**linux系统之Saltstack**

**老男孩老师教学与培训核心思想：重目标、重思路、重方法、重实践、重习惯、重总结。**

# 序：部署规划

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 主机名 | IP | 角色 |
| 1 | linux-node1.example.com | 10.0.0.7 | salt-master、salt-minion |
| 2 | linux-node2.example.com | 10.0.0.8 | salt-minion |

# 1-安装部署Saltstack

## 1.1-安装salt-master和salt-minion（10.0.0.7）

**安装salt-master salt-minion**

yum install salt-master salt-minion -y

**启动salt-master**

/etc/init.d/salt-master start

**修改minion端配置文件**

vim /etc/salt/minion

16行改为 **master: 10.0.0.7**

**启动salt-minion**

/etc/init.d/salt-minion start

**查看所有minion端发送到master的key**

[root@master master]# salt-key

Accepted Keys:

Denied Keys:

Unaccepted Keys:

linux-node1.example.com

linux-node2.example.com

Rejected Keys:

**接受所有minion端发送到master的key**

[root@master master]# salt-key -A

The following keys are going to be accepted:

Unaccepted Keys:

linux-node1.example.com

linux-node2.example.com

Proceed? [n/Y] Y

Key for minion linux-node1.example.com accepted.

Key for minion linux-node2.example.com accepted

## 1.2-安装salt-minion（10.0.0.8）

**修改minion端配置文件**

vim /etc/salt/minion

16行改为 master: 10.0.0.7

/etc/init.d/salt-minion start

**安装salt-minion**

yum install salt-minion -y

**修改minion端配置文件**

vim /etc/salt/minion

16行改为 **master: 10.0.0.7**

**启动salt-minion**

/etc/init.d/salt-minion start

# 2-Saltstack配置管理

## 2.1-state模块

[root@linux-node1 ~]# vim /etc/salt/master #注释打开

file\_roots:

base:

- /srv/salt

[root@linux-node1 ~]# mkdir /srv/salt -p #创建/srv/salt目录

[root@linux-node1 ~]# cd /srv/salt/

[root@linux-node1 salt]# vim apache.sls #编写一个.sls文件，注意空格的个数

apache-install:

pkg.installed:

- names:

- httpd

- httpd-devel

apache-service:

service.running:

- name: httpd

- enable: True

- reload: True

[root@linux-node1 salt]# salt '\*' state.sls apache #使用salt的state模块的sls方法执行apache.sls

[root@linux-node2 ~]# ps aux|grep yum #在minion端可以查看到apache正在安装

root 17325 26.6 2.5 236304 25840 ? S 10:16 0:02 /usr/bin/python /usr/bin/yum -q check-update

root 17343 0.0 0.0 103308 848 pts/3 S+ 10:16 0:00 grep yum

[root@linux-node1 salt]# pwd

/srv/salt

[root@linux-node1 salt]# vim top.sls #在/srv/salt编写入口文件top.sls

base:

'\*':

- apache

[root@linux-node1 salt]# salt '\*' state.highstate #使用salt的state模块的highstate方法执行top.sls

## 2.2-Grains模块

**数据系统Grains：收集系统启动时的启动信息，收集完之后系统信息就不会变了，只能重启minion才能重新收集**

[root@linux-node1 salt]# salt '\*' grains.ls #显示所有Grains的key值

[root@linux-node1 salt]# salt '\*' grains.items #显示所有Grains的key值和相应信息

[root@linux-node1 salt]# salt '\*' grains.item fqdn #显示单个key

linux-node2.example.com:

----------

fqdn:

linux-node2.example.com

linux-node1.example.com:

----------

fqdn:

linux-node1.example.com

[root@linux-node1 salt]# salt '\*' grains.get fqdn #显示单个key的相应信息，不会显示key的名称

linux-node2.example.com:

linux-node2.example.com

linux-node1.example.com:

linux-node1.example.com

**使用Grains用来匹配minion**

例如：在所有CentOS的系统上进行操作

[root@linux-node1 salt]# salt '\*' grains.get os

linux-node2.example.com:

CentOS

linux-node1.example.com:

CentOS

[root@linux-node1 salt]# salt -G os:CentOS cmd.run 'w'

linux-node2.example.com:

11:39:32 up 18:10, 1 user, load average: 0.06, 0.04, 0.00

USER TTY FROM LOGIN@ IDLE JCPU PCPU WHAT

root pts/3 10.0.0.1 00:10 16:32 0.13s 0.13s -bash

linux-node1.example.com:

11:39:33 up 18:10, 1 user, load average: 0.01, 0.03, 0.00

USER TTY FROM LOGIN@ IDLE JCPU PCPU WHAT

root pts/1 10.0.0.1 00:10 2.00s 1.62s 1.40s /usr/bin/python

#-G表示适用Grains进行匹配

**通过修改配置文件自定义Grains**

[root@linux-node1 salt]# vim /etc/salt/minion #打开注释

grains:

roles:

