules.munin), 751
run_all() (in module salt.modules.nagios), 759
run_all_pillar() (in module salt.modules.nagios), 759
run_buildout() (in module salt.modules.zcbuildout), 1028
run_chroot() (in module salt.modules.cmdmod), 549
run_cmd() (in module salt.modules.lxc), 720
run_command() (in module salt.modules.jboss7_cli), 691
run_job() (in module salt.modules.schedule), 892
run_job() (salt.client.LocalClient method), 340
run_operation() (in module salt.modules.jboss7_cli), 691
run_pillar() (in module salt.modules.nagios), 759
run_query() (in module salt.modules.oracle), 805
run_request() (in module salt.modules.state), 929
run_stderr() (in module salt.modules.cmdmod), 550
run_stderr() (in module salt.modules.dockerio), 586
run_stderr() (in module salt.modules.lxc), 720
run_stdout() (in module salt.modules.cmdmod), 550
run_stdout() (in module salt.modules.dockerio), 587
run_stdout() (in module salt.modules.lxc), 721
Runner Function, **1678**
Runner Module, **1678**
runner() (in module salt.modules.publish), 854
runner() (in module salt.modules.raet_publish), 866
runner() (in module salt.modules.saltutil), 888
runner() (in module salt.runners.doc), 1158
runner() (in module salt.states.saltmod), 1404
runner() (salt.netapi.NetapiClient method), 289
runner_argspec() (in module salt.modules.sysmod), 950
runner_dirs
    conf/master, 423
runner_doc() (in module salt.modules.sysmod), 950
RunnerClient (class in salt.runner), 341
running() (in module salt.modules.saltutil), 888
running() (in module salt.modules.state), 929
running() (in module salt.states.dockerio), 1289
running() (in module salt.states.lxc), 1352
running() (in module salt.states.service), 1411
running() (in module salt.states.supervisord), 1416
running_service_owners() (in module salt.modules.introspect), 681
RunSaltAPIHandler (in module salt.netapi.rest_tornado.saltnado), 1065

S

S3Credentials (class in salt.pillar.s3), 1096
safe_accept() (in module salt.runners.manage), 1169

Salt Cloud, **1678**
salt command line option
--args-separator=ARGS_SEPARATOR, 317
--async, 316
--force-color, 318, 332
--grain-pcre, 317, 331
--hide-timeout, 316
--log-file-level=LOG_LEVEL_LOGFILE, 317, 331
--log-file=LOG_FILE, 317, 331
--no-color, 318, 332
--out, 318, 331
--out-file=OUTPUT_FILE, --output-file=OUTPUT_FILE, 318, 332
--out-indent OUTPUT_INDENT, --output-indent OUTPUT_INDENT, 318, 332
--return=RETURNER, 317
--state-output=STATE_OUTPUT, 316
--subset=SUBSET, 316
--version, 316, 330
--versions-report, 316, 331
-C, --compound, 318
-E, --pcre, 317, 331
-G, --grain, 317, 331
-I, --pillar, 318
-L, --list, 317, 331
-N, --nodegroup, 318, 331
-R, --range, 318, 331
-S, --ipcidr, 318
-T, --make-token, 317
-a EAUTH, --auth=EAUTH, 317
-b BATCH, --batch-size=BATCH, 317
-c CONFIG_DIR, --config-dir=CONFIG_dir, 316, 331
-d, --doc, --documentation, 317
-h, --help, 316, 331
-l LOG_LEVEL, --log-level=LOG_LEVEL, 317, 331
-s, --static, 316
-t TIMEOUT, --timeout=TIMEOUT, 316
-v VERBOSE, --verbose, 316
Salt Mine, 143
Salt Reactor, 136
Salt SSH, **1678**
Salt Thin, **1679**
Salt Virt, **1679**
salt-api command line option
--log-file-level=LOG_LEVEL_LOGFILE, 334
--log-file=LOG_FILE, 334
--pid-file=PIDFILE, 334
--version, 334
--versions-report, 334
-c CONFIG_DIR, --config-dir=CONFIG_dir, 334
-d, --daemon, 334
-h, --help, 334
-l LOG_LEVEL, --log-level=LOG_LEVEL, 334
salt-call command line option
--file-root=FILE_ROOT, 314
--force-color, 315
--hard-crash, 314
--id=ID, 314
--local, 314
--log-file-level=LOG_LEVEL_LOGFILE, 315
--log-file=LOG_FILE, 315
--master=MASTER, 314
--metadata, 314
--no-color, 315
--out, 315
--out-file=OUTPUT_FILE, --output-file=OUTPUT_FILE, 315
--out-indent OUTPUT_INDENT, --output-indent OUTPUT_INDENT, 315
--pillar-root=PILLAR_ROOT, 314
--refresh-grains-cache, 314
--retcode-passthrough, 314
--return RETURNER, 314
--skip-grains, 314
--version, 314
--versions-report, 314
-c CONFIG_DIR, --config-dir=CONFIG_dir, 314
-d, --doc, --documentation, 314
-g, --grains, 314
-h, --help, 314
-l LOG_LEVEL, --log-level=LOG_LEVEL, 315
-m MODULE_DIRS, --module-dirs=MODULE_DIRS, 314
salt-cloud command line option
--force-color, 321
--list-images=LIST_IMAGES, 321
--list-locations=LIST_LOCATIONS, 321
--list-profiles, 321
--list-providers, 321
--list-sizes=LIST_SIZES, 321
--no-color, 321
--out, 321
--out-file=OUTPUT_FILE, --output-file=OUTPUT_FILE, 321
--out-indent OUTPUT_INDENT, --output-indent OUTPUT_INDENT, 321
--script-args=SCRIPT_ARGS, 320
--set-password=<USERNAME> <PROVIDER>, 321
--show-deploy-args, 320
--version, 319
--versions-report, 319
-F, --full-query, 320
-H, --hard, 320
-L LOCATION, --location=LOCATION, 319
-P, --parallel, 320
-Q, --query, 320
-S, --select-query, 320
-a ACTION, --action=ACTION, 320

-c CONFIG_DIR, --config-dir=CONFIG_dir, 319
-d, --destroy, 320
-f <FUNC-NAME> <PROVIDER>, --function=<FUNC-NAME> <PROVIDER>, 320
-h, --help, 319
-k, --keep-tmp, 320
-m MAP, --map=MAP, 320
-p PROFILE, --profile=PROFILE, 320
-u, --update-bootstrap, 320
-y, --assume-yes, 320
salt-cp command line option
--grain-pcre, 323
--log-file-level=LOG_LEVEL_LOGFILE, 323
--log-file=LOG_FILE, 323
--version, 323
--versions-report, 323
-E, --pcre, 323
-G, --grain, 323
-L, --list, 323
-N, --nodegroup, 323
-R, --range, 323
-c CONFIG_DIR, --config-dir=CONFIG_dir, 323
-h, --help, 323
-l LOG_LEVEL, --log-level=LOG_LEVEL, 323
-t TIMEOUT, --timeout=TIMEOUT, 323
salt-key command line option
--auto-create, 326
--force-color, 325
--gen-keys-dir=GEN_KEYS_DIR, 326
--gen-keys=GEN_KEYS, 326
--gen-signature, 326
--hard-crash, 324
--include-all, 326
--keysize=KEYSIZE, 326
--log-file-level=LOG_LEVEL_LOGFILE, 325
--log-file=LOG_FILE, 325
--no-color, 325
--out, 325
--out-file=OUTPUT_FILE, --output-file=OUTPUT_FILE, 325
--out-indent OUTPUT_INDENT, --output-indent OUTPUT_INDENT, 325
--priv=PRIV, 326
--pub=PUB, 326
--rotate-aes-key=ROTATE_AES_KEY, 324
--signature-path=SIGNATURE_PATH, 326
--version, 324
--versions-report, 324
-A, --accept-all, 325
-D, --delete-all, 326
-F, --finger-all, 326
-L, --list-all, 325
-P, --print-all, 326
-R, --reject-all, 326
-a ACCEPT, --accept=ACCEPT, 325
-c CONFIG_DIR, --config-dir=CONFIG_dir, 324
-d DELETE, --delete=DELETE, 326
-f FINGER, --finger=FINGER, 326
-h, --help, 324
-l ARG, --list=ARG, 325
-p PRINT, --print=PRINT, 326
-q, --quiet, 324
-r REJECT, --reject=REJECT, 325
-u USER, --user=USER, 324
-y, --yes, 324
salt-master command line option
--log-file-level=LOG_LEVEL_LOGFILE, 327
--log-file=LOG_FILE, 327
--pid-file PIDFILE, 327
--version, 327
--versions-report, 327
-c CONFIG_DIR, --config-dir=CONFIG_dir, 327
-d, --daemon, 327
-h, --help, 327
-l LOG_LEVEL, --log-level=LOG_LEVEL, 327
-u USER, --user=USER, 327
salt-minion command line option
--log-file-level=LOG_LEVEL_LOGFILE, 328
--log-file=LOG_FILE, 328
--pid-file PIDFILE, 328
--version, 328
--versions-report, 328
-c CONFIG_DIR, --config-dir=CONFIG_dir, 328
-d, --daemon, 328
-h, --help, 328
-l LOG_LEVEL, --log-level=LOG_LEVEL, 328
-u USER, --user=USER, 328
salt-run command line option
--hard-crash, 329
--log-file-level=LOG_LEVEL_LOGFILE, 329
--log-file=LOG_FILE, 329
--version, 329
--versions-report, 329
-c CONFIG_DIR, --config-dir=CONFIG_dir, 329
-d, --doc, --documentation, 329
-h, --help, 329
-l LOG_LEVEL, --log-level=LOG_LEVEL, 329
-t TIMEOUT, --timeout=TIMEOUT, 329
salt-syndic command line option
--log-file-level=LOG_LEVEL_LOGFILE, 333
--log-file=LOG_FILE, 333
--pid-file PIDFILE, 333
--version, 332
--versions-report, 333
-c CONFIG_DIR, --config-dir=CONFIG_dir, 333
-d, --daemon, 333
-h, --help, 333

