salt stack 运维工具

版本信息：

日期	版本	修改人	备注
2013-10-21	V1.0	戴儒锋	创建文档，初稿，不完善，仅作测试使用

一稿作者信息：
网名：江江　　交流邮箱：63780668@qq.com　　个人博客：http://www.linuxyw.com

二稿作者信息：
……

Saltstack 介绍

Saltstack是一个新的基础设施管理工具。目前处于快速发展阶段，可以看做是强化的Func+弱化的Puppet的组合。间接的反映出了saltstack的两大功能：远程执行和配置管理。Saltstack 使用 Python 开发的，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。

安装 salt 源

```
wget  http://dl.cpis-opt.com/huanw/shencan/epel-release-5-4.noarch.rpm && rpm -vih  epel-release-5-4.noarch.rpm
```

或

```
rpm -ivh http://mirrors.sohu.com/fedora-epel/6/x86_64/epel-release-6-8.noarch.rpm
```

服务端安装 salt-master

```
yum install salt-master -y
```

客户端安装 salt-minion

```
yum install salt-minion -y
```

启动服务：

```
服务端启动方式：service salt-master start
客户端启动方式：service salt-minion start
```

日志查看路径：（有问题可查日志获取出错信息）

服务端：/var/log/salt/master

客户端：/var/log/salt/minion

服务端 master 配置

（在以下配置中，需要注意的是，每个参数冒号后面都要带一个空格如： interface: 192.168.1.229）

默认情况下，salt master 在所有接口(0.0.0.0)上监听 4505 和 4506 两个端口. 如果想 bind 某个具体的 IP，需要对/etc/salt/master 配置文件中"interface"选项做如下修改:

```
interface: 192.168.1.229
```

注：192.168.1.229 是本机服务端的 IP 地址

修改 auto_accept 为 True，自动接受客户端的 KEY，当然也可以这里不设置，手动接受就行，接受方式：salt-key -a keyname （keyname 即为客户端刚才设置的 id 标识）

```
auto_accept: True
```

客户端 minion 配置

需要修改 minion 的配置文件/etc/salt/minion 中的 master 选项，进行如下操作:

```
master: 192.168.1.229
id :68
```

注：192.168.1.229 是服务端的 IP 地址

id :客户端的标识，用服务端连接时，就是用此标识来连接客户端，如：salt '68' cmd.run 'df -h'

重启以上服务生效

```
服务端启动方式：service salt-master restart
客户端启动方式：service salt-minion restart
```

Master 与 Minion 认证

1.minion 在第一次启动时，会在/etc/salt/pki/minion/（该路径在/etc/salt/minion 里面设置）下自动生成 minion.pem(private key)和 minion.pub(public key)，然后将 minion.pub 发送给 master。

2.master 在接收到 minion 的 public key 后，通过 salt-key 命令 accept minion public key，这样在 master 的/etc/salt/pki/master/minions 下的将会存放以 minion id 命名的 public key，然后 master 就能对 minion 发送指令了。

Master 与 Minion 的连接

Saltstack master 启动后默认监听 4505 和 4506 两个端口。4505(publish_port)为 salt 的消息发布系统，4506(ret_port)为 salt 客户端与服务端通信的端口。如果使用 lsof 查看 4505 端口，会发现所有的 Minion 在 4505 端口持续保持在 ESTABLISHED

```
[root@drfdai-17 salt]# lsof -i :4505
COMMAND     PID USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
salt-mast 10843 root   27u  IPv4  53124      0t0  TCP 192.168.1.229:4505 (LISTEN)
```

联系 Email：63780668@qq.com http://www.linuxyw.com

```
salt-mast   10843   root        29u      IPv4      53214                0t0      TCP
192.168.1.229:4505->192.168.1.68:12183 (ESTABLISHED)
salt-mast   10843   root        30u      IPv4      53215                0t0      TCP
192.168.1.229:4505->192.168.1.230:49306 (ESTABLISHED)
```

KEY 管理：

Salt 在 master 和 minion 数据交换过程中使用 AES 加密，为了保证发送给 minion 的指令不会被篡改，master 和 minion 之间认证采用信任的接受(trusted, accepted)的 key.