- webserver

- memcache

[root@linux-node1 salt]# /etc/init.d/salt-minion restart

[root@linux-node1 salt]# salt -G roles:memcache cmd.run 'echo hehe'

linux-node1.example.com:

hehe

**第二种自定义Grains的方式：**

[root@linux-node1 salt]# vim /etc/salt/grains

web: nginx

**#自定义的Grains（即web）必须唯一**

[root@linux-node1 salt]# /etc/init.d/salt-minion restart

Stopping salt-minion daemon: [ OK ]

Starting salt-minion daemon: [ OK ]

[root@linux-node1 salt]# salt -G web:nginx cmd.run 'w'

linux-node1.example.com:

11:56:06 up 18:26, 1 user, load average: 0.13, 0.05, 0.01

USER TTY FROM LOGIN@ IDLE JCPU PCPU WHAT

root pts/1 10.0.0.1 00:10 1.00s 1.71s 1.37s /usr/bin/python

**第三种方法：使用top.sls进行匹配**

[root@linux-node1 salt]# vim top.sls

base:

'web:nginx':

- match: grain #此处grain没有s

- apache

[root@linux-node1 salt]# salt '\*' state.highstate

## 2.3-Pillar模块

**数据系统：Pillar，给minion指定它可以看到的的数据**

[root@linux-node1 salt]# salt '\*' pillar.items

linux-node2.example.com:

----------

linux-node1.example.com:

----------

#默认配置Pillar是关掉的，所以看不到任何数据

**修改master配置文件开启Pillar**

[root@linux-node1 salt]# vim /etc/salt/master #注释打开并将False改为True

pillar\_opts: True

[root@linux-node1 salt]# /etc/init.d/salt-master restart

[root@linux-node1 salt]# salt '\*' pillar.items

**自定义Pillar**

[root@linux-node1 salt]# vim /etc/salt/master #注释打开并将pillar\_opts: True改为pillar\_opts: False

pillar\_roots:

base:

- /srv/pillar

[root@linux-node1 salt]# /etc/init.d/salt-master restart

[root@linux-node1 pillar]# cat /srv/pillar/apache.sls

{% if grains['os'] == 'CentOS' %}

apache: httpd

{% elif grains['os'] == 'Debian' %}

apache: apache2

{% endif %}

[root@linux-node1 salt]# cat /srv/pillar/top.sls #指定哪些minion可以看到这个Pillar

base:

'\*':

- apache

[root@linux-node1 pillar]# salt '\*' saltutil.refresh\_pillar #刷新pillar

linux-node2.example.com:

True

linux-node1.example.com:

True

[root@linux-node1 pillar]# salt '\*' pillar.items #显示pillar的所有items

linux-node1.example.com:

----------

apache:

httpd

linux-node2.example.com:

----------

apache:

httpd

[root@linux-node1 pillar]# salt -I 'apache:httpd' test.ping #使用pillar定位主机

linux-node2.example.com:

True

linux-node1.example.com:

True

# 3-远程执行：

salt '\*' cmd.run 'w'

命令 目标 模版.方法 参数

#返回结果↓

linux-node2.example.com:

13:50:27 up 20:21, 1 user, load average: 0.00, 0.00, 0.00

USER TTY FROM LOGIN@ IDLE JCPU PCPU WHAT

root pts/3 10.0.0.1 00:10 2:27m 0.13s 0.13s -bash

linux-node1.example.com:

13:50:27 up 20:21, 1 user, load average: 0.13, 0.03, 0.01

USER TTY FROM LOGIN@ IDLE JCPU PCPU WHAT

root pts/1 10.0.0.1 00:10 1.00s 1.86s 1.37s /usr/bin/python

## 3.1-Matching the Minion ID

Each minion needs a unique identifier. By default when a minion starts for the first time it chooses its FQDN as that identifier. The minion id can be overridden via the minion's [id](https://docs.saltstack.com/en/latest/ref/configuration/minion.html#std:conf_minion-id) configuration setting.

Tip

minion id and minion keys

The [minion id](https://docs.saltstack.com/en/latest/glossary.html#term-minion-id) is used to generate the minion's public/private keys and if it ever changes the master must then accept the new key as though the minion was a new host.

### 3.1.1-Globbing（通配符）

The default matching that Salt utilizes is [shell-style globbing](http://docs.python.org/2/library/fnmatch.html#module-fnmatch) around the [minion id](https://docs.saltstack.com/en/latest/glossary.html#term-minion-id). This also works for states in the [top file](https://docs.saltstack.com/en/latest/glossary.html#term-top-file).

Note

You must wrap **salt** calls that use globbing in single-quotes to prevent the shell from expanding the globs before Salt is invoked.

**Match all minions:**

salt '\*' test.ping

**Match all minions in the example.net domain or any of the example domains:**

salt '\*.example.net' test.ping

salt '\*.example.\*' test.ping

**Match all the webN minions in the example.net domain (web1.example.net, web2.example.net … webN.example.net):**

salt 'web?.example.net' test.ping

**Match the web1 through web5 minions:**

salt 'web[1-5]' test.ping

**Match the web1 and web3 minions:**

salt 'web[1,3]' test.ping

**Match the web-x, web-y, and web-z minions:**

salt 'web-[x-z]' test.ping

**Note**

For additional targeting methods please review the [compound matchers](https://docs.saltstack.com/en/latest/topics/targeting/compound.html) documentation.

