Learning SaltStack

Learn how to manage your infrastructure by utilizing the power of SaltStack

Colton Myers
In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 1 "Diving In – Our First Salt Commands"
- A synopsis of the book’s content
- More information on Learning SaltStack

About the Author

Colton Myers is a software engineer living in Salt Lake City, Utah. Since graduating with a BS in computer science from the University of Utah, he has worked professionally writing software in Python. Colton loves working on open source software, and he has represented the US PyCon conference as a speaker multiple times.

He is a SaltStack Certified Trainer and has worked on the Salt open source software for years. At the time of writing this book, Colton was working as a platform engineer for SaltStack.

You can find him on Twitter and GitHub at @basepi.

I would like to thank my friends and family for the support they've given me while writing this book. I especially want to thank my parents, Mark and Annette, as I would not be who or where I am without their support. Finally, I'd like to thank Thomas Hatch, the creator of SaltStack, who gave me the opportunity to help develop such a great piece of software.
Learning SaltStack

SaltStack (or Salt, for short) is an open source project that was started by Thomas Hatch in 2011. It was originally intended to be a lightning-fast remote-execution system. Later, the various pieces that make up the Salt that we know today were added on top of this flexible layer. Salt is now one of the most popular open source projects in the world and one of the most popular infrastructure management platforms.

The project is managed by SaltStack, a company dedicated to preserving the open source nature of the software. SaltStack provides service, long-term support, and custom code in its enterprise product. The company also supports the open source Salt project with a team of dedicated engineers.

Why do you care? What can Salt do for you?

Salt is the easiest, most powerful way to manage your servers. Whether you have a few, hundreds, or even tens of thousands of servers, you can use Salt to manage them from a single, central point. You can use it to flexibly target any subset of your servers to run commands or accomplish tasks. You can use the state system to define the state of your infrastructure in a data-driven way and then enforce that state in seconds, with a single command. You can even create a reactive, self-healing infrastructure using the event system. Salt is written in Python and designed to be easy to extend for your own specific use cases or purposes.

We're going to learn how to do all this and more in these pages. By the end of this book, you will have the knowledge you need to begin making the management of your infrastructure easier with Salt.

Let's get to it!

What This Book Covers

Chapter 1, Diving In – Our First Salt Commands, teaches you how to install Salt and execute basic commands.

Chapter 2, Controlling Your Minions with Remote Execution, covers how to use Salt to accomplish tasks on your minions through remote execution.

Chapter 3, Execution Modules – Write Your Own Solution, teaches you how to write your own custom remote-execution modules to extend Salt for your own purposes.

Chapter 4, Defining the State of Your Infrastructure, covers how to use Salt states to define and enforce the state of your infrastructure.

Chapter 5, Expanding Our States with Jinja2 and Pillar, shows how to make your states more flexible and powerful using Jinja2 and pillar data.
Chapter 6, *The Highstate and Environments*, teaches you how to structure your states into environments and enforce the state of your entire infrastructure using a single command.

Chapter 7, *Using Salt Cloud to Manage Virtual Minions*, covers how to manage your cloud virtual machines using Salt Cloud to create and manage VMs.

Chapter 8, *The Reactor and the Event System*, shows how to make your infrastructure automatically react to changes using the reactor and the event system built into Salt.
Diving In – Our First Salt Commands

Salt is more than just configuration management or remote execution: it is a powerful platform that not only gives you unique tools to manage your infrastructure, but also the power to create new tools to fit your infrastructure’s unique needs. However, everything starts with the foundation of lightning-fast remote execution, so that’s where we will start.

In this chapter, you will learn how to:

• Install Salt
• Configure the master and the minion
• Connect the minion to the master
• Run our first remote execution commands

This book assumes that you already have root access on a device with a common distribution of Linux installed. The machine used in the examples in this book is running Ubuntu 14.04 unless otherwise stated. Most examples should run on other major distributions, such as recent versions of Fedora, RHEL 5/6 or Arch Linux.

Introducing Salt

Before installing Salt, we should learn the basic architecture of a Salt deployment.

The two main pieces of Salt are the **Salt Master** and the **Salt Minion**. The master is the central hub. All minions connect to the master to receive instructions. From the master, you can run commands and apply configuration across hundreds or thousands of minions in seconds.
The minion, as mentioned before, connects to the master and treats the master as the source of all truth. Although minions can exist without a master, the full power of Salt is realized when you have minions and the master working together.

Salt is built on two major concepts: remote execution and configuration management. In the remote execution system, Salt leverages Python to accomplish complex tasks with single-function calls. The configuration management system in Salt, called States, builds upon the remote execution foundation to create repeatable, enforceable configuration for the minions.

With this bird's-eye view in mind, let's get Salt installed so that we can start learning how to use it to make managing our infrastructure easier!