在发送命令到 minion 之前，minion 的 key 需要先被 master 所接受(accepted). 运行 salt-key 可以列出当前 key 的状态

```
[root@drfdai-17 src]#salt-key -L
Accepted Keys:
230
68
Unaccepted Keys:
Rejected Keys:
```

注：

Accepted Keys 为被服务端接受的 KEY（230，68 这二台客户端是被服务端接受了的 KEY,其实 230，68 就是 minion 中的 id 标识号）

Unaccepted Keys:未被服务端接受的 KEY

Rejected Keys：被服务端拒绝的 KEY

salt-key 命令可以接受特定的单个 key 或批量接受 key，使用-A 选项接受当前所有的 key,接受单个 key 可以使用-a keyname.

认证命令为salt-key，常用的有如下命令：

```
-a ACCEPT, --accept=ACCEPTAccept the following key
-A, --accept-all    Accept all pending keys
-r REJECT, --reject=REJECTReject the specified public key
-R, --reject-all    Reject all pending keys
-d DELETE, --delete=DELETEDelete the named key
-D, --delete-all    Delete all keys
```

发送指令：

master 和 minion 之间可以通过运行 test.ping 远程命令判断是否存活

```
[root@drfdai-17 src]# salt -E '230|68' test.ping
230:
      True
68:
      True
```

或者对所有 minion 进行：salt '*' test.ping

返回 True 说明测试是 OK 的，客户端是存活状态

执行命令：

```
salt '68' cmd.run 'df -h'
```

联系 Email：63780668@qq.com http://www.linuxyw.com

```
salt -E '230|68' cmd.run 'df -h'
```

注：把客户端 id 和发送的命令，用单引号括起来，养成习惯，防止出错

在服务端 salt 匹配 minion id

在运行 salt 命令进行匹配时，请使用单引号(')，避免 shell 解析

匹配所有 minion:salt '*' test.ping

匹配下边域的所有 minion:salt '*.example.*' test.ping

匹配 example.net 域中的(web1.example.net、web2.example.net.....webN.example.example.net):salt 'web?.example.net' test.ping

匹配 web1 到 web5 的 minion: salt 'web[1-5]' test.ping

匹配 web-x、web-y 及 web-z minion: salt 'web-[x-z]' test.ping

正则表达式

匹配 web-prod 和 web1-devel minion:

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

指定列表

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

指定组：

在服务务端中打开 master 配置文件

vim /etc/salt/master

添加如下分组

```
nodegroups:
  group1: 'L@230,68'
  group2: '68'
  group3: 'G@os:centos'
  group4: 'G@mem:487'
```

值得注意的是编辑 master 的时候，group1 和 group2 前面是 2 个空格

测试：

```
[root@drfdai-17 salt]#salt -N group2 test.ping
68:
    True
[root@drfdai-17 salt]# salt -N group1 test.ping
230:
    True
68:
True
```

可能大家会好奇 group1 中为什么会有 L@，这代表什么意思？

其实 L 是指客户端列表，我们一组中有多个客户端，所以在前面用 L 表示。

除了有列表匹配外，还有很多匹配方式，如：

联系 Email：63780668@qq.com http://www.linuxyw.com

Letter	含义	例子
G	Grains glob匹配	G@os:Ubuntu
E	PCRE Minion id匹配	E@web\d+\.(dev|qa|prod)\.loc
P	Grains PCRE匹配	P@os:(RedHat|Fedora|CentOS)
L	minions列表	L@minion1.example.com,minion3.domain.com or bl*.domain.com
I	Pillar glob匹配	I@pdata:foobar
S	子网/IP地址匹配	S@192.168.1.0/24 or S@192.168.1.100
R	Range cluster匹配	R@%foo.bar
D	Minion Data匹配	D@key:value

