Saltstack For DevOps

Extremely fast and simple IT automation and configuration management

Aymen El Amri

Saltstack For DevOps

Extremely fast and simple IT automation and configuration management

Aymen El Amri

This book is for sale at http://leanpub.com/saltstackfordevops

This version was published on 2018-04-29

Tweet This Book!

Please help Aymen El Amri by spreading the word about this book on Twitter!

The suggested tweet for this book is:

I just bought SaltStack For DevOps book. Check it out : https://leanpub.com/saltstackfordevops cc @Salt4DevOps

The suggested hashtag for this book is #Salt4DevOps.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

#Salt4DevOps

Also By Aymen El Amri

The Jumpstart Up

Painless Docker

Contents

Dedication

I dedicate this to my Mother and my Father who were always there for me.

Preface

Every Book Has A Story, This Story Has A Book

I wanted to quickly resign from my job, my suggestions about working on a continuous delivery and setting up a deployment pipeline have not been considered by most of my colleagues! Like many of us, I'm lazy when it comes to repeating tasks manually but passionate when I automate them.

I love automation and in my job, there are hundreds of configuration files and thousands of variables to copy from text files then to adjust in some platforms, a huge number of poorly-configured servers and hundreds of servers to manage by a very small team.. I wanted to work on the automation of some weekly procedures. I was aware that this is a good solution but it was not the priority neither for the manager nor for the client.

It was a position within a team of 14 people working on the integration of a number of heavy applications (mainly Java/Oracle, php/Mysql, Nginx, Python/Jython) with a complex architecture, tens of versions/environments to manage and an infrastructure covering Europe.

In the beginning, I was obliged to follow my boss guidelines and the work methods my team has adopted which have one goal: satisfying as fast as possible the unceasing demands of the client.

No, but .. wait, this is not good at all!

I spent almost two weeks searching and working on some solutions before I convinced my boss to give me the time to set up an application prototype that will ease the heavy load, accelerate daily procedures and reduce human errors.

First, I created a configuration management tool using Python/Sqlite3, automated tests using Selenium/Python among other procedures I have set up.

Some weeks later, I started learning SaltStack and found it later a good solution to replace my "home-made" applications. It meets the expectations of the integration process, or rather, the continuous integration, deployments and automatic tests.

I hesitated between several alternatives: Puppet, CFEngine, Chef .. etc. The choice was made based on several criteria (I was looking for a robust, fast configuration management and remote execution tool that everybody can use without learning a new programming language (YAML)) .. I have never regretted my choice.

I found some difficulties and honestly when I started learning Salt, the official documentation was not very detailed.

This book is a fruit of long hours of work and self-learning.

Well, in the beginning, I wanted to resign from my job, just a few days after discovering of Salt, I was in love with my work, with what I was doing and with what I am learning.

I tried Salt first time when I saw my team taking more than 3 days (sometimes more) to configure hosted platforms at each deployment (we had more than 10 environments per application).

After setting it up, the same procedure was taking less than 1/2 hour.

Through this book, it's your turn to discover SaltStack, I will be your guide.

I wish you a pleasant reading.

To Whom Is This Book Addressed?

To developers, system administrators and anyone working in one of these teams in collaboration with the other or simply in an environment that requires knowledge in development, software integration, and system engineering.

Usually, developers think that they are here to serve the machine, or to feed it each period of time with a fresh code to deploy.

The machine is hungry and should eat, otherwise, it will stop working and the boss will be angry.

System administrators think that machines should be happy.

Feeding the machine will make it sick and angry. Especially with obsolete codes. It will stop working and the boss will be angry too.

This is an ironical way to describe the conflicts between developers and system administrators but in many cases, it's true.

Moreover, within the same company there is generally some tension between the two teams:

System administrators accuse developers to write code that consumes memory, does not meet system security standards or not adapted to available machines configuration.

Developers accuse system administrators to be lazy, to lack innovation and to be seriously uncool!

No more mutual accusations, now with the evolution of software development, infrastructure and adopted methodologies (such as Scrum, XP, Kanban), the rise of DevOps culture is the result.

DevOps is more a philosophy and a culture than a job (even if some of the positions I had were called "DevOps Engineer" or "DevOps Architect").

By admitting this, this job seeks closer collaboration and a combination of different roles involved in software development such as the role of developer, responsible for operations and responsible for quality assurance.

The software must be produced at a frenetic pace while at the same time the developing in cascade seems to have reached its limits.

- If you are a fan of configuration management, automation, and the DevOps culture
- If you are a system administrator working on DevOps, SysOps, CloudOps, ResearchOps .. well, Ops in general
- If you are a developer seeking to join the new movement

This book is addressed to you.

Configuration management software are one the most used tools in DevOps environments.

If you are new to configuration management software, this book is also addressed to beginners.

Conventions Used In This Book

Basically, this is a technical book where you will find commands (SaltStack commands) and code (Python, YAML, Jinja2 ..etc).

Commands and code are written in a different format.

Example :

```
python -c 'import urllib; print urllib.urlopen("https://bootstrap.SaltStack.com"\
).read()' | \ sudo  sh -s -- git develop
```

- This book uses *italic* font for technical words such as libraries, modules, languages names. The goal is to get your attention when you are reading and help you identify them.
- You will find two icons, I have tried to be as simple as possible so I have chosen not to use too many symbols, you will only find:

 To highlight useful and important information.

 To highlight a warning or to prevent.

How To Properly Enjoy This Book

This book contains technical explanations and shows in each case an example of a command or a configuration to follow.

The explanation gives you a general idea and the code that follows gives you convenience and help you to practice what you are reading.

Preferably, you should always look both parts for a maximum of understanding.

Like any new tool or programming language you learned, it is normal to find difficulties and confusions in the beginning, perhaps even after.

If you are not used to learning new technologies, you can even have a modest understanding while being in an advanced stage of this book. Do not worry, everyone has passed at least once by this kind of situations.

At the beginning try to make a diagonal reading while focusing on the basic concepts, then try the first practical manipulation on your server or just using your laptop/desktop and occasionally come back to this book for further reading about a specific subject or concept.

This book is neither an encyclopedia nor a documentation but it sets out the most important parts to learn and master, if you find words or concepts that you are not comfortable with, take your time and do your own online research.

Learning can be serial so understanding a topic requires the understanding of another one.

Through this course, you will learn how to install configure and use SaltStack.

Just before finishing it, you will go through a chapter where good examples of practical use cases are explained.

Through these chapters, try to showcase your acquired understanding, and no, it will not hurt to go back to previous chapters if you are unsure or in doubt.

Finally, try to be pragmatic and have an open mind if you encounter a problem.

The resolution begins by asking the right questions.

How To Contribute To This Book?

This work will be always in progress.

I am an adopter of the lean philosophy so the content will be continuously improved in function of many criterions but the most important one is your feedback.

If you have any suggestions please do not hesitate to contact me, you can find me on Twitter[1].

This book is not perfect, so you can find a typo here, some punctuation errors there or missing words.

Every line of the used code was tested before but you may find some errors during your learning path due to a difference of software versions, environment configurations or similar issues.

[1] https://twitter.com/eon01

About The Author

Aymen is a Cloud Architect, Entrepreneur, Author of Best Selling Trainings and Books, Awards Winner, CEO of Eralabs[2] (A DevOps & Cloud Consulting & Training Company) and Founder of DevOpsLinks Community[3].

He actually lives in Paris and helps companies and startups from everywhere (Europe, US ..) develop modern applications, builds multi-tenant cloud infrastructures, scalable applications, highly stable production environments, distributed systems and service-oriented architectures (microservices & PaaS).

You can find Aymen on Twitter[4] and join his newsletters DevOpsLinks[5] and Shipped[6].

For more online training and courses, please visit eralabs.io[7].

[2] http://eralabs.io
[3] http://devopslinks.com
[4] https://twitter.com/eon01
[5] http://devopslinks.com
[6] http://shipped.devopslinks.com
[7] http://eralabs.io

Introduction

```
    \   ^__^
     \  (--)\_______
        (__)\       )\/\
            ||----w |
            ||     ||
```

Configuration Management And Data Center Automation

Configuration management and data center automation are mechanisms to manage a technical description of a system, its components and changes in its configurations.

This happens during the life cycle of a system or during its different processes. Source code deployment on one or multiple environments is a change and it requires the right configuration management and provisioning tool.

An ecosystem composed of development, test, QA and production environments is an example of a slightly complex ecosystem that can be composed many servers, sometimes thousands or hundreds of servers.

From the development, integration, test, and staging, to the production environment, configuration management becomes a process of normalization of the application configuration according to the state of infrastructure and functional components and other requirements. The same process of normalization should be ensured when changing the environment, the infrastructure or when the configuration itself changes.

Imagine that your organization is developing some applications with dependencies between them: databases, different configurations between developer's, staging and production environments..etc

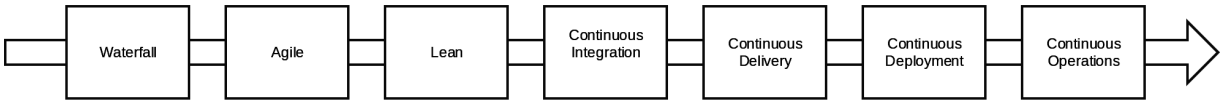
To accomplish this project with a good time-to-market, you may need tools to:

- A tool to rebuild the same server quickly from provisioned images
- Generate dynamic configurations for the same application in different environments
- Execute commands on thousands or hundreds of servers from one central machine
- Create advanced monitoring and reactors systems

- Recover quickly from disasters
- etc.

Configuration management tools like Salt are useful to the use cases quoted below and can do more than this.

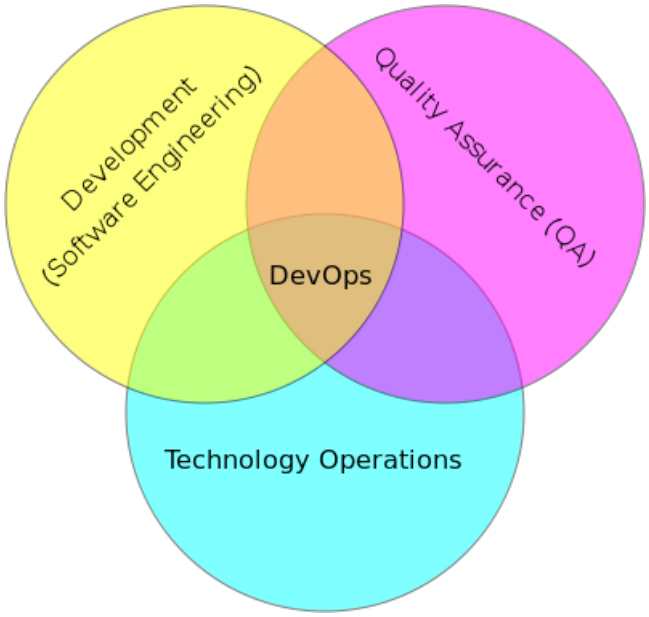
In fact, with the adoption of agile development methods, the process of development; test and deployment of a software component has accelerated, therefore methods of management have become faster, more automated and more adapted to changes.



DevOps Evolution

Even if many specialists consider provisioning, change management and automation a business matter, not an IT one but to make this happen, some special technical skills are required. That's why new positions, teams or rather culture in the IT industry have emerged: DevOps.

The illustration below (taken from Wikipedia) shows the essence of the DevOps philosophy.



DevOps as the intersection of development (software engineering), technology operations and quality assurance (QA)

Automation is important to the success of critical IT processes that are part of the life cycle of a product, including provisioning, change management, release management, patch management, compliance, and security. Therefore, having the right technical skills is important to any "lazy but pragmatic SysAdmin".

This book will help you to learn one of the most known IT automation configuration management and infrastructure automation/orchestration tools.

DevOps Tooling

Currently, several FOSS and proprietary automation and configuration management tools exist. Choosing one of these tools is based on several criteria.

Choice Criteria

- **Performance** : Among memory consumption, the speed of execution and adaptation to increasingly complexes architectures, several performance criteria could help you decide to choose the right tool.
- **License** : You may choose between FOSS and proprietary software. Most of the existing software are Open Source. It remains to be seen what FOSS license you should choose: GPL, BSD, Apache, MIT..etc
- **Programming Language** : A such tool is coded using a programming language, but it does not mean that users will manage and automate operations and servers using the same language. For example, SaltStack is written in *Python* but its users use *Jinja* and *YAML* ..etc

Most of the tools are written in *Python*, *Ruby*, *Java* or *Golang*, but one can also find *perl*, *C* and *C ++* based tools.

- **Authentication Methods** : A configuration management or a data center automation tool is based on a model, roughly consisting of clients and a server. The authentication between a client and a server can be automatic, encrypted, secure, fast .. or not.
- **Agents** : Some tools use agents that must be installed on target servers (clients), some tools do not require remote agents and others offer both choices.
- **Scalability** : A tool that grows and evolves with the enterprise must provide technical means and capabilities to ensure scalability at several features and extended functional scopes.
- **Portability** : Most if not all popular configuration management tools are compatible with *nix systems. Some servers run on *BSD*, *AIX*, H*P-UX*, *Mac OS*, *Solaris*, *Windows* and other OSs. In this case, you must study compatibility.

AIX	BSD	HP-UX	Linux	Mac OS X	Solaris	Windows	Others	
Ansible	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Bcfg	Partial	Yes	No	Yes	Partial	Yes	No	No
CFEngine	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
cdist		Yes		Yes	Yes		No	
Chef	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
ISconf	Yes	Yes	Yes	Yes	Yes	Yes	No	No
Juju				Yes				
LCFG	No	No	No	Partial	Partial	Partial	No	No
OCS Inventory NG	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No
Opsi	No	No	No	Yes	No	No	Yes	No
PIKT	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes
Puppet	Yes	Yes	Yes	Yes	Partial	Yes	Yes	Yes
Quattor	No	No	No	Yes	Partial	Yes	No	No
Radmind	Yes	Yes	No	Yes	Yes	Yes	Yes	No
Rex		Yes		Yes	Yes	Yes	Yes	No
Rudder	Yes	Partial	No	Yes	Partial	Partial	Yes	Yes
Rundeck	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No
SmartFrog	No	No	Yes	Yes	Yes	Yes	Yes	No
Salt	Yes	Yes	Partial	Yes	Yes	Yes	Yes	Partial
Spacewalk	No	No	No	Yes	No	Yes	No	No
STAF	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Synctool	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Vagrant				Yes	Yes		Yes	

Thanks to Wikipedia[8] and its contributors for this comparison concerning the portability of the following tools.

- **Documentation, Support, and Latest Stable Release**: Keep in mind that the quantity and the quality of the official documentation, forums, groups, and paid support differs from a tool to another. A good thing to do is to see the date of the latest stable release, some tools are no more updated which can cause security risks, lack of support, bugs, and problems of integrations with other tools.

Popular tools

Among the popular tools we can find : *Ansible*, *CFEngine*, *Puppet*/*Chef* and SaltStack.

[8] http://en.wikipedia.org/wiki/Comparison_of_open-source_configuration_management_software

Illustration 2: Popular configuration management and automation tools

Ansible : Combines multi-node deployment and ad-hoc task execution. It manages nodes with SSH and requires *Python* (2.4 or later). It uses *JSON* and *YAML* modules and states as descriptions. It is built on *Python* but its modules can be written in any language. Ansible[9] is used by Spotify, Twitter, NASA, and Evernote.

Puppet : Puppet[10] is based on a custom declarative language to describe the system configuration, it uses a distributed client-server paradigm and a library for configuration tasks. *Puppet* requires the installation of a master server and client agents on every system that needs to be managed. It is used by Vmware, Cisco, Paypal, and SalesForce.

SaltStack : Salt[11] is what the next chapters of this book will detail. It is used by Rackspace, Photobucket, Nasa, LinkedIn, Hulu, HP Cloud Services, Harvard University, CloudFlare ..etc

[9] http://www.ansible.com/home
[10] http://puppetlabs.com/
[11] http://saltstack.com/

Getting Started

```
    \   ^__^
     \  (--)\_______
        (__)\       )\/\
            ||----w |
            ||     ||
```

Presentation

Salt is an Open Source project, you can read and modify its source code under the *Apache license*. Its source code is available on github[12].

SALSTACK Inc. is the company behind Salt, it was founded by *Thomas Hatch*, the original creator of SaltStack.

Salt is used by Apple Inc, Rackspace, Photobucket, NASA, LinkedIn, Hulu, HP Cloud Services, Cloud Flare and other know companies.

SaltStack Logo

Salt fundamentally improves the way SysAdmins, integrators, and DevOps configure and manage all aspects of a modern data center infrastructure.

[12] https://github.com/saltstack/salt

It provides a different approach to some existing alternatives such as speed and adaptation to the size of a cloud. Several recognized businesses use SaltStack to orchestrate and control their cloud servers and infrastructure and automate the “DevOps Toolchain”.

It is built on a platform running relatively fast while enabling remote-controlling distributed infrastructures, code and data. A layer of security is established while having two-way communication between the different components of the platform (masters, minions ..etc).

The following chapters are conceived for beginners and experienced system administrators, DevOps professionals and developers seeking to manage and configure multiple servers/applications and software platforms easily.

The infrastructure to manage can be on-premise virtual machines, cloud (Amazon EC2 instances, Rackspace ..etc), containers (e.g. Docker) or bare-metal machines as well as hosted applications and platforms that rely on configuration files.

All you need is a root access, a good understanding of the environment to manage and some basic knowledge (command line, basic commands, Linux ..etc).

Even if it is possible to use a web access to manage Salt but the use of the command line is always more adapted to our needs for several reasons such as speed and efficiency. If you are familiar with CLIs, understanding Salt commands and its syntax will be easier.

Salt is portable and works with these 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

According to the official website, other systems and distributions will be compatible in the future. If you want to stay informed just follow the development branches[13].

In the following sections, we will be most of the time using *Linux*. If you are using *Macos*, there is no real differences for the installation but if you are using *Windows*, I recommend using *Ubuntu* on *Windows*, this is optional if you would like to keep using *Windows* shell.

Windows users, please follow this guide[14] to install *Ubuntu* and *Bash*.

You can find some differences in the installation of Salt according to your system: *Windows*, *FreeBSD* or *Solaris*..etc. Overall, principles and usage are the same.

You can use Salt installed on an operating system to manage other systems (A *Linux* to manage a *Solaris* or a *BSD* to manage a *Windows* ... etc.).

The installation part of this book will cover *Redhat* and *Debian*.

Be sure to check the documentation (docs.saltstack.com) for the installation and the specific use of your particular operating system.

A brief summary

SaltStack is based on some special components:

- One or more *salt-master*, *salt-minion* and *salt-syndic*
- A key management system *salt-key* that allows the authentication of a *salt-minion* on a *salt-master*
- A system of *states* to describe configurations
- A *top.sls* that calls *states*
- A system of *grain* on the minion to manage configurations data
- A system of *pillars* to store other data on the master (such as confidential data)
- A transport and data management system called *ZeroMQ*
- An event management system called *reactors*
- Other components like *returners*, *outputters* ..etc

A *master* can manage configurations or execute remote commands on one or more *minions*. These operations are based on *SLS* files, and these files are calling Salt modules, *grains* and/or *pillars*.

Salt could be used either from the command line (Salt CLI) or in executable scripts (Salt API).

The various components of SaltStack will be explained in this book, some definitions appeal others, that's why we need - in the first order - to have a global view about Salt.

[13] https://github.com/saltstack/salt/branches/all

[14] https://www.howtogeek.com/249966/how-to-install-and-use-the-linux-bash-shell-on-windows-10/

Expanding Salt Use

Some provisioning and testing tools are based on Salt, you may find some or all of them interesting. SaltStack can be interfaced with different cloud providers, integrations tools, and containers .. etc.

Some of the possible integrations are :

Salt + Vagrant

Salty Vagrant is a Vagrant plugin that allows you to use Salt as a provisioning tool. You can use *formulas* and existing configurations to create and build development environments.

The simplest way to use *Salty Vagrant* is configuring it to work in *masterless* mode. Details are explained in the official Vagrant documentation[15].

Through this book, you will learn how to interface *Vagrant* with Salt in order to automate the provisioning of virtual machines.

Salt Cloud

Salt Cloud[16] is a public cloud provisioning tool created to integrate Salt to the major cloud infrastructure providers (AWS, Rackspace, GCP, Azure ..etc) in order to facilitate and accelerate the supply process.

Salt Cloud allows managing a cloud infrastructure based on *maps* and *profiles* of virtual machines. This means that many virtual machines in the cloud can be managed easier and faster.

We will see in this book how to integrate Salt with some Cloud providers (e.g. Linode, AWS)

Halite

Halite[17] is the client-side web interface (Salt GUI). It connects and operates a SaltStack infrastructure. This tool is a graphical complement, but it is not indispensable for the functioning of Salt. For best results, *Halite* works with *Hydrogen* and higher versions.

Â Salt + Docker

Salt can be used to provision and manage *Docker* containers (*Dockerng* / *Dockerio*). We are going to see together how to configure and manage *Docker* containers using Salt.

[15] https://docs.vagrantup.com/v2/provisioning/salt.html
[16] https://github.com/saltstack/salt-cloud
[17] https://github.com/saltstack/halite

Conclusion

The general presentation of Salt is not enough to start using Salt, but it is required if you are not familiar with concepts like configuration management and data center automation.

Installation

```
    \   ^__^
     \  (--)\_______
        (__)\       )\/\
            ||----w |
            ||     ||
```

Introduction

Installation is a little bit different from one OS to another: *Arch Linux*, *Debian*, *Fedora*, *FreeBSD*, *Gentoo*, *OS X*, *RHEL* / *CentOS* / *Scientific Linux* / *Amazon Linux* / *Oracle Linux*, *Solaris*, *Ubuntu*, *Windows* and *SUSE* ...etc.

Salt is made of *masters* and *slaves* typically *Masters* and *Minions*. In the jargon used by the editor, *salt-master* is the *Master* and *salt-minion* is the *Minion*. A syndic called *salt-syndic* is part of the package and its role consists of connecting *Masters*.

Dependencies

Salt has external dependencies:

- *Python* 2.6 >= 2.6 <3.0
- *msgpack-python* - An efficient message exchange format
- *YAML* - Python *YAML*
- *Jinja2* - *Jinja2* a template engine.
- *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
- Requests - *HTTP* library

Depending to the choosed transport method (*ZeroMQ* or *RAET*), dependencies may 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* - UDP protocol

There are other optional dependencies such as:

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

If you will use *salt-cloud* for cloud infrastructure management or *dockerng*/*dockerio* with *Docker* machines, you may install other packages and you will have other dependencies, this is explained later in the book.

By default, Salt uses *ZeroMQ*. Its official website[18] defines it as :

ZeroMQ \zero-em-queue\, \Ã˜MQ: - Connect your code in any language, on any platform. - Carries messages across *inproc*, *IPC*, *TCP*, *TPIC*, *multicast*. - Smart patterns like *pub-sub*, *push-pull*, and *router-dealer*. - High-speed asynchronous *I/O* engines, in a tiny library. - Backed by a large and active open source community. - Supports every modern language and platform. - Build any architecture: centralized, distributed, small, or large. - Free software with full commercial support.

Installation

If your server has Internet access, this shell script will be convenient to automate the installation of the stable version:

```
1 wget -O - http://bootstrap.saltstack.org | sudo sh
```

You can also install the latest version from Github :

```
1 curl -L https://bootstrap.saltstack.com -o install_salt.sh
2 sudo sh install_salt.sh git develop
```

or:

[18] http://zeromq.org/

```
python -m urllib "https://bootstrap.saltstack.com" | sudo sh -s -- git develop
```

 Note that the last version downloaded directly from Github is the development version. If you need a version for your production systems, do not do this.

You can also try a Pythonic way:

```
python -c 'import urllib; print urllib.urlopen("https://bootstrap.saltstack.com"\
).read()' | \
sudo  sh -s -- git develop
```

 The installation script is provided by the official documentation of Salt.

Advanced Installation

The saltbootstrap script[19] offers plenty of command line options.

The installation script can be called as follows:

```
sh bootstrap-salt.sh â€“h
```

It allows 3 types of installations:

- stable (default)
- daily (specific to Ubuntu)
- git

Examples:

[19] http://docs.saltstack.com/en/latest/topics/tutorials/salt_bootstrap.html

```
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
```

Script Options:

```
  -h  #Display this message
  -v  #Display script version
  -n  #No colors.
  -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 a\
ny installation.
  -P  #Allow 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 w\
ith -c and -k.
  -I  #If set, allow insecure connections while downloading any files. For examp\
le, pass '--no-check-certificate' to 'wget' or '--insecure' to 'curl'
```

Installation For Test

If you want to install Salt to test its various features, two possibilities are open to you.

Masterless Salt

Installing a *masterless salt* is about installing a *salt-minion* without a *salt-master*. Functionally, it makes no difference if your goal is to test.

This is a great way to quickly test the basics of SaltStack. The execution of Salt different commands is done using *salt-call* that runs on the minion and displays the execution traces. More details about *salt-call* will be addressed later.

The minion contains extended features to enable it to run in the *Standalone Mode*. An autonomous minion can be used to do a certain number of things:

- Creating a master server (*salt-master*) using *states* (salting a *salt-master*)
- Using *salt-call* commands in a system without connectivity to a master system (*salt-master*)
- Applying *masterless states*: states executed entirely from local configuration files


If it is your first time to use and learn Salt, you may find some new terminologies that you will not understand for now but they will be explained later.

Testing SaltStack In Your Localhost

Another simple alternative is to install a Master and a Minion in the same machine - your local machine:

e.g:

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

```
1 apt-get install salt-master salt-minion
```

The idea of running both on the same machine is very useful for quickly testing all functionalities of Salt while keeping the master/slave or *master/minion* model.

Practical Installation Case

Other installation methods are detailed in the official website[20]. We are going to detail the installation in *RHEL* (*Redhat Enterprise Linux*) and its derivatives OSs (*CentOS*/*Scientific Linux*/*Amazon Linux*/*Oracle Linux*) as well as *Debian*.

Using PIP

Since SaltStack is available in *PyPi* repository, it can be installed using *pip* (*Python* Package Management System):

[20] http://salt.readthedocs.org/en/latest/topics/installation/index.html

```
pip install salt
```

This will not work unless *pip* is installed. Assuming that your OS package manager is yum:

```
yum -y install python-pip
```

Once *pip* is installed, you can create an alias *pip* to *pip-python* if needed.

```
$ echo 'alias pip="/usr/bin/pip-python"' >> $HOME/.bashrc
$ . $HOME/.bashrc
```

Redhat: Using RHEL EPEL

Salt is available on *EPEL* (*The Extra Packages for Enterprise Linux*) since its version 0.9.4 and thus can be installed using *yum*. Above all, you must enable *EPEL* on *RHEL*.

For *RHEL 5*:

```
rpm -Uvh http://mirror.pnl.gov/epel/5/i386/epel-release-5-4.noarch.rpm
```

For *RHEL 6*:

```
rpm -Uvh http://ftp.linux.ncsu.edu/pub/epel/6/i386/epel-release-6-8.noarch.rpm
```

Then, install *salt-master* and *salt-minion* on their respective machines (or both of them on your *localhost*).

On the *master* server :

```
yum install salt-master
```

On the *minion* server:

```
yum install salt-minion
```

Further details about *RHEL* installation are available in the official documentation[21].

If you want your master to start-up with the *OS*:

[21] http://salt.readthedocs.org/en/latest/topics/installation/rhel.html

```
chkconfig salt-master on
```

To start it:

```
service salt-master start
```

Do the same thing for *salt-minion* :

```
chkconfig salt-minion on
service salt-minion start
```

A *master* or a *minion* service is a daemon that can be started respectively by the following commands:

```
salt-master -d
salt-minion -d
```

In the list of system processes, you should find the process running the following command for the *master*:

```
To shutdown the *master* or the *minion*, simply send a termination signal to th\
e *pid* of the process:

e.g:

```bash
kill -9 3139
kill -SIGTERM 3139
```

You can also use the command *pkill* followed by the name of the process :

```
pkill salt-master
pkill salt-minion
pkill salt-syndic
```

You have also the possibility to use services.
```

```
service salt-master stop
# or using systemctl
systemctl stop salt-master
```

Normally, dependencies will be automatically installed and in the opposite case install them manually, reinstall Salt or start a reconfiguration of packages dependencies using the package manager of your operating system.

Using Python Virtual Environment

This method creates a *Python* development environment and its purpose is to install packages and libraries while keeping the same libraries of your operating system untouchable. It is recommended to use *virtualenv* when it comes to installing the development version of SaltStack.

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

Start by installing *virtualenv*:

```
wget http://dl.fedoraproject.org/pub/epel/5/i386/epel-release-5-4.noarch.rpm
rpm -i epel-release-5-4.noarch.rpm
rm epel-release-5-4.noarch.rpm
```

Create the development environment:

```
virtualenv /path/to/your/virtualenv
```

Then turn it on:

```
source /path/to/your/virtualenv/bin/activate
```

Install SaltStack and its dependencies:

```
pip install M2Crypto
pip install pyzmq PyYAML pycrypto msgpack-python jinja2 psutil
pip install -e ./salt #The path to the salt git clone from above
```

Create the needed configuration files for the master and the minion:

```
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
```

This is not our main topic, I invite you to learn more about using *virtualenv* in the online documentation[22].

