Diving In – Our First Salt Commands

- Salt không chỉ là 1 công cụ quản trị và thực thi từ xa, nó là 1 nền tảng mạnh mẽ, cung cấp các công cụ để quan trị hạ tầng, và cũng cho phép tạo ra các công cụ mới phù hợp với môi trường thực thi

Giới thiệu về Salt.

- Các nội dung cơ bản về kiến trúc Salt.

- Salt gồm 2 phần chính: Salt master và Salt minion.

- Salt master là trung tâm.

- Tất cả minion sẽ kết nối tới master để nhận các chỉ thị.

- Từ master, bạn có thể sử dụng các commands và apply các config tới hàng trăm – hàng ngàn minions trong vài giây.

- Minion được kết nối tới master và coi master là nguồn đang tin cậy.

- Minion có thể tồn tại mà ko cần tới master nhưng để hệ thống hoạt động tốt nhất, master và minion cần được làm việc với nhau.

- Salt được xây dựng trên 2 mục tiêu chính: Thực thi từ xa và quản trị cấu hình.

- Về hệ thống thực thi từ xa, Salt sử dụng Python để thực hiện các công việc phức tạp với các hàm đơn.

- Về hệ thống quản trị cấu hình, các trạng thái, config được xấy dựng dựa trên nền tản thực thi từ xa, dễ dàng cho việc sử dụng lại, thực thi trên các minions

Installing Salt

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

• Python 2 – Version 2.6 or greater (Salt is not Python 3-compatible)

• Msgpack – python

• YAML

• Jinja2

• MarkupSafe

• ZeroMQ – Version 3.2.0 or greater

• PyZMQ – Version 2.2.0 or greater

• Tornado

• PyCrypto

• M2Crypto

- Lời khuyên là sử dụng các công cụ hỗ trợ cài đặt để dễ dàng hơn trong việc triển khai SaltStack

Cài đặt:

- Cài đặt saltstack theo seri hướng dẫn

Firewall config

- Minions kết nối với master, nên chỉ cần cấu hình trên master.

- Port mặc định trên master sẽ được sử dụng là 4505 4506. Mở các port mặc định

Salt minion configuration

- Đường dẫn cấu hình của cả master và minion đều là “/etc/salt/”

- Đường dẫn file cấu hình minion “/etc/salt/minion”

- Salt minion được cấu hình để có thể kết nối tới master thông qua trường salt

- Có thể là ip hoặc domain (nó sẽ được phân giải thông qua DNS)

salt=[ip or domain]

- Để cấu hình minion ID sửa dụng trường “id”. ID này sẽ được sử dụng khi kết nối minions với master

id: [Tên muốn gán cho minion]

Starting the Salt master and Salt minion

# sudo service salt-minion restart

# sudo service salt-master restart

Chấp nhận minion key trên master

- Bước cuối để kết nối minion tới master là cho phép minion key tại master node

- Liệt kết các key quản lý bởi master:

# salt-key

Accepted Keys:

Denied Keys:

Unaccepted Keys:

[id chờ]

Rejected Keys:

- Để xem chi tiết key của minion, đảm bảo key tại minion và master chính xác, để kết nối.

## Tại master node

$ salt-key -f minion1

myminion: a8:1f:b0:c2:ab:9d:27:13:60:c9:81:b1:11:a3:68:e1

## Tại minion node

$ salt-call --local key.finger

local: a8:1f:b0:c2:ab:9d:27:13:60:c9:81:b1:11:a3:68:e1

- Khi 2 key đã trùng nhau, tại node master chập nhận minion key:

# salt-key -a myminion

The following keys are going to be accepted:

Unaccepted Keys:

myminion

Proceed? [n/Y] Y

Key for minion myminion accepted.

- Kiểm tra lại sau khi đã kết nối:

# salt-key

Accepted Keys:

myminion

Denied Keys:

Unaccepted Keys:

Rejected Keys:

Các kiểm tra, test sau khi đã kết nối minion tới master

- Cấu lênh test ping. Câu lệnh này sử dụng để check trạng thái của các minion xem nó còn sống ko. Nếu sống sẽ trả lại true.

$ salt '\*' test.ping

myminion:

True

- Cấu lệnh test.\* có sẵn trong module test, được cài mặc định sau khi instal saltstack. Để xem chi tiết các funcion trong module test.\* sử dụng câu lệnh

$ salt 'myminion' sys.list\_functions test

- test.arg

- test.arg\_repr

- test.arg\_type

- test.assertion

- test.attr\_call

- test.collatz

- test.conf\_test

- test.cross\_test

- test.echo

- test.exception

- test.false

- test.fib

- test.get\_opts

- test.kwarg

- test.module\_report

- test.not\_loaded

- test.opts\_pkg

- test.outputter

- test.ping

- test.provider

- test.providers

- test.rand\_sleep

- test.rand\_str

- test.retcode

- test.sleep

- test.stack

- test.true

- test.try\_

- test.tty

- test.version

- test.versions

- test.versions\_information

- test.versions\_report

- Thử sử dụng 1 hàm test khác:

# sudo salt '\*' test.fib

myminion:

Passed invalid arguments to test.fib: fib() takes exactly 1 argument

(0 given)

- Để biết thêm thông tin về 1 funtion, ta sử dụng hàm sys.doc

# sudo salt '\*' sys.doc test.fib

test.fib:

Return a Fibonacci sequence up to the passed number, and the

timeit took to compute in seconds. Used for performance tests

CLI Example:

salt '\*' test.fib 3

- Module sys rất hưu dụng khi tìm hiểu và salt.

- Danh dánh các funtion trong sys module

# sudo salt '\*' sys.list\_functions sys

myminion:

- sys.argspec

- sys.doc

- sys.list\_functions

- sys.list\_modules

- sys.list\_renderers

- sys.list\_returner\_functions

- sys.list\_returners

- sys.list\_runner\_functions

- sys.list\_runners

- sys.list\_state\_functions

- sys.list\_state\_modules

- sys.reload\_modules

- sys.renderer\_doc

- sys.returner\_argspec

- sys.returner\_doc

- sys.runner\_argspec

- sys.runner\_doc

- sys.state\_argspec

- sys.state\_doc

Masterless Salt

- Salt cũng được thiết kế để minions có thể hoạt động mà không cần tới master.

- Tính năng này thường được sử dụng khi chúng ta đang log tại client mà muốn test 1 số function.

- Cmd sử dụng để chạy function trên minions là “salt-call”.

# sudo salt-call test.ping

local:

True

# sudo salt-call test.fib 10

# sudo salt-call sys.doc test.ping

Note:

+ Hàm test.ping sẽ báo lỗi khi service salt-master ngừng hoạt động, vì bất kể chạy hàm test.ping ở master or minions thì nó đều sẽ cố thiết lập kết nối giữa master và client.