匹配中可以使用*and*、*or*及*not*等boolean型操作

这些参数都可以直接在命令行使用,如：

```
salt -S '192.168.1.230' test.ping
salt -G 'os:Centos' test.ping
salt -L '230,68' test.ping
```

minion基本信息的管理

基本使用:

```
salt '*' grains.ls  查看 grains 分类
salt '*' grains.items 查看 grains 所有信息
salt '*' grains.item osrelease 查看 grains 某个信息
```

如：

```
[root@drfdai-17 salt]# salt '*' grains.item osrelease
230:
  osrelease: 6.2
68:
  osrelease: 6.2
```

Salt 命令介绍

salt	Salt 主命令，比如执行命令模块
satl-cp	复制文件到指定的系统上去
salt-key	和 Minion 之间进行身份验证
salt-master	Master 主守护进程，用于控制 Minion
salt-run	前端命令执行
salt-syndic	Salt syndic 守护进程，用于多级 salt-master 使用
备注：具体命令的详细内容，可以查看 man 手册。	

cmd.run

Saltstack 可以远程执行 shell 命令，使用 cmd.run。如：

```
salt '68' cmd.run 'df -h'
```

cmd.script

```
salt '*' cmd.script salt://iptables.sh
```

pkg.install

```
salt '*' pkg.install vim
```

network.interfaces

```
salt '*' network.interfaces
```

更多的 salt 例子可以使用以下命令查看：

```
salt '*' sys.doc | grep "salt '*'"
```

内置执行模块

官方模块地址：http://docs.saltstack.com/ref/modules/all/index.html

salt stack 配置管理

Salt使用State模块文件进行配置管理，使用YAML编写，以.sls结尾。如果进行配置管理首先需要再Master的配置文件中指定"file roots"的选项，Salt支持环境的配置，比如测试环境和生产环境但是base环境是必须的。而且Base环境必须包含入口文件top.sls。

Saltstack 配置管理流程

配置软件安装

告诉 Salt 你的配置管理文件在哪里。根据你是如何安装 Salt，有时你需要自己创建

/srv/salt 目录，服务端 master 配置文件中，默认是：

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

创建一个 top.sls 文件，这个也是入口文件，也就是说，你执行相关命令的时候，会先检测这个文件，这文件提供了其它文件的映射，可以用于作为其它服务器的基础配置文件。
vim /srv/salt/top.sls

```
base:
  '68':
    - apache.apache
  '230':
    - fc7.pack
```

base 语法告诉 Salt 这是基础配置文件，
'68',是指应用在 68 客户机
- apache.apache 是指相关配置在 apache 目录下面的 apache.sls,下面再解释 apache.sls
'230':是指应用在 230 客户机
- fc7.pack：是指相关配置在 fc7 目录下的 pack.sls，下面解释 pack.sls

这些目录和文件名，都是随你怎么命名，以下是我/srv/salt/下面的文件：

```
[root@drfdai-17 salt]# ll
total 16
drwxr-xr-x. 2 root root 4096 Oct 21 17:41 apache
drwxr-xr-x. 2 root root 4096 Oct 21 16:21 fc7
drwxr-xr-x. 2 root root 4096 Oct 22 17:07 _modules
-rw-r--r--. 1 root root   58 Oct 22 17:11 top.sls
```

```
[root@drfdai-17 salt]# tree
.
├── apache
│   ├── apache.sls
│   └── httpd.conf
├── fc7
│   ├── httpd.conf
│   └── pack.sls
├── _modules
│   ├── aa.py
│   └── dl.py
└── top.sls
```