### 3.1.2-Regular Expressions（正则表达式）

Minions can be matched using Perl-compatible [regular expressions](http://docs.python.org/2/library/re.html#module-re) (which is globbing on steroids and a ton of caffeine).

**Match both web1-prod and web1-devel minions:**

salt **-E** 'web1-(prod|devel)' test.ping

**When using regular expressions in a State's** [**top file**](https://docs.saltstack.com/en/latest/glossary.html#term-top-file)**, you must specify the matcher as the first option. The following example executes the contents of webserver.sls on the above-mentioned minions.**

base:

'web1-(prod|devel)':

- match: pcre

- webserver

### 3.1.3-Lists（列表）

**At the most basic level, you can specify a flat list of minion IDs:**

salt -L 'web1,web2,web3' test.ping

### 3.1.4-Grains

Salt comes with an interface to derive information about the underlying system. This is called the grains interface, because it presents salt with grains of information. Grains are collected for the operating system, domain name, IP address, kernel, OS type, memory, and many other system properties.

The grains interface is made available to Salt modules and components so that the right salt minion commands are automatically available on the right systems.

Grain data is relatively static, though if system information changes (for example, if network settings are changed), or if a new value is assigned to a custom grain, grain data is refreshed.

Note

Grains resolve to lowercase letters. For example, FOO, and foo target the same grain.

Important

See [Is Targeting using Grain Data Secure?](https://docs.saltstack.com/en/latest/faq.html#faq-grain-security) for important security information.

**Match all CentOS minions:**

salt -G 'os:CentOS' test.ping

**Match all minions with 64-bit CPUs, and return number of CPU cores for each matching minion:**

salt -G 'cpuarch:x86\_64' grains.item num\_cpus

**Additionally, globs can be used in grain matches, and grains that are nested in a** [**dictionary**](http://docs.python.org/2/library/stdtypes.html#typesmapping) **can be matched by adding a colon for each level that is traversed. For example, the following will match hosts that have a grain called ec2\_tags, which itself is a** [**dict**](http://docs.python.org/2/library/stdtypes.html#typesmapping) **with a key named environment, which has a value that contains the word production:**

salt -G 'ec2\_tags:environment:\*production\*'

**Listing Grains↓**

**Available grains can be listed by using the 'grains.ls' module:**

salt '\*' grains.ls

**Grains data can be listed by using the 'grains.items' module:**

salt '\*' grains.items

**Grains in the Minion Config**

Grains can also be statically assigned within the minion configuration file. Just add the option grains and pass options to it:

grains:

roles:

- webserver

- memcache

deployment: datacenter4

cabinet: 13

cab\_u: 14-15

Then status data specific to your servers can be retrieved via Salt, or used inside of the State system for matching. It also makes targeting, in the case of the example above, simply based on specific data about your deployment.

Grains in /etc/salt/grains

If you do not want to place your custom static grains in the minion config file, you can also put them in /etc/salt/grains on the minion. They are configured in the same way as in the above example, only without a top-level grains: key:

roles:

- webserver

- memcache

deployment: datacenter4

cabinet: 13

cab\_u: 14-15

**Matching Grains in the Top File**

With correctly configured grains on the Minion, the [top file](https://docs.saltstack.com/en/latest/glossary.html#term-top-file) used in Pillar or during Highstate can be made very efficient. For example, consider the following configuration:

'node\_type:web':

- match: grain

- webserver

'node\_type:postgres':

- match: grain

- database

'node\_type:redis':

- match: grain

- redis

'node\_type:lb':

- match: grain

- lb

For this example to work, you would need to have defined the grain node\_type for the minions you wish to match. This simple example is nice, but too much of the code is similar. To go one step further, **Jinja templating** can be used to simplify the [top file](https://docs.saltstack.com/en/latest/glossary.html#term-top-file).

{% set the\_node\_type = salt['grains.get']('node\_type', '') %}

{% if the\_node\_type %}

'node\_type:{{ the\_node\_type }}':

- match: grain

- {{ the\_node\_type }}

{% endif %}

Using Jinja templating, only one match entry needs to be defined.

Note

The example above uses the [grains.get](https://docs.saltstack.com/en/latest/ref/modules/all/salt.modules.grains.html#salt.modules.grains.get) function to account for minions which do not have the node\_type grain set.

Writing Grains

The grains interface is derived by executing all of the "public" functions found in the modules located in the grains package or the custom grains directory. The functions in the modules of the grains must return a Python [dict](http://docs.python.org/2/library/stdtypes.html#typesmapping), where the keys in the [dict](http://docs.python.org/2/library/stdtypes.html#typesmapping) are the names of the grains and the values are the values.

Custom grains should be placed in a \_grains directory located under the [file\_roots](https://docs.saltstack.com/en/latest/ref/configuration/master.html#std:conf_master-file_roots) specified by the master config file. The default path would be /srv/salt/\_grains. Custom grains will be distributed to the minions when [state.highstate](https://docs.saltstack.com/en/latest/ref/modules/all/salt.modules.state.html#salt.modules.state.highstate) is run, or by executing the [saltutil.sync\_grains](https://docs.saltstack.com/en/latest/ref/modules/all/salt.modules.saltutil.html#salt.modules.saltutil.sync_grains) or [saltutil.sync\_all](https://docs.saltstack.com/en/latest/ref/modules/all/salt.modules.saltutil.html#salt.modules.saltutil.sync_all) functions.

