

目 录

[关于版块](#)

[安装部署](#)

[基础知识](#)

[salt state](#)

[更多实例](#)

[grain模块](#)

[pillar模块](#)

关于版块

关于saltstack

SaltStack是一个服务器基础架构集中化管理平台，具备配置管理、远程执行、监控等功能，基于Python语言实现，结合轻量级消息队列（ZeroMQ）与Python第三方模块（Pyzmq、PyCrypto、Pyjinja2、python-msgpack和PyYAML等）构建。

通过部署SaltStack，我们可以在成千万台服务器上做到批量执行命令，根据不同业务进行配置集中化管理、分发文件、采集服务器数据、操作系统基础及软件包管理等，SaltStack是运维人员提高工作效率、规范业务配置与操作的利器。

本文阐述了saltstack的基本知识点及常用用法，更多用法可以参考官网：

<https://docs.saltstack.com/en/latest/ref/modules/all/>

本版块维护人员

版主：子木

QQ：1242119478

交流Q群：526749756

安装部署

基本环境

服务器:

test11: 192.168.37.11

test22: 192.168.37.12

test33: 192.168.37.13

系统版本: CentOS Linux release 7.6.1810 (Core)

测试方便关闭防火墙: `systemctl stop firewalld.service`

安装

安装参考文档: <https://repo.saltstack.com>

添加yum源: `yum install https://repo.saltstack.com/yum/redhat/salt-repo-latest.el7.noarch.rpm`

服务器 test11:

安装master端: `yum install salt-master`

服务器 test22, test33:

安装minion端: `yum install salt-minion`

master添加自定义配置文件,开启file和pillar模块:

```
cat > /etc/salt/master.d/file_dir.conf << EOF
pillar_roots:
  base:
    - /srv/salt/pillar

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

创建目录: `mkdir /srv/salt/{state,pillar} -p`

minion端添加自定义配置文件, 设置salt_master及id:

```
cat > /etc/salt/minion.d/test.conf << EOF
# Set the location of the salt master server. If the master server cannot be
# resolved, then the minion will fail to start.
```

```

master: 192.168.37.11          #指定master端，有条件的建议指定的域名。

# Explicitly declare the id for this minion to use, if left commented the id
# will be the hostname as returned by the python call: socket.getfqdn()
# Since salt uses detached ids it is possible to run multiple minions on the
# same machine but with different ids, this can be useful for salt compute
# clusters.
id: `hostname`
EOF

```

master端启动及添加到开机启动:

```
systemctl start salt-master
```

```
systemctl enable salt-master
```

minion端启动及添加到开机启动:

```
systemctl start salt-minion
```

```
systemctl enable salt-minion
```

查看版本: `salt --version`

salt 2019.2.0 (Fluorine)

在master端查看、添加、删除minion端

查看minion端:

```
salt-key -L
```

添加单个minion端:

```
salt-key -a test22 -y #test22替换成你相应的客户端主机名就可以
```

添加所有minion端:

```
salt-key -A -y
```

删除单个minion端:

```
salt-key -d test22 -y #test22替换成你相应的客户端主机名就可以
```

删除所有minion端:

```
salt-key -D -y
```

salt-master迁移

1、把原salt-master上的pki文件夹打包到新的salt-master机器上:

```
scp -rp /etc/salt/pki root@192.168.139.44:/etc/salt
```

2、更改salt的client的hosts文件，指向新的salt-master

3、重启client的salt-minion

4、到44的机器上测试一下有没有迁移成功了

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

minion端更改主机名

- 1、master上踢除要更改主机名的client，
- 2、在client上更改主机名，删除minion_id文件和pki目录，重启minion服务

基础知识

管理Job

查看所有minion端的状态:

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

查看没在线的minion端:

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

查看在线的minion端:

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

查看所有minion端当前正在运行的jobs:

```
salt-run jobs.active
```

在master端查看正在运行的jobs:

```
salt '*' saltutil.running
```

关闭所有minion端正在运行的job:

```
salt '*' saltutil.kill_job <jid>
```

关闭单个minion(如test22)的job:

```
salt 'test22' saltutil.kill_job <jid>
```

Saltstack匹配Minion ID的多种方法

target也就是目标,目的.指定**master**命令应该对谁执行

1、Globbing通配符

表示target匹配所有