联系 Email：63780668@qq.com http://www.linuxyw.com

先看看指定 68 客户机的配置文件内容 apache.sls（注意每行的缩进和空格，缩进为 2 个空格，冒号后面 1 个空格）

```
apache:
  service:
    - name: httpd
    - running
    - reload: True
    - watch:
      - file: /usr/local/apache/conf/httpd.conf

/usr/local/apache/conf/httpd.conf:
  file:
    - managed
    - source: salt://apache/httpd.conf
    - backup: minion
```

如图所示：

第一行：告诉管理工具，这是一个 id 说明，如我这是管理 apache 配置的，我就写 apache，一看就知道是什么了

第二行：告诉管理工具，这是服务管理（负责管理服务脚本的启用，禁用，启动，停止，重启等等工作，service 下面是运行的方法，方法定义包和服务应该怎么做）

第三行：告诉管理工具，这是 httpd 服务

第四行：告诉管理工具，这个服务需保证是运行状态

第五行：告诉管理工具，这个服务需要自动会重启，这个跟下面 watch 有关，当 watch 中的文件被改动时，就重启服务。

联系 Email：63780668@qq.com http://www.linuxyw.com

第六行：观察下面 file 文件改动，有改动就触发上面的 reload 重启
第七行：指定文件的位置。
第九行：这是文件标识
第十，十一行：文件管理
第十二行：指定源数据在哪里，即服务端的源，当这个文件被改动时，就会触发包管理工具，更改上面 file 指定的文件，可以看成是同步更新吧。
第十三行：告诉包管理工具，当文件更新时，在更新前备份一次，如下图：

```
[root@drfdai-15 conf]# pwd
/var/cache/salt/minion/file_backup/usr/local/apache/conf
[root@drfdai-15 conf]# ll
total 12
-rw-r--r-- 1 root root 3137 Oct 21 17:07 httpd.conf_Mon_Oct_21_17:07:40_897585_2013
-rw-r--r-- 1 root root 3142 Oct 21 17:25 httpd.conf_Mon_Oct_21_17:25:21_392572_2013
-rw-r--r-- 1 root root 3149 Oct 21 17:33 httpd.conf_Mon_Oct_21_17:33:01_406831_2013
```

写好好这些文件后，把 68 客户机上的 httpd.conf 复制一份到服务端，放在/srv/salt/apache/目录下，如我刚才发的 ls 图

```
├──apache
│   ├──apache.sls
│   └──httpd.conf
```

然后，在服务端执行：

```
salt '68' state.highstate
```

这样就可进行对 68 客户端推送文件了，以下是测试结果：

```
68:
----------
    State: - file
    Name:      /usr/local/apache/conf/httpd.conf
    Function:  managed
        Result:    True
        Comment:   File /usr/local/apache/conf/httpd.conf is in the correct state
        Changes:
----------
    State: - service
    Name:      httpd
    Function:  running
        Result:    True
        Comment:   Started Service httpd
        Changes:   httpd: True
```

这里我先发客户端的配置文件最后几行吧，一会我往最后一行添加点数据再测试看：

```
[root@drfdai-15 conf]# tail -6 /usr/local/apache/conf/httpd.conf
<Directory "/usr/local/awstats/wwwroot">
    Options None
    AllowOverride All
    Order allow,deny
    Allow from all
</Directory>
```

现在往服务端的/srv/salt/apache/httpd.conf 最后添加一行：#http://www.linuxyw.com

```
<Directory "/usr/local/awstats/wwwroot">
    Options None
    AllowOverride All
    Order allow,deny
    Allow from all
</Directory>
#http://www.linuxyw.com
```

现在进行推送测试：

```
salt '68' state.highstate
```

```
[root@drfdai-17 apache]# salt '68' state.highstate
68:
----------
    State: - file
    Name:      /usr/local/apache/conf/httpd.conf
    Function:  managed
        Result:    True
        Comment:   File /usr/local/apache/conf/httpd.conf updated
        Changes:   diff: ---
+++
@@ -104,3 +104,4 @@
     Order allow,deny
     Allow from all
 </Directory>
+#http://www.linuxyw.com


----------
    State: - service
    Name:      httpd
    Function:  running
        Result:    True
        Comment:   Started Service httpd
        Changes:   httpd: True
```