-l LOG_LEVEL, --log-level=LOG_LEVEL, 333
-u USER, --user=USER, 333
salt.auth.auto (module), 305
salt.auth.django (module), 305
salt.auth.keystone (module), 307
salt.auth.ldap (module), 307
salt.auth.mysql (module), 307
salt.auth.pam (module), 308
salt.auth.pki (module), 309
salt.auth.stormpath_mod (module), 309
salt.auth.yubico (module), 310
salt.beacons.btmp (module), 1446
salt.beacons.diskusage (module), 1446
salt.beacons.inotify (module), 1447
salt.beacons.journald (module), 1447
salt.beacons.load (module), 1447
salt.beacons.network_info (module), 1448
salt.beacons.service (module), 1449
salt.beacons.sh (module), 1449
salt.beacons.twilio_txt_msg (module), 1450
salt.beacons.wtmp (module), 1450
salt.cloud.clouds.aliyun (module), 345
salt.cloud.clouds.botocore_aws (module), 348
salt.cloud.clouds.cloudstack (module), 348
salt.cloud.clouds.digital_ocean (module), 350
salt.cloud.clouds.digital_ocean_v2 (module), 351
salt.cloud.clouds.ec2 (module), 352
salt.cloud.clouds.gce (module), 359
salt.cloud.clouds.gogrid (module), 364
salt.cloud.clouds.libcloud_aws (module), 365
salt.cloud.clouds.linode (module), 366
salt.cloud.clouds.lxc (module), 369
salt.cloud.clouds.msazure (module), 369
salt.cloud.clouds.nova (module), 371
salt.cloud.clouds.opennebula (module), 374
salt.cloud.clouds.openstack (module), 375
salt.cloud.clouds.parallels (module), 377
salt.cloud.clouds.proxmox (module), 379
salt.cloud.clouds.pyrax (module), 381
salt.cloud.clouds.rackspace (module), 381
salt.cloud.clouds.saltify (module), 383
salt.cloud.clouds.softlayer (module), 383
salt.cloud.clouds.softlayer_hw (module), 384
salt.cloud.clouds.vsphere (module), 386
salt.exceptions (module), 474, 476
salt.fileserver.gitfs (module), 466
salt.fileserver.hgfs (module), 467
salt.fileserver.minionfs (module), 468
salt.fileserver.roots (module), 469
salt.fileserver.s3fs (module), 470
salt.fileserver.svnfs (module), 471
salt.log.handlers.logstash_mod (module), 458
salt.log.handlers.sentry_mod (module), 460
salt.modules.aliases (module), 485
salt.modules.alternatives (module), 486
salt.modules.apache (module), 487
salt.modules.aptpkg (module), 489
salt.modules.archive (module), 495
salt.modules.artifactory (module), 499
salt.modules.at (module), 500
salt.modules.augeas_cfg (module), 501
salt.modules.aws_sqs (module), 502
salt.modules.blockdev (module), 504
salt.modules.bluez (module), 504
salt.modules.boto_asg (module), 506
salt.modules.boto_cfn (module), 508
salt.modules.boto_cloudwatch (module), 509
salt.modules.boto_dynamodb (module), 511
salt.modules.boto_elasticache (module), 512
salt.modules.boto_elb (module), 514
salt.modules.boto_iam (module), 517
salt.modules.boto_route53 (module), 520
salt.modules.boto_secgroup (module), 521
salt.modules.boto_sns (module), 522
salt.modules.boto_sqs (module), 523
salt.modules.boto_vpc (module), 525
salt.modules.brew (module), 532
salt.modules.bridge (module), 535
salt.modules.bsd_shadow (module), 536
salt.modules.btrfs (module), 537
salt.modules.cassandra (module), 539
salt.modules.chef (module), 540
salt.modules.chocolatey (module), 542
salt.modules.cloud (module), 545
salt.modules.cmdmod (module), 547
salt.modules.composer (module), 553
salt.modules.config (module), 554
salt.modules.container_resource (module), 557
salt.modules.cp (module), 557
salt.modules.cpan (module), 560
salt.modules.cron (module), 561
salt.modules.cyg (module), 562
salt.modules.daemontools (module), 563
salt.modules.darwin_sysctl (module), 565
salt.modules.data (module), 565
salt.modules.ddns (module), 566
salt.modules.deb_apache (module), 567
salt.modules.debconfmod (module), 568
salt.modules.debian_ip (module), 569
salt.modules.debian_service (module), 570
salt.modules.defaults (module), 572
salt.modules.devmap (module), 573
salt.modules.dig (module), 573
salt.modules.disk (module), 574
salt.modules.djangomod (module), 575
salt.modules.dnsmasq (module), 575
salt.modules.dnsutil (module), 576
salt.modules.dockerio (module), 578

salt.modules.dpkg (module), 589
salt.modules.drac (module), 590
salt.modules.drbd (module), 593
salt.modules.ebuild (module), 593
salt.modules.eix (module), 598
salt.modules.elasticsearch (module), 598
salt.modules.environ (module), 599
salt.modules.eselect (module), 600
salt.modules.etcd_mod (module), 601
salt.modules.event (module), 602
salt.modules.extfs (module), 603
salt.modules.file (module), 606
salt.modules.firewalld (module), 626
salt.modules.freebsd_sysctl (module), 628
salt.modules.freebsdjail (module), 629
salt.modules.freebsdkmod (module), 630
salt.modules.freebsdpkg (module), 631
salt.modules.freebsdports (module), 634
salt.modules.freebsdservice (module), 636
salt.modules.fsutils (module), 637
salt.modules.gem (module), 637
salt.modules.genesis (module), 640
salt.modules.gentoo_service (module), 641
salt.modules.gentoolkitmod (module), 642
salt.modules.git (module), 644
salt.modules.glance (module), 650
salt.modules.glusterfs (module), 651
salt.modules.gnomedesktop (module), 653
salt.modules.gpg (module), 655
salt.modules.grains (module), 660
salt.modules.groupadd (module), 663
salt.modules.grub_legacy (module), 664
salt.modules.guestfs (module), 665
salt.modules.hadoop (module), 665
salt.modules.haproxyconn (module), 666
salt.modules.hashutil (module), 667
salt.modules.hg (module), 668
salt.modules.hosts (module), 669
salt.modules.htpasswd (module), 670
salt.modules.http (module), 671
salt.modules.ilo (module), 672
salt.modules.img (module), 674
salt.modules.incron (module), 675
salt.modules.influx (module), 676
salt.modules.ini_manage (module), 680
salt.modules.introspect (module), 681
salt.modules.ipset (module), 682
salt.modules.iptables (module), 683
salt.modules.jboss7 (module), 687
salt.modules.jboss7_cli (module), 690
salt.modules.junos (module), 692
salt.modules.key (module), 692
salt.modules.keyboard (module), 692
salt.modules.keystone (module), 693
salt.modules.kmod (module), 698
salt.modules.launchctl (module), 699
salt.modules.layman (module), 700
salt.modules.ldapmod (module), 701
salt.modules.linux_acl (module), 702
salt.modules.linux_lvm (module), 703
salt.modules.linux_sysctl (module), 704
salt.modules.localemod (module), 705
salt.modules.locate (module), 706
salt.modules.logadm (module), 707
salt.modules.logrotate (module), 707
salt.modules.lvs (module), 708
salt.modules.lxc (module), 710
salt.modules.mac_group (module), 723
salt.modules.mac_user (module), 724
salt.modules.macports (module), 726
salt.modules.makeconf (module), 728
salt.modules.match (module), 736
salt.modules.mdadm (module), 738
salt.modules.memcached (module), 740
salt.modules.mine (module), 741
salt.modules.mod_random (module), 742
salt.modules.modjk (module), 743
salt.modules.mongodb (module), 746
salt.modules.monit (module), 748
salt.modules.moosefs (module), 748
salt.modules.mount (module), 749
salt.modules.munin (module), 751
salt.modules.mysql (module), 751
salt.modules.nagios (module), 758
salt.modules.netbsd_sysctl (module), 760
salt.modules.netbsdservice (module), 760
salt.modules.netscaler (module), 762
salt.modules.network (module), 765
salt.modules.neutron (module), 769
salt.modules.nfs3 (module), 787
salt.modules.nftables (module), 787
salt.modules.nginx (module), 792
salt.modules.nova (module), 792
salt.modules.npm (module), 797
salt.modules.omapi (module), 798
salt.modules.openbsd_sysctl (module), 799
salt.modules.openbsdpkg (module), 799
salt.modules.openbsdrcctl (module), 801
salt.modules.openbsdservice (module), 803
salt.modules.openstack_config (module), 804
salt.modules.oracle (module), 805
salt.modules.osxdesktop (module), 806
salt.modules.pacman (module), 807
salt.modules.pagerduty (module), 810
salt.modules.pam (module), 811
salt.modules.parted (module), 811
salt.modules.pecl (module), 815
salt.modules.pillar (module), 816

salt.modules.pip (module), 817
salt.modules.pkg (module), 479
salt.modules.pkg_resource (module), 822
salt.modules.pkgin (module), 823
salt.modules.pkgng (module), 825
salt.modules.pkgutil (module), 837
salt.modules.portage_config (module), 839
salt.modules.postfix (module), 840
salt.modules.postgres (module), 841
salt.modules.poudriere (module), 847
salt.modules.powerpath (module), 849
salt.modules.ps (module), 849
salt.modules.publish (module), 853
salt.modules.puppet (module), 854
salt.modules.pw_group (module), 855
salt.modules.pw_user (module), 856
salt.modules.pyenv (module), 858
salt.modules.qemu_img (module), 859
salt.modules.qemu_nbd (module), 860
salt.modules.quota (module), 860
salt.modules.rabbitmq (module), 861
salt.modules.raet_publish (module), 865
salt.modules.rbenv (module), 866
salt.modules.rdp (module), 867
salt.modules.redismod (module), 868
salt.modules.reg (module), 871
salt.modules.rest_package (module), 872
salt.modules.rest_sample (module), 873
salt.modules.rest_service (module), 873
salt.modules.ret (module), 873
salt.modules.rh_ip (module), 874
salt.modules.rh_service (module), 875
salt.modules.riak (module), 877
salt.modules.rpm (module), 878
salt.modules.rsync (module), 879
salt.modules.runit (module), 880
salt.modules.rvm (module), 881
salt.modules.s3 (module), 884
salt.modules.saltcloudmod (module), 886
salt.modules.saltutil (module), 886
salt.modules.schedule (module), 890
salt.modules.scsi (module), 892
salt.modules.sdb (module), 892
salt.modules.seed (module), 893
salt.modules.selinux (module), 894
salt.modules.sensors (module), 895
salt.modules.serverdensity_device (module), 895
salt.modules.service (module), 896
salt.modules.shadow (module), 897
salt.modules.slack_notify (module), 899
salt.modules.smartos_imgadm (module), 900
salt.modules.smartos_vmadm (module), 901
salt.modules.smf (module), 902
salt.modules.smtp (module), 904
salt.modules.softwareupdate (module), 905
salt.modules.solaris_group (module), 907
salt.modules.solaris_shadow (module), 907
salt.modules.solaris_user (module), 908
salt.modules.solarisips (module), 910
salt.modules.solarispkg (module), 913
salt.modules.solr (module), 916
salt.modules.splunk_search (module), 922
salt.modules.sqlite3 (module), 923
salt.modules.ssh (module), 924
salt.modules.state (module), 927
salt.modules.status (module), 931
salt.modules.sudo (module), 934
salt.modules.supervisord (module), 934
salt.modules.svn (module), 936
salt.modules.swift (module), 939
salt.modules.sysbench (module), 941
salt.modules.syslog_ng (module), 942
salt.modules.sysmod (module), 946
salt.modules.sysrc (module), 951
salt.modules.system (module), 952
salt.modules.systemd (module), 952
salt.modules.test (module), 955
salt.modules.test_virtual (module), 959
salt.modules.timezone (module), 959
salt.modules.tls (module), 960
salt.modules.tomcat (module), 964
salt.modules.twilio_notify (module), 968
salt.modules.upstart (module), 968
salt.modules.useradd (module), 971
salt.modules.uwsgi (module), 973
salt.modules.varnish (module), 973
salt.modules.vbox_guest (module), 974
salt.modules.virt (module), 976
salt.modules.virtualenv_mod (module), 983
salt.modules.win_autoruns (module), 984
salt.modules.win_disk (module), 984
salt.modules.win_dns_client (module), 984
salt.modules.win_file (module), 985
salt.modules.win_firewall (module), 990
salt.modules.win_groupadd (module), 991
salt.modules.win_ip (module), 991
salt.modules.win_network (module), 993
salt.modules.win_ntp (module), 995
salt.modules.win_path (module), 995
salt.modules.win_pkg (module), 996
salt.modules.win_repo (module), 998
salt.modules.win_servermanager (module), 999
salt.modules.win_service (module), 1000
salt.modules.win_shadow (module), 1002
salt.modules.win_status (module), 1002
salt.modules.win_system (module), 1003
salt.modules.win_timezone (module), 1005
salt.modules.win_update (module), 1006

