

**Saltstack的工作原理图**

**用一个简单的命令来演示一下工作顺序：**

Salt ‘\*’ test.ping 此处的\*是代表所有主机，可以针对的写入执行命令在某台主机salt ‘hostname’ “order”

1. 作为用户client在master输入一条命令，master会把命令交给关于salt启动的进程管理器ReqServer
2. 进程管理器ReqServer会选择一个当前空闲的mworker进程来处理client的请求，即在/var/run/salt/master/目录下通过[workers.ipc](https://docs.saltstack.com/en/latest/topics/development/architecture.html" \l "eventpublisher)进行处理（此过程还有另一个方式在minion端执行[salt-call](https://docs.saltstack.com/en/latest/ref/cli/salt-call.html)直接返回关键信息给master，通过4506端口bind路由环境，master将该信息交给闲置的mworker）
3. mworker会开始执行利用eventpulisher，即在/var/run/salt/master/目录下通过[master\_event\_pull.ipc](https://docs.saltstack.com/en/latest/topics/development/architecture.html" \l "eventpublisher)来进行事件推送处理
4. 在Publisher前，要做一个master与minion根据命令的判断，与hostname和用户执行命令权限设置有关（salt -a pam ‘hostname’ “order”)；publisher，即用/var/run/salt/master/目录下的publish\_pull.ipc来处理并通过4505来将命令发布给相对于命令需求的minion端
5. Minion通过4505接收到数据并在本地执行操作
6. Minion把本地执行操作后的数据通过4506端口响应给master，master端的ReqServer接收到响应并交给闲置的mworker
7. Mworker接收数据利用eventpulisher推出响应后的返回状态通知，即在/var/run/salt/master/目录下的[master\_event\_pub.ipc](https://docs.saltstack.com/en/latest/topics/development/architecture.html" \l "eventpublisher)来事件处理后的状态公布
8. 当所有minion都返回结果后，或者显示超时后，此次任务结束

**以上说到minion的响应给master的信息，下面我分享一个收集minion端响应的python脚本以便观测用：**

#!/usr/bin/env python

#encoding=utf-8

import salt.config

import salt.utils.event

opts = salt.config.client\_config('/etc/salt/master')

event = salt.utils.event.get\_event(

'master',

sock\_dir=opts['sock\_dir'],

transport=opts['transport'],

opts=opts)

for nb in event.iter\_events(full=True):

print nb

print '-' \* 30

**工作原理讲述完毕**

首先salt是以收集minion端的服务器信息来判断和指定执行，准确说就是收集minion端的根据服务器相关的所有信息以字典的形式进行收集

**介绍两个命令：salt ‘\*’ grains.items，salt ‘\*’ pillar.items可以查看minion端的字典信息，grains.items列出的是minion静态（固定）的信息，pillar.items列出的是minion动态（二次添加）的信息**

**以下配置和信息最基本的开始分享：**

最常用的命令：

salt-key -L 查看minion列表

salt-key -A 添加没有被添加进master的minion名单里的所有minion

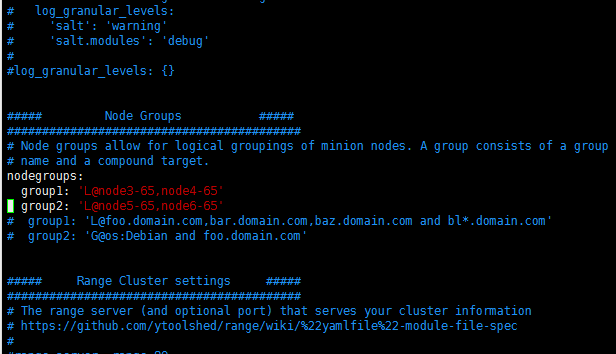
salt-key -a ‘hostname’ 添加指定minion

salt-key -d ‘hostname’ 删除指定minion

可以进行对minion的不同需要进行分组配置来进行组批量执行：

Master

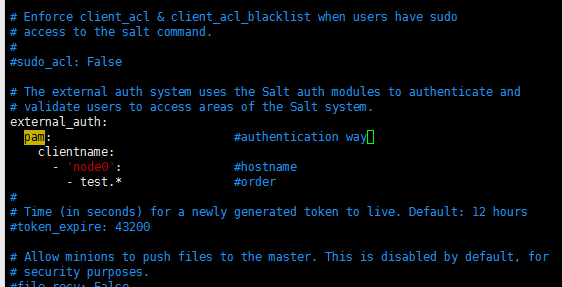
举个例子：



执行命令：salt -N group1 ”order”

可以针对某个用户名对该用户名进行主机分配权限，并限制执行某些命令

Master:



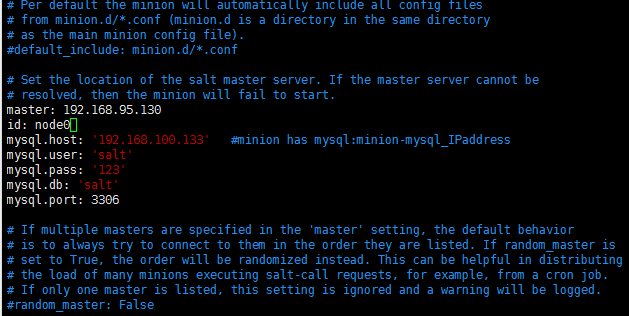
**以上可以扩展minion返回值，利用reactor来进行minion主动提取需求信息**