+ Để giải quyết vấn đề này sử dụng option --local để chỉ test trên minion

# sudo salt-call --local test.ping

local:

True

Controlling Your Minions with Remote Execution

The structure of a remote execution command

- Cấu trúc 1 execute command đơn giản:

# sudo salt '\*' test.ping

myminion:

True

- Thông thường 1 exec cmd sẽ được chia ra làm 5 phần

# sudo salt --help

Usage: salt [options] '<target>' <function> [arguments]

# sudo salt --verbose '\*' test.sleep 2

Executing job with jid 20160218032023792688

- Các phần của 1 exec cmd bao gồm

• The Salt command: salt

• Command-line options: --verbose

• Targeting string: '\*'

• The Salt module function: test.sleep

• Arguments to the remote execution function: 2

Command-line options

- Các tham số tùy chỉnh, cho phép tùy biến cmd

- Salt hỗ trợ 1 số main categories các cli option

- Target option sử dụng cho target minions.

- Output option sử dụng để lấy các thông tin trả về của minions sau khi cmd theo 1 cấu trúc nào đó.

VD: cơ bản các output sẽ hiện thị thông qua nested outputter

# sudo salt '\*' cmd.run\_all 'echo HELLO'

myminion:

----------

pid:

13999

retcode:

0

stderr:

stdout:

HELLO

# sudo salt --out=nested '\*' cmd.run\_all 'echo HELLO'

myminion:

----------

pid:

14020

retcode:

0

stderr:

stdout:

HELLO

- “cmd” execution module thiết kế để cung cấp cách thực thi bất kỳ cmd hoặc programs tại minions.

- “cmd.run\_all” trả lại các kết quả của cmd như pid, code, content std out và std err.

- Có thể đổi lại option output để thấy được raw data trả lại.

# sudo salt --out=raw '\*' cmd.run\_all 'echo HELLO'

{'myminion': {'pid': 14468, 'retcode': 0, 'stderr': '', 'stdout':

'HELLO'}}

- Có thể đổi option output để xem data dạng json.

# sudo salt --out=json '\*' cmd.run\_all 'echo HELLO'

{

"myminion": {

"pid": 14506,

"retcode": 0,

"stderr": "",

"stdout": "HELLO"

}

}

- Có thể tùy chỉnh output hiện thị dạng YAML

# sudo salt --out=yaml '\*' cmd.run\_all 'echo HELLO'

myminion:

pid: 14585

retcode: 0

stderr: ''

stdout: HELLO

- Có thể tùy chỉnh output để không hiện thi ra kết quả

# sudo salt --out=quiet '\*' cmd.run\_all 'echo HELLO'

- Một số tùy chỉnh khác:

• --versions-report: This shows Salt's version and the versions of its

dependencies (essential for bug reports).

• -t TIMEOUT or --timeout=TIMEOUT: This changes the initial timeout of the running command. This timeout is the amount of time the master will wait before checking to see whether any minions are still running the command. It is not a hard timeout for the job.

• --async: This runs a Salt command without waiting for the minions to respond with the results of the run. The minion will still run the job, but the results cannot be seen on the command line and must be retrieved from the job cache later using the provided job ID (jid).

-v or --verbose: This turns on command verbosity—that is, it gives more information about what's happening with the command.

• --show-timeout: This shows which minions time out on a given command. This output is also shown with the --verbose command.

Targeting strings

- Salt stack cho phép target 1 hoặc 1 tập hoặc tất cả minions. Cho phép việc apply các module, cmd trở nên hiệu quả hơn. Giải quyết được đc vấn đề dành cả ngày ssh vào từng máy config.

Glob matching

- Để có thể target tất cả minion sử dụng

# sudo salt '\*' test.ping

myminion:

True

- Các sử dụng target matching cũng tương tự các sử dụng matching file trong linux

VD:

# sudo salt '\*' test.ping

# sudo salt '' test.ping

# sudo salt 'myminion' test.ping

# sudo salt 'my\*' test.ping

# sudo salt 'my\*mini\*' test.ping

# sudo salt 'larry' test.ping

# sudo salt '\*.txt' test.ping

# sudo salt '??minion' test.ping

# sudo salt '??minion' test.ping

Perl-compatible regular expression matching

- Nếu cần thiết sử dụng nhưng matching phức tạp, Salt hỗ trợ việc sử dụng regular expressions thông qua thư viện của python.

- Tham khảo thư viện: Perl-compatible regular expressions (PCRE)

<https://docs.python.org/2/library/re.html>

- Để sử dụng thư viện PCRE sử dụng option -E or -pcre

VD:

# sudo salt -E 'myminion' test.ping

myminion:

True

# sudo salt -E 'my' test.ping

myminion:

True

- Sử dụng “.\*’ để map bất kỳ ký tự nào 1 hay nhiều lần

# sudo salt -E '.\*' test.ping

myminion:

True

# sudo salt -E 'my.\*n' test.ping

myminion:

True

# sudo salt -E 'foo.\*' test.ping

No minions matched the target.

- Sử dụng ký tự “^” map khi bắt đầu dòng, và “&” để map cuối dòng

# sudo salt -E '^my$' test.ping

No minions matched the target.

# sudo salt -E '^myminion$' test.ping

myminion:

True

- Sử dụng “|” để map or giữ 2 chuỗi ký tự A or B “A|B”

# sudo salt -E '((my)|(your))minion' test.ping

myminion:

True

# sudo salt -E 'myminion(s)?' test.ping

myminion:

True

# sudo salt -E '(my)?minion' test.ping

myminion:

True

List matching

- Đơn giản hơn việc sư dụng RE, sử dụng 1 list chi việc match.

- Sử dụng option -L or --list

VD:

# sudo salt -L 'myminion' test.ping

myminion:

True

# sudo salt -L 'myminion,yourminion,theirminion' test.ping

myminion:

True

Grain and pillar matching

Sử dụng grains

- Grains đại diện cho dữ liệu tĩnh trên minion

VD: minions có grain named os\_family, nó đại diện cho hệ điều hành mà minions đang sử dụng.

# sudo salt '\*' grains.item os\_family

myminion:

----------

os\_family:

Debian

- Để biết chính xác hệ điều hành đang chạy trên minions, sử dụng “os” grain

# sudo salt '\*' grains.item os

myminion:

----------

os:

Ubuntu

# sudo salt '\*' grains.item osfinger

myminion:

----------

osfinger:

Ubuntu-14.04

- Có thể sử dụng 1 flag để kiểm tra hệ điều hành minion có đúng như mong đợi, sử dụng tham số -G or --grain:

VD: (ở đây là ubuntu)

# sudo salt --grain 'os\_family:Debian' test.ping

myminion:

True

# sudo salt -G 'os:Ubuntu' test.ping

myminion:

True

# sudo salt -G 'os:ubuntu' test.ping

myminion:

True

# sudo salt -G 'os:u\*' test.ping

myminion:

True

- List các grains.items hiện có, sử dụng grains.items function.

