自动化运维之saltstack

# saltstack快速入门

## saltstack介绍

SaltStack是一个服务器基础架构集中化管理平台，SaltStack基于Python语言实现，也是基于C/S架构，结合轻量级消息队列（ZeroMQ）与Python第三方模块（Pyzmq、PyCrypto、Pyjinjia2、python-msgpack和PyYAML等）构建，SaltStack整合了Puppet和Chef的功能，更适合大规模批量管理服务器，配置简单。

三大主要功能：远程执行、配置管理、云管理

运行方式：local（本地）、Master/Minion（传统方式）、Syndic（分布式）、Salt SSH

[saltstack官方网站](https://saltstack.com/)

## SaltStack工作原理

Saltstack使用Python开发，是一个非常简单易用和轻量级的管理工具。由Master和Minion构成，通过ZeroMQ进行通信。

Saltstack的master端监听4505与4506端口，4505为salt的消息发布系统，4506为salt客户端与服务端通信的端口；salt客户端程序不监听端口，客户端启动后，会主动连接master端注册，然后一直保持该TCP连接，master通过这条TCP连接对客户端控制，如果连接断开，master对客户端就无能为力了。当然，客户端若检查到断开后会定期的一直连接master端的。

salt-master服务启动后会开启两个端口:4505和4506，minion没有端口，通过“双向密钥交换”(可通过tree /etc/salt/pki命令查看)来实现安全管理。salt-master每执行一条命令，所有minion均可收到。

## saltstack安装

saltstack支持多种操作系统，如centos、redhat、debian、ubuntu、FreeBSD等，以及windows（仅支持minion）centos7按以下方法安装

1. sudo yum install <https://mirrors.aliyun.com/saltstack/yum/redhat/salt-repo-latest-2.el7.noarch.rpm>
2. 配置仓库

sed -i 's/repo.saltstack.com/mirrors.aliyun.com\/saltstack/g' /etc/yum.repos.d/salt-latest.repo

1. yum makecache

生成缓存

4. Install the salt-minion, salt-master, or other Salt components:

sudo yum install salt-master master

sudo yum install salt-minion 被控制端

sudo yum install salt-ssh ssh管理方式

sudo yum install salt-syndic 分布式管理方式

sudo yum install salt-cloud 云管理

sudo yum install salt-api

在centos6 64位环境按照以下方法安装

安装 epel--------提供2015版本的salt

centos 6 64位

rpm -Uvh \

<http://mirrors.yun-idc.com/epel/6Server/x86_64/epel-release-6-8.noarch.rpm>

master与minion端都需要安装

安装完成后在master端执行 yum -y install salt-master

在minion端执行yum -y install salt-minion

安装完环境：

|  |  |
| --- | --- |
| 主机名 | IP |
| salt-master | 192.168.132.20 |
| salt-minion01 | 192.168.132.11 |
| salt-minion01 | 192.168.132.10 |

minion端配置使其指向master：

vim /etc/salt/minion

添加 master: 192.168.132.20 为master IP，实际生产环境最好为域名。

master端启动：/etc/init.d/salt-master start

开机自启动：chkconfig salt-master on

minion端启动：/etc/init.d/salt-minion start

开机自启动：chkconfig salt-minion on

安装完后查看salt信息

[root@salt-master app]# salt '\*' test.versions\_report

salt-minion02:

Salt: 2015.5.10

Python: 2.6.6 (r266:84292, Aug 18 2016, 15:13:37)

Jinja2: 2.2.1

M2Crypto: 0.20.2

msgpack-python: 0.4.6

msgpack-pure: Not Installed

pycrypto: 2.0.1

libnacl: Not Installed

PyYAML: 3.10

ioflo: Not Installed

PyZMQ: 14.3.1

RAET: Not Installed

ZMQ: 3.2.5

Mako: Not Installed

Tornado: Not Installed

timelib: Not Installed

dateutil: 1.4.1

salt-minion01:

Salt: 2015.5.10

Python: 2.6.6 (r266:84292, Aug 18 2016, 15:13:37)

Jinja2: 2.2.1

M2Crypto: 0.20.2

msgpack-python: 0.4.6

msgpack-pure: Not Installed

pycrypto: 2.0.1

libnacl: Not Installed

PyYAML: 3.10

ioflo: Not Installed

PyZMQ: 14.3.1

RAET: Not Installed

ZMQ: 3.2.5

Mako: Not Installed

Tornado: Not Installed

timelib: Not Installed

dateutil: 1.4.1

## salt-key

查询帮助 salt-key --help

常用参数

-a -A -r -R -d -D -Y

查看key salt-key

接收minion端的key： salt-key -A -y ---》接收所有的minion且无需交互。接收完成后查看：

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

Accepted Keys:

salt-minion01

salt-minion02

Denied Keys:

Unaccepted Keys:

Rejected Keys:

测试

[root@salt-master ~]# salt '\*' test.ping

salt-minion01:

True

salt-minion02:

True

## saltstack认证原理

salt的数据传输是通过AES加密，master和minion之间在通信前，需要进行认证。通过认证的方式保证安全性，完成一次认证后，master就可以自由的控制minion来完成各项工作了。