salt.modules.win_useradd (module), 1008
salt.modules.xapi (module), 1010
salt.modules.xfs (module), 1014
salt.modules.xmpp (module), 1016
salt.modules.yumpkg (module), 1017
salt.modules.zcbuildout (module), 1026
salt.modules.zfs (module), 1028
salt.modules.zk_concurrency (module), 1030
salt.modules.znc (module), 1031
salt.modules.zpool (module), 1031
salt.modules.zypper (module), 1034
salt.netapi.rest_cherrypy.app (module), 1040
salt.netapi.rest_cherrypy.wsgi (module), 1043
salt.netapi.rest_tornado.saltnado (module), 1058
salt.netapi.rest_tornado.saltnado_websockets (module), 1059
salt.netapi.rest_wsgi (module), 1065
salt.output.compact (module), 1067
salt.output.highstate (module), 1068
salt.output.json_out (module), 1069
salt.output.key (module), 1069
salt.output.nested (module), 1069
salt.output.newline_values_only (module), 1070
salt.output.no_out (module), 1071
salt.output.no_return (module), 1071
salt.output.overstatestage (module), 1072
salt.output.pprint_out (module), 1072
salt.output.progress (module), 1072
salt.output.raw (module), 1072
salt.output.txt (module), 1073
salt.output.virt_query (module), 1073
salt.output.yaml_out (module), 1073
salt.pillar.cmd_json (module), 1076
salt.pillar.cmd_yaml (module), 1076
salt.pillar.cmd_yamlex (module), 1076
salt.pillar.cobbler (module), 1076
salt.pillar.django_orm (module), 1077
salt.pillar.ec2_pillar (module), 1079
salt.pillar.etcd_pillar (module), 1079
salt.pillar.file_tree (module), 1080
salt.pillar.foreman (module), 1082
salt.pillar.git_pillar (module), 1082
salt.pillar.hiera (module), 1084
salt.pillar.libvirt (module), 1084
salt.pillar.mongo (module), 1084
salt.pillar.mysql (module), 1085
salt.pillar.pepa (module), 1089
salt.pillar.pillar_ldap (module), 1093
salt.pillar.puppet (module), 1093
salt.pillar.reclass_adapter (module), 1094
salt.pillar.redismod (module), 1094
salt.pillar.s3 (module), 1095
salt.pillar.svn_pillar (module), 1096
salt.pillar.varstack_pillar (module), 1097
salt.pillar.virtkey (module), 1097
salt.renderers.cheetah (module), 1100
salt.renderers.genshi (module), 1100
salt.renderers.gpg (module), 1100
salt.renderers.jinja (module), 1101
salt.renderers.json (module), 1106
salt.renderers.mako (module), 1107
salt.renderers.msgpack (module), 1107
salt.renderers.py (module), 1107
salt.renderers.pydsl (module), 1108
salt.renderers.pyobjects (module), 1113
salt.renderers.stateconf (module), 1116
salt.renderers.wempy (module), 1120
salt.renderers.yaml (module), 1121
salt.renderers.yamlex (module), 1122
salt.returners.carbon_return (module), 1128
salt.returners.cassandra_return (module), 1129
salt.returners.couchbase_return (module), 1130
salt.returners.couchdb_return (module), 1130
salt.returners.django_return (module), 1131
salt.returners.elasticsearch_return (module), 1132
salt.returners.etcd_return (module), 1133
salt.returners.kafka_return (module), 1134
salt.returners.local (module), 1134
salt.returners.local_cache (module), 1134
salt.returners.memcache_return (module), 1135
salt.returners.mongo_future_return (module), 1136
salt.returners.mongo_return (module), 1137
salt.returners.multi_returner (module), 1138
salt.returners.mysql (module), 1138
salt.returners.odbc (module), 1140
salt.returners.postgres (module), 1143
salt.returners.postgres_local_cache (module), 1145
salt.returners.redis_return (module), 1146
salt.returners.sentry_return (module), 1147
salt.returners.sms_return (module), 1148
salt.returners.smtp_return (module), 1148
salt.returners.sqlite3_return (module), 1149
salt.returners.syslog_return (module), 1151
salt.returners.xmpp_return (module), 1151
salt.roster.ansible (module), 1152
salt.roster.cache (module), 1154
salt.roster.cloud (module), 1155
salt.roster.clustershell (module), 1155
salt.roster.flat (module), 1155
salt.roster.scan (module), 1155
salt.runners.cache (module), 1157
salt.runners.cloud (module), 1158
salt.runners.doc (module), 1158
salt.runners.drac (module), 1159
salt.runners.error (module), 1159
salt.runners.f5 (module), 1160
salt.runners.fileserver (module), 1161
salt.runners.git_pillar (module), 1164

salt.runners.http (module), 1164
salt.runners.jobs (module), 1165
salt.runners.launchd (module), 1166
salt.runners.lxc (module), 1166
salt.runners.manage (module), 1168
salt.runners.mine (module), 1170
salt.runners.nacl (module), 1170
salt.runners.network (module), 1171
salt.runners.pagerduty (module), 1171
salt.runners.pillar (module), 1173
salt.runners.queue (module), 1173
salt.runners.sdb (module), 1175
salt.runners.search (module), 1175
salt.runners.state (module), 1175
salt.runners.survey (module), 1179
salt.runners.test (module), 1179
salt.runners.thin (module), 1180
salt.runners.virt (module), 1180
salt.runners.winrepo (module), 1181
salt.sdb.etcd_db (module), 1450
salt.sdb.keyring_db (module), 1451
salt.sdb.memcached (module), 1452
salt.sdb.sqlite3 (module), 1452
salt.states.alias (module), 1235
salt.states.alternatives (module), 1235
salt.states.apache (module), 1236
salt.states.apache_module (module), 1237
salt.states.apt (module), 1238
salt.states.archive (module), 1238
salt.states.artifactory (module), 1239
salt.states.at (module), 1240
salt.states.augeas (module), 1241
salt.states.aws_sqs (module), 1242
salt.states.blockdev (module), 1243
salt.states.boto_asg (module), 1244
salt.states.boto_cloudwatch_alarm (module), 1248
salt.states.boto_dynamodb (module), 1249
salt.states.boto_elasticache (module), 1252
salt.states.boto_elb (module), 1254
salt.states.boto_iam_role (module), 1258
salt.states.boto_lc (module), 1260
salt.states.boto_route53 (module), 1262
salt.states.boto_secgroup (module), 1264
salt.states.boto_sns (module), 1266
salt.states.boto_sqs (module), 1267
salt.states.chef (module), 1268
salt.states.cloud (module), 1270
salt.states.cmd (module), 1271
salt.states.composer (module), 1278
salt.states.cron (module), 1279
salt.states.cyg (module), 1282
salt.states.ddns (module), 1283
salt.states.debconfmod (module), 1284
salt.states.disk (module), 1286
salt.states.dockerio (module), 1286
salt.states.drac (module), 1292
salt.states.environ (module), 1292
salt.states.eselect (module), 1293
salt.states.event (module), 1293
salt.states.file (module), 1294
salt.states.gem (module), 1314
salt.states.git (module), 1314
salt.states.glusterfs (module), 1317
salt.states.gnomedesktop (module), 1318
salt.states.grafana (module), 1320
salt.states.grains (module), 1323
salt.states.group (module), 1325
salt.states.hg (module), 1326
salt.states.hipchat (module), 1326
salt.states.host (module), 1327
salt.states.htpasswd (module), 1328
salt.states.http (module), 1328
salt.states.incron (module), 1329
salt.states.influxdb_database (module), 1330
salt.states.influxdb_user (module), 1330
salt.states.ini_manage (module), 1331
salt.states.ipset (module), 1332
salt.states.iptables (module), 1334
salt.states.jboss7 (module), 1338
salt.states.keyboard (module), 1343
salt.states.keystone (module), 1344
salt.states.kmod (module), 1346
salt.states.layman (module), 1347
salt.states.libvirt (module), 1347
salt.states.linux_acl (module), 1347
salt.states.locale (module), 1348
salt.states.lvm (module), 1348
salt.states.lvs_server (module), 1349
salt.states.lvs_service (module), 1350
salt.states.lxc (module), 1350
salt.states.makeconf (module), 1353
salt.states.mdadm (module), 1354
salt.states.memcached (module), 1355
salt.states.modjk (module), 1355
salt.states.modjk_worker (module), 1356
salt.states.module (module), 1357
salt.states.mongodb_database (module), 1358
salt.states.mongodb_user (module), 1359
salt.states.mount (module), 1360
salt.states.mysql_database (module), 1361
salt.states.mysql_grants (module), 1361
salt.states.mysql_query (module), 1363
salt.states.mysql_user (module), 1364
salt.states.network (module), 1365
salt.states.nftables (module), 1368
salt.states.npm (module), 1371
salt.states.ntp (module), 1372
salt.states.openstack_config (module), 1372

salt.states.pagerduty (module), 1373
salt.states.pecl (module), 1374
salt.states.pip_state (module), 1374
salt.states.pkg (module), 1378
salt.states.pkgng (module), 1384
salt.states.pkgrepo (module), 1384
salt.states.portage_config (module), 1386
salt.states.ports (module), 1387
salt.states.postgres_database (module), 1388
salt.states.postgres_extension (module), 1389
salt.states.postgres_group (module), 1390
salt.states.postgres_schema (module), 1391
salt.states.postgres_user (module), 1392
salt.states.powerpath (module), 1393
salt.states.process (module), 1393
salt.states.pyenv (module), 1394
salt.states.pyrax_queues (module), 1395
salt.states.quota (module), 1395
salt.states.rabbitmq_cluster (module), 1396
salt.states.rabbitmq_plugin (module), 1396
salt.states.rabbitmq_policy (module), 1397
salt.states.rabbitmq_user (module), 1398
salt.states.rabbitmq_vhost (module), 1398
salt.states.rbenv (module), 1399
salt.states.rdp (module), 1400
salt.states.redismod (module), 1401
salt.states.reg (module), 1401
salt.states.rvm (module), 1402
salt.states.saltmod (module), 1404
salt.states.schedule (module), 1406
salt.states.selinux (module), 1408
salt.states.serverdensity_device (module), 1409
salt.states.service (module), 1410
salt.states.slack (module), 1411
salt.states.smtp (module), 1412
salt.states.splunk_search (module), 1412
salt.states.ssh_auth (module), 1413
salt.states.ssh_known_hosts (module), 1415
salt.states.stateconf (module), 1415
salt.states.status (module), 1416
salt.states.supervisord (module), 1416
salt.states.svn (module), 1417
salt.states.sysctl (module), 1418
salt.states.syslog_ng (module), 1418
salt.states.sysrc (module), 1419
salt.states.test (module), 1419
salt.states.timezone (module), 1421
salt.states.tomcat (module), 1421
salt.states.user (module), 1423
salt.states.vbox_guest (module), 1425
salt.states.virtualenv_mod (module), 1426
salt.states.win_dns_client (module), 1426
salt.states.win_firewall (module), 1427
salt.states.win_network (module), 1427
salt.states.win_path (module), 1428
salt.states.win_servermanager (module), 1429
salt.states.win_system (module), 1430
salt.states.win_update (module), 1430
salt.states.winrepo (module), 1432
salt.states.xmpp (module), 1433
salt.states.zcbuildout (module), 1433
salt.states.zk_concurrency (module), 1434
salt.tops.cobbler (module), 1441
salt.tops.ext_nodes (module), 1441
salt.tops.mongo (module), 1442
salt.tops.reclass_adapter (module), 1443
salt.utils.aggregation (module), 472
salt.utils.serializers (module), 478
salt.utils.serializers.json (module), 478
salt.utils.serializers.msgpack (module), 479
salt.utils.serializers.yaml (module), 478
salt.wheel.config (module), 1444
salt.wheel.error (module), 1444
salt.wheel.file_roots (module), 1444
salt.wheel.key (module), 1445
salt.wheel.minions (module), 1445
salt.wheel.pillar_roots (module), 1446
salt_call() (in module salt.modules.sudo), 934
SaltAPIHandler (in module salt.netapi.rest_tornado.saltnado), 1064
SaltAuthHandler (in module salt.netapi.rest_tornado.saltnado), 1064
SaltClientError, 475, 477
SaltClientTimeout, 475, 477
SaltCloudConfigError, 475, 477
SaltCloudException, 475, 477
SaltCloudExecutionFailure, 475, 477
SaltCloudExecutionTimeout, 475, 477
SaltCloudNotFound, 475, 477
SaltCloudPasswordError, 475, 477
SaltCloudSystemExit, 475, 477
SaltDaemonNotRunning, 475, 477
SaltException, 475, 477
SaltInvocationError, 475, 477
SaltMasterError, 475, 477
SaltRenderError, 476, 477
SaltReqTimeoutError, 476, 477
SaltRunnerError, 476, 477
SaltSyndicMasterError, 476, 477
SaltSystemExit, 476, 478
SaltWheelError, 476, 478
same() (salt.states.cyg.DictDiffer method), 1282
save() (in module salt.modules.dockerio), 587
save() (in module salt.modules.iptables), 687
save() (in module salt.modules.nftables), 791
save() (in module salt.modules.redismod), 870
save() (in module salt.modules.schedule), 892
save_config() (in module salt.modules.mdadm), 739