# sudo salt '\*' grains.items

myminion:

----------

cpu\_model:

Intel(R) Xeon(R) CPU E5-2680 v2 @ 2.80GHz

cpuarch:

x86\_64

id:

myminion

ipv4:

- 127.0.0.1

- 69.164.192.51

ipv6:

- 2600:3c00::f03c:91ff:fe50:3f5d

- ::1

- fe80::f03c:91ff:fe50:3f5d

kernel:

Linux

kernelrelease:

3.15.4-x86\_64-linode45

localhost:

localhost

master:

localhost

os:

Ubuntu

os\_family:

Debian

osarch:

……

Note:

+ grains.items chứa rất nhiều thông tin được đóng gói tại minions. Ta có thể dựa trên những tham số grains để tạo nên nhưng config cho nhưng nền tảng khác nhau.

- Có thể tạo nhưng grains mới

# sudo salt '\*' grains.setval foo bar

salt myminion:

----------

foo:

bar

# sudo salt '\*' grains.item foo

myminion:

----------

foo:

bar

- Tạo grain phức tạp dạng json

# sudo salt '\*' grains.setval baz '["larry", "moe", "curly"]'

myminion:

----------

baz:

- larry

- moe

- curly

# sudo salt '\*' grains.item baz

myminion:

----------

baz:

- larry

- moe

- curly

- Các custom gain sẽ tồn tại cho dù master or minion có restart. Để xóa grains không mong muốn có 2 cách:

+ sử dụng grains.delval (chỉ xóa value)

+ sử dụng grains.delval with destructive=True (xóa cả khóa cả value)

# sudo salt '\*' grains.delval baz

myminion:

None

# sudo salt '\*' grains.item baz

myminion:

----------

baz:

None

# sudo salt '\*' grains.delval baz destructive=True

myminion:

None

# sudo salt '\*' grains.item baz

myminion:

Using pillars

- Pillar data is similar to grains, except that it can be defined more dynamically

and is a secure store for data. We will talk more about the implications of this in

Chapter 5, Expanding Our States with Jinja2 and Pillar. Just note that, since it is also a

key-value store, you can match it just as with grains except that you use the -I or

--pillar flag.

Compound matching- Đây là 1 trong nhưng form matching mạnh mẽ nhất của Salt.

- Cho phép sử dụng các biểu thức matching phức tạp tới các minions. Phương pháp này kết hợp nhiều phương thưc matching

VD:

# sudo salt -C '\*minion and G@os:Ubuntu and not L@yourminion,theirminion'

test.ping

myminion:

True

Remote execution modules and functions

- Thành phân quan trọng của Remote execution command.

- Tất cả các remote execution command đều theo format: <module>.<function>

- Các module được cài đặt mặc định khi install SaltStack.

- Xem chi tiết các module thuộc SaltStack tại:

<https://docs.saltstack.com/en/latest/salt-modindex.html>

Adding users

- Module sử dụng để add user.

# sudo salt '\*' sys.doc user.add

user.add:

Add a user to the minion

CLI Example:

salt '\*' user.add name <uid> <gid> <groups> <home> <shell>

- Trường “name” là trường bắt buộc.

VD:

# sudo salt '\*' user.add larry

myminion:

True

- Xem thông tin user, sử dụng user.info function

# sudo salt '\*' user.info larry

myminion:

----------

fullname:

gid:

1000

groups:

- larry

home:

/home/larry

homephone:

name:

larry

passwd:

x

roomnumber:

shell:

uid:

1000

workphone:

Installing packages

- Sử dụng SaltStack cho việc install packages

- Sử dụng thông qua module pkg.install

# sudo salt '\*' sys.doc pkg.install

pkg.install:

Install the passed package, add refresh=True to update the dpkg

database.

...

CLI Example:

salt '\*' pkg.install <package name>

VD: install apache

# sudo salt '\*' pkg.install apache2

myminion:

----------

apache2:

----------

new:

2.4.7-1ubuntu4.1

old:

apache2-api-20120211:

----------

new:

1

old:

apache2-bin:

----------

new:

2.4.7-1ubuntu4.1

old:

apache2-data:

----------

new:

2.4.7-1ubuntu4.1

old:

- List các package đã install tại minion

# sudo salt '\*' pkg.list\_pkgs

myminion:

----------

adduser:

3.113+nmu3ubuntu3

apache2:

2.4.7-1ubuntu4.1

apache2-api-20120211:

….

- Xem version package

# sudo salt '\*' pkg.version nano

myminion:

2.2.6-1ubuntu1

- Remove package

# sudo salt '\*' pkg.remove htop

myminion:

----------

htop:

----------

new:

old:

1.0.2-3

# sudo salt '\*' pkg.version htop

myminion:

- Sử dụng thuộc tính “--out=json” để get info dạng json

Managing services- quản trị các trạng thái của service, sử dụng module service, giá trị trả lại sẽ ở dạng true or false.

# sudo salt '\*' service.status apache2

myminion:

True

# sudo salt '\*' service.stop apache2

myminion:

True

# sudo salt '\*' service.status apache2

myminion:

False

# sudo salt '\*' service.start apache2

myminion:

True

Monitoring minion states- Sử dụng để quản trị tài nguyên của minion, sử dụng module status

VD: Dung lượng ổ đĩa

# sudo salt '\*' status.diskusage

myminion:

----------

/:

----------

available:

23927775232

total:

25034108928

/dev:

----------

available:

518504448

total:

518508544

VD: quản trị trạng thái trung bình cpu

# sudo salt '\*' status.loadavg

myminion:

----------

1-min:

0.07

15-min:

0.13

5-min:

0.09

VD: Kiểm tra trạng thái memory

# sudo salt '\*' status.meminfo

myminion:

----------

Active:

----------

unit:

kB

value:

420796

Active(anon):

----------

unit:

kB

value:

269900

VD: Trạng thái up time server

# sudo salt '\*' status.uptime

myminion:

22:41:58 up 12 days, 3:44, 1 user, load average: 0.13, 0.10,

0.13

Running arbitrary commands

- Module cmd, sử dụng để thực hiện các cmd từ xa.

- Basic function: cmd.run, nó sẽ trả lại output minion về phía master

# sudo salt '\*' cmd.run 'echo Hello!'

myminion:

Hello!

- Có thể lấy ra nhưng trường đặc biệt của giá trị trả lại như:

+ cmd.run\_stderr, cmd.run\_stdout, cmd.retcode (which return the STDERR,

STDOUT, and return code for the command, respectively)

# sudo salt '\*' cmd.run\_stdout 'echo Hello!'

myminion:

Hello!