Grains are easy to write, and only need to return a dictionary. A common approach would be code something similar to the following:

#!/usr/bin/env python

def yourfunction():

# initialize a grains dictionary

grains = {}

# Some code for logic that sets grains like

grains['yourcustomgrain'] = True

grains['anothergrain'] = 'somevalue'

return grains

Before adding a grain to Salt, consider what the grain is and remember that grains need to be static data. If the data is something that is likely to change, consider using [Pillar](https://docs.saltstack.com/en/latest/topics/pillar/index.html) instead.

Warning

Custom grains will not be available in the top file until after the first [highstate](https://docs.saltstack.com/en/latest/topics/tutorials/states_pt1.html#running-highstate). To make custom grains available on a minion's first highstate, it is recommended to use [this example](https://docs.saltstack.com/en/latest/topics/reactor/index.html#minion-start-reactor) to ensure that the custom grains are synced when the minion starts.

Loading Custom Grains

If you have multiple functions specifying grains that are called from a main function, be sure to prepend grain function names with an underscore. This prevents Salt from including the loaded grains from the grain functions in the final grain data structure. For example, consider this custom grain file:

#!/usr/bin/env python

def \_my\_custom\_grain():

my\_grain = {'foo': 'bar', 'hello': 'world'}

return my\_grain

def main():

# initialize a grains dictionary

grains = {}

grains['my\_grains'] = \_my\_custom\_grain()

return grains

The output of this example renders like so:

# salt-call --local grains.items

local:

----------

<Snipped for brevity>

my\_grains:

----------

foo:

bar

hello:

world

However, if you don't prepend the my\_custom\_grain function with an underscore, the function will be rendered twice by Salt in the items output: once for the my\_custom\_grain call itself, and again when it is called in the main function:

# salt-call --local grains.items

local:

----------

<Snipped for brevity>

foo:

bar

<Snipped for brevity>

hello:

world

<Snipped for brevity>

my\_grains:

----------

foo:

bar

hello:

world

Precedence

Core grains can be overridden by custom grains. As there are several ways of defining custom grains, there is an order of precedence which should be kept in mind when defining them. The order of evaluation is as follows:

Core grains.

Custom grains in /etc/salt/grains.

Custom grains in /etc/salt/minion.

Custom grain modules in \_grains directory, synced to minions.

Each successive evaluation overrides the previous ones, so any grains defined by custom grains modules synced to minions that have the same name as a core grain will override that core grain. Similarly, grains from /etc/salt/minion override both core grains and custom grain modules, and grains in \_grains will override any grains of the same name.

Examples of Grains

The core module in the grains package is where the main grains are loaded by the Salt minion and provides the principal example of how to write grains:

<https://github.com/saltstack/salt/blob/develop/salt/grains/core.py>

Syncing Grains

Syncing grains can be done a number of ways, they are automatically synced when [state.highstate](https://docs.saltstack.com/en/latest/ref/modules/all/salt.modules.state.html#salt.modules.state.highstate) is called, or (as noted above) the grains can be manually synced and reloaded by calling the [saltutil.sync\_grains](https://docs.saltstack.com/en/latest/ref/modules/all/salt.modules.saltutil.html#salt.modules.saltutil.sync_grains) or [saltutil.sync\_all](https://docs.saltstack.com/en/latest/ref/modules/all/salt.modules.saltutil.html#salt.modules.saltutil.sync_all) functions.

### 3.1.5-Pillar

**Pillar data can be used when targeting minions. This allows for ultimate control and flexibility when targeting minions.**

salt -I 'somekey:specialvalue' test.ping

**Like with** [**Grains**](https://docs.saltstack.com/en/latest/topics/targeting/grains.html#targeting-grains)**, it is possible to use globbing as well as match nested values in Pillar, by adding colons for each level that is being traversed. The below example would match minions with a pillar named foo, which is a dict containing a key bar, with a value beginning with baz:**

salt -I 'foo:bar:baz\*' test.ping

### 3.1.6-Subnet/IP Address Matching

**Minions can easily be matched based on IP address, or by subnet (using** [**CIDR**](http://en.wikipedia.org/wiki/Classless_Inter-Domain_Routing) **notation).**

salt -S 192.168.40.20 test.ping

salt -S 10.0.0.0/24 test.ping

**Ipcidr matching can also be used in compound matches**

salt -C 'S@10.0.0.0/24 and G@os:Debian' test.ping

**It is also possible to use in both pillar and state-matching**

'172.16.0.0/12':

- match: ipcidr

- internal

Note

Only **IPv4** matching is supported at this time.

### 3.1.7-Compound matchers

Compound matchers allow very granular minion targeting using any of Salt's matchers. The default matcher is a [glob](http://docs.python.org/2/library/fnmatch.html#module-fnmatch) match, just as with CLI and [top file](https://docs.saltstack.com/en/latest/glossary.html#term-top-file) matching. To match using anything other than a glob, prefix the match string with the appropriate letter from the table below, followed by an @ sign.

| **Letter** | **Match Type** | **Example** | [**Alt Delimiter?**](https://docs.saltstack.com/en/latest/topics/targeting/compound.html#target-alt-delimiters) |
| --- | --- | --- | --- |
| 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 |

Matchers can be joined using boolean and, or, and not operators.