Debian: Installation Of SaltStack On Debian Squeeze

Start by creating a file under *sources.list.d*:

```
/etc/apt/sources.list.d
```

then add the following lines :

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

You can also add them directly to:

```
/etc/apt/sources.list
```

Import the key used for the signature:

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

and to finish :

```
apt-get update
apt-get install salt-master
apt-get install salt-syndic
apt-get install salt-minion
```

Debian : Installation Of SaltStack On Debian Stretch

Import the right key:

[22] http://docs.saltstack.com/en/latest/topics/development/hacking.html

```
wget -O - https://repo.saltstack.com/apt/debian/9/amd64/latest/SALTSTACK-GPG-KEY\
.pub | sudo apt-key add -
```

Add the repository url to your list:

```
tee -a "deb http://repo.saltstack.com/apt/debian/9/amd64/latest stretch main" /e\
tc/apt/sources.list.d/saltstack.list
```

Run a system update:

```
apt-get update
```

and to finish :

```
apt-get update
apt-get install salt-master
apt-get install salt-syndic
apt-get install salt-minion
```

Upgrading Salt

Upgrading could be different from one package manager to another.


The important point to remember in this part is that the master must be upgraded first, followed by minions.

The compatibility between an upgraded *master* and *minions* having an earlier version is managed by SaltStack, so everything will work perfectly except the case where the update concerns security or vulnerabilities patches on *minions*.

Just use `salt --versions` to have the version of Salt and its dependencies.

e.g :

```
Salt: 2014.7.0
Python: 2.7.3 (default, Mar 14 2014, 11:57:14)
Jinja2: 2.6
M2Crypto: 0.21.1
msgpack-python: 0.1.10
msgpack-pure: Not Installed
pycrypto: 2.6
libnacl: Not Installed
PyYAML: 3.10
ioflo: Not Installed
PyZMQ: 13.1.0
RAET: Not Installed
ZMQ: 3.2.3
Mako: 0.7.0
```

Configuration And Troubleshooting

```
    \   ^__^
     \  (--)\_______
        (__)\       )\/\
            ||----w |
            ||     ||
```

Introduction

In the remainder of this guide, we will use a *Linux* virtual machine where we will install the *salt-master* and the *salt-minion* (in the same machine). Basic concepts are explained here so that there will be practically neither functional nor conceptual differences between having master and minion in the same machine and having them in two different machines.

The only differences that might exist are easily noticeable such as the configuration of IP addresses (interfaces).

For the configuration, we will follow these key steps:

- The configuration of the *salt-master*
- The configuration of the *salt-minion*
- The configuration of *salt-key*
- Functional tests
- Diagnostics and troubleshooting

Configuration files for master and minion are in the default configuration directory:

```
/etc/salt
```

Configuration Of A Masterless Minion

To allow a minion execute Salt commands without a master, the option *file_client* must be configured.

By default, it is set to the remote execution mode so that a minion consults a master before running any command.

file_client shall be set to *local*. In the minion configuration file */etc/salt/minion* find and uncomment this line:

```
file_client: remote
```

Now change it to *local*.

```
file_client: local
```

 You can find more details in the *salt-minion* section of the 4th lesson.

Configuration of salt-master

In the configuration file `/etc/salt/master` the default address for the listening interface (binding) is set to `0.0.0.0`:

```
interface: 0.0.0.0
```

You can adapt it to your specific needs.

e.g: You can change it and restrict it to your localhost (`127.0.0.1` or `0.0.0.0`):

```
interface: 127.0.0.1
```

 Make sure that your system settings are well configured, think of *hosts* file for instance.

In our case we will keep the default value of the interface, which means `0.0.0.0` and defaults ports values:

```
publish_port: 4505
ret_port: 4506
```

publish_port is the network port and *ret_port* is the port of the execution and the feedback of minions.

Same thing for *file_roots*, we are just keeping its default value:

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

The *base* environment is always required and must contain the top file. It is basically the entry point for Salt and will be used if no environment is specified. This will be detailed later.


Keep in mind that it is obligatory to configure your *base* environment.

To deliver files to minions, Salt uses a lightweight file server written in ZeroMQ (or 0mq). This file server is running on the master and requires a dedicated port.

The file server uses environments declared in the master configuration file. Each environment can have multiple root directories:

e.g :

```
file_roots:
  base:
    - /srv/salt/
  dev:
    - /srv/salt/dev/subfolder1/
    - /srv/salt/dev/subfolder2/
  prod:
    - /srv/salt/prod/subfolder1
    - /srv/salt/prod/subfolder2
```

A *base* environment must have a top file *top.sls*.

For the sake of simplicity, even if we need to configure different environments, we will keep a single environment in the master configuration file.

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

Instead, we will focus on how these environments are actually implemented and to do that we will set a *state tree* for each one of them. We are going to have multiple *state trees* designating each one of our environments.

For the time being, restart the master:

```
service salt-master restart
```

Configuration of salt-minion

The address of master machine should be configured in the minion configuration file */etc/salt/minion*.

```
master:  127.0.0.1
```


Keep in mind that the value of *master_port* should coincide with the value of *ret_port* previously configured on the master.

A minion may have a nickname, you are free to change it by uncommenting `id:` and changing its value. By default, the minion id/nickname is the name of the machine (in *Linux*, it is the same name obtained with `hostname` command).

This value is stored in the file:

```
/etc/salt/minion_id
```


Once all settings are made, it is recommended to restart the daemon:

```
service salt-minion restart
```

Post Installation

A very important thing to do after the installation and before the configuration is to let the master accept the minion.

To do this, you should use *salt-key*.

Salt-key is the public key management system server. A master can communicate with a minion only when the public key of the latter is processed and accepted by *salt-key* module.

Salt automatically handles the key generation for running minions, the server will be notified of the presence of a minion via *salt-key*.

A minion can have three different *states*:

- **Accepted **: The public key of the minion is accepted and thus registered on the server.
- **Unaccepted **: The public key of the minion has not been accepted yet.
- **Rejected **: This is usually a problem of installation or configuration.

After setting up the master and the minion, type the following command to list all *minions* and their associated *states*:

```
salt-key  -L
```

or

```
salt-key --list all
```

Accepted minions are ready to receive orders from the master. You can also use the same command followed by the following parameters and the id of the minion:

```
salt-key -a <name_of_the_minion>
```

The last command adds a minion to the list of accepted minions.

You can also use the following parameter if you want to accept all of the configured minions:

```
salt-key -A
```

Or

```
salt-key --accept-all
```

e.g:

To add a minion *mynode* we should type the following command:

```
salt-key -a 'mynode'
```

 *name_of_the_minion* and *mynode* are used as examples. You should change it by the name of the minion you are using. Remember `salt-key -L` can help you list the names of all minions.

General Troubleshooting and Prerequisites Checklist

SaltStack offers many possibilities[23] to diagnose a problem.

We are going to see the most commonly used methods for commons problems.

Before launching the master and the minion(s), a list of prerequisites should be checked:

- All of your *salt-master* and *salt-minion* are installed with the satisfied dependencies
- *Python* version is compatible with the version of SaltStack
- *salt-master* and *salt-minion* should be running with the root user
- Communication and execution ports are well configured and opened
- The configuration of any firewall between a master and a minion should be taken into account (e.g. checking your iptables).

The first test to do is to ping the minion from the master machine:

```
1 salt <minion_id> test.ping
```

To ping all the accepted minion use:

```
1 salt "*" test.ping
```

A *True* should appear before each minion that answers.

Troubleshooting salt-master

The first diagnostic step is to launch the *salt-master* process in debug mode :

```
1 salt-master -L debug
```

The output of this command is very important because it contains helpful information about the majority of problems you can encounter when using Salt.

Troubleshooting salt-minion

Like *salt-minion*, *salt-master* can be also executed in debug mode.

[23] http://docs.saltstack.com/en/latest/topics/troubleshooting/

```
salt-minion -L debug
```

The output will be the set of *debug*, *warning* and *error* messages.

Troubleshooting Ports

For *salt-master*, both TCP ports *4505* and *4506* should be opened. By running debug mode, information about these ports should be seen in the debug output.

For *salt-minion*, there are no specific ports to be opened but connectivity must be established:

```
nc -v -z salt.master.ip.address 4505
nc -v -z salt.master.ip.address 4506
```

In the case where a *salt-master* and a *salt-minion* are installed in two different production machines, most of the cases your server is behind a firewall (e.g. Security Groups in *AWS*), check that it allows the establishment of the connection between them.

On a *Redhat* system, *iptables* must allow the following configuration:

```
-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
```

 If you have troubles with setting your firewall using *iptable* file, consider using ufw[24] (Uncomplicated FireWall).

Using salt-call

salt-call is a good way to debug. The command can be used for example with the *states* or *highstates*.

e.g:

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

When *salt-call* is executed on a minion, no *salt-master* is called during the execution. This command allows the local execution, development assistance, and more verbose output.

If you want more verbosity, increase the log level.

e.g:

[24] http://en.wikipedia.org/wiki/Uncomplicated_Firewall

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

Like it is said in the SaltStack official documentation:

> "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."

A Note About Redhat Enterprise Linux 5

Salt works with *Python 2.6* or *2.7* while on *RHEL5*, *Python 2.4* is installed by default. When you install Salt from *Git*, it is recommended to install its dependencies via *EPEL* and run Salt executable with *Python26* usually found here: `/usr/bin/python26`

Basic Concepts

```
    \   ^__^
     \  (--)\_______
        (__)\       )\/\
            ||----w |
            ||     ||
```

Python

Salt is coded in Python[25] and also provides a *Python* client interface. *Python* is an object-oriented, multi-paradigm and multi-platform programming language.

It favors the structured, object-oriented, functional and imperative programming. It is strongly typed and has an automatic memory management with garbage collection and exceptions management system, thus it is similar to *Perl*, *Ruby*, *Scheme*, *Smalltalk*, and *Tcl*.

All *Python* releases are Open Source[26]. Historically, most but not all *Python* releases have also been *GPL* compatible.

It runs on most computing platforms, mainframe supercomputers, from *Windows* to *Unix* through *GNU/Linux*, *Mac OS*, or *Android*, *iOS* and also with *Java* or *.NET*. It is designed to maximize programmer productivity by providing high-level tools and a simple syntax.

It is also appreciated by the educators who find it a language with a syntax that is clearly separated from the low-level mechanisms, a language that enables an easy introduction to basic programming concepts.

Python is also widely used in scripting and operations (dev and sysops). Several DevOps professionals and system administrators adopted this language as their primary language for scripting because it has a large standard library, providing suitable tools for many types of tasks.

Salt is coded in *Python* and offers the power and flexibility that this language has. Some other configuration management like *Ansible* are also using *Python* as the main programming language.

Even if Salt can be used only from the command line, you can also use its *Python* interface and call all its commands from *Python* programs. Mastering *Python* is not obligatory but it is recommended to be at least initiated with this programming language.

A “hello world” in *Python can be written this way:

[25] http://www.python.org
[26] http://opensource.org/

```
1 print ( 'hello world !')
```

One of the advantages of working with *Python* is its community support, its active community and its large number of users and modules.

YAML: The Human Readable Serialization Language

YAML is a recursive acronym of "YAML Is not Markup Language". It is human readable serialization language (*Unicode* data).

It takes some concepts from other languages such as *XML*, or the email format as documented by RFC 2822.

The idea behind *YAML* is that any data can be represented by a combination of lists, tables (*hash*) and scalar data. *YAML* describes these types of data (*YAML* representations) and provide a syntax to describe the data as a stream of characters (the *YAML* stream).

A computer application passes the *YAML* stream to *YAML* representation by a "load operation". The representation is passed to the stream using a "dump operation". These could remind you of *JSON* (load & dump).

The syntax of *YAML* is relatively simple and effective, less verbose than *XML*, less compact than the *CSV* and was established as more readable by humans as possible.

YAML is easily mapped to common high-level languages (*python*, *ruby*, *perl* ..etc) at the level of the above data types.

YAML borrows some notations from other languages (*JSON*, *XML*, *SDL* ..):

- Comments start with the hash sign (#) and extend all along the line
- It is possible to include a *JSON* syntax in a *YAML* syntax
- Lists of items are denoted by a dash (-) followed by a space - one item per line
- Arrays are represented as "key: value" - one pair per line
- Scalar (strings) can be enclosed in double quotes (") or single (')
- Several documents together in a single file are separated by three hyphens (—); three dots (…) are optional to mark the end of a document within a stream
- The indentation, with spaces, shows a tree structure
- Tab characters are never allowed as indentation
- etc ..

YAML is easier to read by a human than *JSON*, it is more focused on data serialization with less documentation in comparison with *XML*.

Example:

```yaml
myApp 1.0:
  end_of_maintenance: 2014-01-01
  is_stable: true
  release_manager: "James Dean"
  description:
    This a stable version
  latest_beta: ~
  latest_minor: 1.0.17
  archives: { source1: [zip, tgz], source2: [zip, tgz] }

myApp 1.1:
  end_of_maintenance: 2014-06-01
  is_stable: true
  release_manager: 'Chuck Norris'
  description:
    Chuck Norris can program a computer to breath
  latest_beta: null
  latest_minor: 1.2.4
  archives:
    source1:
      - zip
      - tgz
    source2:
      - zip
      - tgz
```

phpMyAdmin allows to export a database into a *YAML* file.

Jinja: A Templating Language

Jinja is a text-based template language and one of the most used template engines for *Python*.

Jinja is Unicode-based and it runs on a wide range of *Python* versions from 2.4 to current versions including *Python3*.

It is similar to *Django* template engine but provides Python-like expressions to create a template.

According to Wikipedia:

> It is a text-based template language and thus can be used to generate any markup as well as source code. It is licensed under a *BSD* License. The *Jinja* template engine allows

customization of tags, filters, tests, and globals. Also, unlike the *Django* template engine, Jinja allows the template designer to call functions with arguments on objects. *Jinja*, like *Smarty*, also ships with an easy-to-use filter system similar to the *Unix* pipeline.

Let's move to a basic example of use:

```
from jinja2 import Template

tmpl = Template(u'''\
<!DOCTYPE html>
<html>
  <head>
    <title>{{ variable|escape }}</title>
  </head>
  <body>
  {%- for item in item_list %}
    {{ item }}{% if not loop.last %},{% endif %}
  {%- endfor %}
  </body>
</html>
''')

print tmpl.render(
    variable = 'Value with <unsafe> data',
    item_list = [1, 2, 3, 4, 5, 6]
)

The template above, written in Jinja will give the following HTML code:

``` html
<!DOCTYPE html>
<html>
  <head>
    <title>Value with <unsafe> data</title>
  </head>
  <body>
    1,
    2,
    3,
    4,
    5,
    6
```
```

```
37    </body>
38  </html>
```

Jinja is quite easy to use, and its use with SaltStack is not complicated, this is one of the strengths of *Jinja* and therefore Salt.

SaltStack uses this templating engine with files like *state* files.

The advantage of using *Jinja* is the use of control structures like conditional or redundant elements. The use of these structures (if then, for … in, range from … to … … etc.) makes the repetitive or the conditional templating tasks accessible, fast and easy to use.

In a *state* file (*SLS*: SalStack *states*), we can employ both *YAML* and *Jinja*.

Below is are *SLS* files:

```
1  /home/user/project/integ/myApp/4.2.2/myScript.py:
2    file.managed:
3      - source: salt://integ/myApp/4.2.2/myScript.py
4      - mode: 644
```

In our case, */home/user/project* is already defined in our *grains* file.

```
1  grains['destination_dir']: /home/user/project
```

With Jinja, we can use the next code:

```
1  {{ grains['destination_dir'] }}/integ/myApp/4.2.2/myScript.py:
2    file.managed:
3      - source: salt://integ/myApp/4.2.2/myScript.py
4      - mode: 644
```

The output is the same, since Jinja templating engine will render the code between *{{* and *}}* and will replace it by */home/user/project*.


Don't worry if you haven't understood the whole code above; you will.. For the moment, just focus on how to use Jinja.

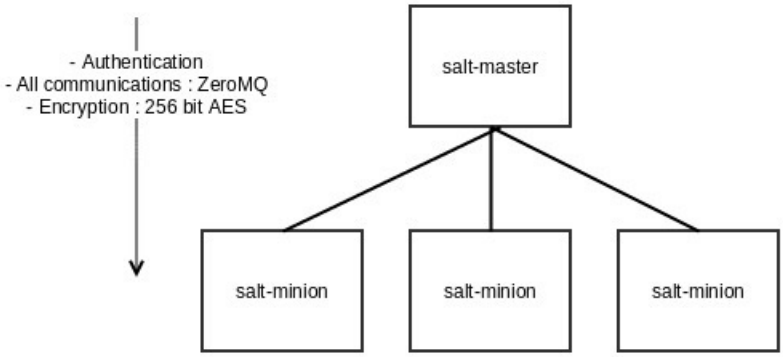
Jinja Loading Util which belongs to the module *salt.renderers.jinja*, is the tool that reads and parses Jinja template files/code and it "process" the code to give it its value.

Since Salt *Formulas* are pre-written Salt *states*, Jinja is used to write *formula states* and it is very helpful.

The Master: salt-master

SaltStack is composed of several modules and two of the most important are *salt-master* and *salt-minion.*

salt-master as its name suggests is the master, *salt-minion* is the listening target. Generally, at least in this documentation, we denote by *salt-master* or *salt-minion* a machine apart (it can be physical or virtual ..etc). The case where both are installed on the same machine (our *localhost*) is an exception - for testing purposes, this does not change almost anything at a functional level. That is why we always assume that each *salt-master* or *salt-minion* is considered as a machine.



A basic topology of a master and three minions

salt-master is the *master* daemon that controls one or more *minions* and can communicate with other *masters* (this will be clearer for you with *salt-syndic* section).

By default, *salt-master* is configured using the file:

```
1  /etc/salt/master
```

Of course you can change the location of the configuration file but we recommend to keep it by default for the moment to avoid all confusions.

salt-master is launched using the command:

```
1  salt-master [options]
```

While you can just use

```
salt-master
```

or

```
salt-master -d
```

The latter command will start the *master* as a daemon that will launch multiple *salt-master* processes, even if 'worker_threads' is set to '1'.

You can use one of the following options (provided by SaltStack in the official online documentation):

```
--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 c\
onfiguration files for Salt master and minions. The default location on most sys\
tems 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
-l LOG_LEVEL, --log-level=LOG_LEVEL
#Console logging log level. One of all, garbage, trace, debug, info, warning, er\
ror, 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, er\
ror, quiet. Default: warning.
```

Generally, the master starts as a daemon except when it is launched in debugging mode:

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

The Minion: salt-minion

salt-minion is the module that runs on the machine to manage (our target), it receives orders from one or more *masters*, executes them and then communicates back with the status of the execution and other details.

The case of a *masterless minion*, lets you use Salt's configuration management for a single machine, but not to execute remote commands or to communicate with other machines.

Note that *masterless minion* is rich with functionalities, you can even use it to stand up a *salt-master* via *states* or what we call "Salting a salt *master*".

Generally a *salt-minion* is the responsible for commands execution. But to do that, it must have the *accepted* state, which means that the *salt-master* can communicate securely with it. The latter uses a key system to communicate safely with one or more minions.

The default configuration file of a minion is in the following file:

```
/etc/salt/minion
```

Like the *master*, a *minion* is launched via the next command:

```
salt-minion [options]
```

Available options are:

```
--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 c\
onfiguration files for Salt *master* and minions. The default location on most s\
ystems 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
-l LOG_LEVEL, --log-level=LOG_LEVEL
#Console logging log level. One of all, garbage, trace, debug, info, warning, er\
ror, 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, er\
ror, quiet. Default: warning.
```

Generally, a minion is executed as a background daemon:

```
salt-minion -d
```

except for the case where the debugging mode is activated:

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

Expanding Salt with salt-syndic

salt-syndic is the module that allows to build and expand Salt topologies. It is used to connect multiple *masters* together and runs on a *master* machine. A *salt-syndic* listening to a master can control the minions attached to that master (the master on which the syndic is running).

salt-syndic transforms the default SaltStack topology to other advanced forms, more suited to a specific architecture or a particular need.

The syndic is a separate daemon to be installed and launched on the master machine. Every syndic is controlled by a higher-level master.

Starting a *salt-syndic* background daemon is done in the same way of starting a master or a minion daemons.

The *salt-syndic* is configured using the same file as the master:

```
cat /etc/salt/master|grep syndic
# 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 pass-through the
# "syndic_master" setting needs to be set to the location of the master server
# masters' syndic interfaces.
# If this master will be running a salt syndic daemon, syndic_master tells
#syndic_master: 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
```

Just uncomment the part dedicated to *salt-syndic* configuration and note that:

- *syndic_master* is the master of the master ip/address
- *syndic_master_port* is the master of the master ret_port
- *syndic_log_file* is the path to the logfile
- *syndic_pidfile* is path to the pidfile

A basic/standard example of a Salt topology using *salt-syndic* will have a top master (a machine where only *salt-master* is installed) and lower-level machines where the master and the syndic should be installed and configured. The machine/node sitting at the bottom of this hierarchic topology needs neither a master nor a syndic, it only needs a configured minion.

Through the topology described above, when a command is issued from a higher-level master, the listening syndic(s) - those having a lower level - will receive those orders and propagate them to their minions and listening syndic(s) having a lower-level.

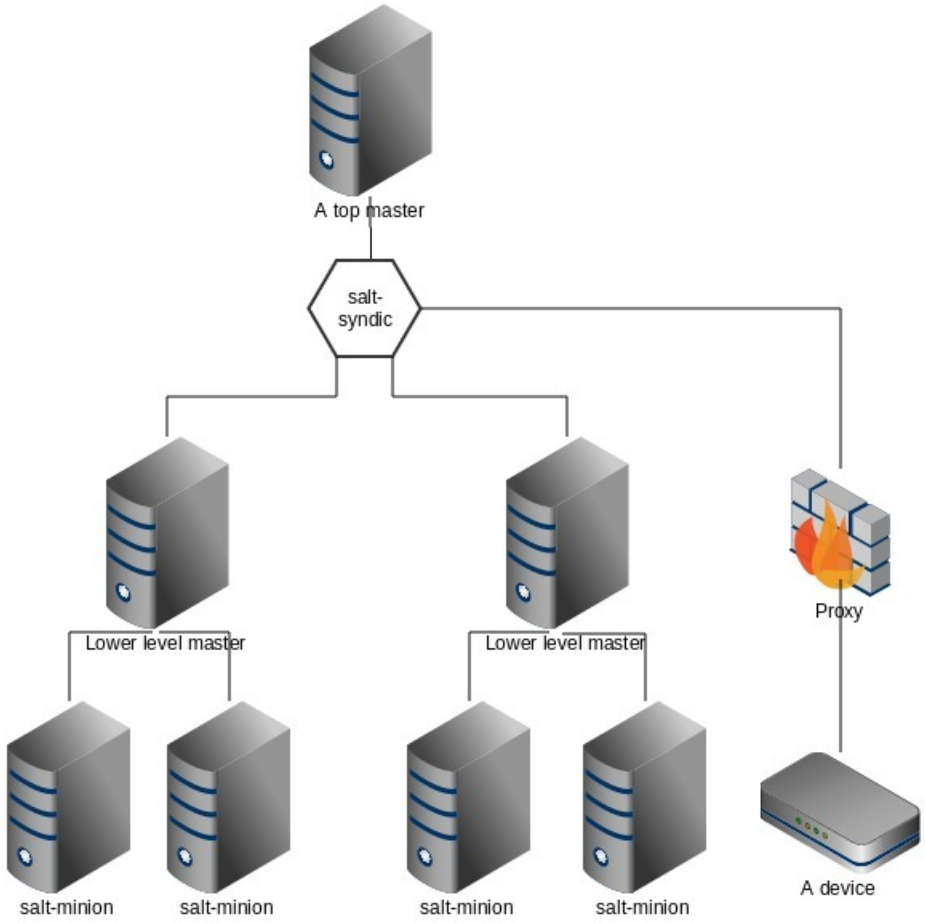

Returned data and executions statuses are generated and dispatched-back using the same reversed hierarchy route.

In some cases, large infrastructure typologies the feedback of a connected minion, master or a syndic could take time to roll up, this could cause a timeout for the execution even if the concerned command is executed in reality but the response driving up through the infrastructure may take time to arrive. To fix this, just add the following configuration to the master configuration file:

```
syndic-wait: 1
```

Another important configuration variable needs to be set to True, the official Salt documentation[27] says that :

> 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 configuration 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.



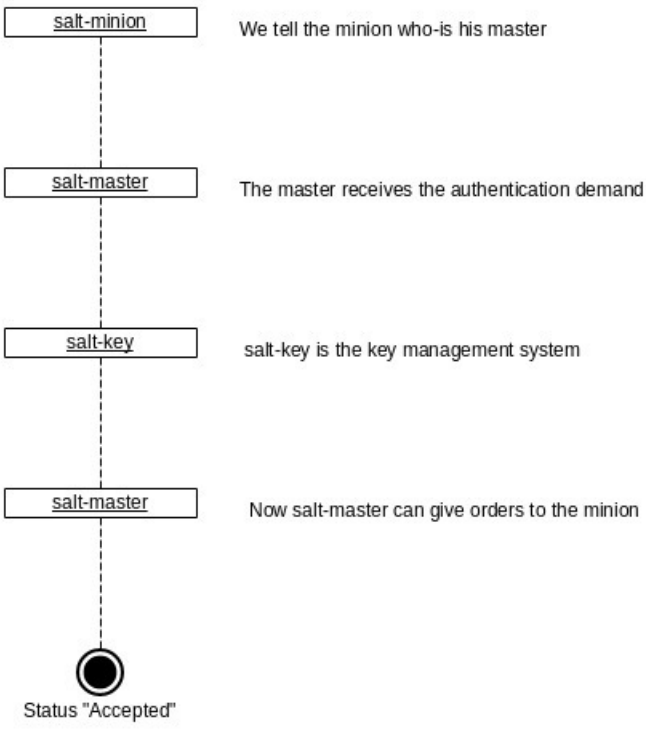
A SaltStack topology using salt-syndic

Key Management with salt-key

salt-key is the system responsible for managing a public key server used for authentication.

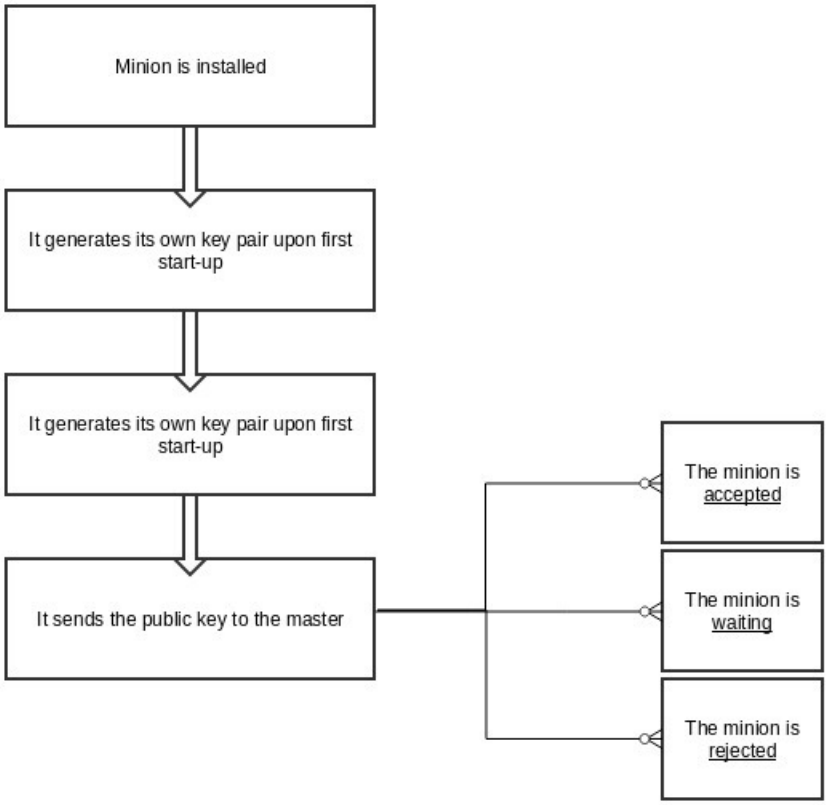
[27] http://docs.saltstack.com

Example: A salt-master communicates with a *salt-minion* only when this key system allows it. The illustration below shows how a *salt-minion* can be accepted through Salt process.



An accepted salt-minion

A *salt-minion* is not always accepted by default, it may be also *rejected* or *waiting*.