# sudo salt '\*' cmd.run\_stderr 'echo Hello!'

myminion:

# sudo salt '\*' cmd.retcode 'echo Hello!'

myminion:

0

- Nếu muốn hiện thị ra tất cả trường

# sudo salt '\*' cmd.run\_all 'echo Hello!'

myminion:

----------

pid:

21243

retcode:

0

stderr:

stdout:

Hello!

- Có thể dùng function cmd.script để chạy các scripts được lưu trên file.

- Để xem docs các function sử dụng sys.docs

Execution Modules – Write Your Own Solution

- Tạo các module riêng, phục vụ cho việc triển khai dịch vụ (own execution module functions)

- Sử dụng python tạo ra các module, yêu cầu có hiểu biết cơ bản về python

Exploring the source

- Về thiết kế, Salt để dễ dang cho việc mở rộng, add thêm nhưng module custom, function. Nó sử dụng các tính chất của ngôn ngữ python.

- Từ đó, các Salt module có thể tương tác với các công cụ mở rộng, các phần mềm thứ 3, thậm chí có thể sử dụng module để hệ thống lại các dữ liệu khác của salt.

VD:

- Hàm cơ bản trong test module. Nó đơn giản là sử dụng function có sẵn trong python để thực thi.

- Chứa docstring, hương dẫn các sử dụng cho user

def sleep(length):

'''

Instruct the minion to initiate a process that will sleep

for a given period of time.

CLI Example:

.. code-block:: bash

salt '\*' test.sleep 20

'''

time.sleep(int(length))

return True

- Không có gì đặc biệt trong stack, ngoài ra ta có thể sử dụng các tool có sẵn của Salt để khiến module trở nên linh hoạt hơn.

Cross-calling execution modules

- Tool đầu tiên Salt cung cấp là khả năng thực thi các module từ 1 module khác

- Sử dụng lại module có sẵn của Salt.

- Từ khóa “\_\_salt\_\_”.

VD:

def refresh\_db():

'''

Updates the APT database to latest packages based upon

repositories

Returns a dict, with the keys being package databases and

the values being the result of the update attempt. Values

can be one of the following:

- ``True``: Database updated successfully

- ``False``: Problem updating database

- ``None``: Database already up-to-date

CLI Example:

.. code-block:: bash

salt '\*' pkg.refresh\_db

'''

ret = {}

cmd = 'apt-get -q update'

out = \_\_salt\_\_['cmd.run\_stdout'](cmd, output\_loglevel='trace')

for line in out.splitlines():

cols = line.split()

if not cols:

continue

ident = ' '.join(cols[1:])

if 'Get' in cols[0]:

# Strip filesize from end of line

ident = re.sub(r' \[.+B\]$', '', ident)

ret[ident] = True

elif cols[0] == 'Ign':

ret[ident] = False

elif cols[0] == 'Hit':

ret[ident] = None

return ret

- Chú ý vào dòng

out = \_\_salt\_\_['cmd.run\_stdout'](cmd, output\_loglevel='trace')

- Nó tham chiếu đến function module thực thi mà ta đã biết.

- cmd.run\_stdout chạy câu lệnh cmd shell trên các minion và trả lại giá trị sdtout lại.

- Sức mạnh của tool “\_\_salt\_\_” được thể hiện tại đây, nó cho phép ta tận dung lại các module có sẵn để thực hiện công việc mong muốn. Các module phụ thuộc vào số module cơ bản được cài đặt sẵn trong salt.

- Có thể viết những function cho việc install packages (pkg.install), add use (user.add), khởi động lại system (system.reboot)

- Thâm chí ta có thể sử dụng “\_\_salt\_\_” để tái tận dụng module mà ta đã viết.

Grains and the \_\_virtual\_\_ function

- 2 tool tiếp theo Salt cung cấp.

VD:

# Define the module's virtual name

\_\_virtualname\_\_ = 'pkg'

def \_\_virtual\_\_():

'''

Confirm this module is on a Debian based system

'''

if \_\_grains\_\_.get('os\_family', False) != 'Debian':

return False

return \_\_virtualname\_\_

- Tập trung vào hàm virtual(), nó là hàm mà Salt sẽ kiểm tra mỗi khi loading execution modules. Nó được sử dụng cho các mục đích:

+ Giúp Salt quyết định khi nào load module

+ Nó có thể đổi tên module khi cần thiết

- Trong trường hợp aptpkg, đầu tiền nó sử dụng “\_\_grains\_\_” (hỗ trợ việc truy cập vào các giá trị mô tả minions) để quyết định khi nào load module cho minion

if \_\_grains\_\_.get('os\_family', False) != 'Debian':

return False

- Nếu “\_\_virtial\_\_” function trả lại false, Salt sẽ không load bất kỳ function nào nữa.

- Nếu sau khi check thành công, đến dòng:

return \_\_virtualname\_\_

- Ở đây, giá trị trả lại là \_\_virtualname\_\_, bất kỳ string nào được trả lại tại dòng nãy sẽ được sử dụng như module name nằm trong Salt

- Nếu dòng này trả lại true hoặc false, nó sẽ load module tiếp theo.

VD: yumpkg.py

|  |  |
| --- | --- |
| # Define the module's virtual name  \_\_virtualname\_\_ = 'pkg'  def \_\_virtual\_\_():  '''  Confine this module to yum based systems  '''  if \_\_opts\_\_.get('yum\_provider') == 'yumpkg\_api':  return False  try:  os\_grain = \_\_grains\_\_['os'].lower()  os\_family = \_\_grains\_\_['os\_family'].lower()  except Exception:  return False  enabled = ('amazon', 'xcp', 'xenserver')  if os\_family == 'redhat' or os\_grain in enabled:  return \_\_virtualname\_\_  return False | # Define the module's virtual name  \_\_virtualname\_\_ = 'pkg'  def \_\_virtual\_\_():  '''  Confirm this module is on a Debian based system  '''  if \_\_grains\_\_.get('os\_family', False) != 'Debian':  return False  return \_\_virtualname\_\_ |

The\_\_opts\_\_ and \_\_pillar\_\_ functions

- 2 tool tiếp theo được cung cấp bởi Salt: \_\_opts\_\_ và \_\_pillar\_\_.

- “\_\_opts\_\_” cho phép truy cập vào các config option tại trong execution modules.

- Có thể sử dụng để thay đổi các config nằm bên trong file cấu hình minions.

- Về “pillar” chúng ta sẽ không xem xét sâu, pillar bảo đảm sự an toàn cho data nằm tại minion

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

Reformatting return data

- Thay đổi list user thông qua custom module.