**For example, the following string matches all Debian minions with a hostname that begins with webserv, as well as any minions that have a hostname which matches the** [**regular expression**](http://docs.python.org/2/library/re.html#module-re) **web-dc1-srv.\*:**

salt -C 'webserv\* and G@os:Debian or E@web-dc1-srv.\*' test.ping

**That same example expressed in a** [**top file**](https://docs.saltstack.com/en/latest/glossary.html#term-top-file) **looks like the following:**

base:

'webserv\* and G@os:Debian or E@web-dc1-srv.\*':

- match: compound

- webserver

New in version 2015.8.0.

**Excluding a minion based on its ID is also possible:**

salt -C 'not web-dc1-srv' test.ping

**Versions prior to 2015.8.0 a leading not was not supported in compound matches. Instead, something like the following was required:**

salt -C '\* and not G@kernel:Darwin' test.ping

**Excluding a minion based on its ID was also possible:**

salt -C '\* and not web-dc1-srv' test.ping

**Precedence Matching**

**Matchers can be grouped together with parentheses to explicitly declare precedence amongst groups.**

salt -C '( ms-1 or G@id:ms-3 ) and G@id:ms-3' test.ping

**Note**

Be certain to note that spaces are required between the parentheses and targets. Failing to obey this rule may result in incorrect targeting!

Alternate Delimiters

New in version 2015.8.0.

Matchers that target based on a key value pair use a colon (:) as a delimiter. Matchers with a Yes in the Alt Delimiters column in the previous table support specifying an alternate delimiter character.

This is done by specifying an alternate delimiter character between the leading matcher character and the @ pattern separator character. This avoids incorrect interpretation of the pattern in the case that : is part of the grain or pillar data structure traversal.

salt -C 'J|@foo|bar|^foo:bar$ or J!@gitrepo!https://github.com:example/project.git' test.ping

### 3.1.8-Node groups

Nodegroups are declared using a compound target specification. The compound target documentation can be found [here](https://docs.saltstack.com/en/latest/topics/targeting/compound.html).

**The** [**nodegroups**](https://docs.saltstack.com/en/latest/ref/configuration/master.html#std:conf_master-nodegroups) **master config file parameter is used to define nodegroups. Here's an example nodegroup configuration within /etc/salt/master:**

nodegroups:

group1: 'L@foo.domain.com,bar.domain.com,baz.domain.com or bl\*.domain.com'

group2: 'G@os:Debian and foo.domain.com'

group3: 'G@os:Debian and N@group1'

group4:

- 'G@foo:bar'

- 'or'

- 'G@foo:baz'

Note

The L within group1 is matching a list of minions, while the G in group2 is matching specific grains. See the [compound matchers](https://docs.saltstack.com/en/latest/topics/targeting/compound.html) documentation for more details.

New in version 2015.8.0.

Note

Nodgroups can reference other nodegroups as seen in group3. Ensure that you do not have circular references. Circular references will be detected and cause partial expansion with a logged error message.

New in version 2015.8.0.

Compound nodegroups can be either string values or lists of string values. When the nodegroup is A string value will be tokenized by splitting on whitespace. This may be a problem if whitespace is necessary as part of a pattern. When a nodegroup is a list of strings then tokenization will happen for each list element as a whole.

**To match a nodegroup on the CLI, use the -N command-line option:**

salt **-N** group1 test.ping

**To match a nodegroup in your** [**top file**](https://docs.saltstack.com/en/latest/glossary.html#term-top-file)**, make sure to put - match: nodegroup on the line directly following the nodegroup name.**

base:

group1:

- match: nodegroup

- webserver

Note

When adding or modifying nodegroups to a master configuration file, the master must be restarted for those changes to be fully recognized.

A limited amount of functionality, such as targeting with -N from the command-line may be available without a restart.

Using Nodegroups in SLS files

To use Nodegroups in Jinja logic for SLS files, the pillar\_opts option in /etc/salt/master must be set to "True". This will pass the master's configuration as Pillar data to each minion.

Note

If the master's configuration contains any sensitive data, this will be passed to each minion. Do not enable this option if you have any configuration data that you do not want to get on your minions.

Also, if you make changes to your nodegroups, you might need to run salt '\*' saltutil.refresh\_pillar after restarting the master.

Once pillar\_opts is enabled, you can find the nodegroups under the "master" pillar. To make sure that only the correct minions are targeted, you should use each matcher for the nodegroup definition. For example, to check if a minion is in the 'webserver' nodegroup:

nodegroups:

webserver: 'G@os:Debian and L@minion1,minion2'

{% if grains.id in salt['pillar.get']('master:nodegroups:webserver', [])

and grains.os in salt['pillar.get']('master:nodegroups:webserver', []) %}

...

{% endif %}

Note

If you do not include all of the matchers used to define a nodegroup, Salt might incorrectly target minions that meet some of the nodegroup requirements, but not all of them.