save_load() (in module salt.returners.couchbase_return), 1130
save_load() (in module salt.returners.django_return), 1132
save_load() (in module salt.returners.etcd_return), 1134
save_load() (in module salt.returners.local_cache), 1135
save_load() (in module salt.returners.memcache_return), 1136
save_load() (in module salt.returners.mongo_future_return), 1137
save_load() (in module salt.returners.multi_returner), 1138
save_load() (in module salt.returners.mysql), 1140
save_load() (in module salt.returners.odbc), 1143
save_load() (in module salt.returners.postgres), 1144
save_load() (in module salt.returners.postgres_local_cache), 1146
save_load() (in module salt.returners.redis_return), 1147
save_load() (in module salt.returners.sqlite3_return), 1151
say() (in module salt.modules.osxdesktop), 806
Scalar() (in module salt.utils.aggregation), 474
scan() (in module salt.modules.bluez), 505
schedule() (in module salt.modules.softwareupdate), 906
Scheduling Jobs, 154
schema_create() (in module salt.modules.postgres), 845
schema_exists() (in module salt.modules.postgres), 845
schema_get() (in module salt.modules.postgres), 845
schema_list() (in module salt.modules.postgres), 846
schema_remove() (in module salt.modules.postgres), 846
screensaver() (in module salt.modules.osxdesktop), 807
Script (class in salt.roster.ansible), 1154
script() (in module salt.cloud.clouds.aliyun), 347
script() (in module salt.cloud.clouds.cloudstack), 350
script() (in module salt.cloud.clouds.digital_ocean), 351
script() (in module salt.cloud.clouds.digital_ocean_v2), 352
script() (in module salt.cloud.clouds.ec2), 358
script() (in module salt.cloud.clouds.gce), 363
script() (in module salt.cloud.clouds.gogrid), 365
script() (in module salt.cloud.clouds.linode), 368
script() (in module salt.cloud.clouds.msazure), 370
script() (in module salt.cloud.clouds.nova), 373
script() (in module salt.cloud.clouds.opennebula), 374
script() (in module salt.cloud.clouds.openstack), 377
script() (in module salt.cloud.clouds.parallels), 378
script() (in module salt.cloud.clouds.proxmox), 380
script() (in module salt.cloud.clouds.rackspace), 382
script() (in module salt.cloud.clouds.saltify), 383
script() (in module salt.cloud.clouds.softlayer), 384
script() (in module salt.cloud.clouds.softlayer_hw), 385
script() (in module salt.cloud.clouds.vsphere), 387
script() (in module salt.modules.cmdmod), 551
script() (in module salt.modules.dockerio), 587
script() (in module salt.states.cmd), 1275
script() (in module salt.states.dockerio), 1291
script_retcode() (in module salt.modules.cmdmod), 551
script_retcode() (in module salt.modules.dockerio), 588
scrub() (in module salt.modules.zpool), 1033
search() (in module salt.modules.dockerio), 588
search() (in module salt.modules.file), 622
search() (in module salt.modules.freebsdports), 635
search() (in module salt.modules.ldapmod), 701
search() (in module salt.modules.pkgin), 825
search() (in module salt.modules.pkgng), 832
search() (in module salt.modules.solarisips), 912
search() (in module salt.modules.zypper), 1039
Search() (salt.modules.win_update.PyWinUpdater method), 1007
Search() (salt.states.win_update.PyWinUpdater method), 1431
search_keys() (in module salt.modules.gpg), 658
secgroup_create() (in module salt.modules.nova), 795
secgroup_delete() (in module salt.modules.nova), 795
secgroup_list() (in module salt.modules.nova), 795
sections_absent() (in module salt.states.ini_manage), 1332
sections_present() (in module salt.states.ini_manage), 1332
securitygroup() (in module salt.cloud.clouds.ec2), 358
securitygroup() (in module salt.cloud.clouds.libcloud_aws), 366
securitygroupid() (in module salt.cloud.clouds.ec2), 358
sed() (in module salt.modules.file), 623
sed_contains() (in module salt.modules.file), 623
seed_non_shared_migrate() (in module salt.modules.virt), 980
seek_read() (in module salt.modules.file), 623
seek_write() (in module salt.modules.file), 624
select_query() (in module salt.modules.cloud), 546
select_query() (in module salt.runners.cloud), 1158
select_query() (salt.cloud.CloudClient method), 344
selfupdate() (in module salt.modules.composer), 553
selinux_fs_path() (in module salt.modules.selinux), 894
send() (in module salt.modules.event), 603
send() (in module salt.modules.mine), 742
send() (in module salt.states.event), 1293
send_message() (in module salt.states.hipchat), 1326
send_msg() (in module salt.modules.smtp), 905
send_msg() (in module salt.modules.xmpp), 1016
send_msg() (in module salt.states.smtp), 1412
send_msg() (in module salt.states.xmpp), 1433
send_sms() (in module salt.modules.twilio_notify), 968
SendMsgBot (class in salt.modules.xmpp), 1016
SendMsgBot (class in salt.returners.xmpp_return), 1152
sense() (in module salt.modules.sensors), 895
Sequence (class in salt.utils.aggregation), 474
serial() (in module salt.modules.dnsutil), 578

serialize() (in module salt.states.file), 1311
serialize() (in module salt.utils.serializers.json), 478
serialize() (in module salt.utils.serializers.msgpack), 479
serialize() (in module salt.utils.serializers.yaml), 479
SerializerExtension (class in salt.utils.jinja), 1105
serve_file() (in module salt.fileserver.gitfs), 467
serve_file() (in module salt.fileserver.hgfs), 468
serve_file() (in module salt.fileserver.minionfs), 469
serve_file() (in module salt.fileserver.roots), 470
serve_file() (in module salt.fileserver.s3fs), 471
serve_file() (in module salt.fileserver.svnfs), 472
server_add() (in module salt.modules.netscaler), 762
server_by_name() (in module salt.modules.nova), 795
server_delete() (in module salt.modules.netscaler), 762
server_disable() (in module salt.modules.netscaler), 762
server_enable() (in module salt.modules.netscaler), 762
server_enabled() (in module salt.modules.netscaler), 762
server_exists() (in module salt.modules.netscaler), 762
server_hardreset() (in module salt.modules.drac), 591
server_list() (in module salt.modules.nova), 795
server_list_detailed() (in module salt.modules.nova), 795
server_poweroff() (in module salt.modules.drac), 591
server_poweron() (in module salt.modules.drac), 591
server_pxe() (in module salt.modules.drac), 592
server_reboot() (in module salt.modules.drac), 592
server_show() (in module salt.modules.nova), 796
server_status() (in module salt.modules.apache), 488
server_update() (in module salt.modules.netscaler), 763
serverinfo() (in module salt.modules.tomcat), 966
servermods() (in module salt.modules.apache), 488
service_absent() (in module salt.states.keystone), 1345
service_create() (in module salt.modules.keystone), 695
service_delete() (in module salt.modules.keystone), 695
service_disable() (in module salt.modules.netscaler), 763
service_enable() (in module salt.modules.netscaler), 763
service_exists() (in module salt.modules.netscaler), 763
service_get() (in module salt.modules.keystone), 695
service_highstate() (in module salt.modules.introspect), 681
service_list() (in module salt.modules.keystone), 696
service_present() (in module salt.states.keystone), 1345
service_up() (in module salt.modules.netscaler), 763
servicegroup_add() (in module salt.modules.netscaler), 763
servicegroup_delete() (in module salt.modules.netscaler), 763
servicegroup_exists() (in module salt.modules.netscaler), 763
servicegroup_server_add() (in module salt.modules.netscaler), 764
servicegroup_server_delete() (in module salt.modules.netscaler), 764
servicegroup_server_disable() (in module salt.modules.netscaler), 764
servicegroup_server_enable() (in module salt.modules.netscaler), 764
servicegroup_server_exists() (in module salt.modules.netscaler), 764
servicegroup_server_up() (in module salt.modules.netscaler), 764
sessions() (in module salt.modules.tomcat), 966
set() (in module salt.states.debconfmod), 1285
set() (in module salt.states.stateconf), 1416
set_() (in module salt.modules.alternatives), 487
set_() (in module salt.modules.debconfmod), 568
set_() (in module salt.modules.etcd_mod), 602
set_() (in module salt.modules.gnomedesktop), 654
set_() (in module salt.modules.logrotate), 707
set_() (in module salt.modules.memcached), 741
set_() (in module salt.modules.openstack_config), 805
set_() (in module salt.modules.parted), 814
set_() (in module salt.modules.quota), 861
set_() (in module salt.modules.sdb), 892
set_() (in module salt.modules.sysrc), 952
set_() (in module salt.runners.sdb), 1175
set_() (in module salt.sdb.etcd_db), 1451
set_() (in module salt.sdb.keyring_db), 1452
set_() (in module salt.sdb.memcached), 1452
set_() (in module salt.sdb.sqlite3), 1453
set_() (in module salt.states.alternatives), 1236
set_() (in module salt.states.eselect), 1293
set_absent() (in module salt.states.ipset), 1333
set_attributes() (in module salt.modules.boto_elb), 517
set_attributes() (in module salt.modules.boto_sqs), 524
set_attributes() (in module salt.modules.win_file), 989
set_auth_key() (in module salt.modules.ssh), 926
set_auth_key_from_file() (in module salt.modules.ssh), 926
set_automaster() (in module salt.modules.mount), 750
set_autostart() (in module salt.modules.virt), 980
set_binary_path() (in module salt.modules.syslog_ng), 945
set_ca_path() (in module salt.modules.tls), 964
set_cflags() (in module salt.modules.makeconf), 732
set_change() (in module salt.modules.bsd_shadow), 536
set_chost() (in module salt.modules.makeconf), 732
set_computer_desc() (in module salt.modules.win_system), 1004
set_computer_name() (in module salt.modules.win_system), 1004
set_config() (in module salt.modules.dnsmasq), 576
set_config_file() (in module salt.modules.syslog_ng), 945
set_cxxflags() (in module salt.modules.makeconf), 733
set_date() (in module salt.modules.shadow), 898
set_default() (in module salt.modules.rvm), 884
set_default_zone() (in module salt.modules.firewalld), 628
set_dhcp_all() (in module salt.modules.win_ip), 992