1. minion在第一次启动时，会在/etc/salt/pki/minion下生成minion.pem（private key）和minion.pub（public key），然后将minion.pub发送给master。
2. master在第一次启动时，会在/etc/salt/pki/master下生成master.pem和master.pub。通过salt-key接收minion的public key，在master的/etc/salt/pki/master/minions目录下存放，以minion id命名的public key，同时minion会保存一份master public key在/etc/salt/pki/minion\_master.pub

认证小结：

minion将自己的公钥发送给master

master认证后会将自己的公钥放置在minion端

由于这里是测试环境，master与minion端都关闭了防火墙及selinux。不然master可能无法接收minion端的key。

## saltstack远程执行

远程执行是saltstack的核心功能之一。主要使用salt模块可以批量给选定的minion端执行相应的命令，并获得返回结果。

如 salt '\*' test.ping \*代表所有，test表示模块，ping表示模块下的方法。

salt '\*' disk.usage 查看磁盘使用情况

saltstack可以使用cmd.run（非常重要） 模块远程执行shell命令

//让所有minion节点运行uptime命令

[root@salt-master ~]# salt '\*' cmd.run 'uptime'

salt-minion02:

10:30:44 up 1 day, 11:53, 1 user, load average: 0.00, 0.00, 0.00

salt-minion01:

10:30:45 up 1 day, 11:53, 1 user, load average: 0.00, 0.00, 0.00

#saltstack提供了大量的命令模块可以使用

#cmd.run可以执行所有的命令，不建议在生产环境中频繁使用。

## saltstack配置管理

salt通过状态模块来识别配置描述文件state状态文件，格式是YAML扩展名，后缀是.sls

YAML基础语法

YAML:

1. 缩进（层级关系） 2个空格，不能使用tab
2. 冒号 key：value ----》以冒号结尾，以冒号为路径的除外，其他必须有空格
3. 短横线

表示列表项，短横线加一个空格 - list1

* list2

配置文件

master端配置文件中有说明配置文件写在什么地方。

cat /etc/salt/master

# file\_roots:

# base: ----》基础环境

# - /srv/salt/

# dev: ---》开发环境

# - /srv/salt/dev/services

# - /srv/salt/dev/states

# prod: -----》生产环境

# - /srv/salt/prod/services

# - /srv/salt/prod/states

这里启用测试环境

file\_roots: ----》打头写

base: ---》缩进 2个空格

- /srv/salt ----》4个空格，文件路径

启用后重启master服务，创建目录 mkdir -p /srv/salt

在实际成产环境中可以根据业务分别创建不同的文件夹，然后在对应的里面写。

实例

这里在salt下创建app目录，然后在app目录中写配置文件。

[root@salt-master app]# pwd

/srv/salt/app

vim vsftpd.sls

[root@salt-master app]# cat vsftpd.sls

vsftpd\_install:

pkg.installed: ------>pkg模块 开头两个空格

- names: ------》下面接多个list 开头4个空格

- vsftpd

- httpd

## saltstack状态

配置文件写好后

执行 salt '\*' state.sls app.vsftpd env=base 默认为base环境

minion端安装完成，如下，如已经安装则忽略。

[root@salt-master salt]# salt '\*' cmd.run 'rpm -qa|grep vsftpd'

salt-minion01:

vsftpd-2.2.2-24.el6.x86\_64

salt-minion02:

vsftpd-2.2.2-24.el6.x86\_64

[root@salt-master salt]# salt '\*' cmd.run 'rpm -qa|grep httpd'

salt-minion01:

httpd-tools-2.2.15-60.el6.centos.6.x86\_64

httpd-2.2.15-60.el6.centos.6.x86\_64

salt-minion02:

httpd-2.2.15-60.el6.centos.6.x86\_64

httpd-tools-2.2.15-60.el6.centos.6.x86\_64

如果要安装后服务启动可以继续编写配置文件，如下：

[root@salt-master app]# cat vsftpd.sls

vsftpd\_install:

pkg.installed:

- names:

- vsftpd

- httpd

vsftpd\_service:

service.running:

- name: vsftpd

- enable: True

执行salt '\*' state.sls app.vsftpd

查看vsftp服务状态，已经启动

[root@salt-master app]# salt '\*' cmd.run 'service vsftpd status'

salt-minion01:

vsftpd (pid 12399) is running...

salt-minion02:

vsftpd (pid 12721) is running...

在实际生产环境中执行后状态是不能回退的，所以可以在执行前，进行检查

如 salt '\*' state.sls app.vsftpd test=True

## saltstack高级状态

如果执行一条语句，如何让不同的minion执行不同的动作，这里就要用到saltstack的高级状态。

salt执行高级状态模块，top只能放置在base环境，即top.sls需要放在base目录 /srv/salt下。

如实例：[root@salt-master salt]# cat top.sls

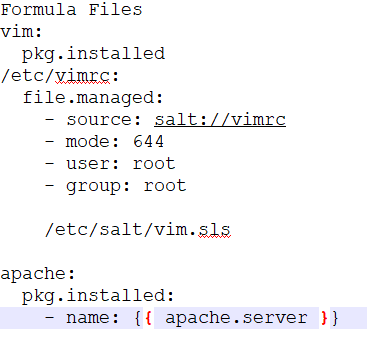
base:

'salt-minion01':