VD:

- cmd list\_users hiện thị ra list user theo dạng cây

# sudo salt '\*' user.list\_users

myminion:

- backup

- bin

- daemon

…

- Ta viết module đơn giản, custom lại cách hiện thị của user list

Bước 1:

- Tạo thư mục chứa các module custom:

# sudo mkdir -p /srv/salt/\_modules/

Bước 2:

- Tạo file customuser.py:

# sudo cd /srv/salt/\_modules/

# sudo vim customuser.py

….

def users\_as\_csv():

'''

Retrieve the users from a minion, formatted

as comma-separated-values (CSV)

CLI Example:

.. code-block:: bash

salt '\*' customuser.users\_as\_csv

'''

user\_list = \_\_salt\_\_['user.list\_users']()

csv\_list = ','.join(user\_list)

return csv\_list

Bước 3:

- Đồng bộ module cho tất cả minions

# sudo salt '\*' saltutil.sync\_all

myminion:

----------

grains:

modules:

- modules.customuser

outputters:

….

Bước 4:

- Gọi hàm vừa tạo:

# sudo salt '\*' customuser.users\_as\_csv

myminion:

backup,bin,daemon,games,gnats,irc,larry,libuuid,list,lp,mail,man,mess

agebus,news,nobody,ntp,proxy,root,sshd,sync,sys,syslog,uucp,www-data

Kiểm tra:

- Kiểm tra các hàm đã tạo

# sudo salt '\*' sys.list\_functions customuser

myminion:

- customuser.users\_as\_csv

- Kiểm tra docs:

# sudo salt '\*' sys.doc customuser.users\_as\_csv

Retrieve the users from a minion, formatted

as comma-separated-values (CSV)

CLI Example:

salt '\*' customuser.users\_as\_csv

- List các module hiện có

# sudo salt '\*' sys.list\_modules

myminion:

...

- config

- cp

- cron

- customuser

- daemontools

- data

- debconf

...

An advanced example

- Truy cập: [https://github.com/saltstack/salt/blob/develop/salt/modules/](https://github.com/saltstack/salt/blob/develop/salt/modules/file.py)

- Xem các ví dụ từ nhưng module thực tế

Defining the State of Your Infrastructure

- Cấu trúc các state và cách viết các state

- How to expand our state declarations to encompass multiple pieces of a state

- About the various pieces of the state declaration

- About ordering states with requisites

Our first state- Tất cả các file Salt đặc biệt ko thuộc python file và có đuôi mở rộng là .sls.

- Mặc định, các states được lưu trữ tại thư mục “/srv/salt/..”.

# mkdir -p /srv/salt

# cd /srv/salt

Trong thư mục, tạo file apache.sls với nội dung:

# vim apache.sls

###

install\_apache:

pkg.installed:

- name: apache2

Note:

+ State file cấu trúc sử dụng YAML (Yet Another Markup Language)

+ gói tin apache2 (cho ubuntu) nếu sử dụng centos sử dụng gói httpd

- Chạy file state

# sudo salt '\*' state.sls apache

myminion:

----------

ID: install\_apache

Function: pkg.installed

Name: apache2

Result: True

Comment: Package apache2 is already installed.

Started: 20:21:00.498735

Duration: 750.402 ms

Changes:

Summary

------------

Succeeded: 1

Failed: 0

------------

Total states run: 1

- Nếu remove package apache và chạy lại state custom, sẽ thấy được state thay đổi.

# sudo salt '\*' pkg.remove apache2

myminion:

----------

apache2:

----------

new:

old:

2.4.7-1ubuntu4.1

# sudo salt '\*' state.sls apache

myminion:

----------

ID: install\_apache

Function: pkg.installed

Name: apache2

Result: True

Comment: The following packages were installed/updated: apache2.

Started: 20:53:30.187526

Duration: 10185.321 ms

Changes:

----------

apache2:

----------

new:

2.4.7-1ubuntu4.1

old:

Summary

------------

Succeeded: 1 (changed=1)

Failed: 0

------------

Total states run: 1

Note:

+ Lưu ý đến trạng thái thay đổi của service, pkg được thiết kế để đưa service đến trạng thái mong muốn với ít xử lý nhất:

- Các thao tác hỗ trợ:

+ pkg.install = install package

+ pkg.installed = bảo đảm gói tin đã được cài đặt, nếu chưa thì nó sẽ cài, nếu đã nó sẽ bỏ qua việc cài

- Để xem được các function và ý nghĩa:

# sudo salt '\*' sys.list\_state\_functions pkg

# sudo salt '\*' sys.state\_doc pkg.removed

The pieces of a state declaration- Các thành phần của 1 State:

|  |  |
| --- | --- |
| <ID Declaration>:  <State Module>.<Function>:  - name: <name>  - <Function Arg>  - <Function Arg>  - <Function Arg>  - <Requisite Declaration>:  - <Requisite Reference> | install\_apache:  pkg.installed:  - name: apache2 |

- Để xem các nâng cao:

<https://docs.saltstack.com/en/latest/ref/states/highstate.html#large-example>

Note:

+ Trường ID

Expanding to encompass multiple pieces of state

- Mở rộng State thành nhiều phần

- Trong ví dụ install apache, sau khi cài đặt, service sẽ ko được chạy và chưa được setup gì cả

- Ta có thể sử dụng module service để quản lý service

# sudo salt '\*' service.status apache2

myminion:

False

# sudo salt '\*' service.start apache2

myminion:

True

# sudo salt '\*' service.status apache2

myminion:

True

# Chắc chắn service hoạt động

# sudo salt '\*' service.status service.running apache2

# Hoạt động khi boot

# sudo salt '\*' service.status service.enable apache2

- Để việc chạy service tự động, ta thêm vào file state apache.sls

install\_apache:

pkg.installed:

- name: apache2

make sure apache is running:

service.running:

- name: apache2

- enable: True

- Trong file trên, ID là “make sure …”, id này thỏa mãn định dang và dễ đọc hiểu cho user:

Dependencies using requisites

- Sử dụng phụ thuộc giữa các thành phần trong state

The require requisite

- về cơ bản, ta thường sử dụng “require”, nó ràng buộc đợi 1 state nào đó hoạt động trước khi nó được thực hiện, và đảm báo 1 thứ tự định sẵn

VD: bảo đảm service apache đã được cài đặt trước khi chạy.

install\_apache:

pkg.installed:

- name: apache2

make sure apache is running:

service.running:

- name: apache2

- enable: True

- require:

- pkg: install\_apache

Note:

- Nếu 1 thành phần thuộc service.running sai thì hoạt động install\_apache vẫn được chạy, nhưng khi đến state service.running thì nó sẽ dừng lại.

The watch requisite

- Trong phần này ta sẽ cung cấp các file cho apache thực thi

- Ta sử dụng module file, đặc biệt sẽ sử dụng “file.managed”, để xem docs:

# sudo salt '\*' sys.state\_doc file.managed

- Trong những state này, ta sẽ cần các tài nguyên có sẵn tên master để cung cấp cho các minions như 1 phần của state.