3 possible statuses for a salt-minion

salt-key is launched using the following command:

```
salt-key [options]
```

The following options can help you to administer *salt-key* :

```
--version
# Print the version of Salt that is running.
--versions-report
# Show dependencies and version number
-h, --help
# Show the help message
-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 is /etc/sa\
lt.
-q, --quiet
# Suppress output
```

```
-y, --yes
# Answer 'Yes' to all questions presented, defaults to False
--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, e\
rror, quiet.
# Default: warning.
--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 Sa\
lt will fall back on the pprint outputter and display the return data using the \
Python pprint standard library module.
--out-indent OUTPUT_INDENT, --output-indent OUTPUT_INDENT
# Print the output indented by the provided value in spaces. Negative values dis\
able 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
-l ARG, --list=ARG
# List the public keys. The args pre, un, and unaccepted will list unaccepted/un\
signed keys. acc or accepted will list accepted/signed keys. rej or rejected wil\
l 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 a\
ddition 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 a\
ddition 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.
--gen-keys=GEN_KEYS
# Set a name to generate a keypair for use with Salt
--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.
```

Some of those commands (available on the official SaltStack documentation) will be used in the next chapter.

 There will be no possible communication between a master and a minion unless the *salt-key* system allows it. Be sure that all *minions* are accepted, before starting anything else.

Salt Runners Interface: salt-run

Consider *salt-run* as your interface to use Salt Runners. These are executable modules implemented in SaltStack. Using the runner is simple as:

```
salt-run RUNNER
```

A good example of using this is when you want to list your *minions*' states (up or down).

Type the next command to show all minions and the status of everyone.

```
salt-run manage.status
```

Showing only all minions with “up” status:

```
salt-run manage.up
```

And for showing all minions with “down” status:

```
salt-run manage.down
```

Another useful usage is when you want to clear Salt cache (this could be helpful also for debugging/troubleshooting).

```
salt-run cache.clear_all
```

This is the official list of SaltStack Runners:

Function	Description
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

Function	Description
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

The Data Aspect of Salt: Grains

"Salt grains of data" is the best way to describe the main role of Salt *grains*.


The *grains* module represents the "data" aspect of SaltStack.

When minion daemon starts, it loads the data from Salt *grains*. *Grains* stored data is static. Salt *grains* can indicate various information such as the operating system of a *minion*, the *kernel* or the disk size ... Etc.

Just after the installation, the configuration and the start-up of a *salt-minion*, we have the possibility to directly manipulate this module and handle the available data.

A Salt data is a pair of a key and its value. The next command shows how to add a key and assign its value:

```
1 salt '*' grains.setval <key> <value>
```

Example: We want to add the key *env_name* (for environment name) and give it the value of *prod* (for production).

```
1 salt '*' grains.setval env_name prod
```

This will add a key/value to *grains* configuration file. We can do a test :

```
1 salt '*' grains.get <key>
```

and for the last example:

```
salt '*' grains.get env_name
```

You will see the 'env_name' value printed out:

```
mynode:
  - integ
```

If you need to massively add multiple *grains*:

```
salt '*' grains.setvals "{'key1':'value1','key2':'value2','key3':'value3'}"
```

We can also delete a key, therefore, its value:

```
salt '*' grains.delval <key>
```

To delete 'env_name' :

```
salt '*' grains.delval env_name
```

Salt can also handle dictionaries and lists.

Dictionaries and lists in *Python* are powerful and simple data structures. *Python* offers multiple functionalities to easily manage them.

Creating and adding elements to a list is simple :

```
salt '*' grains.append list1 element_a;
salt '*' grains.append list1 element_b;
salt '*' grains.append list1 element_c;

salt '*' grains.append list2 element_d;
salt '*' grains.append list2 element_e;
salt '*' grains.append list2 element_f;
```

grains.append is like *Python*'s *list.append* will create a list if it does not exist.

What we have added, could be viewed using *grains.get*:

```
salt '*' grains.get list1
#The output
mynode:
  - element_a
  - element_b
  - element_c
```

To remove an element from the list :

```
salt '*' grains.remove list value_c

#The output
mynode:
  list1:
    - value_a
    - value_b
```

Creating a dictionary of data can be done using *setval*:

```
salt '*' grains.setval key "{'subkey_a':'value_a','subkey_b':'value_b'}"

#The output
mynode:
  key:
    subkey_a:
      value_a
    subkey_b:
      value_b
```

Once all of those elements were added, we can now check all of the *grains* we have on our *minion*. Remember that there every minion has already default *grains* (*os*, *cpu* architecture ..etc).

To view all of them, just type:

```
salt '*' grains.ls
```

This will print a list of keys that the *minion* knows their values.

Example:

```
mynode:
    - biosreleasedate
    - biosversion
    - cpu_flags
    - cpu_model
    - cpuarch
    - defaultencoding
    - defaultlanguage
    - domain
    - fqdn
    - fqdn_ip4
    - fqdn_ip6
    - gpus
    - host
    - id
    - ip_interfaces
    - ipv4
    - ipv6
    - kernel
    - kernelrelease
    - localhost
    - manufacturer
    - master
    - mem_total
    - nodename
    - num_cpus
    - num_gpus
    - os
    - os_family
    - osarch
    - oscodename
    - osfinger
    - osfullname
    - osmajorrelease
    - osrelease
    - path
    - productname
    - ps
    - pythonpath
    - pythonversion
    - saltpath
    - saltversion
```

```
    - saltversioninfo
    - serialnumber
    - server_id
    - shell
    - virtual
```

All of the above output is the list of keys, to print out their values, you should type:

```
salt '*' grains.items
```

Here is an example of all the *grains* (keys, values) that the minion *mynode* has:

```
mynode
  biosreleasedate: 03/05/2009
  biosversion: VirtualBox
  cpu_flags: fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat p\
se36 clflush mmx fxsr sse sse2 syscall nx rdtscp lm constant_tsc up rep_good pni\
 monitor ssse3 lahf_lm
  cpu_model: Intel(R) Celeron(R) CPU G530 @ 2.40GHz
  cpuarch: x86_64
  defaultencoding: UTF8
  defaultlanguage: en_US
  domain: localdomain
  fqdn: mynode
  fqdn_ip4:
  fqdn_ip6:
  gpus:
      model:
          VirtualBox Graphics Adapter
       vendor:
           unknown
host: mynode
  id: mynode
  ip_interfaces: {'lo': ['127.0.0.1'], 'eth0': ['10.0.0.1']}
  ipv4:
      10.0.0.1
      127.0.0.1
  ipv6:
      ::1
      fe88::a00:72ff:fee8:64b
  kernel: Linux
  kernelrelease: 2.6.32-431.1.2.el6.x86_64
```

```
  localhost: mynode
  master: 10.0.254.254
  mem_total: 1500
  nodename: mynode
  num_cpus: 1
  num_gpus: 1
  os: ScientificLinux
  os_family: RedHat
  osarch: x86_64
  oscodename: Carbon
  osfinger: Scientific Linux-6
  osfullname: Scientific Linux
  osmajorrelease:
      6
      4
  osrelease: 6.4
  path: /sbin:/usr/sbin:/bin:/usr/bin
  productname: VirtualBox
  ps: ps -efH
  pythonpath:
      /usr/bin
      /usr/lib64/python26.zip
      /usr/lib64/python2.6
      /usr/lib64/python2.6/plat-linux2
      /usr/lib64/python2.6/lib-tk
      /usr/lib64/python2.6/lib-old
      /usr/lib64/python2.6/lib-dynload
      /usr/lib64/python2.6/site-packages
      /usr/lib/python2.6/site-packages
      /usr/lib/python2.6/site-packages/setuptools-0.6c11-py2.6.egg-info
  pythonversion: 2.6.6.final.0
  saltpath: /usr/lib/python2.6/site-packages/salt
  saltversion: 0.17.4
  saltversioninfo:
      0
      17
      4
  serialnumber: 0
  server_id: 332009357
  shell: /bin/sh
  virtual: VirtualBox
```

As you can see, *salt-minion* knows a lot of information about the system and the environment on

which it is installed, like *Python* version, the serial number or IP addresses of different machine's interfaces.

If you are interested in viewing only the machine's interfaces, use *ip_interfaces* (that figure in *grains.item* output) like this:

```
salt '*' grains.get ip_interfaces
```

This will give you only the IP interface data. Only requested data will be printed out:

```
ip_interfaces:
        eth0:
            - 10.0.0.1
        lo:
            - 127.0.0.1
```

It is also possible to refine your output like getting only the IP of *eth0*:

```
salt '*' grains.get ip_interfaces:eth0
```

The IP *10.0.0.1* will be displayed as a result of your request.

Another feature is combining two requests, like getting *kernel* and *ip_interfaces* information:

```
salt '*' grains.item ip_interfaces kernel
```

Result:

```
mynode:
    ip_interfaces:
        eth0:
            - 10.0.0.1
        lo:
            - 127.0.0.1
    kernel:
        Linux
```

Adding, modifying or deleting *grains* can be made directly to the default configuration file:

```
/etc/salt/grains
```


The configuration file is written in *YAML*. Neglecting standards compliance may lead to some problems.

You can add strings, lists or dictionaries. Open the *grains* file or the configuration file with your favorite editor and add :

```
key: value
```

The last element could be a list:

```
my_list:
  - my_first_value
  - my_second_value
```

It could be a dictionary:

```
my_dict:
  my_first_key: my_first_value
  my_second_key: my_second_value
```

By using *YAML* syntax you can create complex data structures with *list* and *dict* and of course *int*, *string* and *boolean* types ..etc

Here is an example :

```
include:
  - apache

mod-php:
  pkg.installed:
    - name: libapache2-mod-php
    - require:
      - pkg: apache

a2enmod php:
  cmd.run:
    - unless: ls /etc/apache2/mods-enabled/php.load
    - require:
```

```
14        - pkg: mod-php5
15      - watch_in:
16        - module: apache-restart
17
18  /etc/php/apache2/php.ini:
19    file.managed:
20      - source: salt://prog/php_ini
21
22  service.running:
23    - name: {{ apache.service }}
24    - enable: True
```

Notice that *Jinja* syntax is used near the name of the running service. We are going to use it again in many of the following examples, if you are not familiar with it, normally following the examples will be enough to understand its basics. If it is not the case, please refer to the official documentation[28]. If you want to master the use of Jinja, I recommend using it with Flask[29]

Targeting grains

In the previous examples, we used SaltStack commands with the asterisk ' * '. The '*' replaces all machines or minions that salt-master has in its base. It is possible to perform a targeting minions on which commands will be executed Salt.

When we type the next command:

```
1  salt '*' grains.item kernel
```

SaltStack, more precisely the 'salt-master' will get the kernel name for all of the minion that it knows, in other words, all of accepted *salt-minions*, which means minions that will respond with *True* when this command is executed:

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

In all of the last example, since we are using one master and one minion (*mynode*), executing the last command will have the next output:

[28] http://jinja.pocoo.org/docs/dev/
[29] http://flask.pocoo.org/

```
mynode:
  kernel:
    Linux
```

Once another minion is added (following the same configuration and process as the first minion), the output will be different and will show other machines.

```
mynode:
  kernel:
    Linux

another_node:
  kernel:
    Darwin
```

In this section, we call this process : *targeting*.

SaltStack offers many other targeting options; apart from using '*' or the name of the minion instead:

```
salt '*' test.ping
salt mynode test.ping
salt another_node test.ping
```

You can also target minions having specific properties (*grains* data), like *OS* architecture; *OS* family; *kernel* name or simply the name of the *minion*..etc

```
#Targeting minions having i368 processor
salt -G 'osarch:i386' test.ping

#Targeting Debian machines
salt -G 'os_family:Debian' test.ping
#Or
salt -G 'os_family:Deb*' test.ping

#Targeting *mynode* and 'another_node' nodes
salt '*_node' test.ping

#Targeting *mynode*
salt 'm?_node' test.ping
#Or
salt '?y_node' test.ping
# ..etc
```

```
# If we had the following nodes : 'mynode_1', 'mynode_2' and 'mynode_3', to targ\
et them we may use:
salt 'mynode_[1,3]' test.ping
```

Every SysAdmin knows that regular expressions (*Regex*) are necessary to "survive some situations". When it comes to targeting, regular expressions could be helpful. If you are not familiar with *Regex*, it is important to begin learning them. Note that SaltStack is using *PCRE* : *Perl* Compatible Regular Expressions.

Other types of targeting, like *compound matchers*, are provided.

The following table explains well the list of compound matchers:

Letter	Targeting Type	Example
G	Grains glob	G@os:Debian
E	PCRE Minion ID	E@webapp\d+.(dev|staging|prod)
P	Grains PCRE	P@os:(RedHat|Ubuntu|CentOS)
L	List of minions	L@my.node.example.com,another.node.example.com
I	Pillar glob	I@pdata:foobar
S	Subnet/IP address	S@192.168.55.0/24 or S@192.168.56.1
R	Range cluster	R@%foo.bar

Here are some additional examples:

```
# "-C" is used with compound matchers
#
#Targeting all minions having amd64 architecture
salt -C '* and not G@osarch:amd64 test.ping'
#Targeting all minions except "mynode"
salt -C '* and not mynode' test.ping
#Targeting "mynode" and all Debian machines or "my_minion_\d.*"
salt -C 'mynode and G@os:Debian or E@my_minion_\d.*' test.ping
```

Note that all of those targeting expressions and methods could be used in the 'top.sls' file. Top file will be explained hereinafter.

Extending The Minion Features With Saltutil

SaltStack has implemented extended features on minions as standalone operations, the ability to self-configure, the self-management of its modules to automate updates. That's why a minion does

not always need a master.

Among the features *saltutil* allows, purging cache:

```
salt '*' saltutil.clear_cache
```

Or executing a command on the *salt-minion* like it was executed on a master :

```
salt '*' saltutil.cmd
```

Formatting Outputs with Outputter

Salt uses a standard display format to display a list of keys or a configuration of *grains*. *Outputter* feature allows you to format the output produced by a Salt command.

To re-format the output produced by the next command :

```
salt '*' grains.item ip_interfaces kernel
```

we just need to add “–out=<format>” or “–output=<format>” in the end of the command:

```
salt '*' grains.item ip_interfaces kernel --out=json
```

The output will be displayed in *JSON* format:

```
{
    "mynode": {
        "kernel": "Linux",
        "ip_interfaces": {
            "lo": [
                "127.0.0.1"
            ],
            "eth0": [
                "10.0.0.1"
            ]
        }
    }
}
```

Other displaying formats could be used, according to official documentation, Salt uses this list of output modules:

grains

Special *outputter* for *grains*.

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*.

json_out

Display return data in *JSON* format. 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.

key

Display *salt-key* output. The *salt-key* command makes use of this *outputter* to format its output.

nested

Recursively display nested data. This is the default *outputter* for most execution functions.

newline_values_only

Display values only, separated by newlines.

no_out

Display no 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.

overstatestage

Display clean output of an overstate stage. This *outputter* is used to display overState stages, and should not be called directly.

pprint_out

Python pretty-print (*pprint*). The *python pretty-print* system was once the default *outputter*.

raw

Display raw output data structure. This *outputter* simply displays the output as a *Python* data structure, by printing a string representation of it.

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.

virt_query

virt.query outputter

yaml_out

Display return data in *YAML* format. This *outputter* defaults to printing in *YAML* block mode for better readability.

Describing Configurations and States

A Salt "state" is a pre-configured state of an operation/task supported by Salt (like a configuration). Generally noted *SLS*, this term can also refer to *SLS* files.

A Salt state is based on descriptive files (*SLS*) containing information and structured data. These files are by default written in *YAML* and containing data structures like dictionaries, lists, character strings or numbers.

SLS files can be assigned to hosts using the *top.sls* file.

The Top File

The role of this file is to map the *SLS* modules (Salt *states*) to *minions*. Without it a *salt-master* will not know which state configuration to apply on a given *minion*.

As we have already set it in the *master* configuration file, the variable *file_roots* contains:

```
1 file_roots:
2     base:
3         - /srv/salt/
```

When using Salt, some rules should be respected, like the obligation to set a *top.sls* file in the *base* environment. This file must be created on:

```
1 /srv/salt/top.sls
```

Why?

When using Salt with the default configuration, a single environment called *base* is set up by default:

```
#The master configuration file
file_roots:
  base:
    - /srv/salt
```

That's why the *top.sls* file should be under the *base* environment "file_roots":

```
/srv/salt/
```

The *top.sls* will contain something similar to:

```
base:
  '*':
    - <SLS_name>
```

Next is an example of an *SLS* file:

```
/srv/salt/vim.sls
```

If we want to assign the latter to all *minions* (*'*) *in the *base* environment, we need to have a *top.sls* file containing:

```
base:
  '*':
    - vim
```


Note that you don't need to write the whole name of the *SLS* file, writing "vim" instead of *vim.sls* is sufficient.

When changing the path of *vim.sls*, think of changing it in the *top.sls* file also.

If the *vim.sls* path is:

```
/srv/salt/salt_vim/vim.sls
```

The *top.sls* file should be modified:

```
1 base:
2   '*':
3     - salt_vim/vim
```

Since our *base* environment is */srv/salt* we don't need to type all of the path of *vim.sls* which is */srv/salt/salt_vim/vims.sls*. You can just use *salt_vim/vim*, cf. line 3 in the last example.

The content of *top.sls* file depends on the declaration of the environments in the *salt-master* configuration file.

If you want to declare other environments different from the base environment like in the next example:

```
1 file_roots:
2     base:
3          - /srv/salt/base
4     development:
5          - /srv/salt/development
6      pre-production:
7          - /srv/salt/pre-production
8      production:
9          - /srv/salt/production
```

the *top.sls* file should reference all of the different declared environments:

```
1  base:
2      '*':
3          - vim
4
5  development:
6      '*dev*':
7          - dev_sls
8
9  pre-production:
10     '*preprod*':
11         - prepprod_sls
12
13 production:
14     '*production*':
15         - prod_sls
```

The *top.sls* file will be called during the execution of a "highstate" and this is, by the way, its most important purpose.

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

Salt will recursively look for *top.sls* files in the folder representing the *base* environment and any other folders representing the other different environments.

In the last example, the file *vim.sls* will be applied to all *minions*. The file *dev_sls.sls* will be applied to *minions* having an id containing *preprod* and so on ..

We could also use the compound matchers in a *top.sls* file:

```
base:
    'mynode* and G@os:CentOS or E@myminion.*':
        - match: compound
        - vim
```

Using Master's Pillars

Pillars are tree-structured data defined on the master and returned to the *minions*. They allow, for example, storing confidential data securely to be sent only to the targeted *minions*.

Grains are an innate property of a *minion*, while the data in *pillars* are acquired by a *minion* that a master target: This is the fundamental difference since *pillars* are defined in the master (in *SLS* format).

Examples:

- Authentication login/password (database, CMS ..etc)
- API secret key
- A private key

Like *grains*, viewing the list of available *pillars* could be done using:

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

pillars are based on environments (declared in the master configuration file):

```
/etc/salt/master
```

Consider that we are going to keep the default base environment:

```
#pillar_roots:
#base:
#  - /srv/pillar
```

We should create the folder *pillars*:

```
mkdir /srv/pillar
```

then a *top.sls* file :

```
touch /srv/pillar/top.sls
```

We are going to create a file where we will store some sensible data (mysql password as an example):

```
touch /srv/pillar/passwords.sls
```

In the case where the password will be used by all of available *minions*, the *top.sls* file will contain:

```
base:
  '*':
    - passwords
```

Notice the use of ‘’ *to tell Salt that the password will be used by all *minions*.* Notice also that we are not obliged to write *password.sls*. Salt will recognize that *passwords* make reference to *password.sls* file.

For now, let’s add the password of *MySql* to the *passwords.sls* file:

```
mysql_password: aZ?e14E373F7123$aBP
```

After configuring and storing data, it is recommended to make a refresh for *pillars* using:

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

MySQL password will be shown on your screen if you enter:

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

Generally, *pillars* are intended to be used in SLS files (Jinja) and can be called using different ways.

Simple structures:

```
{{ pillar['key'] }}
```

Nested structures:

```
{{ pillar['key_a']['key_b'] }}
```

Conditionally, if "key_a:key_b" is not configured (no value returned), the "value_c" will be used:

```
{{ salt['pillar.get']('key_a:key_b', 'value') }}
```

To reiterate the different elements of a structure:

```
{{ pillar.get('key', {}).items() }}
```

Note that *pillars* data could be configured using CLI:

```
salt '*' state.highstate pillar='{"key": "value"}'
```

For a particular file:

```
salt '*' state.sls my_sls_file pillar='{"key": "value"}'
```

The configured *pillars* may also be used to target *minions*:

```
salt -I 'key:value' test.ping
```

Remote Execution

Remote execution is one the most important SaltStack features. Using *CLI*, you could use the different matchers to to filter *minions*:

```
salt '*' test.ping
salt -G 'os:Debian' test.ping
salt -E 'integ_[a-z]' test.ping
salt -L 'prod,web,indus,integ' test.ping
salt -C 'G@os:Redhat and prod* or E@integ_[a-z]' test.ping
```

Basically, the execution of a distant command is done like this:

```
salt '*' cmd.run '<command to run>'
```

Example:

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

Remote execution can be also used in *SLS* files. The following example launches two remote controls. The second command (echo) only starts when the first is executed.

```
run_script:
  cmd.run:
    - name: /path/to/myscript
    - cwd: /etc
    - stateful: True

run_second_script:
  cmd.wait:
    - name: echo 'Finished running script'
    - cwd: /etc
    - watch:
        - cmd: run_script
```

Event-Based Execution With Reactors

SaltStack uses a system of "reactors" since its version 0.11.0, the goal is to run commands triggered by an event. This system allows you to link an *SLS* file to an "event tag" on the master.

The event system is based on ZeroMQ that triggers events in a bus. The latter is used to send information about operations executions.

Each event has a "tag" and data structure (a dictionary containing event information). The definition of "reactors" must appear on the master configuration file.

Example :

```
reactor:
  - 'salt/minion/*/start':
    - /srv/reactor/sync_grains.sls
    - /srv/reactor/mysql.sls
```

In the last example, when the tag

```
salt/minion/*/start
```

is activated, the “reactor” system will trigger the activation of the different *SLS* files:

```
    - /srv/reactor/sync_grains.sls
    - /srv/reactor/mysql.sls
```

The role of the first file (*sync_grains.sls*) is to synchronize the *grains* at *salt-minion* start-up. This is its content:

```
sync_grains:
  local.saltutil.sync_grains:
    - tgt: {{ data['id'] }}
```

The role of the second file (mysql.sls) is to apply *state.highstate* to *mysql_minion* .

And this is its content:

```
{% if data['id'] == mysql_minion %}
highstate_run:
  local.state.highstate:
    - tgt: mysql_minion
{% endif %}
```

The system of *reactors* enables management of multiple types of events associated with:

- Authentication
- *Minion* service upstart
- Key management
- Jobs management
- Cloud Management

The list below lists some “event tag”:

- when a *minion* performs an authentication check with the *master*.
  - salt/auth
- when a *minion* connects to the Salt *master*.
  - salt/minion/*/start
- when accepting and rejecting *minions* keys on the Salt *master*.
  - salt/key
- when a new job is sent out to *minions*.
  - salt/job/*/new
- when a *minion* returns data for a job.
  - salt/job//*ret*/
- when setting *schedule*.
  - salt/presence/present
- when the *presence* system detects new *minions* connect or disconnect.
  - salt/presence/change
- when *salt-cloud* starts the VM creation process.
  - salt/cloud/*/creating
- when the VM is available and *salt-cloud* begins deploying Salt to the new VM.
  - salt/cloud/*/deploying
- when *salt-cloud* sends the request to create a new VM.
  - salt/cloud/*/requesting

You can find the rest of the list on the official documentation of SaltStack[30]

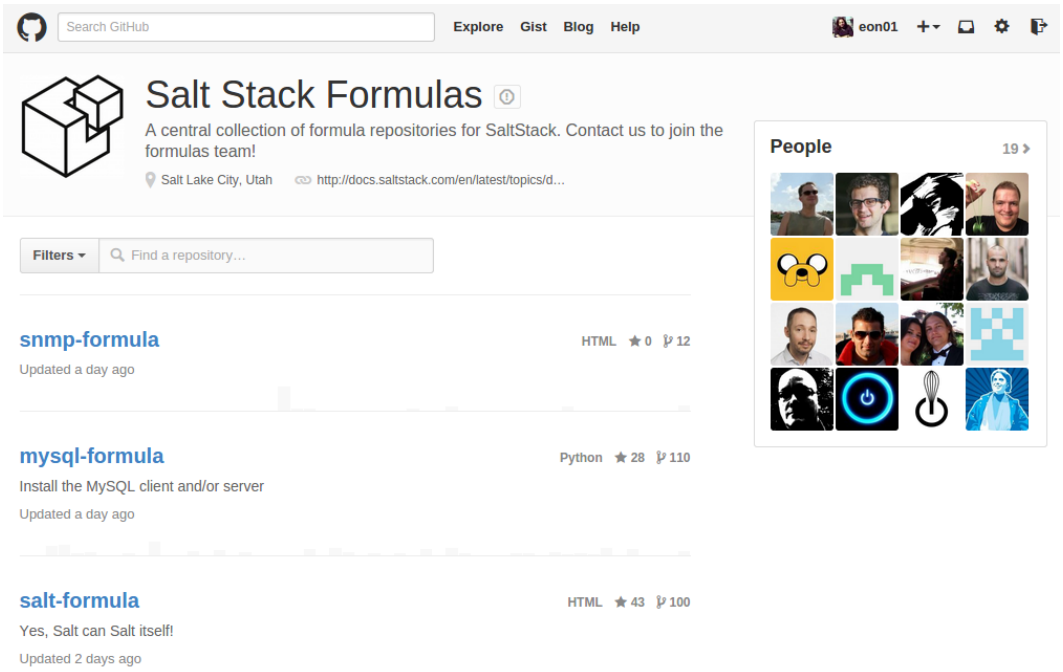
From States To Formulas

When you write a *salt-state* in order to use it later, you had actually written a 'formula'. A Salt *formula* is a pre-written salt *states* that can be used mainly to run common tasks like installing/configuring packages and starting services or configuring users/groups/permissions ..etc.

You can find the official Salt 'formulas' in a git repository[31](saltstack-formulas).

Salt 'formulas' makes life easier, since the repository is maintained by a community of developers working on many different platforms *formulas*. To install *Tomcat*/*Apache* or *MongoDB*, just reference the *formula* in the *top.sls* file and configure the targeted *minions* and environment.

[30] http://docs.saltstack.com/en/latest/topics/event/master_events.html
[31] https://github.com/saltstack-formulas



A central collection of formula repositories for SaltStack.

Here is a list of some useful Salt *formulas* that you can find in the official git repository:

- *mysql-formula* (Install the *MySQL* client and/or server)
- *salt-formula* (Yes, Salt can Salt itself!)
- *maven-formula* (A minimalistic way to install just the maven distribution without any dependencies)
- *apache-formula* (Set up and configure the *Apache* HTTP server)
- *docker-formula* (Install and set up *Docker*)
- *sensu-formula* (SaltStack formula to manage *Sensu*)
- *redis-formula* (*Redis* state)
- *epel-formula* (Install the *EPEL RPM* and *GPG* key)
- *openvpn-formula* (Setup and configure *openvpn* server and client.)
- *iptables-formula*
- *sysstat-formula*
- *sysctl-formula*
- *hadoop-formula*
- *mongodb-formula*
- *circus-formula*
- *squid-formula*
- *jenkins-formula*
- *snmp-formula*
- *pam-ldap-formula*
- ..etc

SaltStack And Vagrant

Between development and live (production) environments, some problems may happen, most of them are due to the difference between the two environments. “Well, it works on my machine”, this is a common sentence that both Dev and Ops guys could use it as an excuse. Even if it could be true - in some cases, a role of a DevOps guy is to take this into consideration and provide the necessary techniques and technologies to solve this kind of problems.

Vagrant seems to be a good solution to implement in the pipeline - the bridge connecting Dev and Ops since it allows developers to have an isolated development environment that could be similar to the production one.

In this part of the book, I am assuming that you have some basic knowledge about Vagrant.

If not, the following introduction will help you to be more familiar with this software but not an expert :-)

A simple definition to start: Vagrant is a software that let you create and configure virtual environments for development purposes (maybe other purposes also, this depends on the context of your product and enterprise).

Vagrant is an open-source (MIT) tool developed by Mitchell Hashimoto and John Bender. It manages virtual machines hosted in Oracle VirtualBox. It has its command-line tool and it uses Ruby (a dynamically typed object-oriented scripting language).

Why Vagrant

According to the official website[32], *Vagrant* is a tool for :

- Developers

  If you are a developer, *Vagrant* will isolate dependencies and their configuration within a single disposable, consistent environment, without sacrificing any of the tools you are used to working with (editors, browsers, debuggers, etc.). Once you or someone else creates a single *Vagrantfile*, you just need to *Vagrant* up and everything is installed and configured for you to work. Other members of your team create their development environments from the same configuration, so whether you are working on Linux, Mac OS X, or Windows, all your team members are running code in the same environment, against the same dependencies, all configured the same way. Say goodbye to “works on my machine” bugs.