**Installing Salt**

The dependencies for running Salt at the time of writing are as follows:

- Python 2—Version 2.6 or greater (not Python 3-compatible)
- msgpack-python
- YAML
- Jinja2
- MarkupSafe
- Apache Libcloud
- Requests
- ZeroMQ—Version 3.2.0 or greater
- PyZMQ—Version 2.2.0 or greater
- PyCrypto
- M2Crypto

The easiest way to ensure that the dependencies for Salt are met is to use system-specific package management systems, such as apt on Ubuntu systems, that will handle the dependency-resolution automatically. You can also use a script called Salt-Bootstrap to handle all of the system-specific commands for you. **Salt-Bootstrap** is an open source project with the goal of creating a Bourne shell-compatible script that will install Salt on any compatible server. The project is managed and hosted by the SaltStack team. You can find more information at https://github.com/saltstack/salt-bootstrap.

We will explore each of these methods of installation in turn.
Installation with system packages (Ubuntu)

The latest release of Salt for Ubuntu is provided in Personal Package Archive (PPA), which is a type of package repository for Ubuntu. The easiest way to access the PPA to install Salt is using the add-apt-repository command, as follows:

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

If the `add-apt-repository` command is not found, you can add it by installing the `python-software-properties` package:

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

If you are using Ubuntu Version 12.10 or greater, this step should not be required as the `add-apt-repository` command should be included in the base system.

After you have added the repository, you must update the package management database, as follows:

```
# sudo apt-get update
```

If the system asks whether you should accept a **gpg** key, press **Enter** to accept.

You should then be able to install the Salt master and the Salt minion with the following command:

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

Assuming there are no errors after running this command, you should be done! Salt is now installed on your machine.

Note that we installed both the Salt master and the Salt minion. The term master refers to the central server—the server from which we will be controlling all of our other servers. The term minion refers to the servers connected to and controlled by a master.
Installing with Salt-Bootstrap

Information about manual installation on other major Linux distributions can be found online, at http://docs.saltstack.com. However, in most cases, it is easier and more straightforward to use a tool called Salt-Bootstrap. In-depth documentation can be found on the project page at https://github.com/saltstack/salt-bootstrap—however, the tool is actually quite easy to use, as follows:

```bash
# curl -L https://bootstrap.saltstack.com -o install_salt.sh
# sudo sh install_salt.sh --h
```

We won’t include the help text for Bootstrap here as it would take up too much space. However, it should be noted that, by default, Bootstrap will install only the Salt minion. We want both the Salt minion and the Salt master, which can be accomplished by passing in the `-M` flag, as follows:

```bash
# sudo sh install_salt.sh -M
```

The preceding command will result in a fully-functional installation of Salt on your machine! The supported operating system list is extensive, as follows:

- Amazon Linux AMI 2012.09
- Arch Linux
- CentOS 5/6
- Debian 6.x/7.x/8 (git installations only)
- Fedora 17/18
- FreeBSD 9.1/9.2/10
- Gentoo Linux
- Linaro
- Linux Mint 13/14
- OpenSUSE 12.x
- Oracle Linux 5/6
- RHEL 5/6
- Scientific Linux 5/6
- SmartOS
- SuSE 11 SP1 and 11 SP2
- Ubuntu 10.x/11.x/12.x/13.x/14.x
The version of Salt used for the examples in this book is the 2014.7 release. Here is the full version information:

```
# sudo salt --versions-report
Salt: 2014.7.0
Python: 2.7.6
Jinja2: 2.7.2
M2Crypto: 0.21.1
msgpack-python: 0.3.0
msgpack-pure: Not Installed
pycrypto: 2.6.1
libnacl: Not Installed
PyYAML: 3.10
ioflo: Not Installed
PyZMQ: 14.0.1
RAET: Not Installed
ZMQ: 4.0.4
Mako: 0.9.1
```

It’s probable that the version of Salt you installed is a newer release and might have slightly different output. However, the examples should still all work in the latest version of Salt.

### Configuring Salt

Now that we have the master and the minion installed on our machine, we must do a couple of pieces of configuration in order to allow them to talk to each other.

#### Firewall configuration

Since Salt minions connect to masters, the only firewall configuration that must be done is on the master. By default, ports 4505 and 4506 must be able to accept incoming connections on the master. The default install of Ubuntu 14.04, used for these examples, actually requires no firewall configuration out-of-the-box to be able to run Salt; the ports required are already open. However, many distributions of Linux come with much more restrictive default firewall settings. The most common firewall software in use by default is **iptables**.
Note that you might also have to change firewall settings on your network hardware if there is network filtering in place outside the software on the machine on which you're working.