再打开 68 客户端上的配置文件看看：

```
[root@drfdai-15 conf]# tail -6 /usr/local/apache/conf/httpd.conf
    Options None
    AllowOverride All
    Order allow,deny
    Allow from all
</Directory>
#http://www.linuxyw.com
```

结果出来了，这只是简单的一种用法，更多的用法，再研究。
上面这种方法，是我 68 客户机上原本就安装好了的环境，用编译安装的 apache，所以，我就只采取了配置文件管理，而不对其进行包安装，下面就会讲到用 rpm 包进行包管理（当然，如果要编译安装怎么办？这个，也是可以的，下回再研究这个。）

回到 top.sls 文件

```
base:
  '68':
    - apache.apache
  '230':
    - fc7.pack
```

看看 fc7 目录下的 pack.sls 文件内容：

```
apache:
  pkg:
    - name: httpd
    - installed
  service:
    - name: httpd
    - running
    - reload: True
    - watch:
      - file: /etc/httpd/conf/httpd.conf

/etc/httpd/conf/httpd.conf:
  file.managed:
    - source: salt://fc7/httpd.conf
    - user: root
    - group: root
    - mode: 644
    - backup: minion
```

如下图：

```
[root@drfdai-17 fc7]# vim pack.sls

 1 apache:
 2   pkg:
 3     - name: httpd
 4     - installed
 5   service:
 6     - name: httpd
 7     - running
 8     - reload: True
 9     - watch:
10       - file: /etc/httpd/conf/httpd.conf
11
12 /etc/httpd/conf/httpd.conf:
13   file.managed:
14     - source: salt://fc7/httpd.conf
15     - user: root
16     - group: root
17     - mode: 644
18     - backup: minion
```

这个图和上面 apache 的有点相似吧，相似的不说了，说下新添加的吧

第二行：安装包管理（pkg 负责包的安装与卸载）

第三行：需要安装的软件名称

第四行：这个软件需要被安装，如果没安装就会执行 yum 安装，当然如果系统不一样，这里可能就要判断了，我这里是针对 Centos 做的实验

其它行，大概都是和上面差不多了，这里就不解释了。

如果需要把服务设置为开机启动可以在 service 中添加使用 - enable:True

现在进行测试吧：

在 230 客户机上检查是否安装 httpd，结果是没有安装的。

```
[root@drfdai-16 www]# rpm -qa httpd
[root@drfdai-16 www]#
```

现在我们在服务端进行推送管理：

```
salt '230' state.highstate
```

服务端信息：

```
[root@drfdai-17 fc7]# salt '230' state.highstate
230:
----------
    State: - file
    Name:      /etc/httpd/conf/httpd.conf
    Function:  managed
        Result:    True
        Comment:   File /etc/httpd/conf/httpd.conf updated
        Changes:   diff: New file

----------
    State: - pkg
    Name:      httpd
    Function:  installed
        Result:    True
        Comment:   The following packages were installed/updated: httpd.
        Changes:   httpd: { new : 2.2.15-29.el6.centos
old :
}

----------
    State: - service
    Name:      httpd
    Function:  running
        Result:    True
        Comment:   Started Service httpd
        Changes:   httpd: True

[root@drfdai-17 fc7]#
```

