Saltsatck

目录

[简述 1](#_Toc507765223)

[salt的基本使用 2](#_Toc507765224)

[SLS文件 5](#_Toc507765225)

[参考 9](#_Toc507765226)

[salt-minion配置文件 详解 13](#_Toc507765227)

[salt-master配置文件详解 16](#_Toc507765228)

[state tree 多版本控制： 21](#_Toc507765229)

[grains和pillar 22](#_Toc507765230)

[salt 认证过程 24](#_Toc507765231)

[Salt WebUI 25](#_Toc507765232)

[salt 扩展认证 PAM 26](#_Toc507765233)

[salt syndic 概述 28](#_Toc507765234)

[salt gitfs 30](#_Toc507765235)

[saltstack同步文件 32](#_Toc507765236)

# 简述

Salt 和 Puppet Chef 一样可以让你同时在多台服务器上执行命令也包括安装和配置软件。Salt 有两个主要的功能：配置管理和远程执行。

**安装**

在centos系统上，通过yum安装需先安装epel源

yum -y install epel\*

yum -y install salt-master    #安装服务器端

yum -y install salt-minion    #安装客户端

**基本配置**

saltstack的配置文件格式

salt默认使用PyAMI语法作为它的模板文件格式，其他很多模板语言在salt中可以使用的。一定要按照正确的格式书写YAML，比如最基本的，它使用到两个空格代替tab， : 或 -后面要有空格

**服务器端配置:**

通过yum安装后配置文件默认在/etc/salt/目录下

修改配置本机中如下对应的值

interface: 0.0.0.0                  # interface (服务端监听IP),0.0.0.0 是监听所有IP

log\_file: /var/log/salt/master      # 记录主控端运行日志

key\_logfile: /var/log/salt/key      # 记录认证证书日志

master默认监听两个端口， 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,所以确保客户端能跟服务端的这2个端口通信

**客户端配置:**

master: 42.121.124.237          # 设置主控端IP

id: ubuntu-server-001           # 设定受控端编号

log\_file: /var/log/salt/minion  # 记录受控端服务日志

key\_logfile: /var/log/salt/key  # 记录认证证书日志

**最基本字段:**

master: 服务端，

主机名 id: 客户端主机名(在服务端看到的客户端的名字，可以不设置，默认与hostname相同)

小技巧：查看配置文件信息，过滤注释语句

sed -e '/^#/d;/^$/d' /etc/salt/minion

**检查服务**

主控端，和受控端 启动各自的服务，确保服务启动后没有任何报错信息,如果异常请检查相应日志文件处理

主控端: service salt-master restart

受控端: service salt-minion restart

**简单的测试**

你可以从master 使用一个内置命令 test.ping 来测试他们之间的连接

salt '\*' cmd.run test.ping

**证书管理**

如果一切顺利，请继续！

saltstack 主控端是依靠openssl证书来与受控端主机认证通讯的，受控端启动后会发送给主控端一个公钥证书文件，在主控端用 salt-key 命令来管理证书。

**salt minion和master的认证过程：**

minion在第一次启动时，会在/etc/salt/pki/minion/下自动生成minion.pem(private key), minion.pub(public key)，然后将minion.pub发送给master

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

salt-key -L               # 用来查看证书情况

salt-key -a               # 用来管理接受证书

受控端证书认证后会显示如下情形：

Accepted Keys:

ubuntu-server-001

Unaccepted Keys:

Rejected Keys:

**简单的测试**

你可以从master 使用一个内置命令 test.ping 来测试他们之间的连接

salt '\*' cmd.run test.ping

它应该有如下输出:

ubuntu-server-001:

  True

测试与外网的连接

salt '\*' cmd.run "ping -c 4 baidu.com"

salt更多命令及手册

salt '\*' sys.doc

# salt的基本使用

**salt按照操作模式可以分为实时管理和配置管理：**

远程执行，也可以称为实时管理：通过执行命令来完成管理任务，优点就是实时性高，缺点就是无法跟踪，保存配置状态；

配置管理，也可以称为状态管理：通过编写特定的文件用来记录目标系统的配置状态，然后将这些受控主机会定期同步这些配置状态信息；

**实时管理**

有时候我们需要临时的查看一台或多台机器上的某个文件，或者执行某个命令，最通用的模块是 cmd.run

基本操作命令通用格式

格式: 命令 对象 执行模块 参数

示例：salt '\*' cmd.run "ping -c 4 baidu.com"

'\*' target,操作对象，可以使用salt命令的扩展模式 -E -G 等

cmd.run func， 执行模块，函数

参数 args， 传递给 执行模块的参数,

salt也将一些常用的命令做了集成,比如,查看所有节点磁盘使用情况:

salt '\*' disk.usage

其他命令示例：

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

salt '\*' pillar.items   查看pillar所有信息

nodegroup

作用：对minion进行分组

参考文档: <http://docs.saltstack.com/topics/targeting/nodegroups.html>

编辑 /etc/salt/master 启用分组，最基本分组示例：

nodegroups:

  UBUNTU: 'ubuntu-12.04-\*'

  CENTOS: 'centos-6.4-\*'

建立分组之后,操作对象使用分组功能才生效，更多示例：

nodegroups:

  group1: '[L@foo.domain.com](mailto:L@foo.domain.com),bar.domain.com,baz.domain.com or bl\*.domain.com'

  group2: '[G@os](mailto:G@os):Debian and foo.domain.com'

在state或者pillar中引用的时候,如下:

base:

  group1:

    - match: nodegroup

    - webserver

请确保这条属性存在

- match: nodegroup

更多请参考文档：<http://docs.saltstack.com/topics/targeting/nodegroups.html>

targeting

从最基本的测试命令来讲述：

salt '\*' test.ping

引号中以实现很强大的minion的过滤与匹配技术

常用命令:

salt 'shell正则' 命令

salt -E 'prel 正则' 命令

salt -N  GroupName  命令

salt -L 'server\_id1,server\_id2,server\_id3' 命令

示例:

salt -C 'webserv\* and [G@os](mailto:G@os):Debian or [E@web-dc1-srv.](mailto:E@web-dc1-srv.)\*' test.ping

**配置管理，或称状态管理**

个人理解，管理一个服务器应用可以从 软件包,配置文件,服务管理这个三个最基本角度来出发，要启用配置管理，首先应对受控端进行额外的配置

配置受控端

配置受控端主机,以root用户身份来接受主控端主机的管理

编辑/etc/salt/minion

user: root

配置主控端，开启配置管理功能

Salt的配置管理指令和文件保存在默认在/srv/salt目录下，这里存放着所有的配置文件，和一些你想要拷贝到从服务器的文件。

配置主控端，编辑 /etc/salt/master 文件

file\_roots:

  base:

    - /srv/salt

手动创建目录和配置文件 /srv/salt/top.sls

/srv/salt/top.sls 为默认入口配置文件，这个配置项可以自定义，基本配置如下：

base:

  '\*':

    - ubuntu.vim

详细解释一下这个本配置文件的参数

base: 默认的的起点配置项：

'\*': 这个引号内的是匹配对象，针对所有受控主机

ubuntu.vim 就是指资源文件/srv/salt/ubuntu/vim.sls

一个简单的例子：ubuntu 基本系统默是不安装 vim 我们可以利用配置管理把被托管的ubuntu主机全部安装上vim

编辑 /srv/salt/ubuntu/vim.sls

vim:

  pkg:

    - name: vim

    - installed

salt '\*' state.highstate

请注意观察返回结果，查看/var/log/salt/下面的日志来调试saltstack配置。

稍微复杂的例子：管理ssh服务

一个更复杂的例子：管理ssh服务，并且使用salt托管配置文件

/srv/salt/top.sls 内容：

base:

  '\*':

    - ubuntu.ssh

/srv/salt/ubuntu/ssh.sls

ssh:

  pkg:

    - name: ssh

    - installed

  service:

    - name: ssh

    - running

    - reload: True

    - watch:

      - file: /etc/ssh/ssh\_config

/etc/ssh/ssh\_config:

  file.managed:

    - source: salt://ubuntu/ssh\_config

    - user: root

    - group: root

    - mode: 644

简要解释一下配置文件

pkg, service , file 这些都是salt的管理模块,pkg 是包管理模块; file是文件管理模块; service 是包服务管理模块

模块下一级是各个管理模块配置项的属性，以 service: 模块为例

name: ssh ubuntu下的服务脚本名称是 ssh

running 状态是持续运行，如果受控端配置了自动同步，每格一段时间就会对其状态进行检查

reload: True 是否重载服务

watch: 监视文件

最后两条属性的整体含义是如果配置文件 /etc/ssh/ssh\_config 发生变化，服务重启更新

source: salt://ubuntu/ssh\_config 托管的配置文件实际存储在 /srv/salt/ubuntu/ssh\_config

同样，使用如下命令来验证结果需要

salt '\*' state.highstate

# SLS文件

SLS（代表Salt State文件）是Salt State系统的核心,SLS描述了系统的目标状态，由格式简单的数据构成。这经常被称作配置管理

top.sls

  top.sls是配置管理的入口文件，一切都是从这里开始，在master主机上，默认存放在/srv/salt/目录

  top.sls默认从base标签开始解析执行，下一级是操作的目标，可以通过正则，grain模块，或分组名，来进行匹配，在下一级是要执行的state文件，不包括扩展名

创建/srv/salt/top.sls

1、通过正则进行批评的示例

base:

  '\*':

    - webserver

2、    通过分组名进行匹配的示例，必须要有 -match: nodegroup

base:

  group1:

    -match: nodegroup

    -webserver

 3、通过grain模块匹配的示例，必须要有 -match: grain

base:

  'os:centos':

    - match: grain

    - webserver

准备好top.sls后，编写一个state文件/srv/salt/webserver.sls

apache:                #标签定义

  pkg:                #state declartion

    - installed        #fuction declaration

第一行被称为(ID declaration)标签定义，在这里被定义为安装包的名。注意:在不同发行版软件包命名不同，比如centos中叫httpd的包，Debian/Ubuntu中叫apache2

第二行被称为(state declaration)状态定义，在这里定义使用(pkg state module)

第三行被称为(function declaration)函数定义，在这里定义使用(pkg state module)调用installed函数

最后在终端中执行命令查看结果

salt '\*' state.highstate

或附件test=True参数 测试执行

salt '\*' state.highstate -v test=True

SLS文件命名空间

注意在以上的例子中,SLS文件 webserver.sls 被简称为webserver. SLS文件命名空间有如下几条基本的规则：

    SLS文件的扩展名 .sls 被省略。 (例如. webserver.sls 变成 webserver)

    子目录可以更好的组织,每个子目录都由一个点来表示.(例如 webserver/dev.sls 可以简称为 webserver.dev）

    如果子目录创建一个init.sls的文件，引用的时候仅指定该目录即可. (例如 webserver/init.sls 可以简称为 webserver）

    如果一个目录下同时存在webserver.sls 和 webserver/init.sls，那么 webserver/init.sls 将被忽略，SLS文件引用的webserver将只引用webserver.sls

state多文件示例

下面是一个state多文件示例，

apache/init.sls

apache/httpd.conf

ssh/init.sls

ssh/server.sls

ssh/banner

ssh/ssh\_config

ssh/sshd\_config

创建一个引用这些目录的server.sls

server:

  - apache

  - ssh

state的层级关系

include示例：

/srv/salt/apache.sls

    apache:

      pkg:

        - installed

      service:

        - running

        - require:

          - pkg: apache

收include可以包换索引state文件而不必重新写

/srv/salt/apache-custom.sls

    include:

      apache

extend示例：

extend和include配合使用，作用是修改或扩展引用的state文件的某个字段

/srv/salt/apache.sls

    apache:

      pkg:

        - installed

      service:

        - running

        - require:

          - pkg: apache

extend默认是替换引用文件的某个字段属性

/srv/salt/apache-change.sls

    include:

      apache

    extend:

      apache

        service

          -watch:

            -file: /etc/redis.conf

state的逻辑关系列表

    match:匹配某个模块，比如match: grain match:nodegroup

    require:依赖某个state，在运行此state之前，先运行依赖的state，依赖可以多个

    watch:在某个state变化时运行此模块

    order:优先级比require和watch低，有order指定的state比没有order指定的优先级高

state的逻辑关系实例

require:依赖某个state，在运行state前，先运行依赖的state，依赖可以有多个

httpd:                                  # maps to "name"

  pkg:

    - installed

  file：                                # maps to State module filename

    - managed:                          # maps to the managed function in the file State module

    - name: /etc/httpd/conf/httpd.conf  # one of many options passed to the manage function

    - source: salt://httpd/httpd.conf

    - require:

      - pkg: httpd

watch:在某个stae变化时运行此模块，watch除了具备require功能外，还增加了关注状态的功能

redis:

  pkg:

    - latest

  file.managed:

    - source: salt://redis/redis.conf

    - name: /etc/redis.conf

    - require:

      - pkg: redis

  service.running:

    - enable: True

    - watch:

      - file: /etc/redis.conf

      - pkg: redis

order:优先级比require和watch低，有order指定的state比没有order指定的优先级高

vim:

  pkg.installed:

    - order: 1

想让某个state最后一个运行，可以用last

使用模板来精简SLS，使用SLS可以使用python的循环、分支、判断等逻辑

{% for item in ['tmp','test'] %}

/opt/{{ item }}:

  file.directory:

    - user: root

    - group: root

    - mode: 755

    - makedirs: True

{% endfor %}

```markdown

httpd:

  pkg.managed:

{% if grains['os'] == 'Ubuntu' %}

    - name: apache2

{% elif grains['os'] == 'CentOS' %}

    - name: httpd

{% endif %}

    - installed

```

通过加载jinja模板引擎，可以模板配置文件按照预定条件来生成最终的配置文件

/opt/test.conf

{% if grains['os'] == 'Ubuntu' %}

host: {{ grains['host'] }}

{% elif grains['os'] == 'CentOS' %}

host: {{ grains['fqdn'] }}

{% endif %}

```markdown

/opt/test.conf:

  file.managed:

    - source: salt://test.conf

    - user: root

    - group: root

    - mode: 644

    - template: jinja

```

# 参考

二进制软件包

rpm deb 不同包管理体系，不同发行版二进制包拆分命名规则不尽相同,相比之下 deb 拆分的力度要更细些。

RHEL6/CentOS 软件包列表

salt

salt-master

salt-minion

salt-api

salt-cloud

Deian/Ubuntu 软件包列表

salt-master

salt-minion

salt-syndic

salt-doc

salt-common

salt-cloud

salt-cloud-doc

salt-api

salt-ssh

下面按照服务端（主控端）和客户端（受控端）来说明主要功能项。

主控端命令列表

/usr/bin/salt 主控命令

/usr/bin/salt-cp 批量复制文件

/usr/bin/salt-key 证书管理

/usr/bin/salt-master 服务端程序

/usr/bin/salt-run 管理虚拟机

/usr/bin/salt-ssh 管理ssh

/usr/bin/salt-syndic master分布式节点服务程序

受控端命令列表

/usr/bin/salt-call

/usr/bin/salt-minion 客户端程序

启用扩展配置文件目录

主控端 /etc/salt/master

default\_include: master.d/\*.conf

受控端 /etc/salt/minion

default\_include: master.d/\*.conf

命令示例

salt

测试与受控主机网络是否通畅

salt '\*' cmd.run test.ping

在全部受控主机行执行命令

salt '\*' cmd.run “uptime”

使用 -E 按照正则匹配操作对象

salt -E 'ubuntu\*' cmd.run “uptime”

使用 -N 按照分组匹配操作对象

salt -N 'UBUNTU-GROUPS' cmd.run “uptime”