- Operations engineers

[32] http://vagrantup.com

If you are an operations engineer, *Vagrant* gives you a disposable environment and consistent workflow for developing and testing infrastructure management scripts. You can quickly test things like shell scripts, Chef cookbooks, Puppet modules, and more using local virtualization such as VirtualBox or VMware. Then, with the same configuration, you can test these scripts on remote clouds such as AWS or RackSpace with the same workflow. Ditch your custom scripts to recycle EC2 instances, stop juggling SSH prompts to various machines, and start using *Vagrant* to bring sanity to your life.

- Designers

If you are a designer, *Vagrant* will automatically set everything up that is required for that web app in order for you to focus on doing what you do best: design. Once > a developer configures *Vagrant*, you don't need to worry about how to get that application running ever again. No more bothering other developers to help you fix your environment so you can test designs. Just check out the code, *vagrant up*, and start designing.

Using SaltStack As Provisioner

The salt provisioner allows you to provision the guest VM using Salt *states*. The current state of a *Vagrant* VM could be described using SaltStack (installed packages, running services, and configurations).

In general, there is no need for a master to provision the VM, a *mastreless* Salt is enough.

The *Vagrant* Salt provisioner allows you to provision the guest machine using Salt *states*: installing and configuring packages, activating and running services and configuring development environments.

Using *Vagrant* for development environments could help both developers and ops having similar environments between development and production, this can be done by using Salt since both *states*/configurations used to create production environments could be used to create development environments.

Using both Salt and *Vagrant* helps automation teams to set up a continuous integration/delivery pipeline. When using *Vagrant*, you will probably test it and build a basic machine and you will surely need to manage to build your virtual *Vagrant* machines, since configurations cloud change from one team to another, from an application to another or even from a software release to another, that's why it using a configuration management like SaltStack is valuable.

In the next chapter, you are going to see some practical examples of using SaltStack as a provisioning tool.

SaltStack And Docker

Containers are relatively new technologies. You have surely heard about containers technologies like *Docker* and *CoreOs* and maybe also other related technologies like *Jubernetes* and *Docker Compose* (formerly *Fig*).

Even if some say that those technologies are not really production-ready yet given some issues with file systems, speed, garbage collection, and security, *Docker* is changing the Cloud Computing industry: 5x growth in its adoption in the last 12 months (from August 2014 to August 2015) according to *DataDog*.

Anyway, *Docker* containers have similar resource isolation and allocation benefits as virtual machines but a different architectural approach. The great benefit of using *Docker* is that it allows you to write code once and run it anywhere. Apart from being lightweight, *Docker* is portable.

If you had already worked with SaltStack to manage *Docker*, you have probably worked with *dockerio* module which is a built-in state module (*salt.states.dockerio*) just like *salt.states.disk* or *salt.states.apache*.

This module requires *docker-py* (version >= 0.6.0), you can simply install the latest version by typing:

```
1 pip install docker-py
```

Once *docker-py* is installed, you can start working with *dockerio* module.


The problem with *dockerio* is that it is deprecated and since the version 2015.8.0, development is only done on *dockerng*, so I highly recommend to learn *dockerng* rather than *dockerio*.

The module *dockerng* will be supported for the future and it allows you to create, build, manipulate and orchestrate *Docker* images.

According to the official documentation of Salt:

> We have received a lot of feedback on our *Docker* support. In the process of implementing recommended improvements, it became obvious that major changes needed to be made to the
> functions and return data. In the end, a complete rewrite was done.


dockerng stands for *Docker* Next Generation.

You will still need *docker-py*:

```
pip install docker-py>=1.4.0
```

Or just use Salt to install *docker-py* on your *minion*:

```
salt mynode pip.install docker-py>=1.4.0
```

Salt Cloud

Salt Cloud is built-in to Salt and is configured on and executed from your Salt *master*.

Salt can provision systems on cloud hosts/hypervisors: - Public clouds: Google Cloud, Amazon Web Services, Linode - Private clouds: Vmware Private Cloud, OpenStack, CloudStack

After provisioning cloud hosts, they are managed using Salt like every other host.

To manage Cloud hosts SaltStack uses *Providers*, *Profiles* and *Maps*.

When using *salt-cloud*, normally you should follow 4 steps:

- Look at the global *Salt Cloud* configuration
- Create your *Salt Cloud Providers* configuration files
- Create virtual machines *Salt Cloud Profiles*
- If needed, create your *Salt Cloud Map* files and start your virtual machines

Salt Cloud Installation

If you are using SaltStack 2014.1.0 or a superior version, *Salt Cloud* is a built-in module, otherwise, you should install *salt-cloud* with your package manager.

Example:

```
sudo apt-get install salt-cloud
```

I am supposing that you had already added Salt repository:

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

If you do not have *python-libcloud* installed, you should also install it:

```
sudo apt-get install python-libcloud
```

If you are installing *salt-cloud* for development, make sure to install *apache-libcloud*:

```
sudo apt-get install apache-libcloud
```

Salt Cloud Configurations

The *Salt Cloud* configuration resides in:

```
/etc/salt/cloud
```

and of course the main Salt configuration file:

```
/etc/salt/master
/etc/salt/minion
```

We are going to detail the usage of *Providers*, *Profiles* and *Maps* in the next part of this section, but the concepts are simple:

- *Providers* are for managing Cloud Providers
- *Profiles* are for configuring Cloud hosts
- *Maps* are used for mapping virtual machines to Profiles

The *salt-cloud* configuration file is similar to the following one:

```
# This file should normally be installed at: /etc/salt/cloud


##########################################
#####          VM Defaults           #####
##########################################

# Set the size of minion keys to generate, defaults to 2048
#
#keysize: 2048


# Set the default os being deployed. This sets which deployment script to
# apply. This argument is optional.
```

```
#
#script: bootstrap-salt


##########################################
#####         Logging Settings       #####
##########################################

# The location of the master log file
#
#log_file: /var/log/salt/cloud


# The level of messages to send to the console.
# One of 'garbage', 'trace', 'debug', info', 'warning', 'error', 'critical'.
#
# Default: 'info'
#
#log_level: info


# The level of messages to send to the log file.
# One of 'garbage', 'trace', 'debug', info', 'warning', 'error', 'critical'.
#
# Default: 'info'
#
#log_level_logfile: info


# 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: '%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] %(me\
ssage)s'


# Logger levels can be used to tweak specific loggers logging levels.
# For example, if you want to have the salt library at the 'warning' level,
# but you still wish to have 'salt.modules' at the 'debug' level:
#
#   log_granular_levels:
#     'salt': 'warning',
#     'salt.modules': 'debug'
#     'saltcloud': 'info'
#
#log_granular_levels: {}


##########################################
#####         Misc Defaults          #####
##########################################

# Whether or not to remove the accompanying SSH key from the known_hosts file
# when an instance is destroyed.
#
# Default: 'False'
#
#delete_sshkeys: False
```

The file is commented so no need to go through all definition but some important ones:

Salt Bootstrap

Salt Cloud uses

```
salt-bootstrap.sh
```

and this script can be found in the official Github account:

```
https://github.com/saltstack/salt-bootstrap
```

The bootstrap script installs dependencies and the *salt-minion* on the new Cloud virtual machine. You can find this file here:

```
/usr/lib/python2.7/dist-packages/salt/cloud/deploy/bootstrap-salt.sh
```

And if you are been using *Vagrant* and *Docker* with SaltStack as provisioner, you could find the bootstrap-salt.sh in other locations:

```
locate bootstrap-salt.sh

/opt/vagrant/embedded/gems/gems/vagrant-1.7.4/plugins/provisioners/salt/bootstra\
p-salt.sh
/usr/lib/python2.7/dist-packages/salt/cloud/deploy/bootstrap-salt.sh
/var/lib/docker/aufs/diff/62f00290f429277acf1c5695a2e0321e8e96dfb18b50fc4451b111\
26f632045d/usr/lib/python2.7/dist-packages/salt/cloud/deploy/bootstrap-salt.sh
```

Salt Cloud Logging

I am explaining this part because I recommend activating logs the first time you will use *Salt Cloud*. It will help you to keep traces and learn from them.

The master log file is located in

```
/var/log/salt/cloud
```

In the *Salt Cloud* configuration file, you can set the nature of logging, you have several choices: garbage, trace, debug, info, warning, error or critical.

I recommend also to activate the debugging mode in the console output. The same thing you have to choose the degree of the verbosity: garbage, trace, debug, info, warning, error or critical.

If you are familiar with *Python* and you want to change the format of the date when logging, you must change this line:

```
log_datefmt: '%Y-%m-%d %H:%M:%S'
```

Same thing for the log format:

```
log_fmt_console: '[%(levelname)-8s] %(message)s'
log_fmt_logfile: '%(asctime)s,%(msecs)03.0f [%(name)-17s][%(levelname)-8s] %(mes\
sage)s'
```

You can set up logging granularities like in the following example:

```
log_granular_levels:
  'salt': 'debug',
  'salt.modules': 'error'
  'saltcloud': 'trace'

### Salt Cloud Providers

Providers are configuration files where credentials for each used Cloud provider\
 are provided (examples: ec2 configuration).
If you want to use Digital Ocean, you should create a file in:

``` bash
/etc/salt/cloud.providers.d/digital_ocean.conf
```

Then you should edit it in order to have a similar configuration to the next one:

```
do:
  provider: digital_ocean
  minion:
    master: your_server_ip

  # DigitalOcean Access Token
  personal_access_token: your_access_token

  # The name of your SSH key
  ssh_key_name: salt-master-root-key

  # The rivate key for your Digital Ocean account                          \

  ssh_key_file: /root/.ssh/id_rsa
```

Note that the Digital Ocean example provided below could be different from AWS configuration file. Every provider has its own configuration schema.

Salt Cloud Profiles

Profiles are system configurations like hardware, location, OS, file systems ..etc (VM profiles)

Salt Cloud profiles are configured in the file:
```

```
/etc/salt/cloud.profiles
```

You can also create your configuration files in:

```
/etc/salt/cloud.profiles.d/*conf
```

Here is a simple example for creating a virtual machine on RackSpace:

```
ubuntu_rackspace:

    provider: rackspace
    image: Ubuntu 14.04
    size: 1024 server
    script: salt-bootstrap
    minion:
        master: salt.example.com
        append_domain: webs.example.com
        grains:
            role: webserver
```

And this an example for working with Azure Cloud:

```
azure-ubuntu:
  provider: azure-west
  image: 'Ubuntu-14_04-LTS-amd64-server-20140724-en-us-30GB'
  ssh_username: user_name
  ssh_password: user_password
  ssh_pubkey: /root/azure.pub
  media_link: 'https://example.blob.core.windows.net/vhds'
  slot: production
  size: Small
  tty: True
  sudo: True
```

Salt Cloud Maps

Map files have a simple format and they are useful at the same time. In *map* files, you can specify a profile and a list of virtual machines to create from said profile.

```
ubuntu_small:
  - web
  - rethindb
ubuntu_medium:
  - nginx
  - mongodb
ubuntu_large:
  - mysql
  - mariadb
```

A *map* file can also include *grains* configuration options:

```
ubuntu_micro:
  - web1:
      grains:
        grain_1: value_1
        grain_2: value_2
  - web2:
      grains:
        grain_1: value_1
        grain_2: value_2
```

A *map* file can also include minion configuration options:

```
ubuntu_micro:
  - web1:
      minion:
        log_level: error
  - web2:
      minion:
        log_level: debug
```

and of course the *map* file can contains both of *minions* and *grains* options:

```
ubuntu_micro:
  - web1:
      minion:
        log_level: error
      grains:
        grain_1: value_1
        grain_2: value_2
  - web2:
      minion:
        log_level: debug
      grains:
        grain_1: value_1
        grain_2: value_2
```

Maps files can be saved in:

```
/etc/salt/cloud.maps.d/*.conf
```

You can create it if it does not exist:

```
mkdir -p /etc/salt/cloud.maps.d
```

Map files can be called to roll out virtual machines using *salt-cloud* command with the -m option:

```
salt-cloud -m /etc/salt/cloud.maps.d/my_map.conf
```

You can use -P to parallelize the creation of virtual machines:

```
salt-cloud -m /etc/salt/cloud.maps.d/my_map.conf -P
```

If you want to destroy all virtual machines that are not mentioned in the *map* file, use the -H option (–hard):

```
salt-cloud -m /etc/salt/cloud.maps.d/my_map.conf -P -H
```

If you really want to use this dangerous option, you should activate it the cloud configuration file:

```
enable_hard_maps: True
```

Creating New Salt Master Cloud Hosts

Creating a salt-master in a Rackspace or AWS virtual machine cloud be done easily using *Salt Cloud maps*. After setting up Cloud Providers and Profiles, you can use Cloud *Maps* to start new Salt masters.

```
ubuntu_micro:
  - vm_1:
      make_master: True
  - vm_2
      make_master: True
  - vm_3
      make_master: True
  - vm_4
      make_master: True
```

In this example, you can notice that creating 4 Cloud virtual machines.

 When using a *Map* file, all bootstrapped *minions* will answer to the newly created *salt-master*.

 To make any of the bootstrapped *minions* answer to the another *salt-master* (not the newly created one), you should add this condition to the configuration file.

Here is a simple example where the created *vm_3 minion* responds to a different salt-master and not to the *vm_1 salt-master*:

```
centos_small:
  - vm_1:
      make_master: True
  - vm_2
  - vm_3:
      minion:
        master: master_vm_3.example.com
        local_master: True
```

Troubleshooting Salt Cloud

Salt Cloud is a built-in module in SaltStack, so basic troubleshooting is not different from troubleshooting Salt. First thing that may come to your mind is activating debugging:

```
salt-cloud -p ubuntu_micro minion_server -l debug
```

Salt Cloud uploads some files that are specific to *salt-minion* on your cloud VM, for debugging and troubleshooting reasons, it could be a good idea to keep them, because by default they are deleted.

```
salt-cloud -p ubuntu_micro minion_server --keep-tmp
```

You can find those file in the directory:

```
/tmp/.saltcloud/
```

Those temporary files are:

- An executable *deploy.sh* script
- .pem and .pub key that should be copied to */etc/salt/pki/minion/*
- *minion* file that should be copied to the */etc/salt/*
- *grains* file if you have used *grains* that should be copied to */etc/salt/*

Changing the log level to be more verbose is also a good idea if you want to troubleshoot Salt, you can update your configurations in *Salt Cloud* configuration file:

```
# The level of messages to send to the console.
# One of 'garbage', 'trace', 'debug', info', 'warning', 'error', 'critical'.
#
# Default: 'info'
#
#log_level: info


# The level of messages to send to the log file.
# One of 'garbage', 'trace', 'debug', info', 'warning', 'error', 'critical'.
#
# Default: 'info'
#
#log_level_logfile: info
```

Some problems may come from version compatibility as said in the official documentation:

> 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.

You cloud have problems due to the fact that the *salt-cloud* is not updated, in this case, you can type:

```
salt-cloud -u
```

A successful update will show you a similar output to the following one:

```
Success:
    ----------
    Files updated:
        - /etc/salt/cloud.deploy.d/bootstrap-salt.sh
        - /usr/lib/python2.7/dist-packages/salt/cloud/deploy/bootstrap-salt.sh
```

Real World Examples

```
    \   ^__^
     \  (--)\_______
        (__)\       )\/\
            ||----w |
            ||     ||
```

Introduction

One of the strengths of SaltStack is its richness and the large number of functionalities it offers. The previous chapters have introduced the main concepts about Salt, however, the purpose of this book is not to detail all about SaltStack but to bring you to the world of configuration management, remote execution, infrastructure automation using the different Salt modules.

Knowledge is not enough and practicing is important. That's why it is a good idea to practice, and the next practical examples will help you.

Vagrant Quick-Start


Note that we are using *Ubuntu 14.04* for the next tutorial.

To install and use *Vagrant*, the first thing you need to do is installing *VirtualBox*:

```
sudo apt-get install virtualbox
```

Install *Vagrant*:

```
sudo apt-get install vagrant
```

And the *VirtualBox* host kernel modules:

```
sudo apt-get install virtualbox-dkms
```

Create a work directory:

```
mkdir vagrantbox
cd vagrantbox
```

Create your first *Vagrant* machine:

```
vagrant box add precise32 http://files.vagrantup.com/precise32.box
vagrant init
```

Then edit the *Vagrantfile* in the same directory

```
config.vm.box = "precise32"
```

If you want to choose another OS/Distro or architecture, please visit this repository of *Vagrant* boxes.

Example:

```
config.vm.box = "precise32"
```

Launch the Vagrant virtual machine:

```
vagrant up
```

And start using it:

```
vagrant ssh
```

Creating A SaltStack Work Environments Using Vagrant

These instructions will help you set up a SaltStack test or development environment using *Vagrant*.

First, download and install *Vagrant*, go to the download page[33] and choose your distribution.

We are using a *Debian*/*Ubuntu* distribution for the next instructions:

[33] https://www.vagrantup.com/downloads.html

```
wget https://releases.hashicorp.com/vagrant/1.8.1/vagrant_1.8.1_x86_64.deb
dpkg -i vagrant_1.8.1_x86_64.deb
```

or use the second method:

```
sudo apt-get install virtualbox
sudo apt-get install vagrant
apt-get install virtualbox-dkms
```

Now get a local copy from *salt-vagrant-demo* github:

```
git clone https://github.com/UtahDave/salt-vagrant-demo.git
cd salt-vagrant-demo
vagrant up
```

> This will download an *Ubuntu VirtualBox* image and create three virtual machines for you. One will be a *salt-master* named *master* and two will be *salt-minion* instances named *minion1* and *minion2*. Each *salt-minion* will point to *master* and their respective keys will already be accepted. Because the keys are pre-generated and reside in the repository, please be sure to regenerate new keys if you use this for production purposes.

Now let's login into the master using ssh:

```
vagrant ssh master
```

Now if you type:

```
su
salt-key --list-all
```

You will find that you have 2 minions in the same *Vagrant* machine.

Let the master accept the two minions:

```
salt-key --accept-all
```

After accepting the two keys, you can verify by listing all minions:

```
Accepted Keys:
minion1
minion2
Denied Keys:
Unaccepted Keys:
Rejected Keys:
```

To verify that everything was ok:

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

And you should see something like the following output:

``` bash
minion2:
    True
minion1:
    True
```

Installation And Configuration Of Apache Web Server

One of the most important feature that *Vagrant* offers is its capability to interface with many automation and configuration management tools such as *Chef*, *Puppet* and of course *SaltStack*. You can, of course, use *shell*/*bash*/*python* scripts to provision your development machine.

In this part of "SaltStack For DevOps", we are more interested in using SaltStack standards as a provisioning tool. The combination of Salt and *Vagrant* allows you to create a powerful automated development environment.

You should have *Vagrant* installed before starting, then you need to install Salt plugin:

```
vagrant plugin install vagrant-salt
```

In this example we will use *Ubuntu Precise 64* as a guest virtual machine:

```
mkdir salt-vagrant_example
cd salt-vagrant_example
vagrant box add precise64 http://files.vagrantup.com/precise64.box
```

In this example, we are going to set up an *Apache* web server.

Create the following directories and files:

```
#The minion configuration file. Please copy the default minion configuration fil\
e from #/etc/salt/minion to your work directory.
salt/minion
#The minion grains
salt/minion.d/vagrant.conf
#The root folder where we will create the "top.sls" file and the other state fil\
es.
salt/roots
salt/roots/top.sls
salt/roots/apache/apache.sls
```

You workspace structure should look like this:

```
salt-vagrant_example
├── salt
|   ├── minion
|   ├── minion.d
|   |   └── vagrant.conf
|   └── roots
|       ├── apache2
|       |   ├── apache2.conf
|       |   └── apache2.sls
|       ├── top.sls
└── Vagrantfile
```

Let's take a look at *Vagrant* configuration file.

You will need to :

- Mount your Salt file root (synchronize salt/roots (in the host) to /srv/salt/ (in the guest).).
- Tell *Vagrant* that we are using SaltStack as a provisioning automation tool.


When we say "host" and "guest", we are referring respectively to your machine and to *Vagrant* machine.

The *Vagrant* configuration file (*Vagrantfile*) will look like this:

```
Vagrant.configure("2") do |config|
  ## Choose your base box
  config.vm.box = "precise64"

  ## For masterless, mount your salt file root
  config.vm.synced_folder "salt/roots/", "/srv/salt/"
  config.vm.synced_folder "salt/minion.d/", "/etc/salt/minion.d/"

  ## Use all the defaults:
  config.vm.provision :salt do |salt|

    salt.minion_config = "salt/minion"
    salt.run_highstate = true
    salt.grains_config = "salt/minion.d/vagrant.conf"

  end
end
```

Let's get back to Salt.

To create the *minion* configuration file, you would have made like this:

```
cp /etc/salt/minion ./salt/minion
```

In this example, you will be using Salt in its *masterless* mode, that's why you need to update your *minion* configuration:

```
file_client: local
```

The *Apache* installation *SLS* file should be called by your local *top.sls* file.

```
#Local top.sls file under salt/roots/top.sls
base:
  '*':
    - apache2/apache2
```

Now, let's create the *Apache SLS* file (salt/roots/apache2/apache2.sls):

```
# Install Apache
apache2:
  pkg.installed:
    - name: apache2
    - file: apache2
# Start the Apache service
  service.running:
    - enable: True
    - require:
      - pkg: apache2
# Copy the configuration template to */etc/apache2/apache2.conf* and render grai\
ns values.
  file.managed:
    - name: /etc/apache2/apache2.conf
    - source: salt://apache2/apache2.conf
    - template: jinja
    - require:
      - pkg: apache2
```

And the configuration template (*salt/roots/apache2/apache2.conf*)

```
LockFile ${APACHE_LOCK_DIR}/accept.lock
PidFile ${APACHE_PID_FILE}
Timeout {{ grains['apache_timeout']  }}
KeepAlive {{ grains['keep_alive'] }}
MaxKeepAliveRequests {{ grains['MaxKeepAliveRequests'] }}
KeepAliveTimeout {{ grains['KeepAliveTimeout'] }}

<IfModule mpm_prefork_module>
    StartServers          5
    MinSpareServers       5
    MaxSpareServers      10
    MaxClients          150
    MaxRequestsPerChild   0
</IfModule>

<IfModule mpm_worker_module>
    StartServers          2
    MinSpareThreads      25
    MaxSpareThreads      75
    ThreadLimit          64
    ThreadsPerChild      25
```

```
    MaxClients          150
    MaxRequestsPerChild   0
</IfModule>

<IfModule mpm_event_module>
    StartServers          2
    MinSpareThreads      25
    MaxSpareThreads      75
    ThreadLimit          64
    ThreadsPerChild      25
    MaxClients          150
    MaxRequestsPerChild   0
</IfModule>

User www-data
Group www-data
AccessFileName .htaccess

<Files ~ "^\.ht">
    Order allow,deny
    Deny from all
    Satisfy all
</Files>

DefaultType None
HostnameLookups Off
ErrorLog ${APACHE_LOG_DIR}/error.log
LogLevel warn
Include mods-enabled/*.load
Include mods-enabled/*.conf
Include ports.conf
LogFormat "%v:%p %h %l %u %t \"%r\" %>s %O \"%{Referer}i\" \"%{User-Agent}i\"" v\
host_combined
LogFormat "%h %l %u %t \"%r\" %>s %O \"%{Referer}i\" \"%{User-Agent}i\"" combined
LogFormat "%h %l %u %t \"%r\" %>s %O" common
LogFormat "%{Referer}i -> %U" referer
LogFormat "%{User-agent}i" agent
Include conf.d/
Include sites-enabled/
```

Note in the last configuration file, that we are using *grains*:

```
Timeout {{ grains['apache_timeout']  }}
KeepAlive {{ grains['keep_alive'] }}
MaxKeepAliveRequests {{ grains['MaxKeepAliveRequests'] }}
KeepAliveTimeout {{ grains['KeepAliveTimeout'] }}
```

Those *grains* will be rendered using the following file:

```
salt/minion.d/vagrant.conf
```

Let's see its content :

```
grains:

  apache_timeout: 600
  keep_alive: On
  MaxKeepAliveRequests: 150
  KeepAliveTimeout: 10
```

We are done with configurations.

Now bring you *Vagrant* machine up:

```
vagrant up
```


At this step, in the virtual machine, SaltStack (precisely *salt-minion*) will be installed and configured (*minion* configuration file) and state files will be copied (into */srv/salt/*).

Log in:

```
vagrant ssh
```

Now, you need to run SaltStack on your virtual machine in order to provision it :

```
vagrant@precise64$ sudo salt-call state.highstate
```

After executing *state.highstate* call, *Apache* will be installed and configured into your *Vagrant* VM. The lat

Creating You Own Private Storage Cloud (SaltStack + Vagrant + OwnCloud)

Before moving to another example, please note that while writing this book, SaltStack was not supported by *Vagrant* : The installation of *Salty Vagrant* was mandatory. Now there is no need for this, SaltStack is provided directly by *Vagrant*. In the following tutorial, we are not using *Salty Vagrant* anymore.

If you have the latest version of SaltStack, so it is sure that you will not need *Salty Vagrant* plugin. In the next example, we are going to use SaltStack to automate the provisioning of a *Vagrant* machine.

The main goal is to create a development machine for *OwnCloud*.

First of all, type:

```
vagrant init
```

Apart from *Vagrantfile*, created automatically after typing the last command, create the other necessary folders and files in order to have a similar tree:

```
├── salt
|   ├── formulas
|   |   └── owncloud
|   ├── minion.yml
|   └── roots
|       └── top.sls
└── Vagrantfile
```

Go to [OwnCloud Formula] (https://github.com/saltstack-formulas/owncloud-formula/), get the archive and extract it to

```
salt/formulas/owncloud/
```

Note that the tree should be like the next after extracting *OwnCloud* archive:

```
├── salt
│   ├── formulas
│   │   └── owncloud
│   │       ├── init.sls
│   │       ├── map.jinja
│   │       ├── mysql.sls
│   │       ├── python-mysqldb.sls
│   │       └── repo.sls
│   ├── minion.yml
│   └── roots
│       └── top.sls
└── Vagrantfile
```

Let configure *Vagrant*.

Put the following code in the *Vagrantfile*:

```
# -*- mode: ruby -*-
# vi: set ft=ruby :
Vagrant.configure(2) do |config|
  config.vm.box = "ubuntu/trusty64"
# The port 80 in the guest system will be accessed using the port 4567
  config.vm.network :forwarded_port, host: 4567, guest: 80
#Folders to synchronize between host and guest machines
  config.vm.synced_folder "salt/roots/", "/srv/salt/"
  config.vm.synced_folder "salt/formulas/", "/srv/formulas/"

#Salt Configuration
  config.vm.provision :salt do |salt|
    salt.verbose = true
    salt.minion_config = "salt/minion.yml"
    salt.run_highstate = true
    salt.colorize = true
    salt.log_level = 'all'
#OwnCloud Formula uses the next pillar pillar['owncloud']['owncloudpass'], which\
 is the password of
    salt.pillar({
      "owncloud" => {
        "owncloudpass" => "SetAPassword"
       }
    })
  end

```

```
27    config.vm.provider "virtualbox" do |v|
28  #You can give a name to your VM (should be unique)
29      v.name = "owncloud_dev"
30    end
31  end
```

In the *minion* configuration (minion.yml), we should configure the *master* to *localhost* and *file_client* to *local*:

```
1  master: localhost
2  file_client: local
3
4  file_roots:
5    base:
6      - /srv/salt
7      - /srv/formulas
```

And finally the *tops.sls* file:

```
1  base:
2    '*':
3      - owncloud
```

If everything is ok, you can start you VM:

```
1  vagrant up
```

You can find all of the code in a GitHub repository[34] that I created for this reason.

Use it by executing these command:

[34] https://github.com/eon01/OwnCloud_Vagrant_SaltStack

```
#Clone the git
git clone https://github.com/eon01/owncloud_vagrant_saltstack.git

#Cd into the folder
cd owncloud_vagrant_saltstack

#'Wake up' your machine :-)
vagrant up
```

Scheduling Monitoring Tasks Using SaltStack

In this part, we are using the scheduling feature, the built-in module *SALT.MODULES.SCHEDULE* will do the job for us after a simple configuration. Let's test how it works.

We will add a scheduled job that will do a simple task (write "test" at the end of a file).

Let's add the job, in our case it is the command:

```
echo "test" >> $HOME/temp
```

The first line of the next code will add a new job, the second part is the output (you should see something similar to this with "True" as "result").

```
mynode ~ # salt '*' schedule.add test_job function='cmd.run' job_args="['echo \"\
test\" >> /home/user/temp']" seconds=1
mynode:
    ----------
    comment:
        Added job: test_job to schedule.
    result:
        True
```

By default, the added job will not be enabled, we should add do this:

```
mynode ~ # salt '*' schedule.enable_job test_job
mynode:
    ----------
    comment:
        Enabled Job test_job in schedule.
    result:
        True
```

We can now see our job when we list all of the scheduled tasks:

```
mynode ~ # salt '*' schedule.list
mynode:
    schedule:
      enabled: false
      test_job:
        args:
        - echo "test" >> /home/user/temp
        enabled: true
        function: cmd.run
        jid_include: true
        maxrunning: 1
        name: test_job
        seconds: 1
```

Now let's enable the scheduling feature:

```
mynode ~ # salt '*' schedule.enable
mynode:
    ----------
    comment:
        Enabled schedule on minion.
    result:
        True
```

A last check:

```
mynode ~ # tail -f /home/user/temp
test
test
test
test
test
test
test
test
test
test
test
test
test
test
test
test
^C
```

Our last test should show us that *test* is added at the end of the file *temp* every second: Success!

