saltstack自动化部署

1. 安装saltstack服务端和客户端：

```
http://repo.saltstack.com      #进入网站安装官方源
yum install salt-master salt-minion     #服务端
yum install salt-minion                 #客户端
```

2. 配置客户端的master：

vim /etc/salt/minion

master: 172.16.1.211

3. 在服务端建立客户端的认证（服务端执行）：

```
 salt-key -A   #认证所有       -a 认证单个
```

二. SaltStack基本操作

1.在所有客户端执行测试命令：

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

2.在所有minion(客户端)上执行‘uptime’命令：

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

3.拷贝文件到所有客户端的tmp目录下面：

4.saltstack的配置管理：

vim /etc/salt/master

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

mkdir /srv/salt

vim apache.sls #安装apache并启动

```
apache-install:
  pkg.installed:
    - names:
      - httpd
```

```
    - httpd-devel

apache-service:
  service.running:
    - name: httpd
    - enable: True
    - reload: True
```

salt ‘*’ state.sls apache #执行状态管理脚本

5.编辑salt的入口文件：

PS: top-file文件要放在base环境下。

vim top.sls

```
base:
  'SH_T_ansiblecli_02.gigold-idc.com':
    - apache
  'SH_T_ansiblecli_03.gigold-idc.com':
    - apache
```

salt ‘*’ state.highstate #通过入口文件执行安装apache的脚本

salt ‘*’ state.highstate test=True #生产环境上面命令很危险，要先测试下

三. SaltStack-数据系统Grains

1.查看客户机的所有grains信息：

```
salt 'SH_T_test_03.gigold-idc.com' grains.items
```

2.查看客户机的IP地址：

```
salt 'SH_T_test_03.gigold-idc.com' grains.get ip_interfaces:eth0
```

3.通过grains判断在哪些客户端上执行‘W’命令：

```
salt -G 'os:Centos' cmd.run 'w'
```

4.客户端自定义grains并通过top筛选:

vim /etc/salt/grains #在客户端定义，写完要重启minion

```
roles:
  - webserver
  - memcache
```

/etc/init.d/salt-minion restart

```
base:
  'roles:webserver':     #匹配roles变量为webserver的主机
    - match: grain      #要定义通过grain方法获取参数
    - apache
```

salt ‘*’ state.highstate #通过top筛选匹配的grain

5.通过master刷新minion的grains参数：

```
salt '*' saltutil.sync_grains
```

vim /srv/salt/_grains/my_grains.py #文件创建在master服务器上

salt ‘*’ saltutil.sync_grains #编辑完后要记得推送到minion上

7. Grains的优先级：

从高到低：系统自带--grains文件---minion配置文件--master自定义的

四. SaltStack-数据系统Pillar

定义： Pillar是动态的，给特定的minion指定特定的数据 ， 只有指定的minion可以看到（所有相对安全，可以用来设置密码）。

1.在服务端创建Pillar的base环境：

vim /etc/salt/master

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

mkdir -p /srv/pillar

/etc/init.d/salt-master restart

2.自己创建pillar：

vim /srv/pillar/apache.sls

```
my-pillar:
  {%if grains['os'] == 'CentOS'%}
  apache: httpd
  {% elif grains['os'] == 'Debian' %}
  apache: apache2
  {% endif %}
```

vim /srv/pillar/top.sls

```
#配置在哪台主机上面使用pillar
base:
  'SH_T_ansiblecli_02.gigold-idc.com':
    - apache
```

```
salt   '*'   saltutil.refresh_pillar #配置完成后要记得刷新pillar
salt   '*'   pillar.items #查看配置的pillar
```

3.刷新pillar：

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

```
salt -I "apache:httpd" cmd.run 'w'
```

5.grains和pillar的区别：

名称	存储位置	数据类型	数据采集更新方式	应用
Grains	Minion端	静态数据	Minion启动时收集，也可以使用saltutil.sync_grains进行刷新。	存储Minion基本数据。比如用于匹配Minion，自身数据可以用来做资产管理等。
Pillar	Master端	动态数据	在Master端定义，指定给对应的Minion。可以使用saltutil.refresh_pillar刷新	存储Master指定的数据，只有指定的Minion可以看到。用于敏感数据保存