```
salt '*' saltutil.running
```

? 表示任意一个字符

```
salt 'test?2' test.ping
```

2、regularexpressions正则表达式

同时匹配test22,test33

```
salt -E 'test(22|33)' test.ping
```

3、List列表

```
salt -L 'test22,test33' test.ping
```

4、grains匹配

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

通过**cmd.run**模块发送命令到**minion**端执行

例:

```
salt '*' cmd.run 'uptime'
```

Saltstack指定脚本到minion端执行

写一个简单的测试脚本,测试完后记得用job管理kill掉:

```
while true
do
echo 1 >>/tmp/1.txt

sleep 1

done
```

把脚本放到test22上执行:

```
salt 'test22' cmd.script salt://script/echo.sh
```

用tail -f 到test22下查看/tmp/1.txt一直在打印

```
[root@test22 ~]# tail -f /tmp/1.txt
1
1
1
1
```

在minion端执行: salt-call

该命令通常在minion上执行, minion自己执行可执行模块, 不是通过master下发job

用法:

```
salt-call [options] [arguments]
```

例:

```
salt-call -l debug state.sls hosts.a
```

salt state

关于salt state

核心是写sls(SaLt State file)文件,sls文件默认格式是YAML格式(以后会支持XML),并默认使用jinja模板,YAML与XML类似,是一种简单的适合用来传输数据的格式,而jinja是根据django的模板语言发展而来的语言,简单并强大,支持for if 等循环判断。salt state主要用来描述系统,软性,服务,配置文件应该处于的状态,常常被称为配置管理!

通常state,pillar会用sls文件来编写。state文件默认是放在/srv/salt中,实际工作中为了更便捷管理,前面我们设置的目录为/srv/salt/state,后期再根据不同的项目或用途,再在state下新建子目录管理。

YAML语法

1、空格和TAB

yaml两个空格为缩进,不能用tab

2、冒号 : 和减号 -

: 和- 之间要有一个空格

3、数字解析

mode: 0644 会解析成为mode: 644 最好使用mode: (0644)

JinJa模板

Jinja 基于Python模板引擎开发,saltstack默认使用yaml_jinja渲染器,渲染流程时先jinja在yaml解析.所以在开始解析yaml的时候可以使用jinja“偷个腥”,用法如下:

1、jinja中使用grains `{{ grains['os'] }}`

2、jinja中使用Pillar `{{ pillar['apache']['PORT'] }}`

3、jinja的逻辑关系

```
{% if grains['os'] == 'RedHat' %}
apache: httpd
{% elif grains['os'] == 'Debian' %}
apache: apache2
{% endif %}
```

sls文件执行的两种方式

建议: srv/salt/state下新建各用途的目录,再在目录中再新建sls文件(默认为init.sls),不要把所有的sls文件都放在state下,显的很乱且不好管理!

NO1:

在/srv/salt/state下新建top.sls, top.sls中指定要执行的sls文件, state.highstate推送匹配的所有sls文件例:

```
[root@test11 state]# pwd
```



```

/srv/salt/state
[root@test11 state]# salt '*' state.highstate #执行top.sls下匹配的所有sls
[root@test11 state]# vim top.sls
base:
  '*': #要推送到的客户端，*指所有mini
on端，也可以指定单个
  - hosts #/srv/salt/state/hosts下的
init.sls
[root@Master salt]# vim init.sls
/tmp/hosts: #推送到的目的地的文件名
  file.managed:
    - source: salt://hosts/templates/hosts #推送的文件存放位置，在/srv/s
alt/etc下
    - user: root
    - group: root
    - mode: 600

```

NO2:

指定推送单个sls文件: `salt '*' state.sls [sls文件]`

例:

按照上述例子推送hosts文件: `salt '*' state.sls hosts`

注: 大家可能注意到上面命令state.sls hosts这个目录就可以了, 原因是salt用的是python写的, 在py下init为默认的初始配置, 所以可以略写, 但如果要执行的是hosts目录下的其它sls文件, 假如是a.sls文件, 可参考如下执行:

```
salt '*' state.sls hosts.a
```

状态之间的依赖关系

1、我依赖谁	#require	
2、谁依赖我[我被谁依赖]	#require_in	
3、我监控谁 启服务	#watch	如果配置文件有修改, 那么会重载、重
4、谁监控我[我被谁监控] 服务	#watch_in	如果配置文件有修改, 那么会重载、重启
5、我引用谁	#include	
6、我扩展谁	#extend	

salt schedule

schedule是salt中的crontab, 结合pillar使用, 就是周期性执行一些函数, 以下例说明:

```

[root@Master pillar]# cat hosts/init.sls
schedule:
  hosts: ##这个是ID, 可以随意起, 全文件唯一
    function: state.sls ##对于master, function就是runner

```

salt state

```
seconds: 30          ##间隔秒数
minutes: 15          ##间隔分数
minutes: 2           ##间隔小时数
args:
  - 'hosts'
```

注：上面sls文件的意思是每隔2小时15分30秒执行hosts的state.sls文件。

更多实例

实例

1、关闭selinux服务

```
[root@test11 state]# cat service/seliunx.sls
seliunx:                                     ##ID
  file.replace:                             ##Function, 替换文件内容的方法
    - name: /etc/selinux/config              ##在这里是要替换内容的文件
    - pattern: SELINUX=enforcing             ##要替换的内容
    - repl: SELINUX=disabled                 ##替换后的内容
    - count: 1                               ##默认为0表示所有的都将进行替换, 这里加了整数的话就表示替换的次数不超过这个整数

setenforce:
  cmd.run:
    - name: setenforce 0                     ##这里就是cmd.run后面跟着的要执行的操作
[root@test11 state]# salt '*' state.sls service.seliunx
```

2、测试脚本的sls文件

```
[root@test11 state]# cat pyscript/init.sls
pyscript:
  cmd.script:
    - source: salt://pyscript/test.sh
    - user: root
```

3、安装nginx

```
[root@test11 state]# cat nginx/init.sls
include:                                     #引用的state
  - repo
  - user.www
  - nginx.package
  - nginx.config
  - nginx.service

extend:                                     #扩展那些属性
  nginx:                                    #所扩展的对象ID, 在sls文件中ID唯一
  pkg:
    - require:                              #依赖谁
```

```

    - file: ubox.repo
    - user: www
nginx.conf:
  file:
    - require:
      - pkg: nginx
nginx-service:
  service:
    - watch:                                #如果配置文件有修改，那么会重启nginx服务
      - pkg: nginx
      - file: nginx.conf
##下面为上面include引用的state.sls文件与extend所扩展的属性
[root@test11 state]# ls repo/files/
ubox-nexus.repo  ubox.repo
[root@test11 state]# ls user/
test.sls  www.sls
[root@test11 state]# cat user/www.sls
www:                                           ##定义对象ID
  user.present:
    - shell: /sbin/nologin
    - createhome: False
[root@test11 state]# cat nginx/package.sls
nginx:
  pkg.installed
[root@test11 state]# cat nginx/config.sls
nginx.conf:
  file.managed:
    - name: /usr/local/nginx/conf/nginx.conf
    - source: salt://nginx/files/nginx.conf
    - user: root
    - group: root
    - mode: 644
    - template: jinja
[root@test11 state]# cat nginx/service.sls
nginx-service:
  service.running:
    - name: nginx
    - enable: True
    - reload: True

```

grain模块

关于grains

Grains是SaltStack的一个组件，存放在SaltStack的minion端，当salt-minion启动时会把收集到的数据静态存放在Grains当中，只有当minion发生过重启时才会进行数据的更新

列出SaltStack默认支持的Grains

以服务器test22为例：

```
[root@test11 ~]# salt 'test22' grains.ls
test22:
  - SSDs
  - biosreleasedate
  - biosversion
  - cpu_flags
  - cpu_model
  - cpuarch
  - disks
  - dns
  - domain
  - fqdn
  .
  .
  .
```

查看所有项的值：

```
[root@test11 ~]# salt 'test22' grains.items
test22:
  -----
  SSDs:
  biosreleasedate:
    12/01/2006
  biosversion:
    VirtualBox
  cpu_flags:
    - fpu
    - vme
    .
    .
  cpu_model:
    Intel(R) Core(TM) i5-4570 CPU @ 3.20GHz
  .
```

.

查看单独项的值：

```
[root@test11 ~]# salt 'test22' grains.item cpu_model
test22:
-----
cpu_model:
    Intel(R) Core(TM) i5-4570 CPU @ 3.20GHz
```

匹配minion