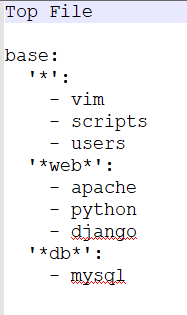
- app.apache

'salt-minion02':

- app.vsftpd

执行：salt '\*' state.highstate

Top 文件示范，top文件和其他的配置文件分开写。



显示saltstack进程名称方式

安装yum -y install python-setproctitle

在本次实例中,结构如下：

Top File

Formula File

apache-install:

pkg.installed:

- name: httpd

apache-service:

service.running:

- name: httpd

- enable: True

base:

'salt-minion01':

- app.apache

'salt-minion02':

- app.vsftpd

vsftpd\_install:

pkg.installed:

- name: vsftpd

vsftpd\_service:

service.running:

- name: vsftpd

- enable: True

## saltstack与ZeroMQ消息队列

我们进行自动化运维大多数情况下，是我们的服务器数量已经远远超过人工SSH维护的范围。saltstack可以支持以千计甚至更多的服务器。这些性能的提供主要是来自与ZeroMQ。因为saltstack底层是基于ZeroMQ进行高效的网络通信。ZeroMQ用于node与node间的通信，node可以是主机也可以是进程。

ZeroMQ是一个简单好用的传输层，像框架一样的一个套接字库，它使得socket编程更加简单、简洁和性能更高。它还是一个消息处理队列库，可在多个线程、内核和主机盒之间弹性伸缩。

发布与订阅

ZeroMQ支持publish/subscribe即发布与订阅模式，简称pub/sub

salt master运行两个网络服务

一个是ZeroMQ pub消息发布与订阅系统，默认监听4505端口，主要用于master对minion下发指令

salt master运行的第二个网络服务是ZeroMQ REP系统，默认监听端口是4506（客户端与服务端通信的端口），同时也主要用于master接收minion执行某个命令后的返回值。

# saltstack数据系统

## Grains

Grains：静态数据

当minion启动的时候收集minion的本地相关状态信息。

操作系统版本、内核版本、CPU、内存、硬盘、设备型号、序列号。

minion收集信息，master可以用于匹配目标主机

minion端设置，提供给master端

Grains收集minion启动时所有的系统信息，存储在minion端

grains.ls 列出所有可打印的模块

grains.item

grains.items 打印所有状态信息

查看ip地址 salt '\*' grains.item fqdn\_ip4

用途：

1. 资产管理，信息查询
2. 用于目标选择

minion端搜索grains进行配置

重启minion服务

salt '\*' grains.item roles

如果需要对特定的服务器如webserver执行动作，可以使用

salt -G 'roles:webserver' cmd.run "df -h"

这种方式需要在minion端进行配置，如果minion端服务器较多，该方式比较繁琐

1. 配置管理中使用

master端编写一个grains的python脚本推送给minion

[root@salt-master salt]# ll

total 0

drwxr-xr-x 2 root root 41 Nov 19 2018 app

drwxr-xr-x 2 root root 26 Nov 18 11:56 \_grains

[root@salt-master salt]# pwd

/srv/salt

[root@salt-master \_grains]# cat my\_grains.py

#!/usr/bin/env python

#-\*- coding:utf-8 -\*-

def my\_grains():

grains = {}

grains['iaas'] = 'openstack'

grains['pass'] = 'docker'

return grains

刷新grains ---推送到minionn 文件存放路径为:/var/cache/salt

salt '\*' saltutil.sync\_grains

grains优先级

1. 系统默认获取的状态信息
2. minion配置文件中定义
3. 在master端自定义grains。自定义grains会存储在/var/cache/salt

[root@salt-minion01 salt]# pwd

/var/cache/salt

[root@salt-minion01 salt]# tree ./

./

`-- minion

|-- accumulator

|-- extmods

| `-- grains

| |-- my\_grains.py

| `-- my\_grains.pyc

|-- files

| `-- base

| |-- \_grains

| | `-- my\_grains.py

| |-- app

| | |-- httpd.sls

| | `-- vsftpd.sls

| `-- top.sls

|-- highstate.cache.p

|-- module\_refresh

|-- pkg\_refresh

|-- proc

`-- sls.p

## Pillar

pillar动态的。

给特定的minion指定特定的数据。只有指定的minion自己能看到自己的数据。

master端设置，提供给minion端。

pillar在saltstack中主要作用是存储和定义配置管理中需要的一些数据，比如软件版本号、用户名密码等信息，它的存储格式跟grains类似，都是YAML格式。

pillar\_roots:

base:

* /srv/pillar

1. master端进行pillar配置，创建目录mkdir /srv/pillar。

[root@salt-master pillar]# cat zabbix.sls

zabbix\_server:192.168.132.11

zabbix\_name:salt-minion01

1. 编写topfile，指定minion端可以使用。

[root@salt-master pillar]# cat top.sls

base:

'salt-minion01':

- zabbix

3.刷新到对应的主机

salt '\*' saltutil.refresh\_pillar

4获取对应的pillar值

salt '\*' pillar.items

salt '\*' pillar.items

salt-minion02:

----------

salt-minion01:

----------

zabbix\_server:

192.168.132.11

pillar使用场景

grains VS pillar

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 数据 | 类型 | 数据采集方式 | 应用场景 | | | 定义位置 |
| grains | 静态 | minion启动时收集 | 数据查询 | 目标选择 | 配置管理 | minion |
| pillar | 动态 | master自定义 | 目标选择 | 配置管理 | 敏感数据 | master |

# saltstack远程执行

深入学习saltstack远程执行：

salt '\*' cmd.run 'w'

命令：salt

目标： '\*'

模块:cmd.run https://docs.saltstack.com/en/latest/salt-modindex.html

返回:执行后返回

目标：一种和minion id有关

通配符 salt\* salt[1|2]

正则表达式 salt -E

列表 salt -L 'salt-minion01,salt-minion02'

和minion id无关

subnet/IP salt -S '192.168.132.0/24' test.ping

grains

compound matchers（复合匹配 and or）

也是在master端中的nodegroups中设置

nodegroups（节点组）------需要在master端配置

root@salt-master:~ # grep '^nodegroups' -A 1 /etc/salt/master

nodegroups:

webserver: 'salt-minion01'

重启master服务

验证

root@salt-master:~ # salt -N 'webserver' test.ping

salt-minion01:

True

在实际生产环境中，可以对web服务器分组，部署web服务时就可以只针对web服务器进行操作。

## pkg模块

0.初始 user crond

1. 安装 pkg salt '\*' pkg.install "php" 卸载使用remove
2. 配置 file
3. 启动 service

软件包管理

pkg

pkg.install pkg.latest pkg.remove

## file模块

file.touch ----创建文件

file.stats --文件状态信息

file.chown ----更改属性

...

## service模块

管理服务

## 返回

执行命令后minion返回的结果

1. 将返回结果写到mysql数据库中进行查看

安装mysql及mysql-python

master端：yum -y install mysql-python mariadb-server mariadb

启动数据库：systemctl start mariadb

启动mysql后授权：grant all on salt.\* to salt@"%" identified by 'passwd';

安装官方文档创建数据库及表

<https://docs.saltstack.com/en/latest/ref/returners/all/salt.returners.mysql.html>

minion端：yum -y install mariadb mysql-python

测试minion端可以连接master的mariadb

mysql -h192.168.132.30 -usalt -p'passwd'

minion端配置：

[root@salt-minion01 ~]# grep mysql /etc/salt/minion

#return: mysql

mysql.host: '192.168.132.30'

mysql.user: 'salt'

mysql.pass: 'passwd'

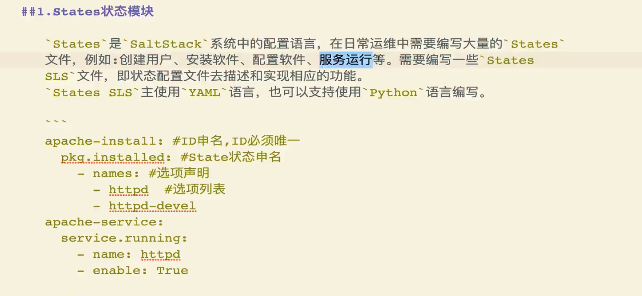
mysql.db: 'salt'

mysql.port: 3306

#return: mysql,slack,redis

# - mysql

# 配置管理



## pkg模块

列出所有状态模块

salt '\*' sys.list\_modules

查看状态模块中的函数：如pkg模块

[root@salt-master \_grains]# salt '\*' sys.list\_state\_functions pkg

salt-minion01:

- pkg.downloaded

- pkg.group\_installed

- pkg.installed

- pkg.latest

- pkg.mod\_aggregate

- pkg.mod\_init

- pkg.mod\_watch

- pkg.patch\_downloaded

- pkg.patch\_installed

- pkg.purged

- pkg.removed

- pkg.uptodate

salt-minion02:

- pkg.downloaded

- pkg.group\_installed

- pkg.installed

- pkg.latest

- pkg.mod\_aggregate

- pkg.mod\_init

- pkg.mod\_watch

- pkg.patch\_downloaded

- pkg.patch\_installed

- pkg.purged

- pkg.removed

- pkg.uptodate



## file模块



root@salt-master app]# mkdir files

[root@salt-master app]# cp /etc/resolv.conf files/

[root@salt-master app]# cat dns.sls

dns-config:

file.managed:

- name: /etc/resolv.conf #file of

- source: salt://app/files/resolv.conf salt代表file根目录

- user: root

- group: root

- mode: 644

- backup: minion ------备份到minion端

修改/srv/salt/app/files下的resolv.conf文件

[root@salt-master files]# cat resolv.conf

# Generated by NetworkManager

nameserver 101.206.4.6

nameserver 114.114.114.114

###########saltstack configure ----------》添加一行注释

salt 'salt-minion01' state.sls app.dns test=True 测试

salt 'salt-minion01' state.sls app.dns

查看

[root@salt-minion01 salt]# cat /etc/resolv.conf

# Generated by NetworkManager

nameserver 101.206.4.6

nameserver 114.114.114.114

###########saltstack configure

file.directory ------创建目录

[root@salt-master app]# cat dir.sls

mkdir\_tmp\_dns:

file.directory:

- name: /tmp/dns

- user: root

- group: root

- mode: 755

- makedirs: True

目录下发 file.recurse

[root@salt-master app]# cat xf.sls

httpd-config:

file.recurse:

- name: /etc/httpd/conf.d/

- source: salt://app/files/httpd\_conf.d

- file\_mode: 644

- dir\_mode: 755

- include\_empty: True

- clean: True #####远程强制删除

mkdir files/httpd\_conf.d

[root@salt-master app]# cat files/httpd\_conf.d/www.conf

server {

www

}

[root@salt-master app]# cat files/httpd\_conf.d/blog.conf

erver {

blog

}

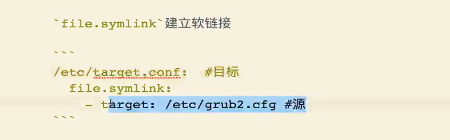
[root@salt-master app]# salt 'salt-minion01' state.sls app.xf

minion01

[root@salt-minion01 tmp]# ls /etc/httpd/conf.d/

autoindex.conf fcgid.conf README userdir.conf www.conf

blog.conf manual.conf ssl.conf welcome.conf



可以直接采用以下方法：

[root@salt-master app]# cat ln.sls

Link:

cmd.run:

- name: ln -s /tmp /mnt

## service模块

httpd\_service:

service.running:

- name: httpd

- enable: True

- reload: True #######允许重载，不写表示restart

- watch: #####监控谁

-pkg.httpd-file

## cron模块

[root@salt-master app]# cat cron.sls

crontab\_scripts:

cron.present:

- name: /usr/bin/uptime >/dev/null 2>&1

- user: root

- minute: '\*/5'

salt 'salt-minion01' state.sls app.cron

结果:

[root@salt-minion01 tmp]# crontab -l

\*/30 \* \* \* \* /usr/sbin/ntpdate time.windows.com

# Lines below here are managed by Salt, do not edit

# SALT\_CRON\_IDENTIFIER:/usr/bin/uptime >/dev/null 2>&1

\*/5 \* \* \* \* /usr/bin/uptime >/dev/null 2>&1

## 高级状态模块

base:

'\*': 匹配所有minion

- app.httpd #base模块下httpd目录下的安装sls文件

#webserver: #定义的分组名称进行匹配,需要定义nodegroups

# - match: nodegroups

# - nginx

'os:centos': #通过grains模块匹配

- match: grains

- app.cron

top.sls需要写在/srv/salt下

执行是salt '\*' state.highstate

## LAMP架构部署

此案例在prod生产环境下部署

修改master端配置文件

file\_roots:

base:

- /srv/salt/base

prod:

- /srv/salt/prod

systemctl restart salt-master

mkdir /srv/salt/{base,prod} -p

把之前的基础环境下写的放在base下

### 先手动安装lamp环境

yum install httpd mariadb mariadb-server php php-mysql php-gd gd -y

[root@salt-master init]# pwd

/srv/salt/prod/init

mkdir files -------相关配置写在该目录下

### 准备相应目录与对应配置文件

cp /etc/httpd/conf/httpd.conf files/

cp /etc/my.cnf files/

cp /etc/php.ini files/

在init目录下写lamp安装，配置 启动

### 部署状态文件lamp.sls

[root@salt-master init]# cat lamp.sls

#yum install httpd mariadb-server php php-mysql php-gd gd -y

httpd-install:

pkg.installed:

- names:

- httpd

- httpd-tools

httpd-config:

file.managed:

- name: /etc/httpd/conf/httpd.conf

- source: salt://init/files/httpd.conf

- user: root

- group: root

- mode: 644

- backup: minion

httpd-service:

service.running:

- name: httpd

- enable: True

- reload: True

php-install:

pkg.installed:

- names:

- php

- php-mysql

- php-gd

- gd

php-config:

file.managed:

- name: /etc/php.ini

- source: salt://init/files/php.ini

- user: root

- group: root

- mode: 644

- backup: minion

mysql-install:

pkg.installed:

- names:

- mariadb

- mariadb-server

mysql-config:

file.managed:

- name: /etc/my.cnf

- source: salt://init/files/my.cnf

- user: root

- group: root

- mode: 644

- backup: minion

mysql-service:

service.running:

- name: mariadb

- enable: True

salt 'salt-minion01' state.sls init.lamp saltenv=prod 指定为prod环境，因为默认为base环境

### 通过高级状态进行管理

通过高级状态来管理更方便，需要在base下写top.sls

[root@salt-master base]# cat top.sls

prod:

'salt-minion01':

- init.lamp

然后执行salt 'salt-minion01' state.highstate

查看目录结构

[root@salt-master base]# tree /srv/salt/

/srv/salt/

├── base

│   ├── app

│   │   ├── cron.sls

│   │   ├── dir.sls

│   │   ├── dns.sls

│   │   ├── files

│   │   │   ├── httpd\_conf.d

│   │   │   │   ├── blog.conf

│   │   │   │   └── www.conf

│   │   │   └── resolv.conf

│   │   ├── httpd.sls

│   │   ├── ln.sls

│   │   ├── service.sls

│   │   ├── vsftpd.sls

│   │   └── xf.sls

│   ├── \_grains

│   │   └── my\_grains.py

│   ├── top.sls

│   └── top.sls20181121

└── prod

└── init

├── files

│   ├── httpd.conf

│   ├── my.cnf

│   └── php.ini

└── lamp.sls

### states状态依赖管理

可以用state模块来定义minion的状态，但是如果一个主机涉及多个状态，并且状态之间有相互关联，需要在执行顺序上有先后之分，那么必须引入requisites 来进行控制。