set_dhcp_dns() (in module salt.modules.win_ip), 993
set_dhcp_ip() (in module salt.modules.win_ip), 993
set_dns() (in module salt.modules.lxc), 721
set_emerge_default_opts() (in module salt.modules.makeconf), 733
set_env() (in module salt.modules.cron), 562
set_expire() (in module salt.modules.bsd_shadow), 536
set_expire() (in module salt.modules.shadow), 898
set_file() (in module salt.modules.debconfmod), 568
set_file() (in module salt.states.debconfmod), 1285
set_fstab() (in module salt.modules.mount), 750
set_gentoo_mirrors() (in module salt.modules.makeconf), 733
set_health_check() (in module salt.modules.boto_elb), 517
set_host() (in module salt.modules.hosts), 670
set_hostname() (in module salt.modules.junos), 692
set_http_port() (in module salt.modules.ilo), 674
set_https_port() (in module salt.modules.ilo), 674
set_hwclock() (in module salt.modules.timezone), 959
set_hwclock() (in module salt.modules.win_timezone), 1006
set_id() (in module salt.modules.parted), 815
set_inactdays() (in module salt.modules.shadow), 898
set_is_polling() (in module salt.modules.solr), 921
set_job() (in module salt.modules.cron), 562
set_job() (in module salt.modules.incron), 675
set_key() (in module salt.modules.redismod), 871
set_key() (in module salt.modules.reg), 872
set_known_host() (in module salt.modules.ssh), 926
set_locale() (in module salt.modules.localemod), 706
set_main() (in module salt.modules.postfix), 840
set_makeopts() (in module salt.modules.makeconf), 733
set_master() (in module salt.modules.postfix), 840
set_maxdays() (in module salt.modules.shadow), 898
set_maxdays() (in module salt.modules.solaris_shadow), 908
set_mindays() (in module salt.modules.shadow), 898
set_mindays() (in module salt.modules.solaris_shadow), 908
set_mode() (in module salt.modules.file), 624
set_mode() (in module salt.modules.win_file), 989
set_network() (in module salt.modules.drac), 592
set_option() (in module salt.modules.ini_manage), 681
set_output_volume() (in module salt.modules.osxdesktop), 807
set_parameter() (in module salt.modules.lxc), 722
set_parameters() (in module salt.modules.syslog_ng), 945
set_pass() (in module salt.states.lxc), 1352
set_password() (in module salt.modules.bsd_shadow), 536
set_password() (in module salt.modules.lxc), 722
set_password() (in module salt.modules.shadow), 898
set_password() (in module salt.modules.solaris_shadow), 908
set_password() (in module salt.modules.win_shadow), 1002
set_permissions() (in module salt.modules.drac), 592
set_permissions() (in module salt.modules.rabbitmq), 864
set_policy() (in module salt.modules.iptables), 687
set_policy() (in module salt.modules.rabbitmq), 864
set_policy() (in module salt.states.iptables), 1338
set_present() (in module salt.states.ipset), 1333
set_replication_enabled() (in module salt.modules.solr), 921
set_salt_view() (in module salt.returners.couchdb_return), 1131
set_selections() (in module salt.modules.aptpkg), 494
set_selinux_context() (in module salt.modules.file), 624
set_servers() (in module salt.modules.win_ntp), 995
set_snmp() (in module salt.modules.drac), 592
set_special() (in module salt.modules.cron), 562
set_ssh_key() (in module salt.modules.ilo), 674
set_ssh_port() (in module salt.modules.ilo), 674
set_static_dns() (in module salt.modules.win_ip), 993
set_static_ip() (in module salt.modules.win_ip), 993
set_sync() (in module salt.modules.makeconf), 733
set_sys() (in module salt.modules.keyboard), 692
set_system_date() (in module salt.modules.win_system), 1004
set_system_time() (in module salt.modules.win_system), 1004
set_tags() (in module salt.cloud.clouds.ec2), 358
set_tags() (in module salt.cloud.clouds.libcloud_aws), 366
set_target() (in module salt.modules.aliases), 486
set_target() (in module salt.modules.eselect), 601
set_template() (in module salt.modules.debconfmod), 569
set_user_tags() (in module salt.modules.rabbitmq), 864
set_var() (in module salt.modules.makeconf), 734
set_vm_status() (in module salt.cloud.clouds.proxmox), 380
set_warndays() (in module salt.modules.shadow), 898
set_warndays() (in module salt.modules.solaris_shadow), 908
set_weight() (in module salt.modules.haproxyconn), 666
set_x() (in module salt.modules.keyboard), 693
set_zone() (in module salt.modules.timezone), 959
set_zone() (in module salt.modules.win_timezone), 1006
SetCategories() (salt.modules.win_update.PyWinUpdater method), 1007
SetCategories() (salt.states.win_update.PyWinUpdater method), 1431
setClockFormat() (in module salt.modules.gnomedesktop), 654
setClockShowDate() (in module salt.modules.gnomedesktop), 654
setenforce() (in module salt.modules.selinux), 894

setenv() (in module salt.modules.environ), 599
setenv() (in module salt.states.environ), 1292
setIdleActivation() (in module salt.modules.gnomedesktop), 654
setIdleDelay() (in module salt.modules.gnomedesktop), 654
SetInclude() (salt.modules.win_update.PyWinUpdater method), 1007
SetInclude() (salt.states.win_update.PyWinUpdater method), 1431
SetIncludes() (salt.modules.win_update.PyWinUpdater method), 1007
SetIncludes() (salt.states.win_update.PyWinUpdater method), 1431
setmem() (in module salt.modules.smartos_vmadm), 901
setmem() (in module salt.modules.virt), 980
setmem() (in module salt.modules.xapi), 1012
setpassword() (in module salt.modules.win_useradd), 1010
setsebool() (in module salt.modules.selinux), 894
setsebools() (in module salt.modules.selinux), 894
setval() (in module salt.modules.environ), 600
setval() (in module salt.modules.grains), 663
setvals() (in module salt.modules.grains), 663
setvalue() (in module salt.modules.augeas_cfg), 502
setvcpus() (in module salt.modules.virt), 980
setvcpus() (in module salt.modules.xapi), 1012
sha256_digest() (in module salt.modules.hashutil), 667
sha512_digest() (in module salt.modules.hashutil), 668
shadow_hash() (in module salt.modules.mod_random), 743
shell() (in module salt.modules.cmdmod), 551
shells() (in module salt.modules.cmdmod), 552
show() (in module salt.modules.bridge), 535
show() (in module salt.modules.cpan), 561
show() (in module salt.modules.darwin_sysctl), 565
show() (in module salt.modules.debconfmod), 569
show() (in module salt.modules.freebsd_sysctl), 629
show() (in module salt.modules.linux_sysctl), 705
show() (in module salt.modules.netbsd_sysctl), 760
show() (in module salt.modules.nova), 796
show() (in module salt.modules.openbsd_sysctl), 799
show() (in module salt.modules.smartos_imgadm), 900
show() (in module salt.modules.systemd), 954
show_address() (in module salt.cloud.clouds.gce), 363
show_backends() (in module salt.modules.haproxyconn), 667
show_conf() (in module salt.modules.logadm), 707
show_conf() (in module salt.modules.logrotate), 707
show_config() (in module salt.modules.cpan), 561
show_config() (in module salt.modules.freebsdjail), 629
show_current() (in module salt.modules.alternatives), 487
show_dbs() (in module salt.modules.oracle), 806
show_delvol_on_destroy() (in module salt.cloud.clouds.ec2), 358
show_disk() (in module salt.cloud.clouds.aliyun), 347
show_disk() (in module salt.cloud.clouds.gce), 363
show_env() (in module salt.modules.oracle), 806
show_floatingip() (in module salt.modules.neutron), 781
show_frontends() (in module salt.modules.haproxyconn), 667
show_fwrule() (in module salt.cloud.clouds.gce), 363
show_hc() (in module salt.cloud.clouds.gce), 363
show_highstate() (in module salt.modules.state), 929
show_ikepolicy() (in module salt.modules.neutron), 782
show_image() (in module salt.cloud.clouds.aliyun), 347
show_image() (in module salt.cloud.clouds.ec2), 358
show_image() (in module salt.cloud.clouds.parallels), 378
show_instance() (in module salt.cloud.clouds.aliyun), 347
show_instance() (in module salt.cloud.clouds.cloudstack), 350
show_instance() (in module salt.cloud.clouds.digital_ocean), 351
show_instance() (in module salt.cloud.clouds.digital_ocean_v2), 352
show_instance() (in module salt.cloud.clouds.ec2), 358
show_instance() (in module salt.cloud.clouds.gce), 363
show_instance() (in module salt.cloud.clouds.gogrid), 365
show_instance() (in module salt.cloud.clouds.linode), 368
show_instance() (in module salt.cloud.clouds.lxc), 369
show_instance() (in module salt.cloud.clouds.msazure), 370
show_instance() (in module salt.cloud.clouds.nova), 373
show_instance() (in module salt.cloud.clouds.opennebula), 374
show_instance() (in module salt.cloud.clouds.openstack), 377
show_instance() (in module salt.cloud.clouds.parallels), 378
show_instance() (in module salt.cloud.clouds.proxmox), 380
show_instance() (in module salt.cloud.clouds.rackspace), 382
show_instance() (in module salt.cloud.clouds.softlayer), 384
show_instance() (in module salt.cloud.clouds.softlayer_hw), 385
show_instance() (in module salt.cloud.clouds.vsphere), 387
show_ipsec_site_connection() (in module salt.modules.neutron), 782
show_ipsecpolicy() (in module salt.modules.neutron), 782
show_keypair() (in module salt.cloud.clouds.digital_ocean), 351
show_keypair() (in module

salt.cloud.clouds.digital_ocean_v2), 352
show_keypair() (in module salt.cloud.clouds.ec2), 358
show_lb() (in module salt.cloud.clouds.gce), 363
show_low_sls() (in module salt.modules.state), 929
show_lowstate() (in module salt.modules.state), 929
show_main() (in module salt.modules.postfix), 841
show_master() (in module salt.modules.postfix), 841
show_network() (in module salt.cloud.clouds.gce), 363
show_network() (in module salt.modules.neutron), 782
show_pillar() (in module salt.modules.oracle), 806
show_pillar() (in module salt.runners.pillar), 1173
show_port() (in module salt.modules.neutron), 783
show_queue() (in module salt.modules.postfix), 841
show_quota() (in module salt.modules.neutron), 783
show_router() (in module salt.modules.neutron), 783
show_security_group() (in module salt.modules.neutron), 783
show_security_group_rule() (in module salt.modules.neutron), 784
show_service() (in module salt.cloud.clouds.msazure), 370
show_sls() (in module salt.modules.state), 929
show_snapshot() (in module salt.cloud.clouds.gce), 364
show_stages() (in module salt.runners.state), 1178
show_subnet() (in module salt.modules.neutron), 784
show_term_protect() (in module salt.cloud.clouds.ec2), 358
show_top() (in module salt.modules.state), 930
show_top() (in module salt.runners.pillar), 1173
show_volume() (in module salt.cloud.clouds.ec2), 358
show_vpnservice() (in module salt.modules.neutron), 784
showconfig() (in module salt.modules.freebsdports), 635
showglobal() (in module salt.modules.mysql), 756
showvariables() (in module salt.modules.mysql), 756
shutdown() (in module salt.cloud.clouds.proxmox), 380
shutdown() (in module salt.modules.redismod), 871
shutdown() (in module salt.modules.smartos_vmadm), 902
shutdown() (in module salt.modules.system), 952
shutdown() (in module salt.modules.virt), 980
shutdown() (in module salt.modules.win_system), 1005
shutdown() (in module salt.modules.xapi), 1012
shutdown_hard() (in module salt.modules.win_system), 1005
sign() (in module salt.modules.gpg), 658
signal() (in module salt.modules.apache), 488
signal() (in module salt.modules.nginx), 792
signal() (in module salt.modules.solr), 922
signal() (in module salt.modules.tomcat), 967
signal_job() (in module salt.modules.saltutil), 888
SimpleParameter (class in salt.modules.syslog_ng), 943
SimpleParameterValue (class in salt.modules.syslog_ng), 943
single() (in module salt.modules.state), 930
slave_lag() (in module salt.modules.mysql), 756
slaveof() (in module salt.modules.redismod), 871
sleep() (in module salt.modules.test), 958
sleep() (in module salt.runners.test), 1180
SLS Module, **1679**
sls() (in module salt.modules.state), 930
sls_id() (in module salt.modules.state), 931
smembers() (in module salt.modules.redismod), 871
sock_dir
  conf/master, 417
  conf/minion, 446
solo() (in module salt.modules.chef), 541
solo() (in module salt.states.chef), 1269
sort_pkglist() (in module salt.modules.pkg_resource), 822
source_list() (in module salt.modules.file), 624
sources_add() (in module salt.modules.gem), 638
sources_list() (in module salt.modules.gem), 638
sources_remove() (in module salt.modules.gem), 639
SPF() (in module salt.modules.dig), 574
SPF() (in module salt.modules.dnsutil), 577
sqlite_version() (in module salt.modules.sqlite3), 924
ssh_interface() (in module salt.cloud.clouds.ec2), 359
ssh_interface() (in module salt.cloud.clouds.libcloud_aws), 366
ssh_interface() (in module salt.cloud.clouds.nova), 373
ssh_interface() (in module salt.cloud.clouds.openstack), 377
ssh_interface() (in module salt.cloud.clouds.rackspace), 382
ssh_minion_opts
  conf/master, 419
ssh_username() (in module salt.cloud.clouds.libcloud_aws), 366
SSHClient (class in salt.client.ssh.client), 344
stack() (in module salt.modules.test), 958
start() (in module salt.cloud.clouds.aliyun), 347
start() (in module salt.cloud.clouds.ec2), 359
start() (in module salt.cloud.clouds.libcloud_aws), 366
start() (in module salt.cloud.clouds.parallels), 378
start() (in module salt.cloud.clouds.proxmox), 380
start() (in module salt.modules.bluez), 505
start() (in module salt.modules.daemontools), 564
start() (in module salt.modules.debian_service), 572
start() (in module salt.modules.dockerio), 588
start() (in module salt.modules.freebsdjail), 629
start() (in module salt.modules.freebsdservice), 637
start() (in module salt.modules.gentoo_service), 642
start() (in module salt.modules.launchctl), 700
start() (in module salt.modules.lxc), 722
start() (in module salt.modules.monit), 748
start() (in module salt.modules.netbsdservice), 762
start() (in module salt.modules.openbsdrcctl), 802
start() (in module salt.modules.openbsdservice), 804