客户端安装信息：

```
Loaded plugins: aliases, changelog, downloadonly, fastestmirror, presto, security,
              : tmprepo, verify, versionlock
Setting up Remove Process
Resolving Dependencies
--> Running transaction check
---> Package httpd.x86_64 0:2.2.15-29.el6.centos will be erased
--> Finished Dependency Resolution
base                                                      | 3.7 kB     00:00
epel/metalink                                             | 6.3 kB     00:00
epel                                                      | 4.2 kB     00:00
epel/primary_db                                           | 5.6 MB     00:09
extras                                                    | 3.4 kB     00:00
updates                                                   | 3.4 kB     00:00

Dependencies Resolved

================================================================================
 Package        Arch           Version                      Repository       Size
================================================================================
Removing:
 httpd          x86_64         2.2.15-29.el6.centos         @updates         2.9 M

Transaction Summary
================================================================================
Remove        1 Package(s)

Installed size: 2.9 M
Is this ok [y/N]: y
Downloading Packages:
Running rpm_check_debug
Running Transaction Test
Transaction Test Succeeded
Running Transaction
```

在 230 客户端上再检查 httpd：

```
[root@drfdai-16 www]# rpm -qa httpd
httpd-2.2.15-29.el6.centos.x86_64
[root@drfdai-16 www]#
```

这回显式被安装了吧，我们在服务端中要求 httpd 是必须要运行的，那再看下是否运行了呢？

```
[root@drfdai-16 www]# lsof -i :80
COMMAND  PID   USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
httpd   3001   root    4u  IPv6  34053      0t0  TCP *:http (LISTEN)
httpd   3003 apache    4u  IPv6  34053      0t0  TCP *:http (LISTEN)
httpd   3004 apache    4u  IPv6  34053      0t0  TCP *:http (LISTEN)
httpd   3005 apache    4u  IPv6  34053      0t0  TCP *:http (LISTEN)
httpd   3006 apache    4u  IPv6  34053      0t0  TCP *:http (LISTEN)
httpd   3007 apache    4u  IPv6  34053      0t0  TCP *:http (LISTEN)
httpd   3008 apache    4u  IPv6  34053      0t0  TCP *:http (LISTEN)
httpd   3009 apache    4u  IPv6  34053      0t0  TCP *:http (LISTEN)
httpd   3010 apache    4u  IPv6  34053      0t0  TCP *:http (LISTEN)
[root@drfdai-16 www]#
```

也是在等待中了，下面再测试配置文件的推送，把 230 客户端中的/etc/httpd/conf/httpd.conf 复制一份到服务端中的/srv/salt/fc7/下面，如

```
[root@drfdai-17 fc7]# tree /srv/salt/fc7/
/srv/salt/fc7/
├──httpd.conf
└──pack.sls
```

我们往/srv/salt/fc7/httpd.conf 最后添加一行：#http://www.linuxyw.com，再推送下试试：

```
#<VirtualHost *:80>
#    ServerAdmin webmaster@dummy-host.example.com
#    DocumentRoot /www/docs/dummy-host.example.com
#    ServerName dummy-host.example.com
#    ErrorLog logs/dummy-host.example.com-error_log
#    CustomLog logs/dummy-host.example.com-access_log common
#</VirtualHost>
#http://www.linuxyw.com
```

推送：

```
[root@drfdai-17 fc7]# salt '230' state.highstate
230:
----------
    State: - file
    Name:      /etc/httpd/conf/httpd.conf
    Function:  managed
        Result:    True
        Comment:   File /etc/httpd/conf/httpd.conf updated
        Changes:   diff: ---
+++
@@ -1007,3 +1007,4 @@
 #    ErrorLog logs/dummy-host.example.com-error_log
 #    CustomLog logs/dummy-host.example.com-access_log common
 #</VirtualHost>
+#http://www.linuxyw.com


----------
    State: - pkg
    Name:      httpd
    Function:  installed
        Result:    True
        Comment:   Package httpd is already installed
        Changes:
----------
    State: - service
    Name:      httpd
    Function:  running
        Result:    True
        Comment:   Service reloaded
        Changes:   httpd: True
```