五. SaltStack-远程执行-进阶

1.多种判断需要执行Minion的方法：

```
salt -L 'SH_T_test_03, SH_T_ansiblecli_02' cmd.run 'w' #通过列表
salt -S '172.16.1.213' test.ping      #通过ip判断
salt -S '172.16.1.0/24' test.ping     #通过网段判断
vim /etc/salt/master      #配置nodegroups来分组
nodegroups:
  web: 'L@SH_T_test_03.gigold-idc.com,SH_T_ansiblecli_02.gigold-idc.com'
salt -N web test.ping      #通过-N来指定
```

2.将执行返回值写入到数据库中：

执行数据库脚本：

```
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;
```

vim /etc/salt/master

```
#文件底部添加
master_job_cache: mysql
mysql.host: '172.16.1.214'
mysql.user: 'root'
mysql.pass: 'Root123'
mysql.db: 'salt'
mysql.port: 3306
```

/etc/init.d/salt-master restart

salt ‘*’ cmd.run ‘free -m’ #执行完成后可以在数据库里面查看

六. SaltStack-配置管理

1.”top.sls”文件必须放在base环境下。

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

2.拷贝文件到Minion上：

vim dns.sls

```
/etc/resolv.conf:
  file.managed:
    - source: salt://files/resolv.conf
```

```
    - user: root
    - group: root
    - mode: 644
    - template: jinja
    - defaults:
      DNS_SERVER: 223.6.6.6
```

salt ‘*’ state.sls dns #直接执行

下面用top来执行

vim top.sls

```
base:
  '*':
    - dns
```

salt ‘*’ state.highstate #使用高级状态执行

3.状态间的依赖关系：

1.我依赖谁：只要依赖的状态为正常，我就执行。

```
- require:
  - pkg: lamp-pkg          (状态模块：状态名)
  - file: apache-config
```

2.我被谁依赖：(一般用不到)

```
- require_in:    (写在被依赖方)
  - service: mysql-service
```

3.我监控谁：监控某个状态，只有发生变化我才执行，

```
- watch:
```

4.我被谁监控：(一般用不到)

```
- watch_in:
```

5.我引用谁：

```
include:
```

6.我扩展谁：

3.jinja模板的使用方法：

1.模板里面赋值：

```
    - template: jinja
    - defaults:
      PORT: 8080
{{ PORT }}
```

2.使用grains参数获取本地IP： {{ grains['fqdn_ip4'][0] }}

```
3.使用salt远程执行模块获取网卡MAC: {{ salt['network.hw_addr']('eth0') }}
4.使用pillar参数：  {{ pillar['apache'] }}
```

- 1
- 2
- 3
- 4
- 5
- 7
- 8

3.include用法：

```
include:
  - init.dns
  - init.history
  - init.sysctl
```

- 1
- 2
- 3
- 4

4.以测试模式执行：

```
salt '*' state.highstate test=True
```

- 1

七. SaltStack-实践案例

https://github.com/unixhot/saltbook-code #SaltStack实践案例源码

1.安装haproxy案例

```
include:
  - pkp.pkg-init

haproxy-install:
  file.managed:
    - name: /usr/local/src/haproxy-1.6.11.tar.gz
    - source: salt://haproxy/files/haproxy-1.6.11.tar.gz
    - user: root
    - group: root
```

```
    - mode: 755
  cmd.run:
    - name: cd /usr/local/src && tar zxf haproxy-1.6.11.tar.gz && cd haproxy-1.6.11
&& make TARGET=linux26 PREFIX=/usr/local/haproxy && make install
PREFIX=/usr/local/haproxy
    - unless: test -d /usr/local/haproxy    #后面条件为“假”时执行
    - require:                  #下面两个条件要执行成功才能执行
      - pkg: pkg-init           #代表pkg-init里面的Pkg
      - file: haproxy-install

haproxy-init:
  file.managed:
    - name: /etc/init.d/haproxy
    - source: salt://haproxy/files/haproxy.init
    - user: root
    - group: root
    - mode: 755
    - require:
      - cmd: haproxy-install
  cmd.run:
    - name: chkconfig --add haproxy
    - unless: chkconfig --list | grep haproxy
    - require:
      - file: haproxy-init

net.ipv4.ip_nonlocal_bind:
  sysctl.present:
    - value: 1

haproxy-config-dir:
  file.directory:
    - name: /etc/haproxy
    - user: root
    - group: root
    - mode: 755
```