```
[root@test11 ~]# salt -G 'os:CentOs' cmd.run 'uptime'
test33:
    14:52:08 up 1 day,  3:40,  1 user,  load average: 0.00, 0.01, 0.05
test22:
    14:52:08 up 1 day,  3:40,  1 user,  load average: 0.04, 0.04, 0.05
```

自定义grains的item

在服务器test22的minion.d目录下添加一个nginx标签文件，并重启minion端：

```
[root@test22 ~]# vi /etc/salt/minion.d/nginx.conf
grains:
  roles:
    - webserver
    - nginx
```

在master端测试自定义的item有没有生效：

```
[root@test11 ~]# salt -G 'roles:nginx' cmd.run 'uptime'
test22:
    15:01:21 up 1 day,  3:49,  1 user,  load average: 0.00, 0.01, 0.05
```

pillar模块

关于pillar

Pillar在salt中是非常重要的组成部分，利用它可以完成很强大的功能，它可以指定一些信息到指定的minion上，不像grains一样是分发到所有Minion上的，它保存的数据可以是动态的

用途：

- 1.比较敏感的数据，比如密码，key等
- 2.特殊数据到特定Minion上
- 3.动态的内容
- 4.其他数据类型

开启pillar，前面部署时已说明，指定目录是/srv/salt/pillar。

基本使用

查看：

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

```
salt '*' pillar.data
```

查看某个：

```
salt '*' pillar.item KEY
```

刷新pillar数据：

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

在state组件中使用pillar

注：Pillar数据解析后是以字典的形式存在的，因此，引用Pillar数据的过程，实际上就是从字典里面读取数据的过程。

例：

```
pillar['apache']
```

或

```
pillar.get('apache', {})
```

注意观察下面的pillar文件，结论：即使是两个不同的pillar sls文件，如果字典的主key一致，则不同sls文件的的数据都会将解析后放入主字典下

```
[root@test11 pillar]# ls
httpd.sls  sshd.sls  top.sls
[root@test11 pillar]# cat top.sls
base:
  '*':
    - sshd
    - httpd
```

```
[root@test11 pillar]# cat httpd.sls
service:
  http_name: httpd
[root@test11 pillar]# cat sshd.sls
service:
  ssh_service: sshd

[root@test11 pillar]# salt 'test22' pillar.data
test22:
  -----
  service:
    -----
    http_name:
      httpd
    ssh_service:
      sshd
```

实例理解pillar

以给minion端安装httpd服务为例，根据不同的minion端赋以不同的端口

- 1、在master上安装httpd,并把httpd的配置文件拷贝到新建的/srv/salt/state/apache/templates下
- 2、修改httpd.conf文件，把监听的端口改成jinja格式

Listen {{ port }}

- 3、sls文件如下：

```
[root@test11 pillar]# cat top.sls
base:
  '*':
    - apache

[root@test11 pillar]# cat apache/init.sls
apache:
  {% if grains['id'] == 'test22' %}
  port: 8081
  {% elif grains['id'] == 'test33' %}
  port: 8082
  {% else %}
  port: 80
  {% endif %}

[root@test11 pillar]# cat /srv/salt/state/apache/init.sls
apache:
  pkg.installed:
    - name: httpd
  file.managed:
    - name: /etc/httpd/conf/httpd.conf
    - source: salt://apache/templates/httpd.conf
    - require:                                     ##依赖系统
```



```

- pkg: apache                                ##表示依赖id
为apache的pkg状态
- template: jinja
- context:
  port: {{ salt['pillar.get']('apache:port',80) }}  ##表示在没有获
到值的情况下，赋默认值80
service.running:
- enable: True
- name: httpd
- watch:                                         #watch检测配
置文件，如果发生变化会重启服务
- pkg: apache
- file: apache

```

pillar与schedule配合使用，定时执行任务

例：每一分钟自动更新前面/srv/salt/state/hosts/templates/下的hosts文件

```

[root@test11 pillar]# cat top.sls
base:
  '*':
    - hosts
[root@test11 pillar]# cat hosts/init.sls
schedule:
  hosts:
    function: state.sls
    minutes: 1
    args:
      - 'hosts'                                #要执行的state.sls文件

```