查看 230 客户机的/etc/httpd/conf/httpd.conf 文件是否也添加了最后一行

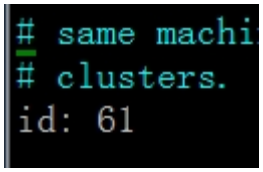
```
[root@drfdai-16 www]# tail -6 /etc/httpd/conf/httpd.conf
#    DocumentRoot /www/docs/dummy-host.example.com
#    ServerName dummy-host.example.com
#    ErrorLog logs/dummy-host.example.com-error_log
#    CustomLog logs/dummy-host.example.com-access_log common
#</VirtualHost>
#http://www.linuxyw.com
[root@drfdai-16 www]#
```

更多的请自己测试了，如重启是否能在文件改动后自动重启呢？我自己测试的时候是可以的，这里就不细说了。

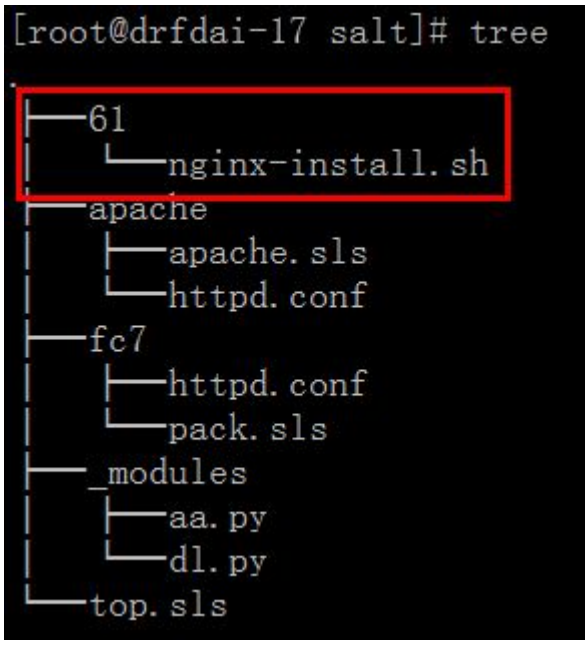
联系 Email：63780668@qq.com http://www.linuxyw.com

salt 管理客户机脚本安装环境

往服务端写一个安装 nginx 的脚本，并把脚本放在/srv/salt/61 目录下，61 指的是新的客户端，这个客户端我在 minion 配置文件中标了 id: 61。



nginx 安装脚本



现在写 top.sls 文件：

```
base:
  '68':
    - apache.apache
  '230':
    - fc7.pack
  '61':
    - 61.nginx
```

如下图：

```
[root@drfdai-17 salt]# vim /srv/salt/top.sls

base:
  '68':
    - apache.apache
  '230':
    - fc7.pack
  '61':
    - 61.nginx
```

在 61 目录中写 nginx.sls 文件，内容如下：

vim /srv/salt/61/nginx.sls

```
nginx_install:
  cmd.script:
    - source: salt://61/nginx-install.sh
    - user: root
    - shell: /bin/bash
```

```
[root@drfdai-17 61]# vim nginx.sls

nginx_install:
  cmd.script:
    - source: salt://61/nginx-install.sh
    - user: root
    - shell: /bin/bash
```

推送并进行安装：

```
salt '61' state.sls 61.nginx
```

检查 61 客户端是否安装成功 nginx：（下图是成功安装的）

```
[root@drfdai-1 ~]# lsof -i :80
COMMAND   PID USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
nginx   18122 root    6u  IPv4  38478      0t0  TCP *:http (LISTEN)
nginx   18124  www    6u  IPv4  38478      0t0  TCP *:http (LISTEN)
nginx   18125  www    6u  IPv4  38478      0t0  TCP *:http (LISTEN)
nginx   18126  www    6u  IPv4  38478      0t0  TCP *:http (LISTEN)
nginx   18127  www    6u  IPv4  38478      0t0  TCP *:http (LISTEN)
```