VD:

- ta sẽ tạo 2 file:

+ mod\_status.conf

/srv/salt/mod\_status.conf

###

<Location /server-status>

SetHandler server-status

Order allow,deny

Allow from all

</Location>

+ mod\_status.load

/srv/salt/mod\_status.load

LoadModule status\_module /usr/lib/apache2/modules/mod\_status.so

- Sửa lại file apache.sls

…..

install\_apache:

pkg.installed:

- name: apache2

sync mod\_status.conf:

file.managed:

- name: /etc/apache2/mods-enabled/mod\_status.conf

- source: salt://mod\_status.conf

- user: root

- group: root

- mode: 600

sync mod\_status.load:

file.managed:

- name: /etc/apache2/mods-enabled/mod\_status.load

- source: salt://mod\_status.load

- user: root

- group: root

- mode: 600

- salt:// là giao thức được cung cấp bởi salt, nó sẽ chọn các phần được lưu trữ trên salt master

- Đặc biệt, vì chúng ta thêm 2 file cấu hình vào sau khi service đã chạy => ta cần phải restart lại service. Nó được hỗ trợ bởi “watch requisite”

- “watch” nó hoạt động khá giống với “require” nhưng nó chỉ được kích hoạt khi có state phụ thuộc thay đổi hay được kích hoạt

VD:

make sure apache is running:

service.running:

- name: apache2

- enable: True

- require:

- pkg: install\_apache

- watch:

- file: sync mod\_status.conf

- file: sync mod\_status.load

sync mod\_status.conf:

…

sync mod\_status.load:

…

Other requisites

- onfail: This requisite is used to run a state only if another state has failed.

It's especially useful for self-healing infrastructures and for rolling back code

deployments after failure.

- onchanges: This requisite is used to run a state only if another state created

changes. It differs from the watch requisite in that when there are no

changes, it does not execute at all.

- prereq: This requisite is one of the most complex requisites. Let's assume

that a service needs to be shut down (service.dead) before changes are

made to a file, but only if changes will be made to the file. This is a use case

for which prereq is perfectly suited. Basically, prereq allows a state to run

only if another state is going to have changes when it is run in the future. We

"look into the future" using a test run of the state that will possibly make the

changes. See the Salt documentation for more details and examples.

- use: This requisite allows a state to inherit the arguments of another state,

overwriting as necessary

- Xem chi tiết tại:

https://docs.saltstack.com/en/latest/ref/states/requisites.html

The \_in requisites:

- Được thêm vào các state sử dụng để ràng buộc, 2 phụ thuộc thường dùng “(watch\_in and require\_in)

VD:

|  |  |
| --- | --- |
| # Đơn giản  install\_apache:  pkg.installed:  - name: apache2  make sure apache is running:  service.running:  - name: apache2  - enable: True  - require:  - pkg: install\_apache | # Sử dụng \_in  install\_apache:  pkg.installed:  - name: apache2  - require\_in:  - service: make sure apache is running  make sure apache is running:  service.running:  - name: apache2  - enable: True |

- Khi sử dụng \_in, pkg sẽ thông báo cho service.running yêu cầu pkg trước khi chạy service.

Expanding Our States with Jinja2 and Pillar

- Sử dụng để config SaltStack đa nển tảng 1 cách linh hoạt

- Cài đặt nhanh Salt Minion thông qua bash script cung cấp

Jinja2

- Là 1 template của Python, cho phép cung cấp các kỹ thuật chỉnh sửa các code block sao cho phù hợp với môi trường, hạ tầng linh hoạt.

- Jinja2 được mô phỏng theo Django template language

- Cú pháp của Jinja2 gần giống với python

- 2 phần chính trong Jinja2 syntanxes khi sử dụng trong Salt.

+ Biến, được hiện thị bởi “{{….}}”

VD:

{{ foo }}

{{ foo.bar }}

{{ foo['bar'] }}

{{ get\_data() }}

+ Các biểu thức, được biểu diễn bằng “{%....%}”

VD:

{% if myvar == 'foo' %}

somecontent

{% elif myvar == 'bar' %}

othercontent

{% else %}

morecontent

{% endif %}

- Ví dụ về 1 template đơn giản

{% for user in ['larry', 'moe', 'curly'] %}

It's user {{ user }}!

Hello {{ user }}!

{% endfor %}

# Set giá trị

{% set myvar = 'foo' %}

apache2 or httpd?

- Ubuntu sử dụng gói tin apache, centos sử dụng httpd

- Sử dụng Jinja để làm linh động việc tùy chọn gói tin cho apache

- Tùy chình apache.sls (chỉnh gói tin và đường dẫn cấu hình)

|  |  |
| --- | --- |
| install\_apache:  pkg.installed:  {% if grains['os\_family'] == 'Debian' %}  - name: apache2  {% elif grains['os\_family'] == 'RedHat' %}  - name: httpd  {% endif %} | make sure apache is running:  service.running:  {% if grains['os\_family'] == 'Debian' %}  - name: apache2  {% elif grains['os\_family'] == 'RedHat' %}  - name: httpd  {% endif %}  - enable: True  - require:  - pkg: install\_apache  - watch:  - file: sync mod\_status.conf  {% if grains['os\_family'] == 'Debian' %}  - file: sync mod\_status.load  {% endif %} |
| sync mod\_status.conf:  file.managed:  {% if grains['os\_family'] == 'Debian' %}  - name: /etc/apache2/mods-enabled/mod\_status.conf  {% elif grains['os\_family'] == 'RedHat' %}  - name: /etc/httpd/conf.d/mod\_status.conf  {% endif %}  - source: salt://mod\_status.conf  - user: root  - group: root  - mode: 600 | {% if grains['os\_family'] == 'Debian' %}  sync mod\_status.load:  file.managed:  - name: /etc/apache2/mods-enabled/mod\_status.load  - source: salt://mod\_status.load  - user: root  - group: root  - mode: 600  {% endif %} |

Defining secure minion-specific data in pillar

- Định nghĩa các giá trị pillar cho minions

- Tất cả file or data trong /srv/salt sẽ được chia sẻ cho minions

- Ta sử dụng Salt để tạo ra các giá trị đặc biệt => nó được gọi là pillar

- Giống như grains, pillar system là 1 cặp khóa được lưu tại Salt. Tuy nhiên, mỗi minion sẽ có 1 tập pillar data riêng, được encrypted trên mỗi minions, giá trị này phục vụ cho những hành động đặc biệt

- Pillar files được lưu trữ trong thư mục “/srv/pillar”

mkdir /srv/pillar

cd /srv/pillar

- Tại thự mục “/srv/pillar/” tạo file “core.sls”. cấu trúc file sử dụng YAML

foo: bar

users:

- larry

- moe

- curly

some\_more\_data: data

- Để minions nhận được data, ta sẽ sử dụng 1 file đặc biệt “/srv/pillar/top.sls”, nó là topfile:

base:

'\*':

- core

'os\_family:debian':

- match: grain

- ssh\_key

- Có rất nhiều vấn đề trong top file.