状态模块：状态间关系

功　　能：条件判断，主要用于cmd状态模块

常用方法：

•onlyif：检查的命令，仅当``onlyif``选项指向的命令返回true时才执行name定义的命令

•unless ：用于检查的命令，仅当``unless``选项指向的命令返回false时才执行name指向的命令

功能名称：requisites

功　　能：处理状态间关系

常用方法：

- require　　#我依赖某个状态
- require_in　　#我被某个状态依赖
- watch　　#我关注某个状态
- watch_in　　# 我被某个状态关注

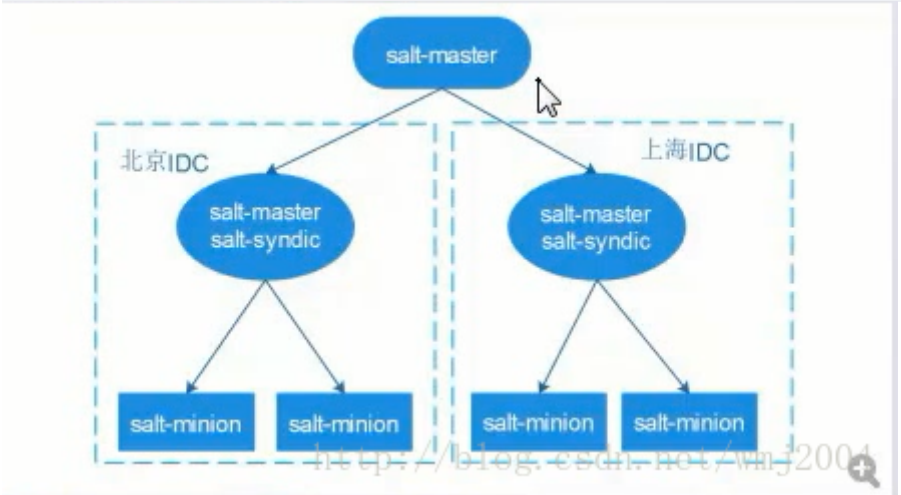
1.salt支持的三中工作模式：

一。本地模式(salt-call)

二。多master模式

三。代理模式（Syndic）：

Syndic必须运行在一个master上,然后连到另外一个更高级的master（这台机器就可以管理syndic-master所管理的机器）。



2.salt自定义grains：

mkdir /srv/salt/base/_grains #自定义的必须放这里

vim my_grains.py

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-

def my_grains():
    #初始化一个grains字典
    grains = {}
    grains['iaas'] = 'openstack'
    grains['edu'] = 'wmjedu'
    return grains
```

salt ‘*’ saltutil.sync_grains #同步自定义的grains到minion

salt ‘*’ grains.item ‘hehe1’ #查看自定义的grains

PS: Grains是静态的，收集一次就不会变了，需要自己用“saltutil.sync_grains”刷新!

3.salt自定义模块：

mkdir /srv/salt/base/_modules #文件夹名称不能变

vim my_disk.py

```
def list():
    cmd = 'df -h'
    ret = __salt__['cmd.run'](cmd) #调用salt自带cmd.run模块执行
    return ret
```

salt ‘*’ saltutil.sync_modules #下发自定义模块到minion

salt ‘*’ my_disk.list #执行自定义模块

PS: “/usr/lib/python2.6/site-packages/salt/modules/” 这个是salt官方模块放的位置，可以参考来写.

4.Salt的无master下运行：

首先要安装salt-minion,然后修改配置文件：

vim /etc/salt/minion

```
file_client: local
```

还要配置:

```
file_roots
```

salt-call –local state.highstate #通过这条命令在本地执行

5. Salt-ssh的使用：

yum install salt-ssh

vim /etc/salt/roster #配置客户端列表

```
SH_T_test_04:
  host: 172.16.1.214
  user: root
  port: 22
```

salt-ssh '*' test.ping -i #第一次连接会自动安装公钥到客户端上
salt-ssh '*' -r ifconfig # -r后面可以直接写命令
salt-ssh '*' state.highstate
salt-ssh '*' state.sls web.apache #执行状态文件