执行命令：salt -a pam ‘clientname’ “order”

###：这个可以对用户进行权限管理，这里的应用：因为一般情况一个salt管理者就可以完成全部主机的管理；但是如果某用户占用了大量主机，这时该用户需要对这些主机进行某些规模性的操作，就可以对该用户进行权限管制

提供一个可观测性可观察命令执行信息，将某一台minion端安装mysql，作为master对该minion和其他minion执行命令的信息反馈数据收集；或者是单独用一台主机来装mysql获取数据

Minion:



执行命令:salt '\*' test.ping --return mysql

以上是简单的命令操作，相对于线上来说，这些操作已经能满足信息拿取是否异常的要求，再加上日志。

**以下来说明推送过程：**

Salt的master推送过程，也有推送上限问题。当遇到一个大的文件需要部署时，推送给n台主机的等待时长是不确定的；这时采用ftp或者用nginx作为web服务进行大资源提供，在master端执行命令让minion去nginx和ftp端进行资源下载，如果大量的主机进行对资源的下载，还可以用lvs进行负载压力。

Salt的管理主要是minion端服务器的minion提供字典形式来进行便利管理

用一个nginx作为推送实验，推送给一台nginx当作web端，另一台当作nginx的proxy端

以下是编写代码：

vim /srv/salt/nginx/init.sls

include:

- nginx.install

- nginx.conf

- nginx.vhost

这是开始执行的默认文本，执行时命令为：salt ‘\*’ state.sls nginx 创建的sls文件应该是vim /srv/salt/nginx.sls 即执行命令的跟配执行文件可以是nginx.sls或者是nginx目录下的init.sls

推送及远程依赖包安装：

vim /srv/salt/nginx/install.sls

nginx\_source:

file.managed:

- name: /opt/nginx-1.9.15.tar.gz

- unless: test -e /tmp/nginx-1.9.15.tar.gz

- source: salt://nginx/files/nginx-1.9.15.tar.gz

#decompression

decompression\_nginx:

cmd.run:

- cwd: /opt

- names:

- tar zxvf nginx-1.9.15.tar.gz

- unless: test -d /opt/nginx-1.9.15

- require:

- file: nginx\_source

#nginx\_pkgs

nginx\_pkg:

pkg.installed:

- pkgs:

- gcc

- openssl-devel

- pcre

- pcre-devel

- zlib-devel

#nginx\_compile

nginx\_compile:

cmd.run:

- cwd: /opt/nginx-1.9.15

- names:

- ./configure --prefix=/usr/local/nginx --with-http\_ssl\_module --with-http\_stub\_status\_module --with-pcre;make;make install

- require:

- cmd: decompression\_nginx

- pkg: nginx\_pkg

- unless: test -d /usr/local/nginx

推送配置文本和启动文本

vim /srv/salt/nginx/conf.sls

include:

- nginx.install

nginx\_conf:

file.managed:

- name: /usr/local/nginx/conf/nginx.conf

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

- template: jinja

- defaults:

num\_cpus: {{grains['num\_cpus']}}

nginx\_service:

file.managed:

- name: /etc/init.d/nginx

- user: root

- mode: 755

- source: salt://nginx/files/nginx

service.running:

- name: nginx

- restart: True

- watch:

- file: /usr/local/nginx/conf/nginx.conf

推送至以pillar定义的字典对应到的主机

vim /srv/salt/nginx/vhost.sls

include:

- nginx.install

{% for vhostname in pillar['vhost'] %}

{% if vhostname['name'] == 'proxy' %}

{{vhostname['name']}}:

file.managed:

- name: {{vhostname['target']}}

- source: salt://nginx/files/vhost\_{{vhostname['name']}}.conf

- target: {{vhostname['target']}}

- template: jinja

- defaults:

server\_name: {{grains['ipv4'][1]}}

log\_name: {{vhostname['name']}}

- watch\_in:

service: nginx

{% else %}

{{vhostname['name']}}:

file.managed:

- name: {{vhostname['target']}}

- source: salt://nginx/files/vhost\_{{vhostname['name']}}.conf

- target: {{vhostname['target']}}

- template: jinja

- defaults:

server\_name: {{grains['ipv4'][1]}}

log\_name: {{vhostname['name']}}

- watch\_in:

service: nginx

{% endif %}

{% endfor %}

touch /srv/salt/nginx/files/{vhost\_proxy.conf, nginx, nginx.conf, vhost\_www.conf}

ls /srv/salt/nginx/files/nginx-1.9.15.tar.gz

vim srv/salt/nginx/files/nginx.conf

#user nobody

worker\_processes {{grains['num\_cpus']}};

error\_log logs/nginx\_error.log notice;

pid /var/run/nginx.pid;

worker\_rlimit\_nofile 65535;

events

{

use epoll;

worker\_connections 65535;