使用 -G 按照查询信息匹配操作对象选项

salt -G 'cpuarch:x86\_64' grains.item num\_cpus

查看受控端模块函数帮助信息

salt '\*' sys.doc

查看受控端模块函数帮助信息

salt '\*' sys.doc service

salt-master

启动服务

salt-master -d

salt-key

查看证书

salt-key -L

接受指定的证书

salt-key -a KeyName

接受所有未认证的证书

salt-key -A

删除所有证书

salt-key -D

删除指定的证书

salt-key -d KeyName

salt-cp

\*批量复制文件到受控主机

salt-cp '\*' /home/vmdisk.img /var/lib/libvirtsh/vmdisk.img

\*拷贝小文件很有效，简单测试，拷贝2.5MB以上的文件就会超时报错

salt-run

salt-run 是用于管理虚拟机的命令

查询虚拟机信息

salt-run virt.hyper\_info

查询虚拟机信息

salt-run virt.query

基于云镜像创建一个新的虚拟机

salt-run virt.init centos1 2 512 salt://centos.img

salt-ssh

编辑配置文件 /etc/salt/roster

ubuntu-12.04-001:

host: 10.8.0.18

user: root

passwd: root

sudo: True

简单的测试

salt-ssh '\*' test.ping

测试执行命令

salt-ssh '\*' -r “ifconfig -a”

salt-syndic

salt的master和minion的交互很大程度上都和网络有关系,比如在管理多个国家的机器的时候(比如大中华局域网),一个master控制多个master,同时被控制的master又可以控制很多的minion，就好比是在 master 和 minions 之间又加了一层的分布式架构。

salt-minion

启动服务

salt-minion -d

salt-call

内置模块列表

内置的状态模块的完整列表： <http://docs.saltstack.com/ref/states/all/index.html>

内置的执行模块的完整列表： <http://docs.saltstack.com/ref/modules/all/index.html>

参考文档

<http://netkiller.github.io/linux/management/saltstack/>

<http://blog.halfss.com/blog/2013/06/15/saltxiang-guan-shi-yong/>

<http://www.ituring.com.cn/article/42398>

唧唧歪歪

saltstack 可能由于比较新的缘故，版本之间支持的模块差异较大，建议最好主控端，受控端使用相同版本的软件包，比如ssh\_auth模块从0.17.2版本才开始出现。

问题：

证书管理 ssh\_auth salt 模块.优先处理 （已完成)

配置仓库，符号链接 （已完成）

salt配置仓库能否托管目录？ （已完成）

一台机器是否属于多个组？ （已完成）

salt 配置仓库 致命错误情况能否回滚？ （已完成）

是否支持，配置模板 （已完成）

怎么支持自定义的模块 （部分）

如何输出给程序解析结果？ （已完成）

如何针对 特例机器进行定义配置？

是否支持 lsattr 如何保证重要配置文件不轻易被更改

主机列表管理功能， 查询当前主机列表的托管状态

自动计划任务以及，以及远程执行结果跟踪分析，

细粒度的主机权限认证控制，（LDAP）

salt-call

建议:

主机命名分组避免使用正则表达式

FAQ:

A: Salt配置仓库托管的配置文件是否支持符号链接？

Q: Salt配置仓库支持文件符号链接。

A: salt 配置仓库中配置项存在致命错误情况能否回滚？

Q: 主要保证受控端主机网络通畅，配置仓库结合svn版本控制工具，就可以实现回滚功能。

A: salt 是否支持测试执行？

Q: Salt支持执行测试，需要做的仅仅是将Test设置为True

salt 'minion1.example.com' state.highstate -v test=True

A: salt 是否支持配置模板?

Q: Salt默认使用yaml\_jinja渲染器，还支持其他渲染器，包括：yaml\_mako，使用Mako模板引擎；yaml\_wempy，使用Wempy模板引擎；py，直接使用Python写SLS文件；pydsl，建立在Python语法基础上的描述语言。

A: 如何输出给程序解析结果？

Q: salt 可以将执行结果是输出为json格式，yaml等格式，见salt PDF文档 756 页！例如:

salt '\*' test.ping --out json

state.show\_highstate --out yaml

state.show\_sls edit.vim --out pprint

# salt-minion配置文件 详解

##### Primary configuration settings #####

##########################################

default\_include: minion.d/\*.conf

#设置include配置文件

master: 192.168.1.7

#master地址

random\_master: False

#设置多master随机请求

ipv6: False

#设置ipv6地址监听

retry\_dns: 30

#设置master hostname解析失败时间

master\_port: 4506

#设置master认证端口

user: root

#SaltStack运行用户

sudo\_user: saltdev

#sudo用户

pidfile: /var/run/salt-minion.pid

#salt-minion进程pid文件

root\_dir: /

#SaltStack工作目录

pki\_dir: /etc/salt/pki/minion

#minion pki信息存储目录

id: node01

#设置minion id

grains:

#设置grains信息，默认为None

cachedir: /var/cache/salt/minion

#设置minion cache目录

verify\_env: True

#服务启动时进程权限设置与验证

cache\_jobs: False

#开始jobs cache

sock\_dir: /var/run/salt/minion

#设置minion scok目录

output: nested

#指定SaltStack命令output类型

color: True

#开始output颜色显示

strip\_colors: False

#剥离颜色显示

backup\_mode: minion

#设置备份文件

acceptance\_wait\_time: 10

#设置等待mstaer公钥时间

acceptance\_wait\_time\_max: 0

#设置等待master公钥最大时间

rejected\_retry: False

#设置拒绝key重试

random\_reauth\_delay: 60

#设置重新认证时间

auth\_timeout: 60

#设置认证timeout时间

auth\_tries: 7

#设置SaltReqTimeoutError重试次数

auth\_safemode: False

#设置safemode模式

ping\_interval: 0

#设置ping master间隔

#####   Minion module management     #####

##########################################

disable\_modules: [cmd,test]

#disable模块

disable\_returners: []

#disable return

modules\_max\_memory: -1

#设置模块最大内存

#####    State Management Settings    #####

###########################################

renderer: yaml\_jinja

#设置statc默认renderer

autoload\_dynamic\_modules: True

#在run highstate之前自动加载动态模块

clean\_dynamic\_modules: True

#设置clean动态模块

environment: None

#设置minion环境

state\_top: top.sls

#设置state入口文件

#####     File Directory Settings    #####

##########################################

file\_client: remote

#设置file client

file\_roots:

  base:

    - /srv/salt

#设置roots file目录

fileserver\_limit\_traversal: False

#设置fileserver遍历限制

hash\_type: md5

#设置文件效验hash类型

pillar\_roots:

  base:

    - /srv/pillar

#设置pollar\_roots目录

######        Security settings       #####

###########################################

open\_mode: False

#设置开启open\_mode模式

permissive\_pki\_access: False

#设置pki文件访问权限

state\_verbose: True

#设置state verbose模式

state\_output: full

#设置state输出

#####         Logging settings       #####

##########################################

log\_file: //var/log/salt/minion

#设置master日志文件

key\_logfile: /var/log/salt/key

#设置key日志文件

log\_level: warning

#设置日志级别

log\_level\_logfile: warning

#设置日志记录级别

log\_datefmt: '%H:%M:%S'

#设置日志时间格式

log\_datefmt\_logfile: '%Y-%m-%d %H:%M:%S'

#设置日志记录时间格式

log\_fmt\_console: '[%(levelname)-8s] %(message)s'

#设置console日志格式

log\_fmt\_logfile: '%(asctime)s,%(msecs)03.0f [%(name)-17s][%(levelname)-8s] %(message)s'

#设置日志记录日志格式

log\_granular\_levels: {}

#设置指定lib库的日志级别

zmq\_monitor: False

#设置zermoq event日志

######      Keepalive settings        ######

############################################

tcp\_keepalive: True

#开启tcp长连接

tcp\_keepalive\_idle: 300

#长连接发生时间

tcp\_keepalive\_cnt: -1

#认定链接失败发送多个tcp探测包

tcp\_keepalive\_intvl: -1

#设置长连接探测频率

# salt-master配置文件详解

##### Primary configuration settings #####

##########################################

default\_include: master.d/\*.conf

#设置include配置文件

interface: 192.168.1.7

#publisg/ret端口监听地址

ipv6: False

#设置ipv6地址监听

publish\_port: 4505

#ZerMQ消息发布端口

user: root

#SaltStack运行用户

max\_open\_files: 100000

#最大文件打开限制

worker\_threads: 5

#SaltStack管理线程数目

ret\_port: 4506

#SaltStack消息接收端口

pidfile: /var/run/salt-master.pid

#salt-master进程pid文件

root\_dir: /

#SaltStack工作根目录

pki\_dir: /etc/salt/pki/master

#SaltStack公钥存储目录

cachedir: /var/cache/salt/master

#SaltStackjobs和cache缓存目录

extension\_modules: <no default>

#自定义模块目录

module\_dirs: <no default>

   - /var/cache/salt/minion/extmods

#自定义模块同步目录

verify\_env: True

#服务启动时进程权限设置与验证

keep\_jobs: 24

#设置jobs消息过期时间，默认单位小时

timeout: 5

#salt和API命令超时时间，默认单位秒

loop\_interval: 60

#设置SaltStack进程检测周期(job cache fileserver backends等等)

output: nested

#指定SaltStack命令output类型

show\_timeout: True

#开启minion timeout提示

color: True

#开始output颜色显示

sock\_dir: /var/run/salt/master

#定义进程sock目录

enable\_gpu\_grains: False

#设置grains收集cpu信息

job\_cache: True

#开始job cache记录

minion\_data\_cache: True

#设置minion grains pillar数据缓存

event\_return: mysql

#设置return存储

event\_return\_queue: 0

#设置return队列

max\_event\_size: 1048576

#设置event大小

preserve\_minion\_cache: False

#设置删除key时是否删除cache数据

#####        Security settings       #####

##########################################

open\_mode: False

#设置开启open\_mode模式

auto\_accept: False

#设置自动签证

autosign\_timeout: 120

#定义自动签证timeout时间，默认单位秒

autosign\_file: /etc/salt/autosign.conf

#定义自动签证规则文件

autoreject\_file: /etc/salt/autoreject.conf

#定义自动拒绝签证规则文件

permissive\_pki\_access: False

#设置pki文件服务权限

client\_acl:

#定义用户模块执行限制，默认值None

client\_acl\_blacklist:

#定义模块和用户黑名单，默认值None

sudo\_acl: False

#关闭利用sudo后cilent\_acl限制

external\_auth:

#指定外部认证方式，默认值None

token\_expire: 43200

#设置token过期时间，默认单位秒

file\_recv: False

#设置minion是否允许push文件到master

file\_recv\_max\_size: 100

#设置minion push文件到master的hard-limit

#####     Salt-SSH Configuration     #####

##########################################

roster\_file: /etc/salt/roster

设置roster文件路径

runner\_dirs: []

#设置其他runner路径，默认为None

cython\_enable: False

#开启cython

#####      State System settings     #####

##########################################

state\_top: top.sls

#设置state入口文件

master\_tops: {}

#设置外面tops方式

external\_nodes: None

#设置external\_nodes

renderer: yaml\_jinja

#设置state默认renderer

state\_verbose: True

#设置state\_verbose模式

state\_output: full

#设置state输出

state\_aggregate: False

#设置state.