start() (in module salt.modules.rest_service), 873
start() (in module salt.modules.rh_service), 877
start() (in module salt.modules.riak), 878
start() (in module salt.modules.runit), 881
start() (in module salt.modules.service), 896
start() (in module salt.modules.smartos_vmadm), 902
start() (in module salt.modules.smf), 904
start() (in module salt.modules.supervisord), 935
start() (in module salt.modules.syslog_ng), 945
start() (in module salt.modules.systemd), 954
start() (in module salt.modules.tomcat), 967
start() (in module salt.modules.upstart), 970
start() (in module salt.modules.virt), 981
start() (in module salt.modules.win_service), 1001
start() (in module salt.modules.xapi), 1013
start() (in module salt.runners.lxc), 1168
start() (in module salt.runners.virt), 1181
start() (salt.modules.xmpp.SendMsgBot method), 1016
start() (salt.returners.xmpp_return.SendMsgBot method), 1152
start_app() (in module salt.modules.rabbitmq), 864
start_time_service() (in module salt.modules.win_system), 1005
start_volume() (in module salt.modules.glusterfs), 653
started() (in module salt.states.glusterfs), 1318
started() (in module salt.states.lxc), 1353
started() (in module salt.states.syslog_ng), 1419
stash() (in module salt.modules.git), 649
State Compiler, **1679**
State Declaration, **1679**
State Function, **1679**
State Module, **1679**
State Run, **1679**
state() (in module salt.modules.lxc), 722
state() (in module salt.states.saltmod), 1405
state_argspec() (in module salt.modules.sysmod), 951
state_doc() (in module salt.modules.sysmod), 951
state_output
    conf/master, 424
    conf/minion, 450
state_top
    conf/master, 423
state_verbose
    conf/master, 424
    conf/minion, 450
Statement (class in salt.modules.syslog_ng), 943
states() (in module salt.loader), 336
states_dirs
    conf/minion, 449
Stats (class in salt.netapi.rest_cherrypy.app), 1057
stats() (in module salt.modules.file), 624
stats() (in module salt.modules.locate), 706
stats() (in module salt.modules.pkgng), 834
stats() (in module salt.modules.quota), 861
stats() (in module salt.modules.syslog_ng), 945
stats() (in module salt.modules.uwsgi), 973
stats() (in module salt.modules.win_file), 989
status() (in module salt.modules.daemontools), 564
status() (in module salt.modules.debian_service), 572
status() (in module salt.modules.freebsdjail), 630
status() (in module salt.modules.freebsdservice), 637
status() (in module salt.modules.gentoo_service), 642
status() (in module salt.modules.git), 650
status() (in module salt.modules.glusterfs), 653
status() (in module salt.modules.jboss7), 689
status() (in module salt.modules.launchctl), 700
status() (in module salt.modules.memcached), 741
status() (in module salt.modules.monit), 748
status() (in module salt.modules.mysql), 756
status() (in module salt.modules.netbsdservice), 762
status() (in module salt.modules.nginx), 792
status() (in module salt.modules.openbsdrcctl), 802
status() (in module salt.modules.openbsdservice), 804
status() (in module salt.modules.puppet), 855
status() (in module salt.modules.rabbitmq), 864
status() (in module salt.modules.rdp), 868
status() (in module salt.modules.rest_service), 873
status() (in module salt.modules.rh_service), 877
status() (in module salt.modules.runit), 881
status() (in module salt.modules.service), 896
status() (in module salt.modules.smf), 904
status() (in module salt.modules.supervisord), 935
status() (in module salt.modules.svn), 938
status() (in module salt.modules.systemd), 954
status() (in module salt.modules.tomcat), 967
status() (in module salt.modules.upstart), 971
status() (in module salt.modules.win_service), 1002
status() (in module salt.modules.zpool), 1034
status() (in module salt.runners.manage), 1169
status() (in module salt.states.disk), 1286
status_raw() (in module salt.modules.supervisord), 936
status_webapp() (in module salt.modules.tomcat), 967
statvfs() (in module salt.modules.file), 624
stdout_print() (in module salt.runners.test), 1180
stop() (in module salt.cloud.clouds.aliyun), 347
stop() (in module salt.cloud.clouds.ec2), 359
stop() (in module salt.cloud.clouds.libcloud_aws), 366
stop() (in module salt.cloud.clouds.parallels), 378
stop() (in module salt.cloud.clouds.proxmox), 381
stop() (in module salt.modules.bluez), 505
stop() (in module salt.modules.daemontools), 564
stop() (in module salt.modules.debian_service), 572
stop() (in module salt.modules.dockerio), 588
stop() (in module salt.modules.freebsdjail), 630
stop() (in module salt.modules.freebsdservice), 637
stop() (in module salt.modules.gentoo_service), 642
stop() (in module salt.modules.launchctl), 700
stop() (in module salt.modules.lxc), 722

stop() (in module salt.modules.mdadm), 740
stop() (in module salt.modules.monit), 748
stop() (in module salt.modules.netbsdservice), 762
stop() (in module salt.modules.openbsdrcctl), 802
stop() (in module salt.modules.openbsdservice), 804
stop() (in module salt.modules.rest_service), 873
stop() (in module salt.modules.rh_service), 877
stop() (in module salt.modules.riak), 878
stop() (in module salt.modules.runit), 881
stop() (in module salt.modules.service), 897
stop() (in module salt.modules.smf), 904
stop() (in module salt.modules.supervisord), 936
stop() (in module salt.modules.syslog_ng), 946
stop() (in module salt.modules.systemd), 954
stop() (in module salt.modules.tomcat), 967
stop() (in module salt.modules.upstart), 971
stop() (in module salt.modules.virt), 981
stop() (in module salt.modules.win_service), 1002
stop() (in module salt.runners.lxc), 1168
stop() (in module salt.states.modjk_worker), 1357
stop_app() (in module salt.modules.rabbitmq), 864
stop_server() (in module salt.modules.jboss7), 689
stop_time_service() (in module salt.modules.win_system), 1005
stop_volume() (in module salt.modules.glusterfs), 653
stopped() (in module salt.states.lxc), 1353
stopped() (in module salt.states.syslog_ng), 1419
stp() (in module salt.modules.bridge), 535
str_encode() (in module salt.modules.mod_random), 743
stream() (in module salt.runners.test), 1180
string() (in module salt.states.redismod), 1401
stringify() (in module salt.modules.pkg_resource), 822
submodule() (in module salt.modules.git), 650
subnet_exists() (in module salt.modules.boto_vpc), 532
subnets() (in module salt.modules.network), 769
subnets() (in module salt.modules.win_network), 995
succeed_with_changes() (in module salt.states.test), 1421
succeed_without_changes() (in module salt.states.test), 1421
summary() (in module salt.modules.monit), 748
summary() (in module salt.modules.puppet), 855
suspend() (in module salt.modules.nova), 796
svnfs_branches
    conf/master, 433
svnfs_env_blacklist
    conf/master, 434
svnfs_env_whitelist
    conf/master, 433
svnfs_mountpoint
    conf/master, 432
svnfs_remotes
    conf/master, 432
svnfs_root
    conf/master, 432
svnfs_tags
    conf/master, 433
svnfs_trunk
    conf/master, 433
SvnPillar (class in salt.pillar.svn_pillar), 1097
swap() (in module salt.states.mount), 1360
swap_memory() (in module salt.modules.ps), 852
swapoff() (in module salt.modules.mount), 750
swapon() (in module salt.modules.mount), 750
swaps() (in module salt.modules.mount), 751
switch() (in module salt.modules.svn), 939
symlink() (in module salt.modules.file), 624
symlink() (in module salt.modules.win_file), 989
symlink() (in module salt.states.file), 1312
symlink_list() (in module salt.fileserver.gitfs), 467
symlink_list() (in module salt.fileserver.roots), 470
symlink_list() (in module salt.runners.fileserver), 1163
sync() (in module salt.modules.eix), 598
sync() (in module salt.modules.layman), 701
sync_all() (in module salt.modules.saltutil), 888
sync_contains() (in module salt.modules.makeconf), 734
sync_grains() (in module salt.modules.saltutil), 888
sync_modules() (in module salt.modules.saltutil), 888
sync_outputters() (in module salt.modules.saltutil), 889
sync_renderers() (in module salt.modules.saltutil), 889
sync_returners() (in module salt.modules.saltutil), 889
sync_states() (in module salt.modules.saltutil), 889
sync_utils() (in module salt.modules.saltutil), 889
syncdb() (in module salt.modules.djangomod), 575
Syndic, **1679**
syndic_log_file
    conf/master, 438
syndic_master
    conf/master, 438
syndic_master_log_file
    conf/master, 438
syndic_master_port
    conf/master, 438
sysctl() (in module salt.modules.freebsdjail), 630
syslog() (in module salt.modules.drac), 592
SyslogNgError, 943
system() (in module salt.states.keyboard), 1343
system() (in module salt.states.locale), 1348
system() (in module salt.states.network), 1368
system() (in module salt.states.timezone), 1421
system_info() (in module salt.modules.drac), 593
system_types() (in module salt.modules.parted), 815
systemctl_reload() (in module salt.modules.systemd), 954

T

tables() (in module salt.modules.sqlite3), 924
tag() (in module salt.modules.dockerio), 589
tar() (in module salt.modules.archive), 497
Target, **1679**