## 3.2常用模块

### 3.2.1-salt.modules.service

The default service module, if not otherwise specified salt will fall back to this basic module

**salt.modules.service.available(name)**

**Returns True if the specified service is available, otherwise returns False.**

**CLI Example:**

salt '\*' service.available sshd

**salt.modules.service.get\_all()**

**Return a list of all available services**

**CLI Example:**

salt '\*' service.get\_all

**salt.modules.service.missing(name)**

**The inverse of service.available. Returns True if the specified service is not available, otherwise returns False.**

**CLI Example:**

salt '\*' service.missing sshd

**salt.modules.service.reload(name)**

**Refreshes config files by calling service reload. Does not perform a full restart.**

**CLI Example:**

salt '\*' service.reload <service name>

**salt.modules.service.restart(name)**

**Restart the specified service**

**CLI Example:**

salt '\*' service.restart <service name>

**salt.modules.service.start(name)**

**Start the specified service**

**CLI Example:**

salt '\*' service.start <service name>

**salt.modules.service.status(name, sig=None)**

**Return the status for a service, returns the PID or an empty string if the service is running or not, pass a signature to use to find the service via ps**

**CLI Example:**

salt '\*' service.status <service name> [service signature]

**salt.modules.service.stop(name)**

**Stop the specified service**

**CLI Example:**

salt '\*' service.stop <service name>

### 3.2.2-salt.modules.network

Module for gathering and managing network information

**salt.modules.network.active\_tcp()**

**Return a dict containing information on all of the running TCP connections**

**CLI Example:**

salt '\*' network.active\_tcp

**salt.modules.network.arp()**

**Return the arp table from the minion**

**Changed in version 2015.8.0: Added support for SunOS**

**CLI Example:**

salt '\*' network.arp

salt.modules.network.calc\_net(ip\_addr, netmask=None)

Returns the CIDR of a subnet based on an IP address (CIDR notation supported) and optional netmask.

CLI Example:

salt '\*' network.calc\_net 172.17.0.5 255.255.255.240

salt '\*' network.calc\_net 2a02:f6e:a000:80:84d8:8332:7866:4e07/64

New in version 2015.8.0.

salt.modules.network.connect(host, port=None, \*\*kwargs)

Test connectivity to a host using a particular port from the minion.

New in version 2014.7.0.

CLI Example:

salt '\*' network.connect archlinux.org 80

salt '\*' network.connect archlinux.org 80 timeout=3

salt '\*' network.connect archlinux.org 80 timeout=3 family=ipv4

salt '\*' network.connect google-public-dns-a.google.com port=53 proto=udp timeout=3

salt.modules.network.default\_route(family=None)

Return default route(s) from routing table

Changed in version 2015.8.0: Added support for SunOS (Solaris 10, Illumos, SmartOS)

CLI Example:

salt '\*' network.default\_route

salt.modules.network.dig(host)

Performs a DNS lookup with dig

CLI Example:

salt '\*' network.dig archlinux.org

salt.modules.network.get\_bufsize(iface)

Return network buffer sizes as a dict

CLI Example:

salt '\*' network.getbufsize

salt.modules.network.get\_hostname()

Get hostname

CLI Example:

salt '\*' network.get\_hostname

salt.modules.network.get\_route(ip)

Return routing information for given destination ip

New in version 2015.5.3.

Changed in version 2015.8.0: Added support for SunOS (Solaris 10, Illumos, SmartOS) Added support for OpenBSD

CLI Example:

salt '\*' network.get\_route 10.10.10.10

salt.modules.network.hw\_addr(iface)

Return the hardware address (a.k.a. MAC address) for a given interface

CLI Example:

salt '\*' network.hw\_addr eth0

salt.modules.network.hwaddr(iface)

This function is an alias of hw\_addr.

Return the hardware address (a.k.a. MAC address) for a given interface

CLI Example:

salt '\*' network.hw\_addr eth0

salt.modules.network.in\_subnet(cidr)

Returns True if host is within specified subnet, otherwise False.

CLI Example:

salt '\*' network.in\_subnet 10.0.0.0/16

salt.modules.network.interface(iface)

Return the inet address for a given interface

New in version 2014.7.0.

CLI Example:

salt '\*' network.interface eth0

salt.modules.network.interface\_ip(iface)

Return the inet address for a given interface

New in version 2014.7.0.

CLI Example:

salt '\*' network.interface\_ip eth0

salt.modules.network.interfaces()

Return a dictionary of information about all the interfaces on the minion

CLI Example:

salt '\*' network.interfaces

salt.modules.network.ip\_addrs(interface=None, include\_loopback=False, cidr=None)

Returns a list of IPv4 addresses assigned to the host. 127.0.0.1 is ignored, unless 'include\_loopback=True' is indicated. If 'interface' is provided, then only IP addresses from that interface will be returned. Providing a CIDR via 'cidr="10.0.0.0/8"' will return only the addresses which are within that subnet.