Now monitoring a website is not very different from what we have done in the last example. We are just going to change the used command and capitalize on the scheduling feature because generally monitoring needs repetitive checks.

What we are going to execute as a scheduled job is just an example. Let's check it:

```
wget "www.eon01.com" --timeout 30 -O - 2>/dev/null | grep "dev & ops" || echo "T\
he site is down"  | mail -s "Your site is down" admin@eon01.com
```

The last command will send an email to *admin@eon01.com* when the string *dev & ops* will not be found: in other words, the site is down or not responding within 30 seconds (notice *–timeout* with *wget*).

Let's add this command the SaltStack *schedule* function:

```
mynode ~ # salt '*' schedule.add sched_monitoring function='cmd.run' job_args="[\
'wget \"www.eon01.com\" --timeout 30 -O - 2>/dev/null | grep \"dev & ops\" || ec\
ho \"The site is down\"  | mail -s \"Your site is down\" admin@eon01.com']" seco\
nds=10
mynode:
    ----------
    comment:
        Added job: sched_monitoring to schedule.
    result:
        True
```

```
mynode ~ # salt '*' schedule.enable_job sched_monitoring
mynode:
    ----------
    comment:
        Enabled Job sched_monitoring in schedule.
    result:
        True
```

```
mynode ~ # salt '*' schedule.disable_job sched_monitoring
mynode:
    ----------
    comment:
        Disabled Job sched_monitoring in schedule.
    result:
        True
```

Now, the *minion* will send an email if the site is down and the check will be done every 10 seconds.

As I said, this is an example, you can imagine many others, like monitoring disk with *df*, monitoring memory with *free* .. etc

I recommend to do a double check on every scheduled task:

```
* check the used resource (memory, disk ..etc) used by the *minion* after config\
uring the scheduled task.
* check the log file in : */var/log/salt/minion*
```

SaltStack *schedule* module offers other possibilities like copying one job to another :

```
salt '*' schedule.copy my_source_job my_target_job
```

Deleting jobs:

```
salt '*' schedule.delete my_job
```

Disabling a job:

```
salt '*' schedule.disable_job my_job
```

Moving a job from a *minion* to another *minion(s)*:

```
salt '*' schedule.move my_job target
```

This part of SaltStack is well explained, so I recommend you to see the other commands in [the official documentation (https://docs.saltstack.com/en/latest/ref/modules/all/salt.modules.schedule.html)

Automating Wordpress / LAMP Installation And Configuration

We can use *formulas* to install Apache, Mysql, and PHP. It's the fastest way you could use in this example. We have seen together how we could use this feature in *OwnCloud* example.

Apart from the advantage of the rapidity of installation and configuration provided by *formulas*, they are also maintained by an active community[35].

If you want to put again into practice the acquired knowledge through this book, this example could help you. In my point of view and I am sure this is shared with most of you, using *formulas* is quite easy but we are going to write our own *states* to automate the installation of *Wordpress* using a *LAMP* stack.

Note that this example is just a fast prototype, if you want to set it up in a production environment, you should focus more on details.

Configurations

We use two *Debian* servers, with *Apache2*, *MySQL-5.5*, *php5*, the files constituting the *Wordpress* and *wp-cli*[36]. The latter will be helpful to automate the installation and configuration of *Wordpress*.

Our two *Debian* servers are on the same subnet. The first one will host the *salt-master* and the other will host the *salt-minion*.

Considering that the IP of the *master* is:

[35] https://github.com/saltstack-formulas/

[36] https://github.com/wp-cli/wp-cli

```
10.10.10.41
```

We should start by changing the *minion* configuration file in order to tell *salt-minion* about the IP of *salt-master* :

```
master: 10.10.10.41
```

In the same server (*minion*), we set up the *file_roots*:

```
file_roots:
  base:
    - /srv/pillar
```

For the *grains* stored in the *minion*, we have the possibility to use files under:

```
/etc/salt/minion.d/*.conf
```

Let's create one:

```
/etc/salt/minion.d/web.conf
```

This file and the data stored within will be used later:

```
grains:
  apache_timeout: 600
  keep_alive: On
  MaxKeepAliveRequests: 150
  KeepAliveTimeout: 10
  html_dir: /var/www/html
  web_user: www-data
  db_name: wp_db
  blog_url: 10.10.10.41/blog
  blog_title: DevOps Blog
  admin_email: admin@myblog.com
```

 The above file should begin with *grains*.

Explanations:

- *apache_timeout*, *keep_alive*, *MaxKeepAliveRequests* and *KeepAliveTimeout* are some of *Apache2* configurations. You have the possibility to use the other configuration options.
- *html_dir* : Where websites file goes. The public *Apache2* server directory.
- *web_user* : A specific user for running *Apache2* and owner of *html_dir*
- *db_name* : The blog database name
- *blog_url* : The blog url
- *blog_title* : The blog title
- *admin_email* : The blog administrator email

In the *master* we will store 2 passwords:

```
mysql_password: TCaXs54rVaVujKqhpEL6
wp_password: zBQHFEKPJuTUuJRmbNhW
```

Those passwords should be declared in the following file:

```
/srv/pillar/passwords.sls
```

Installing and Configuring Apache

In this part, we are going to use the following file tree:

```
/srv/salt/apache2/
├── apache2.conf
└── apache2.sls
```

The file ‘apache2.sls’ contains the following information.

We tell Salt to install *Apache2*:

```
apache2:
  pkg.installed:
    - name: apache2
```

We verify that the service is running:

```
service.running:
  - name: apache2
  - enable: True
```

The *Apache2* package is installed, we are going to configure it :

```
file.managed:
  - name: /etc/apache2/apache2.conf
  - source: salt://apache2/apache2.conf
  - template: jinja
  - require:
  - pkg: apache2
```

According to the above configuration, the file :

```
/etc/apache2/apache2.conf
```

will be powered by a template file (written in *Jinja2*).

```
salt://apache2/apache2.conf
```

salt:// points to the *base* directory declared in *file_roots*. In our case, the previous notation is equivalent to the following:

```
/srv/salt/apache2/apache2.conf
```

The latter template will contain the *Apache2* server configuration:

```
LockFile ${APACHE_LOCK_DIR}/accept.lock
PidFile ${APACHE_PID_FILE}
Timeout {{ grains['apache_timeout']  }}
KeepAlive {{ grains['keep_alive'] }}
MaxKeepAliveRequests {{ grains['MaxKeepAliveRequests'] }}
KeepAliveTimeout {{ grains['KeepAliveTimeout'] }}

<IfModule mpm_prefork_module>
    StartServers          5
    MinSpareServers       5
```

```
    MaxSpareServers      10
    MaxClients          150
    MaxRequestsPerChild   0
</IfModule>

<IfModule mpm_worker_module>
    StartServers          2
    MinSpareThreads      25
    MaxSpareThreads      75
    ThreadLimit          64
    ThreadsPerChild      25
    MaxClients          150
    MaxRequestsPerChild   0
</IfModule>

<IfModule mpm_event_module>
    StartServers          2
    MinSpareThreads      25
    MaxSpareThreads      75
    ThreadLimit          64
    ThreadsPerChild      25
    MaxClients          150
    MaxRequestsPerChild   0
</IfModule>

User www-data
Group www-data
AccessFileName .htaccess

<Files ~ "^\.ht">
    Order allow,deny
    Deny from all
    Satisfy all
</Files>

DefaultType None
HostnameLookups Off
ErrorLog ${APACHE_LOG_DIR}/error.log
LogLevel warn
Include mods-enabled/*.load
Include mods-enabled/*.conf
Include ports.conf
```

```
LogFormat "%v:%p %h %l %u %t \"%r\" %>s %O \"%{Referer}i\" \"%{User-Agent}i\"" v\
host_combined
LogFormat "%h %l %u %t \"%r\" %>s %O \"%{Referer}i\" \"%{User-Agent}i\"" combined
LogFormat "%h %l %u %t \"%r\" %>s %O" common
LogFormat "%{Referer}i -> %U" referer
LogFormat "%{User-agent}i" agent
Include conf.d/
Include sites-enabled/
```

Should not be confused the *Apache2* variables like *${APACHE_LOCK_DIR}* and *Jinja2* variables like *{{ grains['apache_timeout'] }}*.

What we have finally is :

```
apache2:
  pkg.installed:
    - name: apache2
    - file: apache2
  service.running:
    - enable: True
    - require:
      - pkg: apache2
  file.managed:
    - name: /etc/apache2/apache2.conf
    - source: salt://apache2/apache2.conf
    - template: jinja
    - require:
    - pkg: apache2
```

You may ask the question if our server was running *Redhat* or *FreeBSD* (or any other *GNU/Linux* family) instead of *Debian*. This is why the *grain os_family* could be helpful in this case.

```
{% if grains['os_family']=="Debian"  %}
[Debian specefic configuraton]
{ % endif %}
```

To create a generic configuration, I recommend you to see the file *map.jinja* in *saltstack-formula* github repository:

```
{% set apache = salt['grains.filter_by']({
    'Debian': {
        'server': 'apache2',
        'service': 'apache2',
        'mod_wsgi': 'libapache2-mod-wsgi',
        'vhostdir': '/etc/apache2/sites-available',
        'confdir': '/etc/apache2/conf.d',
        'confext': '',
        'default_site': 'default',
        'default_site_ssl': 'default-ssl',
        'logdir': '/var/log/apache2',
        'wwwdir': '/srv',
        'use_require': False,
    },
    'RedHat': {
        'server': 'httpd',
        'service': 'httpd',
        'mod_wsgi': 'mod_wsgi',
        'vhostdir': '/etc/httpd/conf.d',
        'confdir': '/etc/httpd/conf.d',
        'confext': '.conf',
        'default_site': 'default',
        'default_site_ssl': 'default-ssl',
        'logdir': '/var/log/httpd',
        'wwwdir': '/var/www',
        'use_require': False,
    },
    'FreeBSD': {
        'server': 'apache22',
        'service': 'apache22',

        'mod_wsgi': 'ap22-mod_wsgi3',

        'vhostdir': '/usr/local/etc/apache22/Includes',
        'confdir': '/usr/local/etc/apache22/Includes',
        'confext': '',
        'default_site': 'default',
        'default_site_ssl': 'default-ssl',
        'logdir': '/var/log/',
        'wwwdir': '/usr/local/www/apache22/',
        'use_require': False,
    },
```

```
}, merge=salt['grains.filter_by']({
    '14.04': {
        'confext': '.conf',
        'default_site': '000-default.conf',
        'default_site_ssl': 'default-ssl.conf',
        'use_require': True,
    },
    '14.10': {
        'confext': '.conf',
        'default_site': '000-default.conf',
        'default_site_ssl': 'default-ssl.conf',
        'use_require': True,
    },
}, grain='lsb_distrib_release', merge=salt['pillar.get']('apache:lookup'))) %}
```

In this case, the *SLS* file could use variables that adapt to many *contexts* :

```
{% from "apache/map.jinja" import apache with context %}

apache:
  pkg.installed:
    - name: {{ apache.server }}
  service.running:
    - name: {{ apache.service }}
- enable: True
```

We will be working with *Debian* for the remaining parts.

In the 'SLS' file :

```
/srv/salt/apache2/apache2.sls
```

We are going to backup the *apache2.conf* configuration file :

```
backup_conf:
  cmd.wait:
    - name: "cp /etc/apache2/apache2.conf /etc/apache2/apache2.conf.backup"
    - watch:
    - pkg: apache2
```

In the code above, the *cmd.wait* instruction will execute a command

```
cp /etc/apache2/apache2.conf /etc/apache2/apache2.conf.backup
```

but only when the package *Apache2* is installed :

```
- watch:
  - pkg: apache2
```

Afterward, we create a user for the web server. This is a recommended step for better security as you may know. The user will have the name *www-data* and will be added to the *www-data* group.

```
user_creation:
  group:
    - www-data
  user:
    - www-data
    - home: /var/www/html/
    - groups:
      - www-data
    - require:
    - group: www-data
```

Finally, the *apache2/apache2.sls* file will be similar to this :

```
{% if grains['os_family']=="debian"  %}
apache2:
  pkg.installed:
    - name: apache2
    - file: apache2
  service.running:
    - enable: True
    - require:
      - pkg: apache2
  file.managed:
    - name: /etc/apache2/apache2.conf
    - source: salt://apache2/apache2.conf
    - template: jinja
    - require:
      - pkg: apache2

backup_conf:
  cmd.wait:
```

```
    - name: "cp /etc/apache2/apache2.conf /etc/apache2/apache2.conf.backup"
    - watch:
      - pkg: apache2

user_creation:
  group:
    - www-data
  user:
    - www-data
    - home: /var/www/html
    - groups:
      - www-data
    - require:
      - group: www-data
{% endif %}
```

Installing and Configuring Mysql

Create this file tree:

```
/srv/salt/mysql/
└── mysql.sls
```

In the *SLS* file we will tell Salt to install *mysql-server*, *php5-mysql* and *python-mysqldb*.

```
mysql:
  pkg.installed:
    - name: mysql-server
  service.running:
    - enable: True
    - require:
      - pkg: mysql-server

php5-mysql:
  pkg.installed:
    - name: php5-mysql

python-mysqldb:
  pkg.installed
```

Like SaltStack is based on Python, the package *python-mysqldb* will help us manage the installation, configuration and all the operations where a Python-based software should connect to a *Mysql* database.

When we want to install a package, we are not obliged to write its name in " - name " if the identifier has the same name as the package. Look a the difference between *mysql python-mysqldb* installation.

However, while not mandatory, it is highly recommended that you set a password for the *MySQL* administrative *root* user. My method is to prepopulate password fields before running the install process.

```
debconf:
  cmd.run:
    - name: sudo debconf-set-selections <<< "mysql-server mysql-server/root_pass\
word password {{ pillar['mysql_password'] }}"
debconf_bis:
  cmd.run:
    - name: sudo debconf-set-selections <<< "mysql-server mysql-server/root_pass\
word_again password {{ pillar['mysql_password'] }}"
```

The package "phpMyAdmin" is automatically installed with the MySQL server on Debian server, just make accessible from a web browser.

```
{% if 0 == salt['cmd.retcode']("test -d {{ grains['html_dir'] }}/phpmyadmin") %}
sl_phpmyadmin:
  cmd.run:
    - name: "ln -s /usr/share/phpmyadmin {{ grains['html_dir'] }}/phpmyadmin"
    - require:
      - service: mysql
{% endif %}
```

The latter code will check if the folder:

```
/var/www/html/phpmyadmin
```

exists.

Otherwise, it will create a symbolic link from the phpMyAdmin installation directory to the root directory of websites. Since some prefer to have

```
/var/www/
```

as a root directory for sites, others prefer to use a default configuration:

```
/var/www/html/
```

that's why the *html_dir* is a grain. This means that you can adapt your configuration according to user's preferences.

To create a symbolic link, SaltStack gives us the possibility to use *file.symlink* function instead of executing the system command *ln -s*.

```
/path/to/file:
  file.symlink:
    - target: /path/to/target
```

For now, we are going to create a database for our blog:

```
create_db:
  cmd.run:
    - name: mysql -u root -p{{ pillar['mysql_password'] }} -e "CREATE DATABASE I\
F NOT EXISTS {{ grains['db_name'] }}"
```

This is the equivalent of creating a Mysql database using:

```
mysql -u <user> -p<password> -e "<sql query>"
```

According to the configuration file:

```
/etc/salt/minion.d/web.conf
```

the database will have *wp_db* as a name.

If you do not know *mysqltuner.pl*, this is the opportunity to take a look at this script. I was using *Mysql* and *MariaDB* since years and I found that it is really useful to use *mysqltuner.pl* to make security and performance audits.

To use it just download and execute it:

```
wget https://raw.githubusercontent.com/major/MySQLTuner-perl/master/mysqltuner.p\
l -O mt.pl; chmod +x mt.pl; perl mt.pl > mt.log
```

Because we can execute GNU/Linux commands directly from SaltStack, we are going to include the last command in our configuration files:

```
mysql_tuner:
  cmd:
    - run
    - cwd: /tmp
    - name: "wget https://raw.githubusercontent.com/major/MySQLTuner-perl/master\
/mysqltuner.pl -O mt.pl; perl mt.pl>mt.log"
    - require:
    - service: mysql
```

We will get finally a *mysql.sls* file that looks like this:

```
debconf:
  cmd.run:
    - name: sudo debconf-set-selections <<< "mysql-server mysql-server/root_pass\
word password {{ pillar['mysql_password'] }}"

debconf_bis:
  cmd.run:
    - name: sudo debconf-set-selections <<< "mysql-server mysql-server/root_pass\
word_again password {{ pillar['mysql_password'] }}"


mysql:
  pkg.installed:
    - name: mysql-server
  service.running:
    - enable: True
    - require:
      - pkg: mysql-server

php5-mysql:
  pkg.installed:
    - name: php5-mysql

python-mysqldb:
  pkg.installed

{% if 0 == salt['cmd.retcode']('test -d {{ grains['html_dir'] }}/phpmyadmin') %}

sl_phpmyadmin:
  cmd.run:
    - name: "ln -s /usr/share/phpmyadmin {{ grains['html_dir'] }}/phpmyadmin"
```

```
    - require:
      - service: mysql
{% endif %}

create_db:
  cmd.run:
    - name: mysql -u root -p{{ pillar['mysql_password'] }} -e "CREATE DATABASE I\
F NOT EXISTS {{ grains['db_name'] }}"

mysql_tuner:
  cmd:
    - run
    - cwd: /tmp
    - name: "wget https://raw.githubusercontent.com/major/MySQLTuner-perl/master\
/mysqltuner.pl -O mt.pl; perl mt.pl>mt.log"
    - require:
      - service: mysql
```

Installing and Configuring PHP

As usual, we will proceed with the creation of the file tree:

```
/srv/salt/php5/
└── php5.sls
```

The *SLS* file is quite simple:

```
php5:
  pkg.installed:
    - name: php5
php5-cli:
  pkg.installed:
    - name: php5-cli
```

In the last configuration, Salt is asked to install *php5-cli* and *php5*. The first is needed to run dynamic web pages in *PHP* under *Apache2* and the second is necessary to execute *PHP* commands (In our case it will be used with *wp-cli*).

Installing and Configuring Wordpress

First of all, we create the necessary files:

```
/srv/salt/wordpress/
└── wp-cli.sls
```

The second step is to tell Salt to create the directory of our *Wordpress* installation:

```
{{ grains['html_dir'] }}/blog:
  file.directory:
    - user: www-data
    - group: www-data
    - mode: 755
    - makedirs: False
    - recurse:
      - user
      - group
      - mode
```

The directory will be created in:

```
/var/www/html/blog
```

The last directory should be owned (for security reasons) by *www-data* user and the *www-data* group, it will have *755* as a *chmod* configuration. The usage of *recurse* will apply this configuration (user, group, and mode) recursively.

With *makedirs* the directory will not be created if it is already there.

We will then download *wp-cli* and use it to download the *Wordpress* files to *html_dir*.

```
wp_cli_download:
  cmd.run:
    - cwd: {{ grains['html_dir'] }}/blog
    - name: curl -O https://raw.githubusercontent.com/wp-cli/builds/gh-pages/pha\
r/wp-cli.phar > wp-cli.phar

wp_cli_mod:
  cmd.run:
    - cwd: {{ grains['html_dir'] }}/blog
    - name: chmod +x wp-cli.phar

wp_cli_path:
  cmd.run:
    - cwd: {{ grains['html_dir'] }}/blog
```

```
    - name: mv wp-cli.phar /usr/local/bin/wp

core_download:
  cmd.run:
    - cwd: {{ grains['html_dir'] }}/blog
    - name: wp core download --force
    - user: {{ grains['web_user'] }}
```

A *Wordpress* website is based on the configuration in the *wp-config.php* file. We will check if the file already exists and in this case we will delete it to re-create and re-configure it:

```
{% if 0 == salt['cmd.retcode']("test -s {{ grains['html_dir'] }}/blog/wp-config.\
php") %}

wp_config_remove:
  cmd.run:
    - cwd: {{ grains['html_dir'] }}/blog
    - name: rm -f {{ grains['html_dir'] }}/blog/wp-config.php
    - force: True

wp_config:
  cmd.run:
    - cwd: {{ grains['html_dir'] }}/blog
    - name: wp core config --dbname={{ grains['db_name']  }} --dbuser=root --dbp\
ass={{ pillar['mysql_password'] }} --skip-check
    - user: {{ grains['web_user'] }}
    - force: True
{% endif %}
```

Note that other *wp-cli* commands can be found in its online official documentation. By typing the command *wp*, you will have an almost full control of your *Wordpress* website.

All commands issued by Salt must be executed with the user *www-data* and this is why we use:

```
user: {{ grains['web_user'] }}
```

Remember that 'grains' are stored in:

```
/etc/salt/minion.d/*.conf
```

and that we created:

```
/etc/salt/minion.d/web.conf
```

with the following configuration:

```
grains:
  apache_timeout: 600
  keep_alive: On
  MaxKeepAliveRequests: 150
  KeepAliveTimeout: 10
  html_dir: /var/www/html
  web_user: www-data
  db_name: wp_db
  blog_url: 10.10.10.41/blog
  blog_title: DevOps Blog
  admin_email: admin@myblog.com
```

Scheduling Wordpress Monitoring Tasks

You can just use the example detailed in this chapter about scheduling monitoring tasks.

We have used the following command to add a monitoring task:

```
salt '*' schedule.add sched_monitoring function='cmd.run' job_args="['wget \"www\
.eon01.com\" --timeout 30 -O - 2>/dev/null | grep \"dev & ops\" || echo \"The si\
te is down\"  | mail -s \"Your site is down\" admin@eon01.com']" seconds=10
```

Here is another example to use 'Curl' with detailed output about your infrastructure, network, and application.

First create this file:

```
echo -e \
“time_namelookup:  %{time_namelookup}\n
time_connect:  %{time_connect}\n
time_appconnect:  %{time_appconnect}\n
time_pretransfer:  %{time_pretransfer}\n
time_redirect:  %{time_redirect}\n
time_starttransfer:  %{time_starttransfer}\n
----------\n
time_total:  %{time_total}\n” > curl_detailed_response.conf
```

We will add the next command to the scheduled tasks list:

```
curl -w "@curl_detailed_response.conf" -o /dev/null -s "http://10.10.10.41/blog"
```

This is how we do:

```
salt '*' schedule.add sched_monitoring function='cmd.run' job_args="['curl -w \"\
@curl_detailed_response.conf\" -o /dev/null -s \"http://10.10.10.41/blog\" >> cu\
rl_results.log']"
```

It is recommended to go back to the *Scheduling Monitoring Tasks Using SaltStack* part of this book.

Well, you can also imagine other scenarios to monitor where you will put into practice what you have learned through this book.

Conclusion

Finally, you need to execute:

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

to execute the installation of your *Wordpress* blog with *Apache2*, *PHP* and *Mysql*. Everything will be configured like you have set it. A good manner to do things with Salt is to debug before launching anything especially if you are running hotfixes in a production environment:

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

This part of SaltStack For DevOps detailed the various steps in order to practice a concrete example of an installation and a post-installation of a *LAMP* stack with a website that runs, but the security and production environments constraints were not taken into consideration at 100%, so be careful, test before you deploy and think before you type.

Docker Quick-Start

To run Docker, I used Ubuntu 14.04 and a 64-bit installation (required), and I followed the official tutorial from Docker documentation. Normally you can run it on any other Ubuntu version or distributions without problems unless your Linux Kernel is older than 3.10.

Just type the following command to be sure:

```
uname -r
```

If it is ok, add the gpg key:

```
sudo apt-key adv --keyserver hkp://p80.pool.sks-keyservers.net:80 --recv-keys 58\
118E89F3A912897C070ADBF76221572C52609D
```

then ad the repository to your source list:

Create the *docker.list* file if it does not exist:

```
touch /etc/apt/sources.list.d/docker.list
```

then

```
echo “deb https://apt.dockerproject.org/repo ubuntu-trusty main” | tee -a /etc/a\
pt/sources.list.d/docker.list
```

and finally install Docker:

```
apt-get update && apt-get purge lxc-docker && apt-get install docker-engine
```

If you want to use *Docker* with your current user, type:

```
sudo usermod -aG docker your_user_name
```

otherwise, for the next instructions you should use the *root* user:

```
su
```

For more details about installing *Docker* on *Ubuntu* check the official documentation[37].

Getting Docker Container System Information Using SaltStack

Now that ‘Docker’ is installed, let’s create a directory as our workspace:

```
mkdir docker-salt; cd docker-salt
```

If you think about this, there are many ways to work with SaltStack and Docker:

- Install a *masterless minion* inside the *Docker* guest and use *salt-call* to have an established remote-execution environment inside *Docker*.
- Do the same thing but mount your local */srv/salt/* to your guest
- Control the guest *salt-minion* with the host *salt-master*

In this book, I tried to keep it stupidly simple (KISS) and I avoided unnecessary complexity, that’s why we will consider the first scenario and then you will see that it is easy to use the second.

Let’s create the guest *salt-minion*:

[37] https://docs.docker.com/engine/installation/ubuntulinux/

```
echo "file_client: local" > minion
```

Create your *Dockerfile* with your favorite code editor (*vi*, *vim*, *nano* ..etc) and as a basic example, let's put this code:

```
FROM ubuntu:14.04

MAINTAINER Aymen El Amri amri.aymen@gmail.com

RUN apt-get update

RUN DEBIAN_FRONTEND=noninteractive apt-get install -y -q wget
RUN wget -O - https://repo.saltstack.com/apt/ubuntu/14.04/amd64/latest/SALTSTACK\
-GPG-KEY.pub | sudo apt-key add -
RUN echo "deb http://repo.saltstack.com/apt/ubuntu/14.04/amd64/latest trusty mai\
n"|tee -a /etc/apt/sources.list

RUN apt-get update
RUN DEBIAN_FRONTEND=noninteractive apt-get install -y -q salt-minion

ADD minion /etc/salt/

RUN service salt-minion start

EXPOSE 8080
CMD /bin/bash
```

This *Dockerfile* will install *Ubuntu*, *salt-minion* and will copy the *minion* configuration file to */etc/minion* on the *Docker* machine before starting *salt-minion*.

Make sure you are in the same directory containing *Dockerfile* and build your image that we will call *docker-salt*:

```
docker build --rm=true -t docker-salt .
```

Run the container from the built image:

```
docker run  -t -i -d -h docker-salt-host  --name docker-salt-container  docker-s\
alt
```

The last commands will run the image called *docker-salt* on a container that we called *docker-salt-container*. The Docker guest machine will have *docker-salt-host* as a hostname (*/etc/hostname*).

Make sure your container is up.

When you type

```
1 docker ps
```

you should see it:

```
1 CONTAINER ID        IMAGE                COMMAND             CREATED           \
2  STATUS              PORTS               NAMES
3 fa4112583e73        docker-salt:latest   "/bin/bash"         2 seconds ago     \
4  Up 2 seconds                           docker-salt-container
```

In general *CONTAINER ID*(in our case *fa4112583e73*) could be used also when working with *Docker*.

To test if everything is ok, let's execute *salt-call* inside the container.

Remember, *salt-call* command is used to run SaltStack functions on a *masterless minion*.