Target (class in salt.roster.ansible), 1154
targets() (in module salt.roster.ansible), 1154
targets() (in module salt.roster.cache), 1154
targets() (in module salt.roster.cloud), 1155
targets() (in module salt.roster.clustershell), 1155
targets() (in module salt.roster.flat), 1155
targets() (in module salt.roster.scan), 1156
targets() (salt.roster.ansible.Target method), 1154
targets() (salt.roster.flat.RosterMatcher method), 1155
targets() (salt.roster.scan.RosterMatcher method), 1156
tcp_pub_port
    conf/minion, 448
tcp_pull_port
    conf/minion, 448
template() (in module salt.modules.state), 931
template_str() (in module salt.modules.state), 931
templates() (in module salt.modules.lxc), 723
tenant_absent() (in module salt.states.keystone), 1345
tenant_create() (in module salt.modules.keystone), 696
tenant_delete() (in module salt.modules.keystone), 696
tenant_get() (in module salt.modules.keystone), 696
tenant_list() (in module salt.modules.keystone), 696
tenant_present() (in module salt.states.keystone), 1345
tenant_update() (in module salt.modules.keystone), 696
term() (in module salt.modules.daemontools), 564
term() (in module salt.modules.runit), 881
term_job() (in module salt.modules.saltutil), 889
test
    conf/master, 424
test() (in module salt.modules.ipset), 683
threads() (in module salt.modules.sysbench), 941
time() (in module salt.modules.redismod), 871
TimedProcTimeoutError, 476, 478
timeout
    conf/master, 416
toggle() (in module salt.modules.parted), 815
token_expire
    conf/master, 421
token_get() (in module salt.modules.keystone), 696
TokenAuthenticationError, 476, 478
tokenize_grant() (in module salt.modules.mysql), 756
Top File, **1679**
top() (in module salt.modules.dockerio), 589
top() (in module salt.modules.ps), 852
top() (in module salt.modules.state), 931
top() (in module salt.tops.cobbler), 1441
top() (in module salt.tops.ext_nodes), 1442
top() (in module salt.tops.mongo), 1442
top() (in module salt.tops.reclass_adapter), 1443
total_physical_memory() (in module salt.modules.ps), 852
touch() (in module salt.modules.file), 625
touch() (in module salt.states.file), 1313
tpstats() (in module salt.modules.cassandra), 540
traceroute() (in module salt.modules.network), 769
traceroute() (in module salt.modules.win_network), 995
tree() (in module salt.modules.augeas_cfg), 502
tree() (in module salt.modules.etcd_mod), 602
trim_cflags() (in module salt.modules.makeconf), 734
trim_cxxflags() (in module salt.modules.makeconf), 734
trim_emerge_default_opts() (in module salt.modules.makeconf), 734
trim_features() (in module salt.modules.makeconf), 735
trim_gentoo_mirrors() (in module salt.modules.makeconf), 735
trim_makeopts() (in module salt.modules.makeconf), 735
trim_var() (in module salt.modules.makeconf), 735
truncate() (in module salt.modules.file), 625
trust_key() (in module salt.modules.gpg), 659
try_() (in module salt.modules.test), 958
tty() (in module salt.modules.cmdmod), 552
tty() (in module salt.modules.test), 958
tune() (in module salt.modules.blockdev), 504
tune() (in module salt.modules.extfs), 605
tuned() (in module salt.states.blockdev), 1243
TXT() (in module salt.modules.dig), 574
TypedParameter (class in salt.modules.syslog_ng), 943
TypedParameterValue (class in salt.modules.syslog_ng), 944

U

uid_to_user() (in module salt.modules.file), 625
uid_to_user() (in module salt.modules.win_file), 990
umount() (in module salt.modules.mount), 751
umount_image() (in module salt.modules.img), 674
unblock() (in module salt.modules.bluez), 505
unchanged() (salt.states.cyg.DictDiffer method), 1283
uncomment() (in module salt.modules.file), 625
uncomment() (in module salt.states.file), 1313
undefine() (in module salt.modules.virt), 981
undeploy() (in module salt.modules.jboss7), 690
undeploy() (in module salt.modules.tomcat), 968
undeployed() (in module salt.states.tomcat), 1422
unfreeze() (in module salt.modules.lxc), 723
unfreeze() (in module salt.runners.lxc), 1168
unhold() (in module salt.modules.aptpkg), 494
unhold() (in module salt.modules.postfix), 841
unhold() (in module salt.modules.yumpkg), 1025
uninstall() (in module salt.modules.chocolatey), 544
uninstall() (in module salt.modules.cyg), 563
uninstall() (in module salt.modules.gem), 639
uninstall() (in module salt.modules.npm), 798
uninstall() (in module salt.modules.pecl), 815
uninstall() (in module salt.modules.pip), 820
uninstall_python() (in module salt.modules.pyenv), 859
uninstall_ruby() (in module salt.modules.rbenv), 867
unlock() (in module salt.modules.zk_concurrency), 1030
unlock() (in module salt.states.zk_concurrency), 1435

unmask() (in module salt.modules.systemd), 955
unmonitor() (in module salt.modules.monit), 748
unmounted() (in module salt.states.mount), 1361
UnnamedStatement (class in salt.modules.syslog_ng), 944
unpack() (in module salt.modules.genesis), 640
unpair() (in module salt.modules.bluez), 505
unpurge() (in module salt.modules.dpkg), 590
unrar() (in module salt.modules.archive), 498
unzip() (in module salt.modules.archive), 498
up() (in module salt.modules.debian_ip), 570
up() (in module salt.modules.rh_ip), 875
up() (in module salt.runners.manage), 1170
update() (in module salt.fileserver.gitfs), 467
update() (in module salt.fileserver.hgfs), 468
update() (in module salt.fileserver.minionfs), 469
update() (in module salt.fileserver.roots), 470
update() (in module salt.fileserver.s3fs), 471
update() (in module salt.fileserver.svnfs), 472
update() (in module salt.modules.boto_asg), 508
update() (in module salt.modules.chocolatey), 544
update() (in module salt.modules.composer), 554
update() (in module salt.modules.cyg), 563
update() (in module salt.modules.data), 566
update() (in module salt.modules.ddns), 567
update() (in module salt.modules.ebuild), 596
update() (in module salt.modules.eix), 598
update() (in module salt.modules.freebsdports), 635
update() (in module salt.modules.gem), 639
update() (in module salt.modules.hg), 669
update() (in module salt.modules.mine), 742
update() (in module salt.modules.pecl), 816
update() (in module salt.modules.pyenv), 859
update() (in module salt.modules.rbenv), 867
update() (in module salt.modules.saltutil), 889
update() (in module salt.modules.serverdensity_device), 896
update() (in module salt.modules.splunk_search), 923
update() (in module salt.modules.supervisord), 936
update() (in module salt.modules.svn), 939
update() (in module salt.pillar.git_pillar), 1084
update() (in module salt.runners.fileserver), 1164
update() (in module salt.runners.git_pillar), 1164
update() (in module salt.states.composer), 1279
update() (salt.pillar.git_pillar.GitPillar method), 1083
update() (salt.pillar.svn_pillar.SvnPillar method), 1097
update_ca_bundle() (in module salt.modules.http), 671
update_ca_bundle() (in module salt.runners.http), 1165
update_config() (in module salt.wheel.config), 1444
update_datasource() (in module salt.modules.jboss7), 690
update_floatingip() (in module salt.modules.neutron), 784
update_git_repos() (in module salt.modules.win_repo), 999
update_git_repos() (in module salt.runners.winrepo), 1181
update_installed() (in module salt.modules.smartos_imgadm), 900
update_jail() (in module salt.modules.poudriere), 848
update_lxc_conf() (in module salt.modules.lxc), 723
update_network() (in module salt.modules.neutron), 785
update_package_site() (in module salt.modules.pkgng), 834
update_packaging_site() (in module salt.states.pkgng), 1384
update_port() (in module salt.modules.neutron), 785
update_ports_tree() (in module salt.modules.poudriere), 848
update_quota() (in module salt.modules.neutron), 785
update_record() (in module salt.modules.boto_route53), 520
update_restart_services
    conf/minion, 455
update_router() (in module salt.modules.neutron), 786
update_security_group() (in module salt.modules.neutron), 786
update_simple_binding() (in module salt.modules.jboss7), 690
update_subnet() (in module salt.modules.neutron), 786
update_system() (in module salt.modules.gem), 639
update_url
    conf/minion, 455
update_vpnservice() (in module salt.modules.neutron), 787
updated() (in module salt.states.cyg), 1283
updatedb() (in module salt.modules.locate), 706
updating() (in module salt.modules.pkgng), 834
upgrade() (in module salt.modules.aptpkg), 495
upgrade() (in module salt.modules.brew), 534
upgrade() (in module salt.modules.ebuild), 597
upgrade() (in module salt.modules.freebsdpkg), 633
upgrade() (in module salt.modules.macports), 728
upgrade() (in module salt.modules.pacman), 809
upgrade() (in module salt.modules.pip), 821
upgrade() (in module salt.modules.pkgin), 825
upgrade() (in module salt.modules.pkgng), 835
upgrade() (in module salt.modules.pkgutil), 838
upgrade() (in module salt.modules.softwareupdate), 906
upgrade() (in module salt.modules.solarisips), 912
upgrade() (in module salt.modules.win_pkg), 998
upgrade() (in module salt.modules.yumpkg), 1025
upgrade() (in module salt.modules.zypper), 1039
upgrade_available() (in module salt.modules.aptpkg), 495
upgrade_available() (in module salt.modules.brew), 534
upgrade_available() (in module salt.modules.ebuild), 597
upgrade_available() (in module salt.modules.macports), 728

upgrade_available() (in module salt.modules.pacman), 809
upgrade_available() (in module salt.modules.pip), 821
upgrade_available() (in module salt.modules.pkgutil), 838
upgrade_available() (in module salt.modules.softwareupdate), 907
upgrade_available() (in module salt.modules.solarisips), 912
upgrade_available() (in module salt.modules.solarispkg), 916
upgrade_available() (in module salt.modules.win_pkg), 998
upgrade_available() (in module salt.modules.yumpkg), 1026
upgrade_available() (in module salt.modules.zypper), 1039
upgrade_bootstrap() (in module salt.modules.zcbuildout), 1028
uptime() (in module salt.modules.status), 933
uptodate() (in module salt.states.pip_state), 1377
uptodate() (in module salt.states.pkg), 1383
usage() (in module salt.modules.btrfs), 539
usage() (in module salt.modules.disk), 574
usage() (in module salt.modules.win_disk), 984
use_master_when_local
    conf/minion, 451
user
    conf/master, 414
    conf/minion, 444
user_absent() (in module salt.states.keystone), 1346
user_chpass() (in module salt.modules.influx), 678
user_chpass() (in module salt.modules.mysql), 756
user_create() (in module salt.modules.influx), 678
user_create() (in module salt.modules.keystone), 697
user_create() (in module salt.modules.mongodb), 747
user_create() (in module salt.modules.mysql), 757
user_create() (in module salt.modules.postgres), 846
user_delete() (in module salt.modules.keystone), 697
user_exists() (in module salt.modules.influx), 678
user_exists() (in module salt.modules.mongodb), 747
user_exists() (in module salt.modules.mysql), 757
user_exists() (in module salt.modules.postgres), 846
user_exists() (in module salt.modules.rabbitmq), 864
user_exists() (in module salt.states.htpasswd), 1328
user_get() (in module salt.modules.keystone), 697
user_grants() (in module salt.modules.mysql), 757
user_info() (in module salt.modules.mysql), 758
user_keys() (in module salt.modules.ssh), 926
user_list() (in module salt.modules.influx), 679
user_list() (in module salt.modules.keystone), 697
user_list() (in module salt.modules.mongodb), 747
user_list() (in module salt.modules.mysql), 758
user_list() (in module salt.modules.postgres), 847
user_password_update() (in module salt.modules.keystone), 697
user_present() (in module salt.states.keystone), 1346
user_remove() (in module salt.modules.influx), 679
user_remove() (in module salt.modules.mongodb), 747
user_remove() (in module salt.modules.mysql), 758
user_remove() (in module salt.modules.postgres), 847
user_role_add() (in module salt.modules.keystone), 697
user_role_list() (in module salt.modules.keystone), 697
user_role_remove() (in module salt.modules.keystone), 698
user_to_uid() (in module salt.modules.file), 625
user_to_uid() (in module salt.modules.win_file), 990
user_update() (in module salt.modules.keystone), 698
user_update() (in module salt.modules.postgres), 847
user_verify_password() (in module salt.modules.keystone), 698
useradd() (in module salt.modules.apache), 488
useradd() (in module salt.modules.htpasswd), 670
useradd_all() (in module salt.modules.htpasswd), 671
userdel() (in module salt.modules.apache), 488
userdel() (in module salt.modules.htpasswd), 671
ustring() (salt.output.nested.NestDisplay method), 1070