#####      File Server settings      #####

##########################################

file\_roots:

  base:

    - /srv/salt

#设置root file目录

hash\_type: md5

#设置文件效验hash类型

file\_buffer\_size: 1048576

#设置文件最大buffer

file\_ignore\_regex:

#设置同步file忽略文件正则，默认None

file\_ignore\_glob:

#设置fileserver backend

fileserver\_backend:

  - roots

#设置fileserver backend

fileserver\_followsymlinks: False

#设置fileserver允许文件链接

fileserver\_ignoresymlinks: True

#忽略fileserver允许文件链接

fileserver\_limit\_traversal: False

#设置fileserver遍历限制

fileserver\_events: False

#设置fileserver events

gitfs\_provider: gitpython

#设置fileserver gitfs驱动

gitfs\_remotes:

#设置fileserver gitfs\_remotes地址

gitfs\_ssl\_verify: True

#gitfs ssl验证

#####         Pillar settings        #####

##########################################

pillar\_roots:

  base:

    - /srv/pillar

#设置pillar\_roots目录

ext\_pillar:

#设置ext\_pillar方式，默认None

ext\_pillar\_first: False

#开启ext\_pillar\_first

pillar\_gitfs\_ssl\_verify: True

#开启pillar\_gitfs\_ssl验证

pillar\_opts: False

#开启oillar读取opts参数

pillar\_safe\_render\_error: True

#设置开启pillar render错误信息

pillar\_source\_merging\_strategy: smart

#设置pillar配置合并策略

#####          Syndic settings       #####

##########################################

order\_masters: False

#设置开启syndic

syndic\_master: masterofmaster

#设置syndic master

syndic\_master\_port: 4506

#设置syndicmaster端口

syndic\_pidfile: /var/run/salt-syndic.pid

#设置syndic pid文件

syndic\_log\_file: syndic.log

#设置syndic日志文件

#####         Logging settings       #####

##########################################

log\_file: /var/log/salt/master

#设置master日志文件

key\_logfile: /var/log/salt/key

#设置key日志文件

log\_level: warning

#设置日志级别

log\_level\_logfile: warning

#设置日志记录级别

log\_datefmt: '%H:%M:%S'

#设置日志时间格式

log\_datefmt\_logfile: '%Y-%m-%d %H:%M:%S'

#设置日志记录时间格式

log\_fmt\_console: '[%(levelname)-8s] %(message)s'

#设置console日志格式

log\_fmt\_logfile: '%(asctime)s,%(msecs)03.0f [%(name)-17s][%(levelname)-8s] %(message)s'

#设置日志记录日志格式

log\_granular\_levels: {}

#设置指定lib库的日志级别

# state tree 多版本控制：

top file的环境和设置的file\_roots是一致的，在一个简单的，单一环境中，你只设置了base环境，因此你只有一个state tree，例子如下：

file\_roots:

  base:

  - /srv/salt

以上例子意味着，你的top file只有一个环境可以映射。例子如下：

base:

  '\*':

    - server

如果你想使用多个环境，或者分隔file server到多个state tree，file\_roots可以如下设置

/etc/salt/master

file\_roots:

  base:

    - /srv/salt/online

  dev:

    - /srv/salt/dev

  qa:

    - /srv/salt/qa

我们主需要在base根目录下编辑top.sls 引用多个环境即可。

/srv/salt/master/top.sls

base:

  'online\_groups':

    - server

dev:

  'dev\_groups':

    - server

qa:

  'qa\_groups':

    - server

与 版本控制工具 svn 结合

创建 SVN

mkdir /svn/salt

svnadmin create /svn/salt

提取版本作为 state tree

cd /srv/

svn co file:///svn/salt

svn mkdir dev

svn cp dev qa

svn cp dev online

一个基本想法就是把svn的分支和state tree定义的根目录对应起来

# grains和pillar

salt 'node01' grain.items 这个默认会打印一个对象的所有grain值

**grains**

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

自定义grains信息

Grains 信息可以在 minion 配置文件 /etc/salt/minion 中静态定义，自定义的 grains 可以将 Core grains 定义的信息覆盖，示例如下：

grains:

  roles:

    - webserver

    - memcache

  deployment: datacenter4

  cabinet: 13

  cab\_u: 14-15

如果不希望将配置信息定义在 /etc/salt/minion 可以将配置信息写入 /etc/salt/grains,示例如下：

roles:

  - webserver

  - memcache

deployment: datacenter4

cabinet: 13

cab\_u: 14-15

重启 salt-minion 服务后，在 salt 'host' grains.items 可以查看自定义的信息是否生效，下面有几种自定义grains的方式，如果这些定义方式同时存在,则遵守如下由低到高的优先级：

Core grains.

Custom grains in /etc/salt/grains.

Custom grains in /etc/salt/minion.

Custom grain modules in \_grains directory, synced to minions

在用salt进行管理客户端的时候或者写state的时候都可以引用grains的变量

**pillar**

salt敏感信息的管理,只有匹配到的节点才能看到和使用

salt '\*' pillar.items

文档:<http://docs.saltstack.com/topics/tutorials/pillar.html>

开启敏感信息配置管理 Pillar settings

编辑 /etc/salt/master 文件

pillar\_roots:

  base:

    - /srv/pillar

mkdir -p /srv/pillar

默认:pillar数据定义文件存储路径:/srv/pillar

入口文件:/srv/pillar/top.sls

base:

  "targeting":

    - pillar            #名字为pillar.sls的文件来存放对匹配到的minion的变量

$pillar.sls

#基本:

$key: $value

#对应state引用方式: {{ pillar['$key'] }}

#复杂:

users:

  thatch: 1000

  shouse: 1001

  utahdave: 1002

  redbeard: 1003

#state引用方式:

#{% for user, uid in pillar.get('users', {}).items() %}

#  {{user}}:

#    user.present:

#      - uid: {{uid}}

#{% endfor %}

查看节点的pillar数据：

salt 'client2' pillar.data

同步pillar：

salt '\*' saltutil.refresh\_pillar

附：这里我们可以看到，pallar中也可以使用jinja

文档:<http://docs.saltstack.com/topics/tutorials/pillar.html>

# salt 认证过程

salt 认证过程

当初始化安装 minion 启动服务启动后

minion端生成一个秘钥对，并产生一个ID值，minion服务会安装ID值命名的公钥发送给 master ,直到接受为止;

master认证完毕后，会将minion 端发送来的，以ID值命名的公钥存放在 /etc/salt/pki/master/minions 目录中(无扩展名);

master认证完毕后，会将自身的公钥发送给 minion，并存储为 /etc/salt/pki/minion/minion\_master.pub.

minion\_id 值的生成过程

minion 默认按照如下顺序，试图在找到一个不是localhost的值作为ID：

Python函数socket.getfqdn() 获取的值；

/etc/hostname 定义的值(仅限于非Windows系统)；

/etc/hosts (%WINDIR%\system32\drivers\etc\hosts on Windows hosts) 中定义的任何映射 127.0.0.0/8 的主机名做ID值

如果以上都能够产生一个ID， 并且不是localhost，then a sorted list of IP addresses on the minion (excluding any within 127.0.0.0/8) is inspected. The first publicly-routable IP address is used, if there is one.Otherwise, the first privately-routable IP address is used.如果一切都失败了，那么则使用localhost作为备用，最终获取的ID将记录在/etc/salt/minion\_id文件中，该文件可以手动更改，重启服务后不会被重新覆盖。

注意：如果minion端主配置文件 /etc/salt/minion 中启用了 id: xxxx , 那么这个ID值将覆盖 /etc/salt/minion\_id 中记录的ID数值.

参考

master 秘钥对默认存储在

/etc/salt/pki/master/master.pub

/etc/salt/pki/master/master.pem

master 端认证的公钥存储在：

/etc/salt/pki/master/minions/

minion 秘钥对默认存储在

/etc/salt/pki/minion/minion.pub

/etc/salt/pki/minion/minion.pem

minion 存放的master公钥

/etc/salt/pki/minion/minion\_master.pub

minion\_id 默认存储在

/etc/salt/minion\_id

# Salt WebUI

saltstack WebUI平台界面

saltapi项目主页： <http://salt-api.readthedocs.org/en/latest/>

halite 项目主页： <https://github.com/saltstack/halite>

搭建过程：

cd /var/www/

git clone <https://github.com/saltstack/halite>

cd halite/halite

./genindex.py -C

安装 salt-api

apt-get install salt-api 或 yum install salt-api

在master端添加配置文件

在/etc/salt/master.d中加入saltweb.conf,需要用到salt的eauth系统,添加如下配置

rest\_cherrypy:

 host: 0.0.0.0

 port: 8080

 debug: true

 disable\_ssl: True

 static: /var/www/halite/halite

 app: /var/www/halite/halite/index.html

external\_auth:

   pam:

     salt:

     - .\*

     - '@runner'

     - '@wheel'

说明，我在这里设置了 disable\_ssl 使用 external\_auth pam认证的用户进行登录,操作完毕后进行如下操作：

重启master;

/etc/init.d/salt-master restart

添加登陆用户

useradd salt

passwd salt

建立用户后进行测试

salt -a pam \\*  test.ping

输入用户和密码 如看到minion返回信息 则表示登陆验证成功

启动 web server

cd /var/www/halite/halite

python server\_bottle.py -d -C -l debug -s cherrypy

或开启一个后台守护进程

start-stop-daemon --start --background --exec /var/www/halite/halite/server\_bottle.py -- -d -C -l debug -s cherrypy

然后打开<http://ip:8080/app>，通过salt/salt登陆即可

其他参考文档

<http://rfyiamcool.blog.51cto.com/1030776/1275443>

<http://rfyiamcool.blog.51cto.com/1030776/1345518>

# salt 扩展认证 PAM

salt.states.external\_auth

概述

salt 扩展认证 PAM，可以利用PAM 认证机制对系统账户做出功能操作上的限制

认证用户不允许使用root

可以限制认证用户可使用的功能模块

可以限定认证用户可管理的主机

官方文档：<http://docs.saltstack.com/topics/eauth/index.html>

开启PAM认证

编辑 /etc/salt/master

external\_auth:

  pam:

    salt:

      - '\*'

创建 salt 用户 设置密码

验证

salt PAM 认证用户 只能通过 “ -a pam ” 参数才能使用

salt -a pam test.ping

会提示登陆用户，输入密码，如果正确返回结果，说明pam认证已经生效

实例 ：

/etc/salt/master

external\_auth:

  pam:

    admin：

      - '\*'                      # 所有模块

    salt:

      - test.ping                  # test.ping 模块

    test:

      - 'dev\*'：                   # 主机配置规则

        - test.ping                # Salt 模块

解释：

admin 用户可管理所有主机，使用所有salt功能模块

salt 用户可管理所有主机，仅能使用test.ping模块

test 用户只能管理部分主机，仅能使用test.ping模块

实例1： 以salt用户为例,认证用户必须打开PAM 参数才能操作

失败操作：

[salt@fabrizio-vaio](mailto:salt@fabrizio-vaio):~$ salt '\*' test.ping

Failed to authenticate, is this user permitted to execute commands?