Let's get some system/networking information about our host like (*SSDs*, *bios* release date, *bios* version, *cpu* flags, *cpu* model, *cpu* architecture, *fqdn*, *hostname*, network interfaces, *kernel* and *kernel* release ..etc

One simple command:

```
1 docker exec -t -i  docker-salt-container salt-call grains.items
```

will print all the above information:

```
local:
    ----------
    SSDs:
    biosreleasedate:
        /dev/mem: No such file or directory
    biosversion:
        /dev/mem: No such file or directory
    cpu_flags:
        - fpu
        - vme
        - de
        - pse
        - tsc
        - msr
        - pae
        - mce
        - cx8
        - apic
        - sep
        - mtrr
        - pge
        - mca
        - cmov
        - pat
        - pse36
        - clflush
        - dts
        - acpi
        - mmx
        - fxsr
        - sse
        - sse2
        - ss
        - ht
        - tm
        - pbe
        - syscall
        - nx
        - pdpe1gb
        - rdtscp
        - lm
        - constant_tsc
```

```
        - arch_perfmon
        - pebs
        - bts
        - rep_good
        - nopl
        - xtopology
        - nonstop_tsc
        - aperfmperf
        - eagerfpu
        - pni
        - pclmulqdq
        - dtes64
        - monitor
        - ds_cpl
        - vmx
        - est
        - tm2
        - ssse3
        - fma
        - cx16
        - xtpr
        - pdcm
        - pcid
        - sse4_1
        - sse4_2
        - movbe
        - popcnt
        - tsc_deadline_timer
        - aes
        - xsave
        - avx
        - f16c
        - rdrand
        - lahf_lm
        - abm
        - ida
        - arat
        - epb
        - xsaveopt
        - pln
        - pts
        - dtherm
```

```
        - tpr_shadow
        - vnmi
        - flexpriority
        - ept
        - vpid
        - fsgsbase
        - tsc_adjust
        - bmi1
        - avx2
        - smep
        - bmi2
        - erms
        - invpcid
    cpu_model:
        Intel(R) Core(TM) i5-4200U CPU @ 1.60GHz
    cpuarch:
        x86_64
    domain:
    fqdn:
        docker-salt-host
    fqdn_ip4:
        - 172.17.0.10
    fqdn_ip6:
    gpus:
    host:
        docker-salt-host
    hwaddr_interfaces:
        ----------
        eth0:
            02:42:ac:11:00:0a
        lo:
            00:00:00:00:00:00
    id:
        docker-salt-host
    init:
        unknown
    ip4_interfaces:
        ----------
        eth0:
            - 172.17.0.10
        lo:
            - 127.0.0.1
```

```
    ip6_interfaces:
        ----------
        eth0:
            - fe80::42:acff:fe11:a
        lo:
            - ::1
    ip_interfaces:
        ----------
        eth0:
            - 172.17.0.10
            - fe80::42:acff:fe11:a
        lo:
            - 127.0.0.1
            - ::1
    ipv4:
        - 127.0.0.1
        - 172.17.0.10
    ipv6:
        - ::1
        - fe80::42:acff:fe11:a
    kernel:
        Linux
    kernelrelease:
        3.16.0-38-generic
    locale_info:
        ----------
        defaultencoding:
            None
        defaultlanguage:
            None
        detectedencoding:
            ANSI_X3.4-1968
    localhost:
        docker-salt-host
    lsb_distrib_codename:
        trusty
    lsb_distrib_description:
        Ubuntu 14.04.3 LTS
    lsb_distrib_id:
        Ubuntu
    lsb_distrib_release:
        14.04
```

```
    manufacturer:
        /dev/mem: No such file or directory
    master:
        salt
    mdadm:
    mem_total:
        7636
    nodename:
        docker-salt-host
    num_cpus:
        4
    num_gpus:
        0
    os:
        Ubuntu
    os_family:
        Debian
    osarch:
        amd64
    oscodename:
        trusty
    osfinger:
        Ubuntu-14.04
    osfullname:
        Ubuntu
    osrelease:
        14.04
    osrelease_info:
        - 14
        - 4
    path:
        /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
    productname:
        /dev/mem: No such file or directory
    ps:
        ps -efHww
    pythonexecutable:
        /usr/bin/python
    pythonpath:
        - /usr/bin
        - /usr/lib/python2.7
        - /usr/lib/python2.7/plat-x86_64-linux-gnu
```

```
        - /usr/lib/python2.7/lib-tk
        - /usr/lib/python2.7/lib-old
        - /usr/lib/python2.7/lib-dynload
        - /usr/local/lib/python2.7/dist-packages
        - /usr/lib/python2.7/dist-packages
    pythonversion:
        - 2
        - 7
        - 6
        - final
        - 0
    saltpath:
        /usr/lib/python2.7/dist-packages/salt
    saltversion:
        2015.8.3
    saltversioninfo:
        - 2015
        - 8
        - 3
        - 0
    serialnumber:
        /dev/mem: No such file or directory
    server_id:
        324894828
    shell:
        /bin/sh
    virtual:
        physical
    virtual_subtype:
        Docker
    zmqversion:
        4.0.4
```

Monitoring Docker Containers Using SaltStack

We will not have many changes in our *Vagrantfile*, just installing *psutil* library that Salt uses to get system metrics.

```
FROM ubuntu:14.04

MAINTAINER Aymen El Amri amri.aymen@gmail.com

RUN apt-get update

RUN DEBIAN_FRONTEND=noninteractive apt-get install -y -q wget
RUN wget -O - https://repo.saltstack.com/apt/ubuntu/14.04/amd64/latest/SALTSTACK\
-GPG-KEY.pub | sudo apt-key add -
RUN echo "deb http://repo.saltstack.com/apt/ubuntu/14.04/amd64/latest trusty mai\
n"|tee -a /etc/apt/sources.list

RUN apt-get update
RUN DEBIAN_FRONTEND=noninteractive apt-get install -y -q salt-minion

RUN apt-get install python-dev -y
RUN wget https://bootstrap.pypa.io/get-pip.py
RUN python get-pip.py
RUN /usr/bin/yes | pip install psutil

ADD minion /etc/salt/

RUN service salt-minion start

EXPOSE 8080
CMD /bin/bash
```

Build and run then you can get monitoring information easily:

- *Boot* time:

```
docker exec -t -i  docker-salt-container salt-call ps.boot_time --output json
```

Output

```
{
    "local": 1450992213
}
```

- *CPU* usage percentage:

```
docker exec -t -i  docker-salt-container salt-call ps.cpu_percent --output json
```

Output

```
{
    "local": 25.0
}
```

- *Disk I/O*:

```
docker exec -t -i  docker-salt-container salt-call ps.disk_io_counters --output \
json
```

Output

```
{
    "local": {
        "read_time": 819864,
        "write_bytes": 3998978048,
        "read_bytes": 1336084480,
        "write_time": 2441316,
        "read_count": 54610,
        "write_count": 146160
    }
}
```

You can use other function like:

- disk_partition_usage
- cpu_times
- disk_partition_usage
- disk_partitions
- disk_usage
- get_pid_list
- get_users
- network_io_counters
- num_cpus
- proc_info <pid>
- swap_memory
- top
- total_physical_memory
- virtual_memory

Provisioning Docker Containers With SaltStack

Follow the previous steps to set up your *Docker* build, then the container where the *masterless salt-minion* is installed.

```
docker build --rm=true -t docker-salt .
docker run  -t -i -d -h docker-salt-host  --name docker-salt-container  docker-s\
alt
```

Make sure your container is running using *ps*:

```
docker ps -l
```

You should see a container with the name *docker-salt-container*:

```
CONTAINER ID        IMAGE               COMMAND                CREATED         \
    STATUS              PORTS               NAMES
43591d8dff92        docker-salt:latest   "/bin/sh -c /bin/bas   11 minutes ago  \
    Up 11 minutes       8080/tcp            docker-salt-container
```


Note that if you want to stop your container you should use the following command:

```
docker stop docker-salt-container
```


Stopping a container is not removing it. If you would like to remove it type:

```
docker rm docker-salt-container
```

Now we want to install packages inside the running container using SaltStack. Well, it is simple, let's install *vim*:

```
docker exec -t -i  docker-salt-container  salt-call pkg.install vim
```

You should see an output similar to the following one:

```
[INFO    ] Executing command ['dpkg-query', '--showformat', '${Status} ${Package\
} ${Version} ${Architecture}\n', '-W'] in directory '/root'
[INFO    ] Executing command ['apt-get', '-q', '-y', '-o', 'DPkg::Options::=--fo\
rce-confold', '-o', 'DPkg::Options::=--force-confdef', 'install', 'vim'] in dire\
ctory '/root'
[INFO    ] Executing command ['dpkg-query', '--showformat', '${Status} ${Package\
} ${Version} ${Architecture}\n', '-W'] in directory '/root'
local:
    ----------
    libgpm2:
        ----------
        new:
            1.20.4-6.1
        old:
    vim:
        ----------
        new:
            2:7.4.052-1ubuntu3
        old:
    vim-runtime:
        ----------
        new:
            2:7.4.052-1ubuntu3
        old:
```

Let's see what was the version of *vim*:

```
docker exec -t -i  docker-salt-container  salt-call  pkg.version "vim"
```

Output:

```
[INFO    ] Executing command ['dpkg-query', '--showformat', '${Status} ${Package\
} ${Version} ${Architecture}\n', '-W'] in directory '/root'
local:
    2:7.4.052-1ubuntu3
```

If you would like to remove *vim*, just type:

```
docker exec -t -i  docker-salt-container  salt-call  pkg.remove "vim"
```

and it will be removed from system:

```
[INFO    ] Executing command ['dpkg-query', '--showformat', '${Status} ${Package\
} ${Version} ${Architecture}\n', '-W'] in directory '/root'
local:
```


Please note that if the container exits you will lose data: vim will not be installed. You should commit changes to your container. Type the next command:

```
docker commit 5ac064ad5d91 docker-salt
```

This will commit the changes happened in the container with the mentioned id to the image called *docker-salt*. Of course, replace *5ac064ad5d91* by you container id (get it with *docker ps* command). Note that *docker-salt* is the image name.

Let's verify if everything is committed to the image, we stop the image, remove it, run it again and get *vim* version. If we will have the version then the package is installed (even after removing the container) and if we are not going to see a version, this certainly means that *vim* is not installed.

```
docker stop docker-salt-container
docker rm docker-salt-container
docker run  -t -i -d -h docker-salt-host  --name docker-salt-container  docker-s\
alt
docker exec -t -i  docker-salt-container  salt-call  pkg.version "vim"
```

Using Salt Cloud With Linode Cloud

Linode (a portmanteau of the words *Linux* and node) is a virtual private server/cloud hosting provider offering *SSD Linux* servers and simple management tools.

Since we are just learning, we will be using the *Linode 1GB* server so that everyone reading this book could be able to work with *Salt Cloud* on a real remote server without paying too much.

	RAM	CPU	Storage	Transfer	Network In	Network Out	Price	
Linode **1GB**	1 GB	1 Core	24 GB SSD	2 TB	40 Gbps	125 Mbps	**$.015 / hr** ($10 / mo)	Sign Up
Linode **2GB**	2 GB	2 Cores	48 GB SSD	3 TB	40 Gbps	250 Mbps	**$.03 / hr** ($20 / mo)	Sign Up
Linode **4GB**	4 GB	4 Cores	96 GB SSD	4 TB	40 Gbps	500 Mbps	**$.06 / hr** ($40 / mo)	Sign Up
Linode **8GB**	8 GB	6 Cores	192 GB SSD	8 TB	40 Gbps	1000 Mbps	**$.12 / hr** ($80 / mo)	Sign Up
Linode **16GB**	16 GB	8 Cores	384 GB SSD	16 TB	40 Gbps	2000 Mbps	**$.24 / hr** ($160 / mo)	Sign Up
Linode **32GB**	32 GB	12 Cores	768 GB SSD	20 TB	40 Gbps	4000 Mbps	**$.48 / hr** ($320 / mo)	Sign Up
Linode **48GB**	48 GB	16 Cores	1152 GB SSD	20 TB	40 Gbps	6000 Mbps	**$.72 / hr** ($480 / mo)	Sign Up
Linode **64GB**	64 GB	20 Cores	1536 GB SSD	20 TB	40 Gbps	8000 Mbps	**$.96 / hr** ($640 / mo)	Sign Up
Linode **96GB**	96 GB	20 Cores	1920 GB SSD	20 TB	40 Gbps	10000 Mbps	**$1.44 / hr** ($960 / mo)	Sign Up

Linode Pricing

Follow these steps to set up a Linode key that will be used with *Salt Cloud*:

- Go to Linode[38]
- Create a free account
- After confirming the account from your email, log in
- Click on *My Profile*
- Re-authenticate to unlock the settings
- Click on *API keys*
- Generate a key (I gave it the name *salt_cloud_demo*)


Save the generated key on your local machine since you will not be able to view it again on the web manager.

Please note that I will use this file structure when I work with *Salt Cloud*:

[38] https://manager.linode.com/session/signup

```
/etc/salt/
├── cloud
├── cloud.conf.d/
├── cloud.deploy.d/
├── cloud.maps.d/
├── cloud.profiles.d/
```

Let's set up the *provider* configuration file for Linode:

```
/etc/salt/cloud.providers.d/linode.providers.conf
```

Linode Provider Configuration

We are using a simple configuration that I called *linode-config*.

```
linode-config:
  provider: linode
  apikey: O61KWcOQ
  password: **********
  ssh_pubkey: ssh-rsa xzt4Gg3WjHZgFvRU8EMUyVSxwwilKogg0TK0BkbY0gq+IADKMxhuq9yW4O\
nxlWcO+n2VkdaFZlAIh+/Sll7HdfmBc1vGbw== root@eonSpider
  ssh_key_file: /root/.ssh/id_rsa
```

As you can see in the configuration, we are using the *apikey* provided by Linode with the password. You should also configure the *ssh_pubkey* that you are using on your machine (in this example, I am executing *Salt Cloud* command from my local machine on a remote Linode server). If you do not have a public key, generate a key pair (public/private) using this command:

```
ssh-keygen
```

Linode Profile Configuration

To create a Linode server using *Salt Cloud*, you need to configure a *profile* for a VM.

Before trying that, it will be a good idea to test if we do not have problems with the *provider* configuration. To test it, let's play with some commands.

Let's view the different Linode sizes:

```
salt-cloud --list-sizes linode-config
```

You should see something like this:

```
linode-config:
    ----------
    linode:
        ----------
        Linode 1024:
            ----------
            bandwidth:
                2000
            disk:
                24576
            driver:
            extra:
                ----------
            get_uuid:
            id:
                1
            name:
                Linode 1024
            price:
                10.0
            ram:
                1024
            uuid:
                03e18728ce4629e2ac07c9cbb48afffb8cb499c4
        Linode 16384:
            ----------
            bandwidth:
                16000
            disk:
                393216
            driver:
            extra:
                ----------
            get_uuid:
            id:
                7
            name:
                Linode 16384
            price:
                160.0
            ram:
                16384
```

```
            uuid:
                a93683ef7f41a3a7ccb14777d95f03bb6b046ec6
        Linode 2048:
            ----------
            bandwidth:
                3000
            disk:
                49152
            driver:
            extra:
                ----------
            get_uuid:
            id:
                2
            name:
                Linode 2048
            price:
                20.0
            ram:
                2048
            uuid:
                5a1c572b48a26aa78799facaad413d901a73a673
        Linode 32768:
            ----------
            bandwidth:
                20000
            disk:
                786432
            driver:
            extra:
                ----------
            get_uuid:
            id:
                8
            name:
                Linode 32768
            price:
                320.0
            ram:
                32768
            uuid:
                b3586dbcfcb67d3aba9093c472b852bc860846a6
```

```
        Linode 4096:
            ----------
            bandwidth:
                4000
            disk:
                98304
            driver:
            extra:
                ----------
            get_uuid:
            id:
                4
            name:
                Linode 4096
            price:
                40.0
            ram:
                4096
            uuid:
                c08ae025461ecc5f845ede0d95e1cb3e61a8f13b
        Linode 49152:
            ----------
            bandwidth:
                20000
            disk:
                1179648
            driver:
            extra:
                ----------
            get_uuid:
            id:
                9
            name:
                Linode 49152
            price:
                480.0
            ram:
                49152
            uuid:
                954b931001923ab473861c7d427863c3b1df8de7
        Linode 65536:
            ----------
```

```
            bandwidth:
                20000
            disk:
                1572864
            driver:
            extra:
                ----------
            get_uuid:
            id:
                10
            name:
                Linode 65536
            price:
                640.0
            ram:
                65536
            uuid:
                ca78fb5bfed412bd33d434dbd06340de377423c8
        Linode 8192:
            ----------
            bandwidth:
                8000
            disk:
                196608
            driver:
            extra:
                ----------
            get_uuid:
            id:
                6
            name:
                Linode 8192
            price:
                80.0
            ram:
                8192
            uuid:
                9501e716d5ee4502bbeaecf0d4daa2fe2862200f
        Linode 98304:
            ----------
            bandwidth:
                20000
```

```
            disk:
                1966080
            driver:
            extra:
                ----------
            get_uuid:
            id:
                12
            name:
                Linode 98304
            price:
                960.0
            ram:
                98304
            uuid:
                d0dc3fdd2a6cdf54aff0e0213e493dff6d80e5cf
```

If you had the latter output, congratulation, your configuration works.

Let's try to see the distributions that we can use:

```
salt-cloud --list-images linode-config
```

You will see the images that you can use on your Linode server:

```
linode-config:
    ----------
    linode:
        ----------
        Arch Linux 2015.02:
            ----------
            driver:
            extra:
                ----------
                64bit:
                    1
                pvops:
                    1
            get_uuid:
            id:
                138
            name:
```

```
                Arch Linux 2015.02
            uuid:
                74a8e6d46c4e30312ab39aa499702dbc615db1ea
        Arch Linux 2015.08:
            ----------
            driver:
            extra:
                ----------
                64bit:
                    1
                pvops:
                    0
            get_uuid:
            id:
                142
            name:
                Arch Linux 2015.08
            uuid:
                7fb9cf7b1ece5101e4056f9de9fc7f1945cde0ef
        CentOS 5.6:
            ----------
            driver:
            extra:
                ----------
                64bit:
                    1
                pvops:
                    1
            get_uuid:
            id:
                60
            name:
                CentOS 5.6
            uuid:
                f17af5e85ef920dba6c54efb1f7e059bb814e17a
        CentOS 6.5:
            ----------
            driver:
            extra:
                ----------
                64bit:
                    1
```

```
60                  pvops:
61                      1
62              get_uuid:
63              id:
64                  127
65              name:
66                  CentOS 6.5
67              uuid:
68                  f12d308795a507cc73a3cf5f7aacdf2d86fbcf4a
69          CentOS 7:
70              ----------
71              driver:
72              extra:
73                  ----------
74                  64bit:
75                      1
76                  pvops:
77                      1
78              get_uuid:
79              id:
80                  129
81              name:
82                  CentOS 7
83              uuid:
84                  87e25228eddb0c32db87b502b3bf4a9aaf64d15c
85          Debian 6:
86              ----------
87              driver:
88              extra:
89                  ----------
90                  64bit:
91                      1
92                  pvops:
93                      1
94              get_uuid:
95              id:
96                  78
97              name:
98                  Debian 6
99              uuid:
100                 87d8bc1d0236a365f444138cf4158e404c121e75
101         Debian 7:
```

```
            ----------
            driver:
            extra:
                ----------
                64bit:
                    1
                pvops:
                    1
            get_uuid:
            id:
                130
            name:
                Debian 7
            uuid:
                2619a13cd4e5bc5f10fa04b617231c7085167e6a
        Debian 8.1:
            ----------
            driver:
            extra:
                ----------
                64bit:
                    1
                pvops:
                    1
            get_uuid:
            id:
                140
            name:
                Debian 8.1
            uuid:
                57ccf3e88161c13987b6f98976c7bdf791237dbf
        Fedora 20:
            ----------
            driver:
            extra:
                ----------
                64bit:
                    1
                pvops:
                    1
            get_uuid:
            id:
```

```
                122
            name:
                Fedora 20
            uuid:
                b69c15db35d7b987ecf8d6d206c47b22b97e2e81
        Fedora 21:
            ----------
            driver:
            extra:
                ----------
                64bit:
                    1
                pvops:
                    1
            get_uuid:
            id:
                134
            name:
                Fedora 21
            uuid:
                35a3258a19bbaf60f137ffbefcf06475e145a0d8
        Fedora 22:
            ----------
            driver:
            extra:
                ----------
                64bit:
                    1
                pvops:
                    1
            get_uuid:
            id:
                141
            name:
                Fedora 22
            uuid:
                fd93443f93e0248202f9f2e2dad0935697537f20
        Gentoo 2013-11-26:
            ----------
            driver:
            extra:
                ----------
```

```
                64bit:
                    1
                pvops:
                    1
            get_uuid:
            id:
                118
            name:
                Gentoo 2013-11-26
            uuid:
                aec7ded0f24b21dc761d3eacaa2f696c07067a56
        Gentoo 2014.12:
            ----------
            driver:
            extra:
                ----------
                64bit:
                    1
                pvops:
                    1
            get_uuid:
            id:
                137
            name:
                Gentoo 2014.12
            uuid:
                a69dd4adb372e2b1599c4fab23994df70ca3253f
        Slackware 13.37:
            ----------
            driver:
            extra:
                ----------
                64bit:
                    1
                pvops:
                    1
            get_uuid:
            id:
                87
            name:
                Slackware 13.37
            uuid:
```

```
            8ad9944af892c7a355f43062b0056652e4aea626
    Slackware 13.37 32bit:
        ----------
        driver:
        extra:
            ----------
            64bit:
                0
            pvops:
                1
        get_uuid:
        id:
            86
        name:
            Slackware 13.37 32bit
        uuid:
            83c081a5c8d2355942b6cd1b03efee145559c546
    Slackware 14.1:
        ----------
        driver:
        extra:
            ----------
            64bit:
                1
            pvops:
                1
        get_uuid:
        id:
            117
        name:
            Slackware 14.1
        uuid:
            076e7f332f204bf9ae08896bf3d10a2e464a054b
    Ubuntu 12.04 LTS:
        ----------
        driver:
        extra:
            ----------
            64bit:
                1
            pvops:
                1
```

```
            get_uuid:
            id:
                126
            name:
                Ubuntu 12.04 LTS
            uuid:
                f5cadba59311f026e739794ae454d04228323aa9
        Ubuntu 14.04 LTS:
            ----------
            driver:
            extra:
                ----------
                64bit:
                    1
                pvops:
                    1
            get_uuid:
            id:
                124
            name:
                Ubuntu 14.04 LTS
            uuid:
                18be6ebe9bb4f9a818f95a522ac213cfdf295e84
        Ubuntu 15.04:
            ----------
            driver:
            extra:
                ----------
                64bit:
                    1
                pvops:
                    1
            get_uuid:
            id:
                139
            name:
                Ubuntu 15.04
            uuid:
                28c371ac941cf095abd16c29591d271d2417a925
        Ubuntu 15.10:
            ----------
            driver:
```

```
            extra:
                ----------
                64bit:
                    1
                pvops:
                    1
            get_uuid:
            id:
                144
            name:
                Ubuntu 15.10
            uuid:
                e9884caca236aa493211b58cecfe5513f1f949fc
        openSUSE 13.1:
            ----------
            driver:
            extra:
                ----------
                64bit:
                    1
                pvops:
                    1
            get_uuid:
            id:
                120
            name:
                openSUSE 13.1
            uuid:
                e676f73133eb146eafb0df9218ef6520f5b01b56
        openSUSE 13.2:
            ----------
            driver:
            extra:
                ----------
                64bit:
                    1
                pvops:
                    1
            get_uuid:
            id:
                135
            name:
```

```
                openSUSE 13.2
            uuid:
                29d348ebca4fb639eaa4809939ecef4bbfe5b9c9
```

You can request other information like the different location that you can choose when deploying a new Linode machine:

```
salt-cloud --list-location linode-config
```

Here is the output:

```
linode-config:
    ----------
    linode:
        ----------
        Atlanta, GA, USA:
            ----------
            country:
                US
            driver:
            id:
                4
            name:
                Atlanta, GA, USA
        Dallas, TX, USA:
            ----------
            country:
                US
            driver:
            id:
                2
            name:
                Dallas, TX, USA
        Frankfurt, DE:
            ----------
            country:
                ??
            driver:
            id:
                10
            name:
                Frankfurt, DE
```

```
        Fremont, CA, USA:
            ----------
            country:
                US
            driver:
            id:
                3
            name:
                Fremont, CA, USA
        London, England, UK:
            ----------
            country:
                GB
            driver:
            id:
                7
            name:
                London, England, UK
        Newark, NJ, USA:
            ----------
            country:
                US
            driver:
            id:
                6
            name:
                Newark, NJ, USA
        Singapore, SG:
            ----------
            country:
                ??
            driver:
            id:
                9
            name:
                Singapore, SG
        Tokyo, JP:
            ----------
            country:
                JP
            driver:
            id:
```

```
                8
            name:
                Tokyo, JP
```

Salt Cloud offers many others possibilities to interface with *Linode API*, the following list of functionalities can be used:

```
salt.cloud.clouds.linode.avail_images(call=None)
salt.cloud.clouds.linode.avail_locations(call=None)
salt.cloud.clouds.linode.avail_sizes(call=None)
salt.cloud.clouds.linode.boot(name=None, kwargs=None, call=None)
salt.cloud.clouds.linode.clone(kwargs=None, call=None)
salt.cloud.clouds.linode.create(vm_)
salt.cloud.clouds.linode.create_config(kwargs=None, call=None)
salt.cloud.clouds.linode.create_disk_from_distro(vm_, linode_id, swap_size=None)
salt.cloud.clouds.linode.create_private_ip(vm_, linode_id)
salt.cloud.clouds.linode.create_swap_disk(vm_, linode_id, swap_size=None)
salt.cloud.clouds.linode.destroy(name, call=None)
salt.cloud.clouds.linode.get_config_id(kwargs=None, call=None)
salt.cloud.clouds.linode.get_configured_provider()
salt.cloud.clouds.linode.get_datacenter_id(location)
salt.cloud.clouds.linode.get_disk_size(vm_, swap, linode_id)
salt.cloud.clouds.linode.get_distribution_id(vm_)
salt.cloud.clouds.linode.get_ips(linode_id=None)
salt.cloud.clouds.linode.get_linode(kwargs=None, call=None)
salt.cloud.clouds.linode.get_linode_id_from_name(name)
salt.cloud.clouds.linode.get_password(vm_)
salt.cloud.clouds.linode.get_plan_id(kwargs=None, call=None)
salt.cloud.clouds.linode.get_private_ip(vm_)
salt.cloud.clouds.linode.get_pub_key(vm_)
salt.cloud.clouds.linode.get_swap_size(vm_)
salt.cloud.clouds.linode.get_vm_size(vm_)
salt.cloud.clouds.linode.list_nodes(call=None)
salt.cloud.clouds.linode.list_nodes_full(call=None)
salt.cloud.clouds.linode.list_nodes_min(call=None)
salt.cloud.clouds.linode.list_nodes_select(call=None)
salt.cloud.clouds.linode.reboot(name, call=None)
salt.cloud.clouds.linode.show_instance(name, call=None)
salt.cloud.clouds.linode.show_pricing(kwargs=None, call=None)
salt.cloud.clouds.linode.start(name, call=None)
salt.cloud.clouds.linode.stop(name, call=None)
salt.cloud.clouds.linode.update_linode(linode_id, update_args=None)
```

You can find more details about this on the *Salt Linode* documentation page[39].

Using Salt Cloud To Automate Linode VPS Creation

We had already made a progress concerning the configurations of *Salt Linode* as a provider, and now we are going to test the *profile*.

The profile will allow us to create Cloud VMs so let's create a profile for a testing machine.

This is the configuration of our testing VM: - We are going to choose a *Linode 1024* server - We are going to install Ubuntu 14.04 TLS - The region will be Frankfurt - Our new machine will be set up as a minion and its master will be your local machine - This machine is for testing purpose, we will add a grain called *role* with the value *testing_server*


To configure the master of the new minion machine you can just put your IP. To get your IP without leaving your terminal, you can use the following command: *curl icanhazip.com*.