V

valid_fileproto() (in module salt.modules.config), 557
validate() (in module salt.pillar.pepa), 1093
values() (in module salt.wheel.config), 1444
var_contains() (in module salt.modules.makeconf), 735
vcpu_pin() (in module salt.modules.xapi), 1013
verify() (in module salt.modules.gpg), 659
verify() (in module salt.modules.rpm), 879
verify() (in module salt.modules.yumpkg), 1026
verify() (in module salt.modules.zypper), 1039
verify_env
    conf/master, 416
    conf/minion, 445
verify_master_pubkey_sign
    conf/minion, 452
version() (in module salt.modules.apache), 489
version() (in module salt.modules.aptpkg), 495
version() (in module salt.modules.bluez), 506
version() (in module salt.modules.brew), 534
version() (in module salt.modules.btrfs), 539
version() (in module salt.modules.cassandra), 540
version() (in module salt.modules.chocolatey), 544
version() (in module salt.modules.dnsmasq), 576
version() (in module salt.modules.dockerio), 589
version() (in module salt.modules.ebuild), 597
version() (in module salt.modules.firewalld), 628
version() (in module salt.modules.freebsdpkg), 634
version() (in module salt.modules.grub_legacy), 664
version() (in module salt.modules.hadoop), 665
version() (in module salt.modules.ipset), 683

version() (in module salt.modules.iptables), 687
version() (in module salt.modules.linux_acl), 702
version() (in module salt.modules.linux_lvm), 704
version() (in module salt.modules.locate), 706
version() (in module salt.modules.macports), 728
version() (in module salt.modules.modjk), 745
version() (in module salt.modules.mysql), 758
version() (in module salt.modules.nftables), 791
version() (in module salt.modules.nginx), 792
version() (in module salt.modules.openbsdpkg), 801
version() (in module salt.modules.oracle), 806
version() (in module salt.modules.pacman), 810
version() (in module salt.modules.pip), 821
version() (in module salt.modules.pkg_resource), 823
version() (in module salt.modules.pkgin), 825
version() (in module salt.modules.pkgng), 836
version() (in module salt.modules.pkgutil), 838
version() (in module salt.modules.postgres), 847
version() (in module salt.modules.poudriere), 848
version() (in module salt.modules.rest_package), 872
version() (in module salt.modules.rsync), 880
version() (in module salt.modules.smartos_imgadm), 900
version() (in module salt.modules.solarisips), 913
version() (in module salt.modules.solarispkg), 916
version() (in module salt.modules.solr), 922
version() (in module salt.modules.sqlite3), 924
version() (in module salt.modules.status), 933
version() (in module salt.modules.syslog_ng), 946
version() (in module salt.modules.test), 958
version() (in module salt.modules.tomcat), 968
version() (in module salt.modules.varnish), 974
version() (in module salt.modules.win_pkg), 998
version() (in module salt.modules.yumpkg), 1026
version() (in module salt.modules.znc), 1031
version() (in module salt.modules.zypper), 1039
version() (in module salt.runners.drac), 1159
version_clean() (in module salt.modules.ebuild), 597
version_clean() (in module salt.modules.pkg_resource), 823
version_cmp() (in module salt.modules.aptpkg), 495
version_cmp() (in module salt.modules.ebuild), 597
versions() (in module salt.modules.pyenv), 859
versions() (in module salt.modules.rbenv), 867
versions() (in module salt.runners.manage), 1170
versions_information() (in module salt.modules.test), 958
versions_report() (in module salt.modules.test), 959
vg_absent() (in module salt.states.lvm), 1349
vg_present() (in module salt.states.lvm), 1349
vgcreate() (in module salt.modules.linux_lvm), 704
vgdisplay() (in module salt.modules.linux_lvm), 704
vgextend() (in module salt.modules.linux_lvm), 704
vgremove() (in module salt.modules.linux_lvm), 704
vhost_exists() (in module salt.modules.rabbitmq), 864
vhosts() (in module salt.modules.apache), 489
virt_type() (in module salt.modules.virt), 981
virtual_interface_create() (in module salt.cloud.clouds.nova), 373
virtual_interface_create() (in module salt.modules.cloud), 546
virtual_interface_list() (in module salt.cloud.clouds.nova), 373
virtual_interface_list() (in module salt.modules.cloud), 547
virtual_memory() (in module salt.modules.ps), 852
vm_cputime() (in module salt.modules.virt), 981
vm_cputime() (in module salt.modules.xapi), 1013
vm_diskstats() (in module salt.modules.virt), 981
vm_diskstats() (in module salt.modules.xapi), 1013
vm_info() (in module salt.modules.smartos_vmadm), 902
vm_info() (in module salt.modules.virt), 982
vm_info() (in module salt.modules.xapi), 1013
vm_info() (in module salt.runners.virt), 1181
vm_netstats() (in module salt.modules.virt), 982
vm_netstats() (in module salt.modules.xapi), 1014
vm_state() (in module salt.modules.virt), 982
vm_state() (in module salt.modules.xapi), 1014
vm_virt_type() (in module salt.modules.smartos_vmadm), 902
vmstats() (in module salt.modules.status), 933
volume_absent() (in module salt.states.cloud), 1271
volume_attach() (in module salt.cloud.clouds.nova), 373
volume_attach() (in module salt.modules.cloud), 547
volume_attach() (in module salt.modules.nova), 796
volume_attached() (in module salt.states.cloud), 1271
volume_create() (in module salt.cloud.clouds.nova), 373
volume_create() (in module salt.modules.cloud), 547
volume_create() (in module salt.modules.nova), 796
volume_delete() (in module salt.cloud.clouds.nova), 373
volume_delete() (in module salt.modules.cloud), 547
volume_delete() (in module salt.modules.nova), 796
volume_detach() (in module salt.cloud.clouds.nova), 373
volume_detach() (in module salt.modules.cloud), 547
volume_detach() (in module salt.modules.nova), 797
volume_detached() (in module salt.states.cloud), 1271
volume_list() (in module salt.cloud.clouds.nova), 373
volume_list() (in module salt.modules.cloud), 547
volume_list() (in module salt.modules.nova), 797
volume_present() (in module salt.states.cloud), 1271
volume_show() (in module salt.modules.nova), 797
vserver_add() (in module salt.modules.netscaler), 764
vserver_delete() (in module salt.modules.netscaler), 764
vserver_exists() (in module salt.modules.netscaler), 765
vserver_servicegroup_add() (in module salt.modules.netscaler), 765
vserver_servicegroup_delete() (in module salt.modules.netscaler), 765
vserver_servicegroup_exists() (in module salt.modules.netscaler), 765

vserver_sslcert_add() (in module salt.modules.netscaler), 765
vserver_sslcert_delete() (in module salt.modules.netscaler), 765
vserver_sslcert_exists() (in module salt.modules.netscaler), 765

W

w() (in module salt.modules.status), 933
wait() (in module salt.modules.dockerio), 589
wait() (in module salt.states.cmd), 1276
wait() (in module salt.states.event), 1293
wait() (in module salt.states.module), 1358
wait() (in module salt.states.tomcat), 1422
wait_call() (in module salt.states.cmd), 1276
wait_for_created() (in module salt.cloud.clouds.proxmox), 381
wait_for_event() (in module salt.states.saltmod), 1405
wait_for_instance() (in module salt.cloud.clouds.ec2), 359
wait_for_ip() (in module salt.cloud.clouds.vsphere), 387
wait_for_state() (in module salt.cloud.clouds.proxmox), 381
wait_script() (in module salt.states.cmd), 1277
wait_until() (in module salt.cloud.clouds.parallels), 379
waitfor_job() (in module salt.cloud.clouds.linode), 368
waitfor_status() (in module salt.cloud.clouds.linode), 368
war_deployed() (in module salt.states.tomcat), 1423
warn() (in module salt.modules.quota), 861
Webhook (class in salt.netapi.rest_cherrypy.app), 1052
WebhookSaltAPIHandler (in module salt.netapi.rest_tornado.saltnado), 1065
WebsocketEndpoint (class in salt.netapi.rest_cherrypy.app), 1056
wheel() (in module salt.modules.saltutil), 890
wheel() (in module salt.runners.doc), 1159
wheel() (in module salt.states.saltmod), 1406
wheel() (salt.netapi.NetapiClient method), 289
WheelClient (class in salt.wheel), 342
which() (in module salt.modules.cmdmod), 552
which() (in module salt.modules.pkgng), 836
which_bin() (in module salt.modules.cmdmod), 552
win_gitrepos
    conf/master, 442
win_repo
    conf/master, 442
win_repo_mastercachefile
    conf/master, 442
wipe() (in module salt.modules.blockdev), 504
wipefacls() (in module salt.modules.linux_acl), 702
with_lists (salt.pillar.mysql.merger attribute), 1088
wm_preferences() (in module salt.states.gnomedesktop), 1319
wol() (in module salt.modules.network), 769
wol() (in module salt.runners.network), 1171
wollist() (in module salt.runners.network), 1171
Worker, 1679
worker_activate() (in module salt.modules.modjk), 745
worker_activated() (in module salt.states.modjk), 1355
worker_disable() (in module salt.modules.modjk), 745
worker_disabled() (in module salt.states.modjk), 1355
worker_edit() (in module salt.modules.modjk), 745
worker_recover() (in module salt.modules.modjk), 746
worker_recover() (in module salt.states.modjk), 1356
worker_status() (in module salt.modules.modjk), 746
worker_stop() (in module salt.modules.modjk), 746
worker_stopped() (in module salt.states.modjk), 1356
worker_threads
    conf/master, 415
workers() (in module salt.modules.modjk), 746
wrapper() (in module salt.modules.rvm), 884
write() (in module salt.modules.file), 626
write() (in module salt.wheel.file_roots), 1445
write() (in module salt.wheel.pillar_roots), 1446
write_conf() (in module salt.modules.lxc), 723
write_config() (in module salt.modules.syslog_ng), 946
write_cron_file() (in module salt.modules.cron), 562
write_cron_file_verbose() (in module salt.modules.cron), 562
write_cron_file_verbose() (in module salt.modules.incron), 676
write_incron_file() (in module salt.modules.incron), 676
write_launchd_plist() (in module salt.runners.launchd), 1166
write_version() (in module salt.modules.syslog_ng), 946

X

xorg() (in module salt.states.keyboard), 1343

Y

yaml_utf8
    conf/master, 424

Z

zcard() (in module salt.modules.redismod), 871
zero() (in module salt.modules.lvs), 710
zip_() (in module salt.modules.archive), 498
zone_compare() (in module salt.modules.timezone), 960
zone_compare() (in module salt.modules.win_timezone), 1006
zpool_list() (in module salt.modules.zpool), 1034
zrange() (in module salt.modules.redismod), 871