Firewall configuration is a topic that deserves its own book. However, our needs for the configuration of Salt are fairly simple. First, you must find the set of rules currently in effect for your system. This varies from system to system; for example, the file is located in /etc/sysconfig/iptables on RedHat distributions, while it is located in /etc/iptables/iptables.rules in Arch Linux.

Once you find that file, add the following lines to that file, but be sure to do it above the line that says DROP:

```
-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
```

For more information about configuring on your operating system of choice so that your Salt minion can connect successfully to your Salt master, see the Salt documentation at http://docs.saltstack.com/en/latest/topics/tutorials/firewall.html.

In version 2014.7.0, a new experimental transport option was introduced in Salt, called RAET. The use of this transport system is beyond the scope of this book. This book will deal exclusively with the default, ZeroMQ-based transport in Salt.

**Salt minion configuration**

Out of the box, the Salt minion is configured to connect to a master at the location salt. The reason for this default is that, if DNS is configured correctly such that salt resolves to the master's IP address, no further configuration is needed. The minion will connect successfully to the master.

However, in our example, we do not have any DNS configuration in place, so we must configure this ourselves.

The minion and master configuration files are located in the /etc/salt/ directory.

The /etc/salt/ directory should be created as part of the installation of Salt, assuming you followed the preceding directions. If it does not exist for some reason, please create the directory, and create two files, minion and master, within the directory.
Open /etc/salt/minion with your text editor of choice (remember to use sudo!). We will be making a couple of changes to this file.

First, find the commented-out line for the configuration option master. It should look like this:

```plaintext
#master: salt
```

Uncomment that line and change salt to localhost (as we have this minion connected to the local master). It should look like this:

```plaintext
master: localhost
```

If you cannot find the appropriate line in the file, just add the line shown previously to the top of the file.

You should also manually configure the minion ID so that you can more easily follow along with the examples in this text. Find the ID line:

```plaintext
#id:
```

Uncomment it and set it to myminion:

```plaintext
id: myminion
```

Again, if you cannot find the appropriate line in the file, just add the line shown previously to the top of the file.

Save and close the file.

Without a manually-specified minion ID, the minion will try to intelligently guess what its minion ID should be at startup. For most systems, this will mean the minion ID will be set to the Fully-Qualified Domain Name (FQDN) for the system.

**Starting the Salt master and Salt minion**

Now we need to start (or restart) our Salt master and Salt minion. Assuming you're following along on Ubuntu (which I recommend), you can use the following commands:

```plaintext
# sudo service salt-minion restart
# sudo service salt-master restart
```
Packages in other supported distributions ship with init scripts for Salt. Use whichever service system is available to you to start or restart the Salt minion and Salt master.

**Accepting the minion key on the master**

There is one last step remaining before we can run our first Salt commands. We must tell the master that it can trust the minion. To help us with this, Salt comes with the `salt-key` command to help us manage minion keys:

```bash
# sudo salt-key
Accepted Keys:
Unaccepted Keys:
myminion
Rejected Keys:
```

Notice that our minion, `myminion`, is listed in the *Unaccepted Keys* section. This means that the minion has contacted the master and the master has cached that minion's public key, and is waiting for further instructions as to whether to accept the minion or not.

If your minion is not showing up in the output of `salt-key`, it's possible that the minion cannot reach the master on ports 4505 and 4506. Please refer to the Firewall section described previously for more information.

Troubleshooting information can also be found in the Salt documentation at [http://docs.saltstack.com/en/latest/topics/troubleshooting/](http://docs.saltstack.com/en/latest/topics/troubleshooting/).

We can inspect the key's fingerprint to ensure that it matches our minion's key, as follows:

```bash
# sudo salt-key -f myminion
Unaccepted Keys:
```

We can use the `salt-call` command to run a command on the minion to obtain the minion's key, as follows:

```bash
# sudo salt-call --local key.finger
```
Since the fingerprints match, we can accept the key on the master, as follows:

```
# sudo salt-key -a myminion
The following keys are going to be accepted:
Unaccepted Keys:
myminion
Proceed? [n/Y] Y
Key for minion myminion accepted.
```

We can check that the minion key was accepted, as follows:

```
# sudo salt-key
Accepted Keys:
myminion
Unaccepted Keys:
Rejected Keys:
Success! We are ready to run our first Salt command!
```

### A game of ping pong

Here's our first command:

```
# sudo salt '*' test.ping
myminion:
  True
```

Was that a bit underwhelming?

Don't worry. We're going to get to the more impressive stuff soon enough. The command we just ran was called a remote execution command. Basically, we sent a message to all (one) of our minions and told them to run a function from one of the execution modules that is built into Salt. In this case, we just told our minion to return `True`. It's a good way to check which of our minions are alive. We will explore the various parts of this command in more detail in the next chapter.