成功操作：

[salt@fabrizio-vaio](mailto:salt@fabrizio-vaio):~$ salt -a pam '\*' test.ping

username: salt

password:

ubuntu-master:

True

实例2： 以salt用户为例,执行未授权模块被拒绝

[salt@fabrizio-vaio](mailto:salt@fabrizio-vaio):~$ salt -a pam '\*' state.highstate

username: salt

password:

Failed to authenticate, is this user permitted to execute commands?

实例3： 以test用户为例,操作未授权主机被拒绝

失败操作

[salt@fabrizio-vaio](mailto:salt@fabrizio-vaio):~$ salt -a pam '\*' test

username: test

password:

Failed to authenticate, is this user permitted to execute commands?

成功操作

[salt@fabrizio-vaio](mailto:salt@fabrizio-vaio):~$ salt -a pam 'test-001' test

username: test

password:

test-001:

    True

# salt syndic 概述

一个基本的salt配置方式是一个master指挥一群minion，为了不再有假设使用任何单一拓扑结构，考虑多种布局的情况下，开发者引入了syndic，主控master可以控制一群master，通过syndic将操作命令传输给受控master，受控master来完成对自己旗下minion的管理，并将结果传回主控master，从而实现了主控master对所有minion的间接管理。

注意事项：各个 syndic 必须提供自己的file\_roots目录，文件不会在 master-master 之间自动分发！

软件包的安装

master端: salt-master

syndic端: salt-master,salt-syndic

minion端：salt-minion

基本配置

salt-master 需要开启的基本配置，修改配置文件 /etc/salt/master

order\_masters: True

salt-syndic 需要开启的基础配置，修改配置文件 /etc/salt/master

syndic\_master: (指向主控 master 端的 ip或hostname）

file\_roots:

  base:

    - /srv/salt

salt-minion 需要开启的基础配置,修改配置文件 /etc/salt/minion

master: (指向主控 syndic端的 ip或hostname）

证书认证：

所有的 syndic 和 主控 master 进行认证;

所有的 minions 和对应的 syndic 进行认证;

基本测试

在 top master 与所有 minions 的进行连接测试:

salt \\* test.ping

返回结果如下

salt-minion:

    True

salt-master:

    True

salt-syndic:

    True

由于文件不会在 master-master 之间自动分发，配置好各个 syndic 主机的file\_roots 分发SLS资源文件后，(关于SLS资源文件的分发建议使用Git作为 fileserver\_backend) 在 top master进行基本管理任务测试,

示例：

salt \\* state.highstate test=True

返回结果如下

...

salt-minion:

    ----------

    pkg\_|-nginx\_|-nginx\_|-installed:

        ----------

        \_\_run\_num\_\_:

            2

        changes:

            ----------

        comment:

            Package nginx is already installed

        name:

            nginx

        result:

            True

salt-syndic:

    ----------

    pkg\_|-nginx\_|-nginx\_|-installed:

        ----------

        \_\_run\_num\_\_:

            2

        changes:

            ----------

        comment:

            Package nginx is already installed

        name:

            nginx

        result:

            True

salt-master:

    ----------

    pkg\_|-nginx\_|-nginx\_|-installed:

        ----------

        \_\_run\_num\_\_:

            2

        changes:

            ----------

        comment:

            Package nginx is already installed

        name:

            nginx

        result:

            True

...

参考文档

<http://docs.saltstack.com/en/latest/topics/topology/syndic.html>

<http://docs.saltstack.com/en/latest/topics/tutorials/gitfs.html>

# salt gitfs

[**http://my.oschina.net/u/877567**](http://my.oschina.net/u/877567)

**概述**

salt默认使用 /srv/salt作为state的文件路径，另外还支持多个路径，或者远程git仓库。

**准备工作**

GitFS 需要 GitPython 0.3.0 或更新的版本，如果你的系统软件仓库自带的版本不满足需求 (比如 Ubuntu 12.04 LTS)，请使用如下方式安装

pip install GitPython

**开启配置**

GitPython成功安装好后,基本配置实例如下，修改配置文件 /etc/salt/master

fileserver\_backend: - git gitfs\_remotes: - https://domain.com/salt.git

**测试**

salt \\* state.highstate test=True

**其他参考**

**gitfs\_root**

上面的例子中远端仓库根目录即为base环境的根目录，如果是base环境对应的是git仓库内的一个子目录，可以使用gitfs\_root配置项来指定

gitfs\_root: salt

或

gitfs\_root: salt/site

**fileserver\_backend**

fileserver\_backend 可以指定多个后端存储方式,比如一个典型的应用场景就是SLS文件存在git仓库中，大的文件存储在master本地

fileserver\_backend: - roots - git

**gitfs\_remotes**

* gitfs\_remotes 支持配置使用仓库内的指定的目录

gitfs\_remotes: - file:///repository.git - root: gitfs\_root: somefolder/otherfolder

* 指定多个仓库

gitfs\_remotes: - git://github.com/example/first.git - https://github.com/example/second.git - file:///root/third

**参考文档**

* <http://docs.saltstack.com/en/latest/topics/tutorials/gitfs.html>
* <http://docs.saltstack.com/en/latest/ref/configuration/master.html#std:conf_master-fileserver_backend>

# saltstack同步文件

sudo salt 'haproxy\*' cp.get\_file salt://haproxy/whiteurl.lst /etc/haproxy/whiteurl.lst

sudo salt 'haproxy\*' cp.get\_file salt://haproxy/haproxy.cfg /etc/haproxy/haproxy.cfg

sudo salt 'haproxy\*' cp.get\_file salt://haproxy/blackip.lst /etc/haproxy/blackip.lst

sudo service haproxy restart