According to our description below, the *Salt Cloud profile* file will be similar to the next one:

```
ubuntu_linode:
  provider: linode-config
  size: Linode 1024
  image: Ubuntu 14.04 LTS
  location: Frankfurt, DE
  minion:
    master: 20a1:e35:9beb:6a00:2293:5229:2400:a57f # This is an IP v6.
  grains:
    role: testing_server
```

Nice! Let's start our machine *testing_server* based on *ubuntu_linode* profile:

```
sudo salt-cloud -p ubuntu_linode testing_server
```

You can debug your output:

```
sudo salt-cloud -p ubuntu_linode testing_server -l debug
```

You will be able to see the detailed logs ef the execution:

[39] https://docs.saltstack.com/en/latest/ref/clouds/all/salt.cloud.clouds.linode.html

```
DEBUG   ] Reading configuration from /etc/salt/cloud
[DEBUG   ] Reading configuration from /etc/salt/master
[DEBUG   ] Using cached minion ID from /etc/salt/minion_id: eonSpider
[DEBUG   ] Missing configuration file: /etc/salt/cloud.providers
[DEBUG   ] Including configuration from '/etc/salt/cloud.providers.d/linode.prov\
iders.conf'
[DEBUG   ] Reading configuration from /etc/salt/cloud.providers.d/linode.provide\
rs.conf
[DEBUG   ] Missing configuration file: /etc/salt/cloud.profiles
[DEBUG   ] Including configuration from '/etc/salt/cloud.profiles.d/linode.profi\
les.conf'
[DEBUG   ] Reading configuration from /etc/salt/cloud.profiles.d/linode.profiles\
.conf
[DEBUG   ] Configuration file path: /etc/salt/cloud
[INFO    ] salt-cloud starting
[DEBUG   ] Could not LazyLoad parallels.avail_sizes
[DEBUG   ] LazyLoaded parallels.avail_locations
[DEBUG   ] LazyLoaded proxmox.avail_sizes
[DEBUG   ] Could not LazyLoad saltify.destroy
[DEBUG   ] Could not LazyLoad saltify.avail_sizes
[DEBUG   ] Could not LazyLoad saltify.avail_images
[DEBUG   ] Could not LazyLoad saltify.avail_locations
[DEBUG   ] LazyLoaded rackspace.reboot
[DEBUG   ] LazyLoaded openstack.list_locations
[DEBUG   ] LazyLoaded rackspace.list_locations
[DEBUG   ] Could not LazyLoad linode.optimize_providers
[DEBUG   ] The 'linode' cloud driver is unable to be optimized.
[DEBUG   ] Could not LazyLoad parallels.avail_sizes
[DEBUG   ] LazyLoaded parallels.avail_locations
[DEBUG   ] LazyLoaded proxmox.avail_sizes
[DEBUG   ] Could not LazyLoad saltify.destroy
[DEBUG   ] Could not LazyLoad saltify.avail_sizes
[DEBUG   ] Could not LazyLoad saltify.avail_images
[DEBUG   ] Could not LazyLoad saltify.avail_locations
[DEBUG   ] LazyLoaded rackspace.reboot
[DEBUG   ] LazyLoaded openstack.list_locations
[DEBUG   ] LazyLoaded rackspace.list_locations
[DEBUG   ] Generating minion keys for 'testing_server'
[DEBUG   ] MasterEvent PUB socket URI: ipc:///var/run/salt/master/master_event_p\
ub.ipc
[DEBUG   ] MasterEvent PULL socket URI: ipc:///var/run/salt/master/master_event_\
pull.ipc
```

```
[DEBUG   ] Sending event - data = {'_stamp': '2016-02-25T18:34:26.465790'}
[DEBUG   ] Sending event - data = {'profile': 'ubuntu_linode', 'event': 'startin\
g create', '_stamp': '2016-02-25T18:34:27.281133', 'name': 'testing_server', 'pr\
ovider': 'linode-config:linode'}
[INFO    ] Creating Cloud VM testing_server
[DEBUG   ] MasterEvent PUB socket URI: ipc:///var/run/salt/master/master_event_p\
ub.ipc
[DEBUG   ] MasterEvent PULL socket URI: ipc:///var/run/salt/master/master_event_\
pull.ipc
[DEBUG   ] Sending event - data = {'_stamp': '2016-02-25T18:34:32.476974'}
[DEBUG   ] Sending event - data = {'_stamp': '2016-02-25T18:34:32.490534', 'even\
t': 'requesting instance', 'kwargs': {'name': 'testing_server', 'ex_rsize': 2444\
8, 'image': 'Ubuntu 14.04 LTS', 'ex_swap': 128, 'ex_private': False, 'size': 'Li\
node 1024', 'location': 'Frankfurt, DE'}}
[INFO    ] Rendering deploy script: /usr/lib/python2.7/dist-packages/salt/cloud/\
deploy/bootstrap-salt.sh
[DEBUG   ] MasterEvent PUB socket URI: ipc:///var/run/salt/master/master_event_p\
ub.ipc
[DEBUG   ] MasterEvent PULL socket URI: ipc:///var/run/salt/master/master_event_\
pull.ipc
[DEBUG   ] Sending event - data = {'_stamp': '2016-02-25T18:34:45.385425'}
[DEBUG   ] Sending event - data = {'_stamp': '2016-02-25T18:34:45.386326', 'even\
t': 'executing deploy script', 'kwar
...
...
...
 *  INFO: System Information:
 *  INFO:   CPU:          GenuineIntel
 *  INFO:   CPU Arch:     x86_64
 *  INFO:   OS Name:      Linux
 *  INFO:   OS Version:   4.4.0-x86_64-linode63
 *  INFO:   Distribution: Ubuntu 14.04
...
...
...
DEBUG   ] Connection to 1.1.1.1 closed.
[DEBUG   ] Removed /tmp/.saltcloud-b0a0a690-2106-44f2-a81a-25d16487f2bb/minion
[DEBUG   ] MasterEvent PUB socket URI: ipc:///var/run/salt/master/master_event_p\
ub.ipc
[DEBUG   ] MasterEvent PULL socket URI: ipc:///var/run/salt/master/master_event_\
pull.ipc
[DEBUG   ] Sending event - data = {'_stamp': '2016-02-25T18:36:51.304932'}
```

```
[DEBUG   ] Sending event - data = {'host': '1.1.1.1', 'name': 'testing_server', \
'_stamp': '2016-02-25T18:36:51.305734', 'event': 'testing_server has been deploy\
ed at 1.1.1.1'}
[INFO    ] Salt installed on testing_server
[INFO    ] Created Cloud VM 'testing_server'
[DEBUG   ] 'testing_server' VM creation details:
<Node: uuid=737ff2078c4e822e9b992c9de99e94d23f80a331, name=testing_server, state\
=3, public_ips=['1.1.1.1'], private_ips=[], provider=Linode ...>
[DEBUG   ] MasterEvent PUB socket URI: ipc:///var/run/salt/master/master_event_p\
ub.ipc
[DEBUG   ] MasterEvent PULL socket URI: ipc:///var/run/salt/master/master_event_\
pull.ipc
[DEBUG   ] Sending event - data = {'_stamp': '2016-02-25T18:36:51.357829'}
[DEBUG   ] Sending event - data = {'profile': 'ubuntu_linode', 'event': 'created\
 instance', '_stamp': '2016-02-25T18:36:51.362601', 'name': 'testing_server', 'p\
rovider': 'linode-config:linode'}
[DEBUG   ] LazyLoaded nested.output
```

Here is an example without activating the debug:

```
testing_server:
    ----------
    _uuid:
        None
    deployed:
        True
    driver:
    extra:
        ----------
        ALERT_BWIN_ENABLED:
            1
        ALERT_BWIN_THRESHOLD:
            10
        ALERT_BWOUT_ENABLED:
            1
        ALERT_BWOUT_THRESHOLD:
            10
        ALERT_BWQUOTA_ENABLED:
            1
        ALERT_BWQUOTA_THRESHOLD:
            80
        ALERT_CPU_ENABLED:
```

```
            1
        ALERT_CPU_THRESHOLD:
            90
        ALERT_DISKIO_ENABLED:
            1
        ALERT_DISKIO_THRESHOLD:
            10000
        BACKUPSENABLED:
            0
        BACKUPWEEKLYDAY:
            0
        BACKUPWINDOW:
            0
        CREATE_DT:
            2016-02-25 13:49:49.0
        DATACENTERID:
            10
        DISTRIBUTIONVENDOR:
            Ubuntu
        ISKVM:
            1
        ISXEN:
            0
        LABEL:
            testing_server
        LINODEID:
            1682323
        LPM_DISPLAYGROUP:
        PLANID:
            1
        STATUS:
            0
        TOTALHD:
            24576
        TOTALRAM:
            1024
        TOTALXFER:
            2000
        WATCHDOG:
            1
    id:
        1682323
```

```
    image:
        None
    name:
        testing_server
    private_ips:
    public_ips:
        - 1.1.1.1
    size:
        None
    state:
        3
```

Now you can check either your *salt-minion* is set up by using:

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

The result:

```
testing_server:
    True
```

Congratulation, you have a running *salt-minion* Linode server.

If you want to set another profile, you can use the same profile file. In the next example, we are creating a testing server and a production server:

```
ubuntu_linode_test:
  provider: linode-config
  size: Linode 1024
  image: Ubuntu 14.04 LTS
  location: Frankfurt, DE
  minion:
    master: 92.169.23.1
  grains:
    role: testing_server

ubuntu_linode_production:
  provider: linode-config
  size: Linode 1024
  image: Ubuntu 14.04 LTS
  location: Frankfurt, DE
  minion:
```

```
    master: 92.169.23.1
  grains:
    role: production_server
```

You can also create two separate files *ubuntu_linode_test* and *ubuntu_linode_production*.

To create multiple Linode machines having the same profile, you can just simply type:

```
sudo salt-cloud -p ubuntu_linode_test test_machine_1 test_machine_2
```

Note that you can use the *-P* to create the two machines in parallel.

```
sudo salt-cloud -P -p ubuntu_linode_test test_machine_1 test_machine_2
```

In the next step, we are going to use *Salt Cloud Maps*.

Imagine you have a list of VMs to create across multiple cloud providers. Provisioning your infrastructure means creating multiple machines with different configurations. When using *Salt Cloud Maps*, creating your infrastructure is easier than creating it with only *Providers* and *Profiles*. In the map file, any VMs that already exist will remain the same and will not be modified and any new VMs will be created.

To keep things organized, we will save all of our *Salt Cloud Maps* in the following directory:

```
/etc/salt/cloud.maps.d/
```

Imagine we had already set several profiles with Linode as a cloud provider. - linode_1024 - linode_2048 - linode_4096

and we would like to create three environments: - Development - Staging - Production

and in every environment we need 3 servers: - Nginx - Mysql - Redis

This map file will simplify the creation of this infrastructure:

```
linode_1024:
  - redis_dev
  - redis_staging
  - redis_production
linode_2048:
  - nginx_dev
  - nginx_staging
  - nginx_production
linode_4096:
  - mysql_dev
  - mysql_staging
  - mysql_production
```

Then we save the file under:

```
/etc/salt/cloud.maps.d/my_infrastructure.maps.conf
```

To create those servers we type:

```
sudo salt-cloud -m /etc/salt/cloud.maps.d/my_infrastructure.maps.conf
```

Parallelize the last operation like this:

```
sudo salt-cloud -m /etc/salt/cloud.maps.d/my_infrastructure.maps.conf -P
```

Debug it like this:

```
sudo salt-cloud -m /etc/salt/cloud.maps.d/my_infrastructure.maps.conf -P -l debug
```

And send the debug to a file with

```
--out-file
```

Manipulating Linode VMs with Salt Cloud

With *Salt Cloud* you can reboot, destroy and execute other operations on your remote Cloud VMs. To destroy a VM:

```
salt-cloud -d minion_server
```

To reboot it:

```
salt-cloud -a reboot minion_server
```

To stop it:

```
salt-cloud -a stop minion_server
```

To get details about your created VMs:

```
salt-cloud --query
```

To clone a VM:

```
salt-cloud -f clone linode-config linode_id=123123 datacenter_id=1 plan_id=1
```

You can even get the pricing:

```
salt-cloud -f show_pricing linode-config profile=linode-profile
```

You have probably noticed that sometimes with use *-a* and sometimes *-f*. The difference is that:

- *a* is for the action like executing operations
- *f* is for the function like getting information

This is the list of all actions by provider:

```
all providers:
    - reboot
ec2:
    - start
    - stop
joyent:
    - stop
linode:
    - start
    - stop
```

Functions are more specific to each provider (or driver) and need specific names due to the difference between Cloud providers *APIs*, but there are three universal functions:

- *list_nodes*: To get general information about the instances for a given provider
- *list_nodes_full*: Same thing as *list_nodes* but with more information
- *list_nodes_select*: Returns select information about the instances for the given provider

This is a complete list of what you can do with *Linode Rest API*:

```
salt.cloud.clouds.linode.avail_images(call=None)
salt.cloud.clouds.linode.avail_locations(call=None)
salt.cloud.clouds.linode.avail_sizes(call=None)
salt.cloud.clouds.linode.boot(name=None, kwargs=None, call=None)
salt.cloud.clouds.linode.clone(kwargs=None, call=None)
salt.cloud.clouds.linode.create(vm_)
salt.cloud.clouds.linode.create_config(kwargs=None, call=None)
salt.cloud.clouds.linode.create_disk_from_distro(vm_, linode_id, swap_size=None)
salt.cloud.clouds.linode.create_private_ip(vm_, linode_id)
salt.cloud.clouds.linode.create_swap_disk(vm_, linode_id, swap_size=None)
salt.cloud.clouds.linode.destroy(name, call=None)
salt.cloud.clouds.linode.get_config_id(kwargs=None, call=None)
salt.cloud.clouds.linode.get_configured_provider()
salt.cloud.clouds.linode.get_datacenter_id(location)
salt.cloud.clouds.linode.get_disk_size(vm_, swap, linode_id)
salt.cloud.clouds.linode.get_distribution_id(vm_)
salt.cloud.clouds.linode.get_ips(linode_id=None)
salt.cloud.clouds.linode.get_linode(kwargs=None, call=None)
salt.cloud.clouds.linode.get_linode_id_from_name(name)
salt.cloud.clouds.linode.get_password(vm_)
salt.cloud.clouds.linode.get_plan_id(kwargs=None, call=None)
salt.cloud.clouds.linode.get_private_ip(vm_)
salt.cloud.clouds.linode.get_pub_key(vm_)
salt.cloud.clouds.linode.get_swap_size(vm_)
salt.cloud.clouds.linode.get_vm_size(vm_)
salt.cloud.clouds.linode.list_nodes(call=None)
salt.cloud.clouds.linode.list_nodes_full(call=None)
salt.cloud.clouds.linode.list_nodes_min(call=None)
salt.cloud.clouds.linode.list_nodes_select(call=None)
salt.cloud.clouds.linode.reboot(name, call=None)
salt.cloud.clouds.linode.show_instance(name, call=None)
salt.cloud.clouds.linode.show_pricing(kwargs=None, call=None)
salt.cloud.clouds.linode.start(name, call=None)
salt.cloud.clouds.linode.stop(name, call=None)
salt.cloud.clouds.linode.update_linode(linode_id, update_args=None)
```

More details can be found in the official documentation: https://docs.saltstack.com/en/latest/ref/clouds/all/salt.cloud

Using Salt Cloud With Amazon Web Services (AWS)

Working with *ec2* machines is almost similar to working with Linode VMs.

AWS Provider Configuration

The first thing we are going to do here is to check our key pairs.

If you have multiple key pairs, you are going to use one of them in the *Salt Cloud Provider* configuration, that's why you should search for your key name.

You can use this link to list all of your key pairs:

```
https://eu-west-1.console.aws.amazon.com/ec2/v2/home?region=eu-west-1#KeyPairs:s\
ort=keyName
```

If you are using a different region, change the *eu-west-1* by the region you are considering. In the next example we are using *us-east-1* as the region:

```
https://console.aws.amazon.com/ec2/v2/home?region=us-east-1#KeyPairs:sort=keyName
```

To check which key name corresponds to the used key pair, use this command (after installing AWS kit):

```
ec2-fingerprint-key kp.pem
```

This will help you match your key pair to the fingerprint and allows you to check the name of your key.

In this example, our key is called *kp* and it is located under:

```
/etc/salt/kp.pem
```

Now go and get your security credentials for accessing your *ec2* instances, you can find the *id* here:

```
https://console.aws.amazon.com/iam/home?#security_credential
```

You can not get your *key* from there, but normally you would have kept this in a secret place, if you lost this, you should generate two other key pairs.

You should also know other things like the name of the *security group* you want to use and your *ssh_username*:

- Amazon Linux -> ec2-user
- RHEL -> ec2-user
- CentOS -> ec2-user

- Ubuntu -> ubuntu
- etc ..

Another thing to set up is the *ssh_interface* that could have two different values:

- *private_ips* -> The *salt-cloud* command is run inside the *ec2*
- *public_ips* -> The *salt-cloud* command is run outside of *ec2*

This is a generic example of a *provider* configuration using *private_ips*:

```
ec2-private:

  # Set up the location of the salt master
  minion:
    master: saltmaster.myhost.com

  # Set up grains information, which will be common for all nodes using this pro\
vider
  grains:
    env: test

  # Specify whether to use public or private IP for deploy script.
  ssh_interface: private_ips


  # Set the EC2 access credentials
  id: 'use-instance-role-credentials'
  key: 'use-instance-role-credentials'

  # Make sure this key is owned by root with permissions 0400.
  private_key: /etc/salt/test_key.pem
  keyname: test_key

  securitygroup: default

  # This is optional but you can set up your default region and availability zon\
e (optional)
  location: eu-west-1
  availability_zone: eu-west-1c

  # Salt Cloud will use this user name to deploy, it depends on your AMI.
```

```
  # Amazon Linux -> ec2-user
  # RHEL         -> ec2-user
  # CentOS       -> ec2-user
  # Ubuntu       -> ubuntu
  #
  ssh_username: ubuntu

  # Optionally add an IAM profile
  iam_profile: 'arn:aws:iam::123456789012:instance-profile/ExampleInstanceProfil\
e'

  provider: ec2
```

With a *public_ips*, we will have something similar to this configuration:

```
ec2-public:

  minion:
    master: saltmaster.myhost.com

  ssh_interface: public_ips

  id: 'use-instance-role-credentials'
  key: 'use-instance-role-credentials'

  private_key: /etc/salt/test_key.pem
  keyname: test_key

  securitygroup: default

  location: eu-west-1
  availability_zone: eu-west-1c

  ssh_username: ubuntu

  iam_profile: 'my other profile name'

  provider: ec2
```

Please note two things:

- Previously, the suggested provider for AWS *ec2* was the AWS provider. This has been deprecated in favor of the ec2 provider.

- The provider parameter in cloud provider definitions was renamed to "driver" (since the version 2015.8.0).

AWS Profile Configuration

Let's set up the *profile* to provide more "ec2-specific" configuration options.

In the *profile* configuration we should provide the *provider*, the *image* id, the *size* of the instance and the *ssh_username* which is *Ubuntu* since our *image* is also based on *Ubuntu*.

```
  provider: ec2-private
  image: ami-a609b6d5
  size: t2.micro
  ssh_username: ubuntu
```

If we want to add a volume (10Gb), we can do it like this:

```
  volumes:
    - { size: 10, device: /dev/sdf }
```

Suppose we want to add two other volumes while choosing the *iops* (*Input/Output* per second), we could add a similar configuration to the next one:

```
  volumes:
    - { size: 10, device: /dev/sdf }
    - { size: 300, device: /dev/sdg, type: io1, iops: 3000 }
    - { size: 300, device: /dev/sdh, type: io1, iops: 3000 }
```

Note that to use an *EBS* optimized *ec2* instance we should declare it:

```
ebs_optimized: True
```

We can also add tags to our new instance and it will be applied to all *ec2* instances created using this *profile*:

```
  tag: {'env': 'test', 'role': 'redis'}
```

We have the possibility to force *grains* synchronization by adding:

```
sync_after_install: grains
```

One thing that I usually automate is setting my own configurations, like the *.vimrc* file, you can automate things like this by adding a script that will be executed:

```
script:  /etc/salt/cloud.deploy.d/configure_vim.sh
```

Network configuration is also accessible using *Salt Cloud*, here is an example where the primary IP address is the private address and the ec2 instance will have a public IP (not an elastic IP) with subnet id and a *security group* id:

```
network_interfaces:
  - DeviceIndex: 0
    PrivateIpAddresses:
      - Primary: True
    AssociatePublicIpAddress: True
    SubnetId: subnet-142f4bdd
    SecurityGroupId:
      - sg-750af531
```

If you prefer the *EIP* (Elastic IP):

```
allocate_new_eips: True
```

We want to delete root volume when we destroy our *ec2* instance:

```
del_root_vol_on_destroy: True
```

When a machine is terminated, we want to delete all not-root *EBS* volumes for an instance:

```
del_all_vol_on_destroy: True
```

Now we have a functional *ec2 profile*:

```
base_ec2_private:
  provider: ec2-private
  image: ami-a609b6d5
  size: t2.micro
  ssh_username: ubuntu

  volumes:
    - { size: 10, device: /dev/sdf }
    - { size: 300, device: /dev/sdg, type: io1, iops: 3000 }
    - { size: 300, device: /dev/sdh, type: io1, iops: 3000 }

  tag: {'env': 'test', 'role': 'redis'}

  sync_after_install: grains

  script:  /etc/salt/cloud.deploy.d/configure_vim.sh

  network_interfaces:
    - DeviceIndex: 0
      PrivateIpAddresses:
        - Primary: True
      #auto assign public ip (not EIP)
      AssociatePublicIpAddress: True
      SubnetId: subnet-813d4bbf
      SecurityGroupId:
        - sg-750af531

  del_root_vol_on_destroy: True
  del_all_vol_on_destroy: True
```

When we want to create a similar *profile* to the last one but we would like to change one or two options, we can use *extends* like in the following example:

```
base_ec2_private:
  provider: ec2-private
  image: ami-a609b6d5
  size: t2.micro
  ssh_username: ubuntu

  volumes:
    - { size: 10, device: /dev/sdf }
    - { size: 300, device: /dev/sdg, type: io1, iops: 3000 }
```

```
10      - { size: 300, device: /dev/sdh, type: io1, iops: 3000 }
11
12    tag: {'env': 'test', 'role': 'redis'}
13
14    sync_after_install: grains
15
16    script:  /etc/salt/cloud.deploy.d/configure_vim.sh
17
18    network_interfaces:
19      - DeviceIndex: 0
20        PrivateIpAddresses:
21          - Primary: True
22        AssociatePublicIpAddress: True
23        SubnetId: subnet-813d4bbf
24        SecurityGroupId:
25          - sg-750af531
26
27    del_root_vol_on_destroy: True
28    del_all_vol_on_destroy: True
29
30  base_ec2_public:
31    provider: ec2-private
32    extends: base_ec2_private
```


The last example was just provided to help you understand the use of *extends*, it was not tested.

Using Salt Cloud To Automate AWS EC2 Creation

Starting a private* ec2* instance can be done like this:

```
1  salt-cloud -p base_ec2_private private_minion
```

and launching a public one can be done using this command:

```
1  salt-cloud -p base_ec2_public public_minion
```

Like we have done when we saw how to use *Salt Maps* with Linode, nothing is different from using it with AWS:

```
ec2_private:
  - redis
  - mysql

ec2_public:
  - web_1
  - web_2
```

and then we can start our *ec2* instances using:

```
salt-cloud -m /etc/salt/cloud.map.app -P
```

Salt Cloud allows getting, setting and deleting tags after launching the *ec2* instance using the instance name (or the instance id):

```
salt-cloud -a get_tags ec2_minion
salt-cloud -a set_tags ec2_minion tag1=value1 tag2=value2
salt-cloud -a del_tags ec2_minion tag1,tag2
```

It also allows renaming the machine:

```
salt-cloud -a rename ec2_minion newname=ec2_my_minion
```

To enable termination protection, *Salt Cloud* can be used like in the following command:

```
salt-cloud -a enable_term_protect ec2_minion
```

Other options are available:

```
salt-cloud -a show_term_protect ec2_minion
salt-cloud -a disable_term_protect ec2_minion
```

using *Salt Cloud* from command line allows adding volumes and specific configurations like choosing a snapshot to create a volume:

Creating a simple volume in a specific zone:

```
salt-cloud -f create_volume ec2 zone=eu-west-1c
```

Adding size:

```
salt-cloud -f create_volume ec2 zone=eu-west-1c size=100
```

Choosing a snapshot:

```
salt-cloud -f create_volume ec2 zone=eu-west-1c snapshot=snapshot_id
```

Selecting the type (standard, *gp2*, *io1* ..etc):

```
salt-cloud -f create_volume ec2 size=100 type=standard
salt-cloud -f create_volume ec2 size=100 type=gp2
salt-cloud -f create_volume ec2 size=200 type=io1 iops=2000
```

Detaching a volume then deleting it:

```
salt-cloud -a detach_volume ec2_minion volume_id=vol_id
salt-cloud -f delete_volume ec2 volume_id=vol_id
```

SaltStack Cheat Sheet

```
    \   ^__^
     \  (--)\_______
        (__)\       )\/\
            ||----w |
            ||     ||
```

This is basically a small project that I started on Github[40] that contains some basic SaltStack commands, may be the most used ones and it is suitable for people who want to start learning practical examples quickly. The Github repository is public and everyone can contribute to it.

Installing SaltStack - Ubuntu 14.*

```
wget -O - https://repo.saltstack.com/apt/ubuntu/ubuntu14/latest/SALTSTACK-GPG-KE\
Y.pub | sudo apt-key add -
echo 'deb http://repo.saltstack.com/apt/ubuntu/ubuntu14/latest trusty main' | su\
do tee -a /etc/apt/sources.list
sudo apt-get update

# Master installation
apt-get install salt-master

# Minion installation
apt-get install salt-minion

# Salt ssh installation
apt-get install salt-ssh

# Salt syndic installation
apt-get install salt-syndic

# Salt API installation
apt-get install salt-api
```

[40] https://github.com/eon01/SaltStackCheatSheet

Bootstrapping Salt Minion

```
curl -L https://bootstrap.saltstack.com -o install_salt.sh && sudo sh install_sa\
lt.sh
```

Salt Key Management

```
# Listing Salt requests
salt-key -L

# Accepting all requests
salt-key -A

# Accepting a single request (from myNode)
salt-key -a myNode

# Removing the key of a Salt 'myNode' Minion
salt-key -d minion_id
```

Debugging

```
# Debugging the master
salt-master -l debug

# Debugging the minion
salt-minion -l debug

# Restarting the minion without cache
stop master/minion
rm -rf /var/cache/salt
start master/minion
```

SaltStack Documentation

```
# Viewing all the documentation
salt '*' sys.doc

# Viewing a module documentation
salt '*' sys.doc module_name

#Examples:
salt '*' sys.doc status
salt '*' sys.doc pkg
salt '*' sys.doc network
salt '*' sys.doc system
salt '*' sys.doc cloud

# Viewing a function documentation
salt '*' sys.doc module_name function_name

# Examples:
salt '*' sys.doc  auth django
salt '*' sys.doc sdb sqlite3
```

SaltStack Modules And Functions

```
salt '*' sys.list_modules
salt '*' sys.list_functions
```

Compound Matchers

Letter	Match Type	Example	Alt Delimiter?]
G	Grains glob	G@os:Ubuntu	Yes
E	PCRE Minion ID	E@web\d+.(dev|qa|prod).loc	No
P	Grains PCRE	P@os:(RedHat|Fedora|CentOS)	Yes
L	List of minions	L@minion1.example.com,minion3.domain.com or bl*.domain.com	No
I	Pillar glob	I@pdata:foobar	Yes
J	Pillar PCRE	J@pdata:^(foo|bar)$	Yes
S	Subnet/IP address	S@192.168.1.0/24 or S@192.168.1.100	No
R	Range cluster	R@%foo.bar	No

Other examples:

```
# Examples taken from: https://docs.saltstack.com/en/latest/topics/targeting/com\
pound.html

# Joining
salt -C 'webserv* and G@os:Debian or E@web-dc1-srv.*' test.ping
salt -C '( ms-1 or G@id:ms-3 ) and G@id:ms-3' test.ping

# Excluding
salt -C 'not web-dc1-srv' test.ping
```

Upgrades & Versions

```
#
# Listing upgrades
salt '*' pkg.list_upgrades

# Upgrading
salt '*' pkg.upgrade

# List the packages currently installed as a dict
salt '*' pkg.list_pkgs versions_as_list=True

# Refresh the pkgutil repo database
salt '*' pkgutil.refresh_db

# Check the version of a package
salt '*' pkgutil.version mongodb
```

Packages Manipulation

```
# Installation
salt '*' pkg.install apache2

# Latest version installation
salt '*' pkgutil.latest_version mysql-common

# Removing package(s)
salt '*' pkg.remove vim

# Purging package(s)
salt '*' pkg.purge apache2 mysql-server
```

Reboot & Uptime

```
# Reboot
salt '*' system.reboot

#Uptime
salt '*' status.uptime
```

Using Grains

```
# Syncing grains
salt '*' saltutil.sync_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

# Grains have values that could be called via ‘grains.get <grain_name>’ (path is\
 the name of a grain)
salt '*' grains.get path
```

Syncing Data

```
# Syncing grains
salt '*' saltutil.sync_grains

# Syncing everything from grains to modules, outputters, renderers, returners, s\
tates and utils.
salt '*' saltutil.sync_all
```

Running System Commands

```
salt "*" cmd.run "ls -lrth /data"
salt "*" cmd.run "df -kh /data"
salt "*" cmd.run "du -sh /data"
```

Working With Services

```
# Apache example

# Checking if service is available
salt '*' service.available apache2

# Manipulating Apache2 service
salt '*' service.status apache2
salt '*' service.start apache2
salt '*' service.restart apache2
salt '*' service.stop apache2
```

Network Management

```
# Get IP of your minion
salt '*' network.ip_addrs

# Ping a host from your minion
salt '*' network.ping localhost

# Traceroute a host from your minion
salt '*' network.traceroute localhost

# Get hostname
salt '*' network.get_hostname

# Modify hostname to 'myNode'
salt '*' network.mod_hostname myNode

# Information on all of the running TCP connections
salt '*' network.active_tcp

# Return the arp table from the minion
salt '*' network.arp

# Test connectivity
salt '*' network.connect google-public-dns-a.google.com port=53 proto=udp timeou\
t=3

# Get default route
salt '*' network.default_route

# Execute dig
```

```
salt '*' network.dig eon01.com

# Get the MAC address of eth0 interface
salt '*' network.hw_addr eth0

# Get the inet address of eth1 interface
salt '*' network.interface eth1

# Get the IP address of tun interface
salt '*' network.interface_ip tun
```

Working With HTTP Requests

```
# Get the HTML source code of a page
salt-run http.query http://eon01.com text=true

# Get the header of a page
salt-run http.query http://eon01.com headers=true

# Get the response code from a web server
salt-run http.query http://eon01.com status=true

# Sending a post request
salt '*' http.query http://domain.com/ method=POST params='key1=val1&key2=val2'

#
```

Job Management

```
# List active jobs
salt-run jobs.active

# List all jobs with the id and other information
salt-run jobs.list_jobs

# List multiple information about the job with the id:20151101225221651308 like \
the result output
salt-run jobs.lookup_jid 20151101225221651308

# Kill the job with the id:20151101225221651308
salt 'server' saltutil.kill_job 20151101225221651308
```

Scheduling Feature

```
# Schedule a job called "scheduled_job"
salt '*' schedule.add scheduled_job function='cmd.run' job_args="['']" seconds=10

# Enable the job
salt '*' schedule.enable_job scheduled_job

# Disable the job
salt '*' schedule.disable_job scheduled_job
```

Working With SLS

```
salt '*' state.show_sls
```

Testing States

```
salt '*' state.highstate test=True
salt '*' state.sls test=True
salt '*' state.single test=True
```

Load testing

```
# Starting 20 minions
wget https://raw.githubusercontent.com/saltstack/salt/develop/tests/minionswarm.\
py; python minionswarm.py -m 20 --master salt-master;
```

State Declaration Structure

```
# Source: https://docs.saltstack.com/en/latest/ref/states/highstate.html#state-d\
eclaration

# 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>
```

SaltStack Github Repositories

- *Django* with SaltStack https://github.com/wunki/django-salted
- Salt GUI pad https://github.com/tinyclues/saltpad
- *OpenStack* automation with SaltStack https://github.com/CSSCorp/openstack-automation
- A curated collection of working salt *states* and configurations for use in your SaltStack setup. https://github.com/saltops/saltmine
- These are all of the configuration files needed to build a *Wordpress* development environment with *Vagrant*, *Virtual Box* and SaltStack https://github.com/paulehr/saltstack-wordpress
- *Java* bindings for the SaltStack API https://github.com/SUSE/saltstack-netapi-client-java
- *Vim* snippets for SaltStack *states* files https://github.com/StephenPCG/vim-snippets-salt
- Metrics for SaltStack https://github.com/pengyao/salt-metrics
- Salt GUI https://github.com/saltstack/halite

Best Practices

```
    \   ^__^
     \  (--)\_______
        (__)\       )\/\
            ||----w |
            ||     ||
```

Introduction

Recently, I saw a question on Reddit[41] that caught my attention.