1. require 我依赖某个状态，我依赖谁
2. require\_in 我被某个状态依赖，谁依赖我
3. watch 我监控某个状态，如当状态发生变化时，就进行restart或reload操作
4. watch\_in 我被某个状态关注
5. include 我引用谁

require写法

httpd-install:

pkg.installed:

- names:

- httpd

- httpd-tools

httpd-config:

file.managed:

- name: /etc/httpd/conf/httpd.conf

- source: salt://init/files/httpd.conf

- user: root

- group: root

- mode: 644

- backup: minion

- require:

- pkg: httpd-install 因为要修改配置需要先安装软件

watch写法

httpd-service:

service.running:

- name: httpd

- enable: True

- reload: True

- require:

- file: httpd-config

- watch:

- file: httpd-config

watch\_in写法

php-config:

file.managed:

- name: /etc/php.ini

- source: salt://init/files/php.ini

- user: root

- group: root

- mode: 644

- backup: minion

- watch\_in:

- service: httpd-service

利用include将lamp拆分，如将mysql拆分出来

[root@salt-master init]# cat mysql.sls

mysql-install:

pkg.installed:

- names:

- mariadb

- mariadb-server

mysql-config:

file.managed:

- name: /etc/my.cnf

- source: salt://init/files/my.cnf

- user: root

- group: root

- mode: 644

- backup: minion

- require:

- pkg: mysql-install

mysql-service:

service.running:

- name: mariadb

- enable: True

- require:

- file: mysql-config

[root@salt-master init]# mv lamp.sls apache.sls

在prod目录建立lamp.sls

[root@salt-master prod]# cat lamp.sls

include:

- init.apache

- init.mysql

执行salt 'salt-minion01' state.sls lamp saltenv=prod

通过高级状态管理：修改topfile文件如下：

prod:

'salt-minion01':

- lamp

## Jinja模板使用

配置文件一般都是固定无法灵活多变，实际生产过程中，需要配置文件灵活多变，不能写死，有时还需要进行流程控制，这就需要用到jinjia，可以很灵活地进行配置管理及简化配置过程。

### Jinja2模板包含变量和表达式

变量用{{...}}包围，表达式用{%...%}包围

看一个简单的例子

[root@salt-master prod]# cat var.sls

{% set var= 'hello world!' %}

test\_var:

cmd.run:

- name: echo "{{ var }}"

**字符串类型**

{% set var= 'hello world!' %}

**列表类型**

{% set list = ['one','two' ] %}

{{ list[1]}}

**字典类型**

{% set dict = {'first':'value','second':'value1'} %}

{{ dict['first'] }}

### Jinja模板使用步骤

1. 告诉file状态模块，需要使用jinja

- template: jinja

2列出参数列表

- defaults：

PORT: 88

3.配置文件引用jinja模板

{{ PORT }}

配置实例，修改apache端口

httpd-config:

file.managed:

- name: /etc/httpd/conf/httpd.conf

- source: salt://init/files/httpd.conf

- user: root

- group: root

- mode: 644

- backup: minion

- require:

- pkg: httpd-install

- template: jinja

- defaults:

PORT: 8000

files目录下http配置文件中将Listen 80改为Listen {{ PORT }}

然后执行salt 'salt-minion01' state.highstate

查看minion端端口已改动

[root@salt-minion01 ~]# netstat -tnlp|grep 8000

tcp6 0 0 :::8000 :::\* LISTEN 3915/httpd

模板支持grains pillar进行赋值

#### grains方式

通过grains获取minion端IP

[root@salt-master prod]# salt 'salt-minion01' grains.item fqdn\_ip4

salt-minion01:

----------

fqdn\_ip4:

- 192.168.132.31

如果想在http配置文件中使用IP:PORT

Listen {{ IPADDR}}:{{ PORT }}

apache文件中修改

- defaults:

PORT: 8000

IPADDR: {{ grains['fqdn\_ip4'][0]}} ---取列表中的第一个值即IP地址

#### pillar方式：

apache:

IP: {{ grains['fqdn\_ip4'][0] }}

PORT: 8008

引用

{{ pillar['apache'][IP] }}

{{ pillar['apache'][port] }}

实际场景中，如系统有centos和ubuntu，都要安装apache，由于安装的包不一样，则可以根据grains静态数据，使用jinja2流程控制

httpd\_install:

pkg.installed:

- template: jinja

{% if grains['os\_family'] == 'Debian' %}

- name: apache2

{% elif grains['os\_family'] == 'RedHat' %}

- name: http

{% endif %}

## saltstack Job管理

### Job概述

salt每次运行任务都会将任务发布到pub-sub总线，minion会对任务作出响应，为区分不同的任务，saltmaster每次发布一个任务都会为该任务创建一个jobid。

master默认情况下会缓存24小时内的所有job的详细操作

master缓存目录：/var/cache/salt/master/jobs/

minion端每次执行任务都会缓存在/var/cache/salt/minion/proc/目录下，任务执行完成后文件会被删除。

在master端执行一个长时间的任务

salt 'salt-minion01' cmd.run "sleep 100"