CLI Example:

salt '\*' network.ip\_addrs

salt.modules.network.ip\_addrs6(interface=None, include\_loopback=False, cidr=None)

Returns a list of IPv6 addresses assigned to the host. ::1 is ignored, unless 'include\_loopback=True' is indicated. If 'interface' is provided, then only IP addresses from that interface will be returned. Providing a CIDR via 'cidr="2000::/3"' will return only the addresses which are within that subnet.

CLI Example:

salt '\*' network.ip\_addrs6

salt.modules.network.ip\_in\_subnet(ip\_addr, cidr)

Returns True if given IP is within specified subnet, otherwise False.

CLI Example:

salt '\*' network.ip\_in\_subnet 172.17.0.4 172.16.0.0/12

salt.modules.network.ipaddrs(interface=None, include\_loopback=False, cidr=None)

This function is an alias of ip\_addrs.

Returns a list of IPv4 addresses assigned to the host. 127.0.0.1 is ignored, unless 'include\_loopback=True' is indicated. If 'interface' is provided, then only IP addresses from that interface will be returned. Providing a CIDR via 'cidr="10.0.0.0/8"' will return only the addresses which are within that subnet.

CLI Example:

salt '\*' network.ip\_addrs

salt.modules.network.ipaddrs6(interface=None, include\_loopback=False, cidr=None)

This function is an alias of ip\_addrs6.

Returns a list of IPv6 addresses assigned to the host. ::1 is ignored, unless 'include\_loopback=True' is indicated. If 'interface' is provided, then only IP addresses from that interface will be returned. Providing a CIDR via 'cidr="2000::/3"' will return only the addresses which are within that subnet.

CLI Example:

salt '\*' network.ip\_addrs6

salt.modules.network.is\_loopback(ip\_addr)

Check if the given IP address is a loopback address

New in version 2014.7.0.

Changed in version 2015.8.0: IPv6 support

CLI Example:

salt '\*' network.is\_loopback 127.0.0.1

salt.modules.network.is\_private(ip\_addr)

Check if the given IP address is a private address

New in version 2014.7.0.

Changed in version 2015.8.0: IPv6 support

CLI Example:

salt '\*' network.is\_private 10.0.0.3

salt.modules.network.mod\_bufsize(iface, \*args, \*\*kwargs)

Modify network interface buffers (currently linux only)

CLI Example:

salt '\*' network.getBuffers

salt.modules.network.mod\_hostname(hostname)

Modify hostname

Changed in version 2015.8.0: Added support for SunOS (Solaris 10, Illumos, SmartOS)

CLI Example:

salt '\*' network.mod\_hostname master.saltstack.com

salt.modules.network.netstat()

Return information on open ports and states

Note

On BSD minions, the output contains PID info (where available) for each netstat entry, fetched from sockstat/fstat output.

Changed in version 2014.1.4: Added support for OpenBSD, FreeBSD, and NetBSD

Changed in version 2015.8.0: Added support for SunOS

CLI Example:

salt '\*' network.netstat

salt.modules.network.ping(host, timeout=False, return\_boolean=False)

Performs an ICMP ping to a host

Changed in version 2015.8.0: Added support for SunOS

CLI Example:

salt '\*' network.ping archlinux.org

New in version 2015.5.0.

Return a True or False instead of ping output.

salt '\*' network.ping archlinux.org return\_boolean=True

Set the time to wait for a response in seconds.

salt '\*' network.ping archlinux.org timeout=3

salt.modules.network.reverse\_ip(ip\_addr)

Returns the reversed IP address

Changed in version 2015.8.0: IPv6 support

CLI Example:

salt '\*' network.reverse\_ip 172.17.0.4

salt.modules.network.routes(family=None)

Return currently configured routes from routing table

Changed in version 2015.8.0: Added support for SunOS (Solaris 10, Illumos, SmartOS)

CLI Example:

salt '\*' network.routes

salt.modules.network.subnets(interfaces=None)

Returns a list of IPv4 subnets to which the host belongs

CLI Example:

salt '\*' network.subnets

salt '\*' network.subnets interfaces=eth1

salt.modules.network.subnets6()

Returns a list of IPv6 subnets to which the host belongs

CLI Example:

salt '\*' network.subnets

salt.modules.network.traceroute(host)

Performs a traceroute to a 3rd party host

Changed in version 2015.8.0: Added support for SunOS

CLI Example:

salt '\*' network.traceroute archlinux.org

salt.modules.network.wol(mac, bcast='255.255.255.255', destport=9)

Send Wake On Lan packet to a host

CLI Example:

salt '\*' network.wol 08-00-27-13-69-77

salt '\*' network.wol 080027136977 255.255.255.255 7

salt '\*' network.wol 08:00:27:13:69:77 255.255.255.255 7

## 3.3-配置client\_acl

The salt client ACL system is a means to allow system users other than root to have access to execute select salt commands on minions from the master.