If you have a starter or an intermediate level, without following some basic guidelines and respecting best practices, any task could be exhausting.

I will try to share and comment some of that Reddit comments and other best practices. I am also trying to give some general recommendations for you as a new user of Salt.

Structuring You Pillar

> Structure your pillar carefully, or it will become a mess ~ acid_sphinx4

Pillars are used to store data, especially sensitive data that you won't share with all *minions*. They are stored in the *master* and shared only with the targeted *minion*.

Pillars are called using the *top.sls* file.

```
tree /srv/pillar/
/srv/pillar/
├── sched
│   └── init.sls
└── top.sls
```

The latter targets all or some *minions*:

[41] https://www.reddit.com/r/saltstack/comments/3maxi3/what_did_you_guys_wish_you_knew_before_setting_up/

```
base:
  'os:Ubuntu':
    - nginx
```

Choosing meaningful names for state files like *nginx* here and using it to only store *Nginx* related data is a good habit to have when working with *pillars*. Keeping the configuration file as simple as possible would help other users to maintain and ameliorate it.

Organizing Grains:

Salt *grains* are stored by the default in the folder:

```
etc/salt/grains
```

There is another alternative:

```
/etc/salt/minion
```


Hint: Search for *#grains* in your default *minion* file

Now, imagine that you have hundreds of variables put into thousands of files. In this case, I usually create a folder, that I call *config* under */etc/salt/*:

```
/etc/salt/config/app1/roles.conf
```

```
/etc/salt/config/app2/roles.conf
```

```
/etc/salt/config/app3/roles.conf
```

and then I call each one of them in the *minion* file.

```
include:
  - /etc/salt/config/app1/roles.conf
  - /etc/salt/config/app2/roles.conf
  - /etc/salt/config/app3/roles.conf
```

Then I add my *grains* to those files, here is how you can do it :

```
grains:
  variable1: value1
  variable2: value2
```

Don't forget to type *grains:* at the beginning of the file and then you will be able to declare your *grains* without having errors during the execution of *state.sls* or *state.highstate*.

 Note that the default_include in SaltStack is minion.d/*.conf

Your Own Execution Modules vs Working With Jinja

> It is easier and more maintainable to write your own python execution modules than it is to do complex logic in jinja ~ FakingItEveryDay

Salt modules are *python* files and writing your own modules is not that difficult as it seems to be. *Python* is an easy-to-learn language. It is real that *Jinja* files become complex sometimes but it is up to you to see and compare what should be better for your configuration management.

You could choose to keep *Jina* configuration or write your own execution modules in function of many criteria:

- Readability
- Complexity
- Execution time
- Maintainability
- Modularity

I would not recommend using one of them over the other unless I can attach the choice to a specific context and then choose in function of some criteria like the ones listed below. This could be controversial, from a side using *Jinja* was supposed to make things easier while in some cases using *Python* could be faster to code. In both cases, keep things as simple as possible.

Using The Documentation

> I say all the following and it sounds like I'm just a bitchy jerk face but SaltStack is legit black magic that makes my life so much better. Once you get it working.
>
> The documentation is wrong all the time. It is also very often incomplete. Be prepared to read the > source code. If you don't know Python yet, you need to start learning. ~ bbbryson

In my humble opinion, SaltStack documentation is not disastrous at this point but it is not sufficient, which means that you will need to search for other content like blog posts and books. This is normal and I think most of us are used to work like this: looking at the official documentation at the first time and then moving to practical examples posted in community blogs.

You do not need to go to the official website to search for a documentation, like Linux man pages just type use it form your terminal:

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

This will output all the documentation. However, if you need *pkg* module documentation, type the following command:

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

and you will see just the documentation to help you use *pkg* with Salt.

Try also:

```
salt '*' sys.doc network
salt '*' sys.doc system
salt '*' sys.doc status
```

This is a part of the output of *status* documentation:

```
'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


'status.cpuinfo:'

    Return the CPU info for this minion

    CLI Example:

        salt '*' status.cpuinfo


'status.cpustats:'

    Return the CPU stats for this minion

    CLI Example:

        salt '*' status.cpustats


'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 from and custom is this function. It is followed
    by a list of keys that we want to be returned.

    This function is meant to replace all_status(), which returns
    anything and everything, which we probably don't want.

```

```
43      By default, nothing is returned. Warning: Depending on what you
44      include, there can be a LOT here!
45
46      CLI Example:
47
48          salt '*' status.custom
49
50
51  'status.diskstats:'
52
53      Return the disk stats for this minion
54
55      CLI Example:
56
57          salt '*' status.diskstats
58
59
60  'status.diskusage:'
61
62      Return the disk usage for this minion
63
64      Usage::
65
66          salt '*' status.diskusage [paths and/or filesystem types]
67
68      CLI Example:
69
70          salt '*' status.diskusage         # usage for all filesystems
71          salt '*' status.diskusage / /tmp  # usage for / and /tmp
72          salt '*' status.diskusage ext?    # usage for ext[234] filesystems
73          salt '*' status.diskusage / ext?  # usage for / and all ext filesystems
```

As you can see, the documentation is quite simple and comes with examples ready to use, like disk usage for / and */tmp*

```
1  salt '*' status.diskusage / /tmp
```

or *CPU* statistics:

```
1  salt '*' status.cpustats
```

As a beginner, following and mastering the official examples is quite enough, in my opinion, to practice Salt and master more complicated examples. So no worries with the documentation, on the contrary, I recommend to follow and understand the official documentation before moving to something else. The official examples follow the best practices, do not hesitate to copy and modify them for your needs.

Following Guidelines

> I would like to see at least a recommended guideline from somebody. The ability to have a standard (and maybe some namespacing on formulas) would be much appreciated.

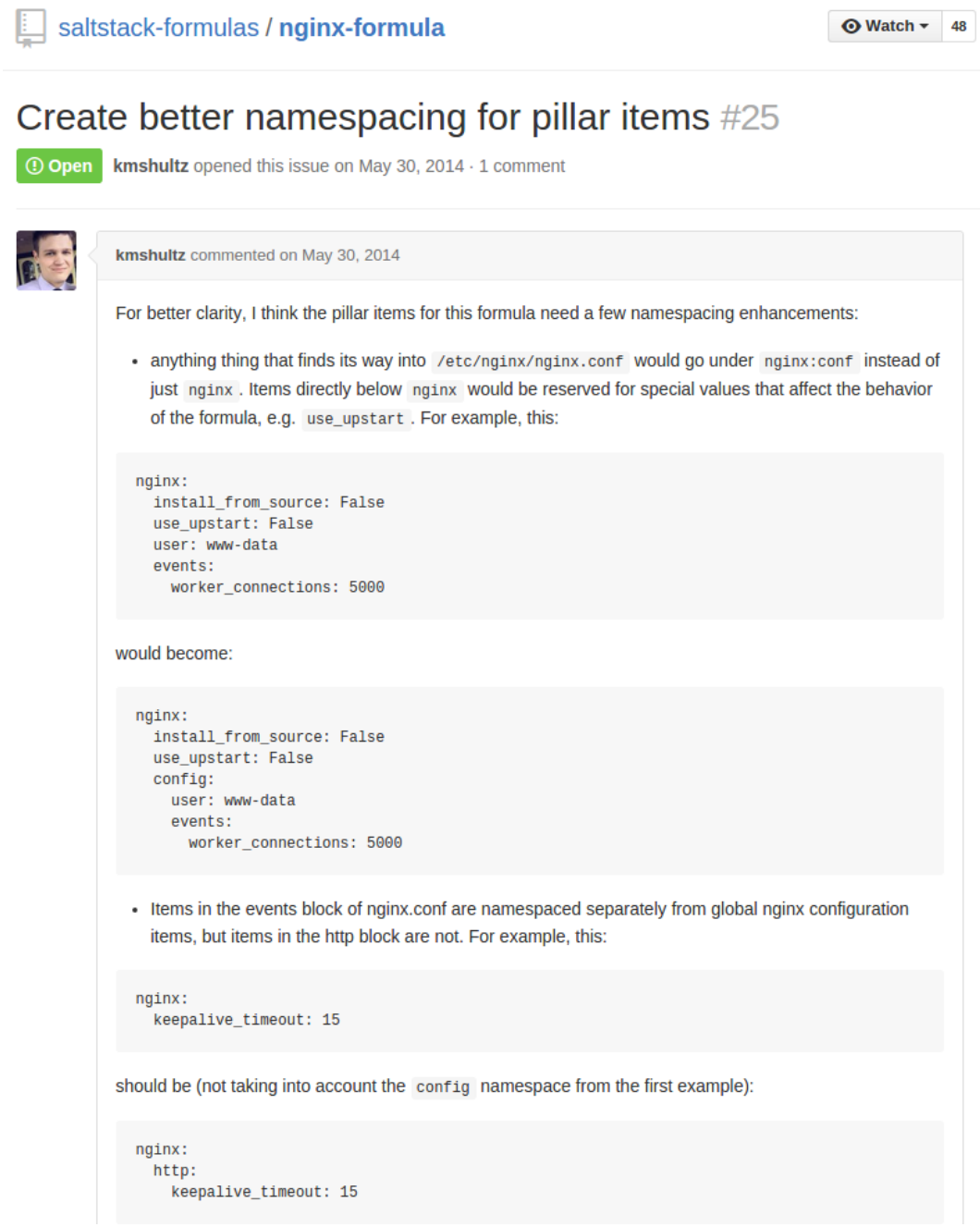
I totally agree with this, some guidelines and *namespacing* are mentioned in the official documentation. Without standards, everyone could have it own *namespaces* conventions and sharing code could be less efficient without this.

Here is the *SLS* file *namespace* that the official documentation describes:

The *namespace* for *SLS* files when referenced in *top.sls* or an include declaration 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 can be represented with a dot (following the *python import* model) or a slash. *webserver/dev.sls* can also be referred to as *webserver.dev*
    b. Because slashes can be represented as dots, *SLS* files cannot contain dots in the name beside the dot for the *SLS* suffix. The *SLS* file *webserver_1.0.sls* can not be matched, and *webserver_1.0* would match the directory/file *webserver_1/0.sls*
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*.

Working on a SaltStack *Formula* would need to pay more attention to *namespacing* since it is shared and maintained in a collaborative way (usually on Github). Some efforts are being made to do this.



saltstack-formulas / **nginx-formula** Watch 48

Create better namespacing for pillar items #25

Open **kmshultz** opened this issue on May 30, 2014 · 1 comment

kmshultz commented on May 30, 2014

For better clarity, I think the pillar items for this formula need a few namespacing enhancements:

- anything thing that finds its way into `/etc/nginx/nginx.conf` would go under `nginx:conf` instead of just `nginx`. Items directly below `nginx` would be reserved for special values that affect the behavior of the formula, e.g. `use_upstart`. For example, this:

```
nginx:
  install_from_source: False
  use_upstart: False
  user: www-data
  events:
    worker_connections: 5000
```

would become:

```
nginx:
  install_from_source: False
  use_upstart: False
  config:
    user: www-data
    events:
      worker_connections: 5000
```

- Items in the events block of nginx.conf are namespaced separately from global nginx configuration items, but items in the http block are not. For example, this:

```
nginx:
  keepalive_timeout: 15
```

should be (not taking into account the `config` namespace from the first example):

```
nginx:
  http:
    keepalive_timeout: 15
```

Contributors on Github

SaltStack has also published an official and complete list of conventions[42] to be used when creating a *formula*. I recommend focussing on this when writing your first *Formula*.

[42] https://docs.saltstack.com/en/latest/topics/development/conventions/formulas.html

Automate Your Automation

With the growing needs of IT systems, automation is becoming one of the best practices for a system administrator. SaltStack is one of the tools that offers a great list of automation possibilities, but you can still automate the automation, you can call Salt commands from *Python*, *Shell* or *Bash* scripts. As most of system administrators, DevOps and developers were using scripts to automate builds, deployments or any other operation, reforming those scripts by adding SaltStack could be a brilliant idea: If you are migrating your infrastructure management to use SaltStack, make it piece by piece.

Using the *Python Client API* could help you. Using tools like *Jenkins*, *Fabric* ..etc are great. Using both of them is just perfect.

Daily SysAdmin routine, like checking the status of a service could, better should be automated.

As an example, checking if *Mongodb* daemon (*mongd*) is up could done using the code below:

```
salt 'myNode' service.status mongod
```

The execution of the last command will output the process id of every instance of *Mongo* if it is already running.

In this example, I am running a *Docker* machine that starts a *Mongo* instance:

```
$ ps -aux|grep docker

root      4060  0.0  0.3 401224 23756 ?        Ssl  15:06   0:01 /usr/bin/docker\
 daemon
root      4218  0.0  0.2 210368 17920 ?        Sl   15:06   0:00 docker-proxy -p\
roto tcp -host-ip 0.0.0.0 -host-port 28017 -container-ip 172.17.0.1 -container-p\
ort 28017
root      4225  0.0  0.2 144832 18284 ?        Sl   15:06   0:00 docker-proxy -p\
roto tcp -host-ip 0.0.0.0 -host-port 27017 -container-ip 172.17.0.1 -container-p\
ort 27017
```

Let's take a look the output of *service.status* command:

```
#Command
salt 'myNode' service.status mongod
#Output
myNode:
    4239
```

If I would run this from a *Python* script using SaltStack *API*, I will code something similar to this:

```
#!/usr/bin/env python

# Import the Salt client library
import salt.client

# Create a local client object
client = salt.client.LocalClient()

# Make calls with the cmd method
ret = client.cmd('myNode', 'service.status', ['mongod'])

# Print the pid of the 'Mongo' database instance
print ret
```

Executing the last script will print the following output:

```
{'myNode': '4239'}
```

If you are familiar with *Python* you will notice that the last output is a *dict* (a *dict* or a dictionary is a mapping object maps *hashable* values to arbitrary objects). You could also notice that we can directly access *4239* by changing:

```
print ret
```

in the last code by

```
ret['myNode']
```

More elegant way to check the same thing.

```
#!/usr/bin/env python

# Import the Salt client library
import salt.client

# Create a local client object
client = salt.client.LocalClient()

# make calls with the cmd method
ret = client.cmd('myNode', 'service.status', ['mongod'])

```

```
12 if ret:
13         exit(0)
14
15 exit(1)
```

The last script could be interfaced with *Jenkins* or another continuous integration server since the build will automatically fail when the exit value is not 0.

Start Your Cheat Sheet

Getting started with SaltStack is not that hard, but working on complex environments and live production sites will relatively complicate your work. Start your own SaltStack cheat sheet where you will take notes every time you learn a new thing.

I started a Github repository[43] with some useful commands this could help you to start your own cheat sheet, just fork and contribute.

The Usual Lecture From The Local System Administrator

The famous *sudo* warning is the first best practice you should learn as a SysAdmin or a DevOps.

> We trust you have received the usual lecture from the local System Administrator. It usually boils > down to these three things: #1) Respect the privacy of others. #2) Think before you type. #3) With great power comes great responsibility.

When working with automation and remote execution tools, operations become easier than before but risky : You would like to remove */data* from a specific server (*myNode*) but you forgot to target it and put *'' instead of *'myNode'*.

```
1 salt '*' cmd.run 'rm /data'
```

This is irreversible, you removed */data* directory from all your servers. When working with *states* remember to test before making changes. SaltStack has a test interface that will simulate the execution of a state and report the same result:

[43] https://github.com/eon01/SaltStackCheatSheet

```
salt '*' state.highstate test=True
salt '*' state.sls test=True
salt '*' state.single test=True
```

Updates, Upgrades And Backups

On an Github opened issue[44] a user having 4 environments (“base”, “qa”, “dev”, “master”) defined in *file_roots” discovered that the documentation describing that

> If an environment has info in *top.sls* in *base*, it will always override all other *top.sls*.

Was not describing the real case: When checking the code he found that there is no distinction between *base* and any other name, and no specific decision on ordering that seems alphabetical

> Even though in the *top.sls* and *file_roots*, “qa” is second down, because it is last alphabetically, that overrides all other *top.sls* files in all other environments.

The documentation is clear but the code seems to do another thing: merge and overwrite in alphabetical order.

The issue was opened on May 2, 2014, and it was fixed. Even if this issue has not any effect on my servers, but problems can happen. If there are any recommendations to say here, it is actually a general recommendation that every SysAdmin should know like:

- Make sure to have the last stable updates and upgrades
- Backup, backup, backup and test your backups
- Document everything
- Have B plans for everything
- ..etc

Following Official Guidelines

You can find on this page[45] a list of recommendations and best practices to follow when working with SaltStack. I strongly recommend to follow this guide since it is edited collaboratively on Github[46] and maintained officially by SaltStackInc.

The official guide includes general rules and recommendations about:

[44] https://github.com/saltstack/salt/issues/12483
[45] https://docs.saltstack.com/en/latest/topics/best_practices.html
[46] https://github.com/saltstack/salt/blob/develop/doc/topics/best_practices.rst

- Structuring States and Formulas
- Structuring Pillar Files
- Variable Flexibility
- Modularity Within States
- Storing Secure Data

Community, Support & Commercial Sercices

```
    \   ^__^
     \  (--)\_______
        (__)\       )\/\
            ||----w |
            ||     ||
```

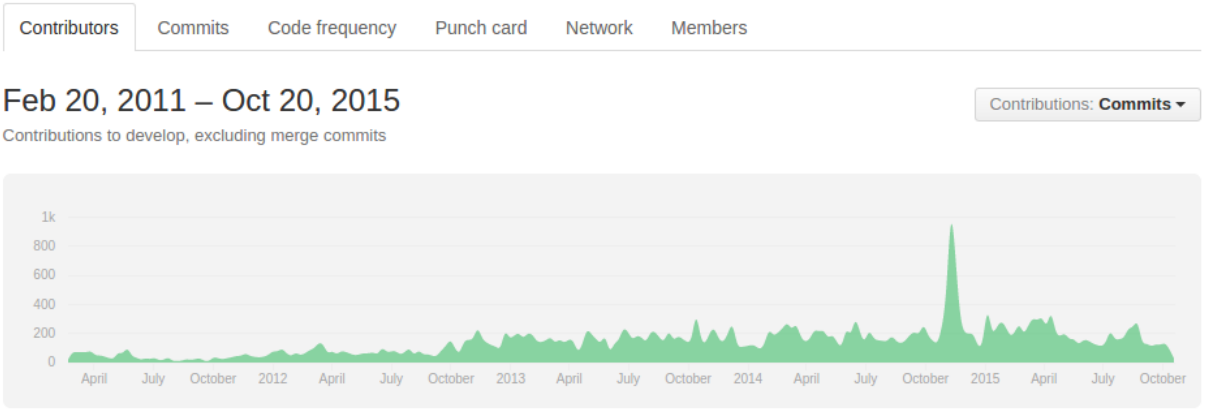
SaltStack Source Code On Github

SaltStack is an Open Source software provided under the Apache 2.0 license.

Like most of Open Source and Free Softwares, Salt has the advantages of auditability, cost, freedom, flexibility and also its active community and contributors.

One of the other advantages of using Open Source Software in general and specifically Salt is the freedom to access and customize its source code. Salt source code is available on Github[47].

Currently, the project has more than 2k forks and 1235 contributors. It was starred by more than 5k Github user.



SaltStack contributors on Github

[47] https://github.com/saltstack/salt

SaltStack documentation is increasingly becoming richer and the number of blogs and tutorials about it is increasing. If you started using SaltStack since 1 or 2 years you have probably noticed the significant increase in online resources.

If you need to ask questions or need the community support, you can use Github, Salt IRC channel, the official Google group and StackExchange (generally StackOverflow).

If you are looking for news, updates, and speeches, you can subscribe to the official Salt blog and its YouTube channel.

Official Resources

This is the full list of the official online resources:

- Github[48]: The official GitHub git repository where you can find the source code, follow the issues and of course contribute ..etc
- Documentation[49] : The official SaltStack documentation. This is may be the most helpful online resource for SaltStack users.
- IRC channel[50]: The official IRC channel where you can find volunteers that may have answers to your questions.
- IRC logs[51]: Where you can find all of the IRC discussions' history.
- Google discussion group[52]: The official discussion group and mailing list. Actually, more than 5K topics were opened. You may get a good quality support as many of SaltStack users are using this online resource.
- Youtube channel[53]: The official Youtube channel where you can find news, speeches and some tutorials.
- SaltStack blog[54] : The official blog.
- SaltConf[55]: A website dedicated to the Salt conference (speeches, training, certifications).
- Linkedin group[56]: The official Linkedin discussion group where you can share links, ask questions and find good quality tutorials and blog posts.
- SaltStack events[57]: A dedicated page for official events and events organized by partners.

[48] https://github.com/saltstack/salt
[49] http://docs.saltstack.com
[50] http://webchat.freenode.net/?channels=salt
[51] http://irclog.perlgeek.de/salt/
[52] https://groups.google.com/forum/#!forum/salt-users
[53] http://www.youtube.com/saltstack
[54] http://www.saltstack.com/salt-blog
[55] http://saltconf.com/
[56] https://www.linkedin.com/groups/4877160
[57] http://saltstack.com/events/

Community Ressources

You can visit the community page[58] where you can find the resources provided above and maybe some other useful links.

SaltStack Reddit[59] is also a very good place where you can find useful links and where you can ask your questions and contribute to the community. This Reddit has more than 1k users and it's getting more and more popular every day.

Meet-ups Around SaltStack

You can also join one of these community discussion and meet-up groups around SaltStack:

- Amsterdam[60]
- Atlanta[61]
- Austin[62]
- Birmingham, AL[63]
- Bucharest[64]
- Charlotte[65]
- Chicago[66]
- China[67]
- Dallas[68]
- Denver[69]
- Durham, NC[70]
- Germany[71]
- The SaltStack LinkedIn group[72]
- London[73]

[58] http://saltstack.com/community/
[59] https://www.reddit.com/r/SaltStack
[60] http://www.meetup.com/Amsterdam-Salt-Stack/
[61] http://www.meetup.com/Atlanta-SaltStack-Meetup/
[62] http://www.meetup.com/Austin-Saltstack-User-Group/
[63] http://www.meetup.com/Birmingham-SaltStack-Users-Group/
[64] http://www.meetup.com/Bucharest-SaltStack-Meetup/
[65] http://www.meetup.com/Charlotte-SaltStack-Meetup/
[66] http://www.meetup.com/SaltStack-Chicago/
[67] http://saltstack.cn/
[68] http://www.meetup.com/SaltStack-Dallas-Meetup/
[69] http://www.meetup.com/Denver-SaltStack-Users-Group
[70] http://www.meetup.com/TriSalt/
[71] mailto:salt-users-de@googlegroups.com
[72] https://www.linkedin.com/groups?home=&gid=4877160
[73] http://www.meetup.com/SaltStack-user-group-London

- Los Angeles[74]
- Michigan[75]
- Minneapolis[76]
- Montreal[77]
- Nashville[78]
- New York City[79]
- Paris[80]
- Portland[81]
- Salt Lake City[82]
- San Francisco[83]
- Sao Paulo[84]
- Seattle[85]
- Silicon Valley[86]
- Stockholm[87]
- Sydney[88]
- Washington DC[89]
- Zurich[90]

SaltStackInc

SaltStackInc is based at 3400 N. Ashton Blvd, Suite 110 Lehi, UT 84043 and they can be contacted by phone +1 801.207.7440 or by email info@saltstack.com

[74] http://www.meetup.com/SaltStack-Los-Angeles-Meetup/
[75] http://www.meetup.com/AWS-Michigan/
[76] http://www.meetup.com/SaltStack-Minneapolis-Meetup/
[77] http://www.meetup.com/Montreal-SaltStack-Meetup/
[78] http://www.meetup.com/Nashville-SaltStack-Users-Group/
[79] http://www.meetup.com/SaltStack-NYC/
[80] http://www.meetup.com/Paris-Salt-Meetup/events/221368010/
[81] http://www.meetup.com/Portland-SaltStack-Users-Group/
[82] http://www.meetup.com/SaltStack-user-group-Salt-Lake-City/
[83] http://www.meetup.com/Salt-Stack-DevOps/
[84] http://www.meetup.com/saltstackbrasil/
[85] http://www.meetup.com/Seattle-SaltStack-Meetup/
[86] http://www.meetup.com/SaltStack-user-group-Silicon-Valley/
[87] http://www.meetup.com/SaltStack-Stockholm/
[88] http://www.meetup.com/Sydney-SaltStack-User-Group/
[89] http://www.meetup.com/NoVA-Saltstack/
[90] http://www.meetup.com/Zurich-SaltStack/

SaltStackInc on Google Maps

SaltStack offers also paid consulting and support services to SaltStack Enterprise customers, training session, on-site training, quick-start workshops certifications (SaltStack Certified Engineer).

About eralabs

eralabs is a company I created with the goal of helping companies create and deploy Cloud-Native applications. We also help them accelerate the DevOps learning curve by providing online and onsite training.

Among other technologies, we use SaltStack, integrate it in our clients DevOps pipelines and provide onsite training to help developers and ops use this technology.

Afterword

DevOps is a culture that aims to improve the process of development, integration and deployment with more collaboration and transparency between the Dev, the Ops and any other team collaborating on the software development process.

SaltStack is a comprehensive tool that also allows you to simplify your infrastructure manipulation and master the flow between multiple environments (development, integration, staging, and production).

In these environments, we use other software such as Vagrant, Docker, Jenkins, Rundeck..etc

SaltStack will allow you to interface with many of these software to be present throughout your "DevOps Toolchain".

I hope that SaltStack For DevOps has brought you what you were looking for.

You can find other books, training and courses in eralabs website[91] like Painless Docker[92] and Practical AWS[93].

If you find this course helpful, if it helped you to solve some of your problems or you simply liked my work, I'll be really glad if you share your feedback with me using this form[94].

Credits

- DevOps as the intersection of development (software engineering), technology operations and quality assurance (QA) is a "Creative Commons derivative work[95]. This file was derived from Devops.png" by Rajiv.Pant and licensed under CC BY 3.0.
- Salt Crystals (Cover) is a "Creative Commons work[96] by Mschel and licensed under CC BY 3.0.
- Comparison of open-source configuration management software is Creative Commons work[97] licensed under CC BY 3.0.

[91] http://eralabs.io
[92] http://painlessdocker.com
[93] http://practicalaws.com
[94] https://eon01.typeform.com/to/A2HE4D
[95] https://en.wikipedia.org/wiki/DevOps#/media/File:Devops.svg
[96] https://commons.wikimedia.org/wiki/File:Salt_Crystals.JPG
[97] https://en.wikipedia.org/wiki/Comparison_of_open-source_configuration_management_software