}

http

{

include mime.types;

default\_type application/octet-stream;

charset utf-8;

server\_names\_hash\_bucket\_size 128;

client\_header\_buffer\_size 32k;

large\_client\_header\_buffers 4 32k;

client\_max\_body\_size 128m;

sendfile on;

tcp\_nopush on;

keepalive\_timeout 60;

tcp\_nodelay on;

server\_tokens off;

client\_body\_buffer\_size 512k;

gzip on;

gzip\_min\_length 1k;

gzip\_buffers 4 16k;

gzip\_http\_version 1.1;

gzip\_comp\_level 2;

gzip\_types text/plain application/x-javascript text/css application/xml;

gzip\_vary on;

log\_format main '$remote\_addr - $remote\_user [$time\_local] "$request" '

'$status $body\_bytes\_sent "$http\_referer" '

'"$http\_user\_agent" "$http\_x\_forwarded\_for" "$host"' ;

include vhost\*.conf;

}

vim srv/salt/nginx/files/vhost\_proxy.conf

upstream pro\_server {

server 192.168.95.132:80 weight=1;

server 192.168.95.133:80 weight=1;

}

server {

listen 80;

server\_name {{ server\_name }};

access\_log logs/{{ log\_name }}-access.log main;

location / {

root /root;

index index.php index.html index.htm;

proxy\_pass http://pro\_server;

proxy\_redirect off;

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

client\_max\_body\_size 10m;

client\_body\_buffer\_size 128k;

proxy\_connect\_timeout 90;

proxy\_send\_timeout 90;

proxy\_read\_timeout 90;

proxy\_buffer\_size 4k;

proxy\_buffers 4 32k;

proxy\_busy\_buffers\_size 64k;

proxy\_temp\_file\_write\_size 64k;

}

}

vim srv/salt/nginx/files/vhost\_www.conf

server

{

listen 80;

server\_name {{ server\_name }};

index index.html index.htm ;

root html;

#location ~ .\*\.(php|php5)?$

# {

# try\_files $uri =404;

# fastcgi\_pass unix:/tmp/php-cgi.sock;

# fastcgi\_index index.php;

# include fcgi.conf;

# }

location /status {

stub\_status on;

}

location ~ .\*\.(gif|jpg|jpeg|png|bmp|swf)$

{

expires 30d;

}

location ~ .\*\.(js|css)?$

{

expires 1d;

}

access\_log logs/{{ log\_name }}-access.log main;

}

vim srv/salt/nginx/files/nginx

#!/usr/bin/env python

import sys

import os

from sys import argv

DAEMON='/usr/local/nginx/sbin/nginx'

CONFIGFILE='/usr/local/nginx/conf/nginx.conf'

PIDFILE='/var/run/nginx.pid'

NGSTART='%s -c %s' % (DAEMON,CONFIGFILE)

NGSTOP='killall nginx'

def be\_start():

if os.path.isfile(PIDFILE):

print "nginx is already running!"

else:

if os.system(NGSTART) == 0:

print "nginx start is ok!"

def be\_stop():

if os.path.isfile(PIDFILE):

if os.system(NGSTOP) == 0:

print "nginx stop is ok!"

else:

print "nginx is not running!"

def be\_restart():

be\_stop()

be\_start()

if argv == "-h" or argv == "--help" or len(argv)==1:

print "Usage: %s {start|stop|restart}" % argv[0]

elif argv[1] == 'start':

be\_start()

elif argv[1] == 'stop':

be\_stop()

elif argv[1] == 'restart':

be\_restart()

else:

print "Usage: %s {start|stop|restart}" % argv[0]

Pillar字典更新命令的首个执行文本

vim /srv/pillar/top.sls

base:

'\*':

- vhost

根据首个执行文本的信息找到了vhost执行设置pillar字典

vim /srv/pillar/vhost.sls

vhost:

{% if 'node5-65' in grains['id'] %}

- name: proxy

target: /usr/local/nginx/conf/vhost\_proxy.conf

{% elif 'node6-65' in grains['id'] %}

- name: proxy

target: /usr/local/nginx/conf/vhost\_proxy.conf

{% else %}

- name: www

target: /usr/local/nginx/conf/vhost\_www.conf

{% endif %}

以上是根据先用[pillar](https://docs.saltstack.com/en/latest/topics/pillar/)动态定义字典，在不同的主机上定义不同的字典，调用这些被定义的字典，然后推送文件进行部署，讲nginx的web和proxy进行分离部署

执行命令如下：

先更新pillar字典：salt '\*' saltutil.refresh\_pillar

再执行：salt ‘\*’ state.sls nginx

这样就可以将上述的nginx分离部署到不同服务器，访问nginx代理可以得到nginx的web信息

上述用到了[jinja](https://docs.saltstack.com/en/latest/ref/renderers/all/salt.renderers.jinja.html)模块进行变量设置渲染，可以灵活的调整配置文件的需求。