Salt 编写自定模块：

官网文档：http://docs.saltstack.com/ref/modules/index.html#grains-data

Master 上创建存放模块的目录：

```
mkdir -pv /srv/salt/_modules
cd /srv/salt/_modules
```

编写一个简单的模块 xzr.py：

```
[root@host109 _modules]# cat xzr.py
#coding:utf-8
import random
def test():
        '''随机一个 1 到 100 的数为双数就返回 True'''
        return random.randint(1,100)%2==0
def  echo(text):
        return text
def myscript(*t,**kv):
        '''引用 salt 本身的模块'''
        ret = __salt__['cmd.script'](*t,**kv)
        return ret
```

同步 master 上的自定模块到 minion 上：

同步前最好在本机运行一下，看有没错误

```
[root@host109 _modules]# salt '*'  saltutil.sync_modules
host100:
    - modules.xzr
host101:
    - modules.xzr
```

运行自定义模块：

```
[root@host109 _modules]# salt '*' xzr.test
host101:
    False
host100:
    True
```

修改一下模块里的 test 函数让他输出为字符串：

```
#coding:utf-8
import random
#定义输出格式，不定义的话默认为字典
__outputter__ = {
                'test': 'txt'
                }
```

联系 Email：63780668@qq.com http://www.linuxyw.com

```
def test():
        '''随机一个 1 到 100 的数为双数就返回 True'''
        return random.randint(1,100)%2==0
def  echo(text):
        return text
def myscript(*t,**kv):
        '''引用 salt 本身的模块'''
        ret = __salt__['cmd.script'](*t,**kv)
        return ret
```

修改后再次运行：

```
[root@host109 _modules]# salt '*' xzr.test
host101: False
host100: False
```

在这里查看更多模块信息：

https://github.com/saltstack/salt/tree/master/salt/modules

编写自定返回处理 Returners：

Returners 可以让客户端把模块执行的返回结果在本机额外处理。

官方文档：

http://docs.saltstack.com/ref/returners/index.html

和自定模块一样在/srv 下新建存放 Returners 的目录_

```
mkdir -pv /srv/salt/_returners
cd  /srv/salt/_returners
```

编写一个返回处理，返回处理的函数名必须为 returner：

```
[root@host109 _returners]# cat my_returners.py
#coding:utf-8

def __virtual__():
        '''调用时的名字'''
        return 'writefile'

def returner(ret):
        '''简单的把返回结果写到文件里'''
        f = open('/tmp/salt_return','a+')
        f.write(str(ret))
        #f.write(str(ret['return']))
        f.close()
```

同步 returners

```
[root@host109 _returners]# salt '*'  saltutil.sync_returners
host101:
    - returners.my_returners
host100:
    - returners.my_returners
```

执行模块时加入上—return 使用返回处理，可使用多个用逗号分开。

```
[root@host109 _returners]# salt '*'  xzr.test --return writefile
host101: False
host100: True
```

去客户端那边看下，写入的是一个字典：

```
[root@host100 ~]# cat /tmp/salt_return
{'jid': '20131024040453764597', 'return': True, 'retcode': 0, 'success': True,
'fun': 'xzr.test', 'id': 'host100'}
```

键 return 就是我们的模块的返回结果。

再看下别的机器：

```
[root@host109 _returners]# salt '*'  cmd.run 'cat /tmp/salt_return'
host100:
    {'jid': '20131024040453764597', 'return': True, 'retcode': 0, 'success': True,
'fun': 'xzr.test', 'id': 'host100'}
host101:
    {'jid': '20131024040453764597', 'return': False, 'retcode': 0, 'success':
True, 'fun': 'xzr.test', 'id': 'host101'}
```

官方的 returners 参考：
https://github.com/saltstack/salt/tree/develop/salt/returners

小结：

使用 salt 的自定模块和返回处理，只要写一个监控主机的状态的模块（salt 自带很多获取主机信息的模块），再把模块的返回结果入数据库，这样就可以轻松实现分布式的监控。

参考资料：
http://wiki.saltstack.cn/docs
http://www.shencan.net/