The `test` module actually has a few other useful functions. To find out about them, we're actually going to use another module, called `sys`, as follows:

```
# sudo salt 'myminion' sys.list_functions test
myminion:
    - test.arg
    - test.arg_repr
```
Diving In – Our First Salt Commands

- test.arg_type
- test.collatz
- test.conf_test
- test.cross_test
- test.echo
- test.exception
- test.fib
- test.get_opts
- test.kwarg
- test.not_loaded
- test.opts_pkg
- test.outputter
- test.ping
- test.provider
- test.providers
- test.rand_sleep
- test.rand_str
- test.retcode
- test.sleep
- test.stack
- test.tty
- test.version
- test.versions_information
- test.versions_report

Let's try one of the other functions on the list. Maybe test.fib:

```
# sudo salt '*' test.fib
myminion:

    TypeError encountered executing test.fib: fib() takes exactly 1 argument (0 given). See debug log for more info. Possibly a missing arguments issue:  ArgSpec(args=['num'], varargs=None, keywords=None, defaults=None)
```
Well, that didn't work. To find out more information about a function, including examples of how to use it, we can use the `sys.doc` function, as follows:

```bash
# sudo salt '*' sys.doc test.fib
test.fib:

    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
```

Aha! We need to give it a number to which it should calculate the Fibonacci sequence, as follows:

```bash
# sudo salt '*' test.fib 30

myminion:
  __-
    - 0
    - 1
    - 1
    - 2
    - 3
    - 5
    - 8
    - 13
    - 21
    - 1.09672546387e-05
```

As it turns out, the Fibonacci sequence is not very hard for computers to calculate quickly.

Note that you can actually use `sys.doc` to retrieve the documentation for a whole module's worth of functions at a time, as follows:

```bash
# sudo salt '*' sys.doc test
```

I didn't include the output as it is lengthy.
The `sys` module is going to be one of the most useful modules in your quest to learn Salt. Keep it handy and turn to it any time you want to learn more about something you're working with. Remember that the `sys` module can target itself. The following code shows you how to use the `sys` module:

```bash
# sudo salt '*' sys.list_functions sys

myminion:
  - sys.argspec
  - sys.doc
  - sys.list_functions
  - sys.list_modules
  - sys.list_returner_functions
  - sys.list_returners
  - sys.list_runner_functions
  - sys.list_runners
  - sys.list_state_functions
  - sys.list_state_modules
  - sys.reload_modules
  - sys.returner_doc
  - sys.runner_doc
  - sys.state_doc
```

We are going to discuss remote execution and the execution modules in much greater detail in the next chapter.

**Masterless Salt**

In this chapter, we’ve taken the time to set up Salt in a master-minion relationship. This will allow us to take advantage of all the power of Salt and scale to multiple minions easily later on. However, Salt is also designed such that a minion can run without a master.

We'll run through a few examples of how to run commands on a minion. This will also be useful even when we do have a master because, if we're logged into a minion for some reason and want to run a command while we're there, we can do so using these same concepts.
To start, we'll leave our master running. The command for running commands on the minion is `salt-call`, and it can take any of the same execution module functions that we used with the `salt` command, as follows:

```bash
# sudo salt-call test.ping
local:
   True
```

Note that it doesn't display our minion's ID because we're just running it locally:

```bash
# sudo salt-call test.fib 10
local:
    - 0
    - 1
    - 1
    - 2
    - 3
    - 5
    - 8
    - 5.00679016113e-06
```

```bash
# sudo salt-call sys.doc test.ping
local:
---------
test.ping:
   Used to make sure the minion is up and responding. Not an ICMP ping.
   Returns `"True"`.
   CLI Example:

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

Now let's stop our master and try again:

```bash
# sudo service salt-master stop
# sudo salt-call test.ping
Failed sign in
Diving In – Our First Salt Commands

The example shown previously will take a fairly long time to terminate. Basically, `salt-call` is trying to establish a connection with the master just in case it needs to copy files from the master or other similar operations.

To get `salt-call` to operate properly masterless, we need to tell it there's no master. We do this with the `--local` flag, as follows:

```
# sudo salt-call --local test.ping
local: True
```

Success! You can now operate a Salt minion without a master!

Start your master again before moving on to the next chapter of this book:
```
# salt-master --daemon --log-level debug
```

Summary

We've covered a lot of ground in this chapter. We've installed the Salt minion and Salt master on our machines and configured them to talk to each other, including accepting the minion's key on the master. We've also run our first Salt commands, both from the master and from the minion without involving the master.

However, we've only just begun! In the next chapter, we're going to go much more in-depth into the topic of remote execution and show how powerful this tool is.
Where to buy this book

You can buy Learning SaltStack from the Packt Publishing website.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

Click here for ordering and shipping details.