在minion01上用strings查看文件内容

[root@salt-minion01 proc]# strings 20181125111411231106

tgt\_type

glob

20181125111411231106

salt-minion01

user

root

sleep 100

cmd.run

### Job管理

通过salt-run 命令来管理job也可以通过salt util模块

在master中执行一个长时间的命令

salt 'salt-minion01' cmd.run "sleep 1000;echo hello"

然后ctrl+c结束，获取jobid后登陆查看

salt 'salt-minion01' saltutil.find\_job-id

salt 'salt-minion01' saltutil.kill\_job jobid 停掉任务

查看master上cache的所有job

salt 'salt-minion' saltutil.runner jobs.list\_jobs

# saltstack项目实践

## 自动化部署集群架构开篇

规划：

系统初始化配置

基础模块配置管理

nginx keepalived php mysql ...

业务模块配置管理

根据业务类型划分，然后包含基础模块配置

web服务，论坛、 bbs

## 系统环境初始化配置

master端环境配置，这里只配置base及prod环境

file\_roots:

base:

- /srv/salt/base

prod:

- /srv/salt/prod

配置pillar

pillar\_roots:

base:

- /srv/pillar/base

prod:

- /srv/pillar/prod

重启master：systemctl restart salt-master

内核优化，limit，时间同步都可以通过file模块进行配置文件管理。

yum源安装

epel源 rpm -Uvh <http://mirrors.yun-idc.com/epel/7Server/x86_64/Packages/e/epel-release-7-11.noarch.rpm>

cd /etc/pki/rpm-gpg

wget <https://archive.fedoraproject.org/pub/epel/RPM-GPG-KEY-EPEL-7>

nginx源：

[root@salt-master yum.repos.d]# cat nginx.repo

[nginx]

name=nginx repo

baseurl=http://nginx.org/packages/centos/7/x86\_64/

gpgcheck=0

enabled=1

基础yum源，选用阿里云的wget <http://mirrors.aliyun.com/repo/Centos-7.repo>

通过file的目录分发进行repo下发

[root@salt-master yum.repos.d.template]# ll

total 20

-rw-r--r-- 1 root root 2523 Jun 16 06:22 Centos-7.repo

-rw-r--r-- 1 root root 2424 Oct 25 01:22 docker-ce.repo

-rw-r--r-- 1 root root 951 Oct 3 2017 epel.repo

-rw-r--r-- 1 root root 96 Nov 25 13:59 nginx.repo

-rw-r--r-- 1 root root 253 Jul 16 22:24 salt-latest.repo

[root@salt-master yum.repos.d.template]# pwd

/srv/salt/base/init/files/yum.repos.d.template

[root@salt-master init]# cat repo.sls

repo-config:

file.recurse:

- name: /etc/yum.repos.d

- source: salt://init/files/yum.repos.d.template

- user: root

- group: root

- file\_mode: 644

- dir\_mode: 755

- include\_empty: True

- clean: True -------远程和本地目录保持一直（远程强制删除）

salt 'salt-minion01' state.sls init.repo

基础环境安装

[root@salt-master init]# cat pkg.sls

include:

- init.repo

pkg-install:

pkg.installed:

- names:

- tree

- telnet

- sysstat

- iftop

- iotop

- dstat

- lrzsz

- gcc

- make

- require:

- file: repo-config

创建用户，所有web服务全在www用户下运行

[root@salt-master init]# cat user.sls

www-group:

group.present:

- name: ww

- gid: 5000

www-user:

user.present:

- name: www

- shell: /bin/nologin

- uid: 5000

- gid: 5000

## 基础模块编写-nginx

在prod环境，以下为目录结构

[root@salt-master modules]# tree /srv/salt/prod

/srv/salt/prod

├── cluster

│   ├── keepalivved

│   └── proxy

└── modules

├── mysql

│   └── files

├── nginx

│   └── files

└── php

└── files

10 directories, 0 files

将nginx的安装与启动分开来写：

[root@salt-master nginx]# pwd

/srv/salt/prod/modules/nginx

[root@salt-master nginx]# tree ./

./

├── files

│   └── nginx.conf.template

├── install.sls

└── service.sls

1 directory, 3 files

[root@salt-master nginx]# cat install.sls

nginx-install:

pkg.installed:

- name: nginx

nginx-config:

file.managed:

- name: /etc/nginx/nginx.conf

- source: salt://modules/nginx/files/nginx.conf.template

- user: root

- group: root

- mode: 644

- require:

- pkg: nginx-install

- backup: minion

[root@salt-master nginx]# cat service.sls

include:

- modules.nginx.install

nginx-service:

service.running:

- name: nginx

- enable: True

- reload: True

- require:

- file: nginx-config

- watch:

- file: nginx-config

用一台minion测试下：

salt 'salt-minion01' state.sls modules.nginx.service saltenv=prod

service文件里已经包含了install。

安装过程中有错误：

Delta RPMs disabled because /usr/bin/applydeltarpm not installed

错误原因：

未安装deltarpm。

解决方案：

依次在minion端执行下面两条命令，即可解决。

yum provides '\*/applydeltarpm'

yum install deltarpm

安装完成后查看minion端

[root@salt-minion01 tools]# netstat -tnlp|grep nginx

tcp 0 0 0.0.0.0:80 0.0.0.0:\* LISTEN 6578/nginx: master

## 基础模块编写-php

php模块编写类似与nginx，也是安装，配置与启动。

[root@salt-master php]# ll

total 8

drwxr-xr-x 3 root root 85 Nov 29 21:27 files

-rw-r--r-- 1 root root 865 Nov 29 21:45 install.sls

-rw-r--r-- 1 root root 333 Nov 29 21:28 service.sls

[root@salt-master php]# pwd

/srv/salt/prod/modules/php

[root@salt-master php]# cat install.sls

php-install:

pkg.installed:

- names:

- php

- php-devel

- php-fpm

- php-pdo

- php-mysql

php-ini-config:

file.managed:

- name: /etc/php.ini

- source: salt://modules/php/files/php.ini.template

- user: root

- group: root

- mode: 644

- require:

- pkg: php-install

- backup: minion

php-fpm-config:

file.managed:

- name: /etc/php-fpm.conf

- source: salt://modules/php/files/php-fpm.conf.template

- user: root

- group: root

- mode: 644

- require:

- pkg: php-install

- backup: minion

php-fpm-dir-config:

file.recurse:

- name: /etc/php-fpm.d

- source: salt://modules/php/files/php-fpm.d.template

- user: root

- group: root

- file\_mode: 644

- dir\_mode: 755

- clean: True

- include\_empty: True

- require:

- pkg: php-install

php有两个配置文件及一个目录，都进行配置管理。

[root@salt-master php]# cat service.sls

include:

- modules.php.install

php-service:

service.running:

- name: php-fpm

- enable: True

- reload: True

- require:

- file: php-ini-config

- file: php-fpm-config

- file: php-fpm-dir-config

- watch:

- file: php-ini-config

- file: php-fpm-config

- file: php-fpm-dir-config

服务启动依赖与php安装配置。

## 基础模块编写mysql

[root@salt-master mysql]# cat install.sls

mysql-install:

pkg.installed:

- names:

- mariadb

- mariadb-server

mysql-config:

file.managed:

- name: /etc/my.cnf

- source: salt://modules/mysql/files/my.cnf.template

- user: root

- group: root

- mode: 644

- backup: minion

- require:

- pkg: mysql-install

[root@salt-master mysql]# cat service.sls

include:

- modules.mysql.install

mysql-service:

service.running:

- name: mariadb

- enable: True

- require:

- file: mysql-config

## 业务模块引入code

[root@salt-master webnode]# pwd

/srv/salt/prod/cluster/webnode

[root@salt-master webnode]# tree ./

./

├── files

│   └── conf.d.template

│   └── bbs.conf

└── vhosts.sls

2 directories, 2 files

bbs配置如下：

[root@salt-master webnode]# cat files/conf.d.template/bbs.conf

server {

listen 80;

server\_name bbs.bgx.com;

root /code/bbs;

index index.php index.html index.htm;

location / {

root /code/bbs;

index index.php index.html index.htm;

}

location ~ \.php$ {

root /code/bbs;

fastcgi\_pass 127.0.0.1:9000;

fastcgi\_index index.php;

fastcgi\_param SCRIPT\_FILENAME /code/bbs$fastcgi\_script\_name;

include fastcgi\_params;

}

}

创建一个虚拟主机配置。

[root@salt-master webnode]# cat vhosts.sls

include:

- modules.nginx.service

- modules.php.service

vhost-config:

file.recurse:

- name: /etc/nginx/conf.d

- source: salt://cluster/webnode/files/conf.d.template

- user: root

- group: root

- file\_mode: 644

- dir\_mode: 755

- include\_empty: True

- clean: True

- require\_in:

- service: nginx-service

- service: php-service

代码配置，实际工作中不需要，代码一般都是自动化部署。

[root@salt-master webnode]# cat code.sls

include:

- cluster.webnode.vhosts

bbs-code-dir:

file.directory:

- name: /code/bbs

- user: root

- group: root

- mode: 755

- makedirs: True

- require:

- file: vhost-config

bbs-code:

file.managed:

- name: /code/bbs/index.php

- source: salt:///cluster/webnode/files/index.php

- user: root

- group: root

- mode: 644

- backup: minion

- require:

- file: bbs-code-dir

在电脑添加hosts解析192.168.132.32 bbs.bgx.com。

然后浏览器访问域名。

通过高级状态模块来管理

[root@salt-master base]# cat top.sls

prod:

'\*':

- cluster.webnode.code

# saltstack架构扩展

## 双master架构

1. 安装一台新的master主机，不要启动，因为启动后会产生秘钥文件
2. 同步旧master的配置文件，状态文件、秘钥文件到新的master

rsync -avz /etc/salt/master xxxxxxx 配置文件

rsync -avz /srv/ xxxxxxx 状态文件

rsync -avz /etc/salt/pki/master/master.\* xxxxx:/ /etc/salt/pki/master 秘钥文件

3.修改minion端配置

master:

- server1

- server2

同步minion配置到其他minion端，或者在master端下发minion的配置文件，通过file模块管理。

利用旧的master重启minion服务

salt-key -A -y接收minion端的key

4.启动新的master服务

在实际双主架构中可以通过rsync+inotify或sersync实时同步配置文件状态文件及秘钥文件。

## syndic分布式架构