The client ACL system is configured in the master configuration file via the client\_acl configuration option. Under the client\_acl configuration option the users open to send commands are specified and then a list of regular expressions which specify the minion functions which will be made available to specified user. This configuration is much like the peer configuration:

client\_acl:

# Allow thatch to execute anything.

thatch:

- .\*

# Allow fred to use test and pkg, but only on "web\*" minions.

fred:

- web\*:

- test.\*

- pkg.\*

**配置实战：**

[root@linux-node1 pillar]# vim /etc/salt/master 打开注释并修改用户名

client\_acl:

oldboy:

- test.ping

- network.\*

[root@linux-node1 pillar]# chmod 755 /var/cache/salt /var/cache/salt/master /var/cache/salt/master/jobs /var/run/salt /var/run/salt/master

[root@linux-node1 pillar]# chmod 777 /var/log/salt/master

[root@linux-node1 pillar]# /etc/init.d/salt-master restart

[root@linux-node1 pillar]# su - oldboy

[oldboy@linux-node1 ~]$ salt '\*' test.ping

linux-node2.example.com:

True

linux-node1.example.com:

True

[oldboy@linux-node1 ~]$ salt '\*' cmd.run 'w'

Failed to authenticate! This is most likely because this user is not permitted to execute commands, but there is a small possibility that a disk error occurred (check disk/inode usage).

## 3.4-返回（return）

先安装依赖包：MySQL-python

[root@linux-node1 ~]# yum install MySQL-python -y

方法一：

**Use the following mysql database schema（创建数据库）:**

CREATE DATABASE `salt`

DEFAULT CHARACTER SET utf8

DEFAULT COLLATE utf8\_general\_ci;

USE `salt`;

--

-- Table structure for table `jids`

--

DROP TABLE IF EXISTS `jids`;

CREATE TABLE `jids` (

`jid` varchar(255) NOT NULL,

`load` mediumtext NOT NULL,

UNIQUE KEY `jid` (`jid`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE INDEX jid ON jids(jid) USING BTREE;

--

-- Table structure for table `salt\_returns`

--

DROP TABLE IF EXISTS `salt\_returns`;

CREATE TABLE `salt\_returns` (

`fun` varchar(50) NOT NULL,

`jid` varchar(255) NOT NULL,

`return` mediumtext NOT NULL,

`id` varchar(255) NOT NULL,

`success` varchar(10) NOT NULL,

`full\_ret` mediumtext NOT NULL,

`alter\_time` TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

KEY `id` (`id`),

KEY `jid` (`jid`),

KEY `fun` (`fun`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

--

-- Table structure for table `salt\_events`

--

DROP TABLE IF EXISTS `salt\_events`;

CREATE TABLE `salt\_events` (

`id` BIGINT NOT NULL AUTO\_INCREMENT,

`tag` varchar(255) NOT NULL,

`data` mediumtext NOT NULL,

`alter\_time` TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

`master\_id` varchar(255) NOT NULL,

PRIMARY KEY (`id`),

KEY `tag` (`tag`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

mysql> grant all on salt.\* to salt@'%' identified by 'salt';

**salt-minion配置文件添加一下内容到最后：**

[root@linux-node2 ~]# vim /etc/salt/minion

mysql.host: '10.0.0.7'

mysql.user: 'salt'

mysql.pass: 'salt'

mysql.db: 'salt'

mysql.port: 3306

[root@linux-node2 ~]# /etc/init.d/salt-minion restart

salt-master执行命令：

[root@linux-node1 pillar]# salt '\*' test.ping --return mysql

mysql> select \* from salt.salt\_returns\G

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

fun: test.ping

jid: 20151123172044880830

return: true

id: linux-node2.example.com

success: 1

full\_ret: {"fun\_args": [], "jid": "20151123172044880830", "return": true, "retcode": 0, "success": true, "fun": "test.ping", "id": "linux-node2.example.com"}

alter\_time: 2015-11-23 17:20:45

1 row in set (0.00 sec)

**第二种方法：**

[root@linux-node1 pillar]# vim /etc/salt/master

return: mysql

mysql.host: '10.0.0.7'

mysql.user: 'salt'

mysql.pass: 'salt'

mysql.db: 'salt'

mysql.port: 3306

[root@linux-node1 pillar]# salt '\*' test.ping #无需加--return mysql

# 4-扩展Grains

1、第一步，我们需要在master上编写一个python脚本。这个脚本的主要内容就是去定义如何手机你想要的信息。在脚步的最后把采集到的各种信息返回

2、第二步，需要把这个脚本sysnc同步到所有Minion（或指定Minion）上。

按照之前我们讲的内容，我们需要重启Minion端，但是因为在sync同步Grains脚本的时候Minion就会去收集刷新Grains。所以此处无需重启Minion就可以收集到最新的Grains。

例子：

1、python脚本

[root@linux-node1 \_grains]# pwd

/srv/salt/base/\_grains

[root@linux-node1 \_grains]# cat example.py

#!/usr/bin/python

def grains():

local={}

test={'key':'value','key1':'value1','key2':'value2'}

local['list'] = [1,2,3,4]

local['string'] = 'str'

local['dict'] = test

return local

2、同步这个脚本到Minion上

[root@linux-node1 \_grains]# salt 'linux-node1\*' saltutil.sync\_grains

3、在master端查看python脚本定义的Grains信息

[root@linux-node1 \_grains]# salt '\*' grains.item list string dict

linux-node2.example.com:

----------

dict:

list:

string:

linux-node1.example.com:

----------

dict:

----------

key:

value

key1:

value1

key2:

value2

list:

- 1

- 2

- 3

- 4

string:

str