+ Sử dụng YAML

+ Có cấu trúc đặc biệt

- Dòng đâu tiên “base:” xác định môi trường, sử dụng default environment

base:

- Đến dòng tiếp theo, thuộc phần target string. Có thể định nghĩa lại ‘\*’ nếu cần thiết. ở đây tất cả minions (‘\*’) sẽ nhận giá trị từ list các file (ở đây là core.sls)

'\*':

- core

- Dòng tiếp theo cũng là target string ('os\_family:debian'). Tại đây, ta sử dụng grain matching thay vì globbing (- match: grain), Debian minions cũng sẽ nhận được pillar data định nghĩa trong ssh\_key.sls (- ssh\_key).

'os\_family:debian':

- match: grain

- ssh\_key

- Pillar data được tự động reset khi ta chạy bất kỳ 1 state nào. Tùy nhiên, trong 1 số trường hợp ta cần reset ngay lập tức, sử dụng function “saltutil.refresh\_pillar”

# sudo salt '\*' saltutil.refresh\_pillar

centminion:

True

myminion:

True

- Kiểm tra lại pillar

# sudo salt '\*' pillar.items

# sudo salt '\*' pillar.item users

Using pillar data in states

- Sử dụng pillar data để định nghĩa states.

- Tạo state file trong /srv/salt/users\_and\_ssh.sls

{% for user in pillar['users'] %}

add\_{{ user }}:

user.present:

- name: {{ user }}

{% endfor %}

{% if 'my\_ssh\_key' in pillar %}

manage\_my\_ssh\_key:

file.managed:

- name: /root/.ssh/{{ pillar['my\_ssh\_key\_name'] }}

- mode: 600

- contents\_pillar: my\_ssh\_key

- show\_diff: False

{% endif %}

- Trên vd, ta sử dụng pillar để tạo user và chỉ có minions chứa “my\_ssh\_key” mới nhận được chứng chỉ ssh

The Highstate and Environments

- Quản lý các state, thi hành đồng loạt các state với 1 cmd

- Cho tới hiện tại, ta chủ yếu sử dụng các state file, nhưng khi hệ thống mở rộng, chúng ta sẽ có nhiều state file và sinh ra nhiều module. Vậy làm thế nào để quản lý?

- Tại phần trước, chúng ta sử dụng top.sls file để thi hành áp dụng các pillar. Top file còn có thể sử dụng trong state system để target các minions bằng pillar.

- thêm mới file “/srv/salt/top.sls” file

|  |  |
| --- | --- |
| # /srv/pillar/top.sls  base:  '\*':  - core  'osfinger:CentOS-6':  - match: grain  - ssh\_key | # /srv/salt/top.sls  base:  '\*minion':  - apache  'os\_family:debian':  - match: grain  - users\_and\_ssh |

- File mới gần như giống với cấu trức của topfile của pillar, nhưng tại đây nó sẽ gọi các state thay vì đưa ra các pillar

Note:

+ Ở đây có 2 topfile trong “/srv/pillar” và “/srv/salt”

- Để chạy state file này, sử dụng

# sudo salt '\*' state.highstate

###

myminion:

----------

……

centminion:

---------

……

Environments- Salt cung cấp khái niệm về environments cho mục đích tổ chức các state.

- Tới thời điểm hiện tại, ta chủ yếu sử dụng base environment

- Chúng ta có thể tạo ra nhiều environment để quản lý hạ tầng và tổ chức kho lưu trữ mỗi env theo các cách khác nhau.

- Cấu hình location cho môi trường trên “master:” nằm tại master config file “/etc/salt/master”.

VD: Tạo mới environment “webserver”

file\_roots:

base:

- /srv/salt

webserver:

- /srv/web

Note:

+ Environment sử dụng để chia cắt các tài nguyên thành nhiều phần, hỗ trợ cho việc quản lý các state thay vì để tất cả chung vào 1 env.

+ VD: sau này ta có thể mở rộng thành env cho database, web, ..

+ VD: có thể quản lý env theo môi trường: prod, test, deploy …

- Tổ chức lại các file của web server: Tạo thêm thư mục “/srv/web”

# sudo mkdir /srv/web

# sudo mv /srv/salt/apache.sls /srv/web/

# sudo mv /srv/salt/mod\_status.\* /srv/web/

- Các file sau khi thay đổi thư mục gần như ko phải thay đổi gì vì đã chia lại env

- Sửa lại cấu trúc “/srv/salt/top.sls”

base:

'os\_family:debian':

- match: grain

- users\_and\_ssh

webserver:

'\*minion':

- apache

- Kiểm tra lại:

# sudo salt '\*' state.highstate

- Nếu chính xác, kết quả vẫn giống như khi chưa chia env

- Trong một số trường hợp ta có thể tách riêng thành 2 top.sls cho 2 env mà ko gây xung đột. Nhưng tốt nhất và sử dụng 1 top file, và lưu nó tại root env “/srv/salt/top.sls”

|  |  |
| --- | --- |
| # /srv/salt/top.sls  base:  'os\_family:debian':  - match: grain  - users\_and\_ssh | # /srv/web/top.sls  webserver:  '\*minion':  - apache |

Environments in pillar

- Bỏ qua phần này, tìm hiểu trong docs

Expanding our base environment

- Tổ chức lại các state, module theo chức năng riêng của nó

VD:

###/srv/salt/top.sls

base:

'\*':

- myuser.user

- myuser.dotfiles

'os\_family:debian':

- match: grain

- users\_and\_ssh

webserver:

'\*minion':

- apache

- “myuser.user” tương đương với “myuser/user.sls”, “myuser” đại diện cho folder, nó gần giống như các python import directory

- Tạo thư mục /srv/salt/myuser

# sudo mkdir /srv/salt/myuser

- Tạo “/srv/salt/myuser/user.sls”, state file này sử dụng để install package và tạo user basepi.

install\_zsh:

pkg.installed:

- name: zsh

add\_user\_basepi:

user.present:

- name: basepi

- shell: /bin/zsh

- require:

- pkg: install\_zsh

- Tạo “/srv/salt/myuser/dotfiles.sls”.

include:

- myuser.user

install\_git:

pkg.installed:

- name: git

clone\_dotfiles:

git.latest:

- name: git://github.com/basepi/dotfiles.git

- rev: master

- target: /home/basepi/dotfiles

- user: basepi

- submodules: True

- require:

- pkg: install\_git

- user: add\_user\_basepi

install\_dotfiles\_if\_changed:

cmd.run:

- name: 'python install.py -y'

- cwd: '/home/basepi/dotfiles'

- user: basepi

- onchanges:

- git: clone\_dotfiles

- trong file sử dụng “include:” để có thể tạo ràng buộc giữa 2 state khác nhau

- Chạy và xem kết quả

# sudo salt '\*' state.highstate

The Reactor and the Event System

- Khiến Salt trở nên linh hoạt, nó sẽ phản ứng lại và tự sửa lỗi thông qua events và reactor.

The Salt event system

- Event là 1 thành phần không thể thiếu của Salt

- Nó được thiết kế để ko nhìn thấy và hoạt động bên dưới, cung các tính liên thông giữa các tiến trình, làm giảm sự phức tạp của hệ thống

- Có thể sử dụng nó để giải quyết 1 số vấn đề phát sinh

Listening for events- Trước khi tìm hiểu reacting to events, hoặc firing our own, cần biết được khi nào event xảy ra

- Salt runner function (state.event) được cung cấp cho mục đích này

- Sử dụng cmd để lắng nghe các sự kiện xảy ra

# sudo salt-run state.event

Note:

+ Có thể sẽ ko hiện thị giá trị gì

+ Chạy test.ping để xem 1 event được sinh ra

# sudo salt-run state.event

###

salt/job/20160513030202312339/new {"tgt\_type": "glob",

"jid": "20160513030202312339", "tgt": "\*", "\_stamp": "2016-05-

13T03:02:02.312623", "user": "root", "arg": [], "fun": "test.ping",

"minions": ["centminion", "myminion"]}

salt/auth {"\_stamp": "2016-05-13T03:07:55.004984", "act":

"accept", "id": "myminion", "pub": "-----BEGIN PUBLIC KEY-----\

nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAtHFZaLGBe71IIbjPyJGe\

n1fx/hTUJNMhT+O/Wh21YzdAN8JdQVgDLpevu+Ww5DoYQHrYnyoxIl94foj35fEak\

nIoL2mA+aWBaAQrV6CI2I/PVMAL8uOOQfypuTQYyqHw9sj2zbGvDAS4zCrw8nUdtS\n1ZbPW

Ja5SwK5Xf3XsQDJFIfrGNAQLvdrvlDIYO8w6pTZgqVudb0cgU6gB2nL7g+y\nkwsXY4Ggzbi

NKVu1I0h8MRZM5bD73+XsesABgNoHuSXnpgRTWgVVrXoYjPhoNHxa\n66snGwK/w0rvEFuyTM

s6WG4+YsmKGRQkB0MdGLglrNpXkb51cAvxKuVVAQFYXQ2C\nUwIDAQAB\n-----END PUBLIC

KEY-----", "result": true}

salt/job/20160513030202312339/ret/myminion {"fun\_args": [], "jid":

"20160513030202312339", "return": true, "retcode": 0, "success": true,

"cmd": "\_return", "\_stamp": "2016-05-13T03:02:02.495713", "fun": "test.

ping", "id": "myminion"}

salt/job/20160513030202312339/ret/centminion {"fun\_args": [], "jid":

"20160513030202312339", "return": true, "retcode": 0, "success": true,

"cmd": "\_return", "\_stamp": "2016-05-13T03:02:02.500055", "fun": "test.

ping", "id": "centminion"}

- Xem qua ta có thể thấy được quá trình phát sinh 1 event khi 1 job được tạo.

- event data chứa function “test.ping” như 1 tham số gửi về client.

- Event format thứ 2 nhằm mục đích xác thực lại giữa master và minion. Event này sẽ thường xuyên xảy ra

- 2 event tiếp theo trả lại kết quả của 2 minion sau khi thực thi câu job test.ping. Ta có thể thấy trường job ID (jid), kết quả trả lại (true), func, id và cả thời gian.

Event structure

- Event được tại ra từ 2 phần.

+ event tag: được sử dụng để xác định events

+ event data

VD:

salt/job/20141127001155262018/ret/myminion

- Custom event tag cần được theo 1 cấu trúc nào đó, nó gần giống 1 URL.

- Cần có “salt” để xác định nó là 1 event

- “/job” để xác định nó 1 job; “/jid” xác định id job; “~/ret” return data; “~/myminion” minion id.

- Có thể tạo 1 custom tag

salt/custom/mycoolapp/failure

Firing events from Salt

- Có 1 số cách để sinh ra event từ phía minion.

- Các đơn giản nhất là sử dụng “event.send”

# sudo salt '\*' sys.doc event.send

- Sử dụng cmd salt-call để chạy function này tại phía minions

# sudo salt-call event.send 'salt/custom/mycoolapp/failure' '{"foo":

"bar"}'

local:

True

- Tại master, ta có thể thấy event này

salt/custom/mycoolapp/failure {"data": {"\_\_pub\_fun": "event.send", "\_\_

pub\_jid": "20160513030755036183", "foo": "bar", "\_\_pub\_pid": 3072, "\_\_

pub\_tgt": "salt-call"}, "\_stamp": "2016-05-13T03:07:55.046303", "cmd": "\_

minion\_event", "tag": "salt/custom/mycoolapp/failure", "id": "myminion"}

Firing events from custom code- Salt cung cấp api cho phép sinh ra 1 event thông qua Python

- Sinh ra event tại master:

# Import the proper library

import salt.utils.event

# Create and fire event

sock\_dir = '/var/run/salt/master'

payload = {'sample-msg': 'this is a test',

'example': 'this is the same test'}

event = salt.utils.event.SaltEvent('master', sock\_dir)

event.fire\_event(payload, 'salt/mycustomtag')

- Sinh ra event tại client:

import salt.client

caller = salt.client.Caller()

caller.function('event.send',

'salt/mycustomtag',

{'foo': 'bar'})

Xem thêm: <https://docs.saltstack.com/en/latest/ref/clients/>

Reacting to events

- phản ứng lại khi phát sinh event.

- Reacter config gồm 2 phần.

+ master config file, định nghĩa events

+ reactor file, nó sẽ định nghĩa hành động sẽ xảy ra khi event phát sinh.

VD:

Tại master config file:

- master sẽ đáp ứng tất cả event có tag “salt/custom/\*”, thực hiện hành động trong file reactor.sls

(/etc/salt/master):

reactor:

- 'salt/custom/\*':

- salt://reactor.sls

- Tạo file reactor.sls trong “/srv/salt”

/srv/salt/reactor.sls

remove\_marker:

local.cmd.run:

- tgt: '\*'

- arg:

- 'rm /tmp/marker'

{% if data['tag'] == 'salt/custom/mycustomapp/failure' %}

run\_a\_highstate:

local.state.highstate:

- tgt: '\*'

- kwarg:

pillar:

trigger\_event\_tag: {{ data['tag'] }}

{% endif %}

- Tại minion, nếu sinh ra event “salt/custom”, nó sẽ chạy state reactor.sls

# salt-call event.send 'salt/custom/somecustomevent' '{}'

# sudo salt-call event.send 'salt/custom/mycustomapp/failure' '{}'

local:

True

- Xem event với function:

# sudo salt-run jobs.list\_jobs

20141128051734805456:

----------

Arguments:

|\_

----------

\_\_kwarg\_\_:

True

….

e