🡪

Redhat Ansible是一个自动化的配置管理和配置工具，使用python开发， 类似于Chef，Puppet或Salt。这是一款很简单也很容易入门的部署工具，它使用SSH连接到服务器并运行配置好的任务，服务器上不用安装任何多余的软件，只需要开启ssh。

Ansible 特点：

部署简单，只需在主控端部署 Ansible 环境，被控端无需做任何操作。

默认使用 SSH（Secure Shell）协议对设备进行管理。

主从集中化管理。

配置简单、功能强大、扩展性强。

支持 API 及自定义模块，可通过 Python 轻松扩展。

通过 Playbooks 来定制强大的配置、状态管理。

对云计算平台、大数据都有很好的支持。

提供一个功能强大、操作性强的 Web 管理界面和 REST API 接口 —- AWX 平台。

🡪

剧本(playbook)是Ansible的配置、部署语言，由它对描述你想要远程机器执行的策略或步骤。  
使用剧本不仅可以简单的对远程机器进行管理，还有很多更强大的功能 。\

每一个Ansible剧本都是一个YAML格式的文件.

YAML文件中的host对应某个inventory, 表示将要被管理的对象们.

🡪

inventory文件，用于定义将要管理的服务器。这个文件可以命名为任何名字，但我们通常会命名为hosts或者项目的名称。 在这个文件中，根据标签设置所属的主机，以后我们可以告诉ansible对哪个inventory文件里面的哪些标签🡪

Playbook（剧本）, 描述需要进行的任务。在playbook中可以含有一个或者多个play, 每个play中有一个tasks列表.

🡪

使用ansible的三种方式：

Ad-Hoc (直接命令行执行，不使用playbook)

ansible-playbooks （Ansible和Playbook一起使用）

Automation Framework (Ansible Tower)

🡪

Check Mode在执行playbook的时候加上--check选项，可以在实际改变被管理的机器前，进行验证。Playbook会打印出将会执行的变化，而不是真正的执行他们。

🡪

Playbook中直接一个Play的例子（不适用Roles）：

- hosts: control-node # 将要执行任务的主机(可为一组或多组)， 执行的对象为control-node分组，已经在/etc/ansible/hosts文件中定义好了

remote\_user: root # 在目标主机上执行任务时的用户身份

vars: # 定义变量， 接下来{{变量名}} 便可以使用

- pkg: httpd

# task 下面是任务列表

tasks: # 表示开始申明tasks列表区域, **每一个playbook只包含一个tasks列表，但一个tasks列表可以有多个 task.** 一个 task 在其所对应的所有主机上执行完毕之后, 下一个 task 才会执行。每个 task 的目标在于执行一个 moudle, 通常是带有特定的参数来执行. 在参数中可以使用变量（variables）

# 使用yum模块进行操作

- name: ensure apache is at the latest version # 每一个 task 必须有一个名称 name

yum: pkg=httpd state=latest # 这个task将运用yum模块, 参数是httpd这个包， 状态时最新latest

- name: write the apache config file # 第二个task，任务名称

template: src=/srv/httpd. j2 dest=/etc/httpd.conf # 这个task将运用template模块

# 定义一个触发器

notify: # 第二个任务配置的notify, 当任务完成以后notify所指定名称的handler

- restart apache # 指定notify所要唤醒的handler的名称

- name: ensure apache is running # 第三个task，任务名称

service: name=httpd state=started # task调用service模块， name=httpd调用httpd service, 状态要变为 started

# 和上面定义的触发器对应，当触发器被触发时执行下面的操作

handlers: # 定义handlers区域

- name: restart apache # 此handler的名称

service: name=httpd state=restarted # 当被唤醒后执行ansible service模块， 进行httpd restarted

🡪

在执行一个 playbook 之前,想看看这个 playbook 的执行会影响到哪些 hosts：

ansible-playbook playbook.yml --list-hosts

🡪

**Ansible + playbook**

-i 选择特定的inventory文件

--key-file 选择指定的 key文件，使用该文件去验证连接

-u 指定远程用户， 连接到远端后使用该用户名

🡪

角色(roles)：如果我们使用playbook写成一个文件，这个文件会很大，但是不方便组织，我们可以分组，把playbook根据功能，如handler，tasks等分门别类的放在在各自的子目录下，形成一个集合，就是角色。 然后在playbook中使用该角色，建议把角色放在固定的目录下，目录可以ansible的配置文件中的roles\_path来定义。

* 目录结构清晰
* 可以重复使用

**角色(role)的定义：**

在roles目录下，分别创建不同的目录，每个目录就是不同的角色，roles目录下是角色的集合：

如roles/创建如下的目录

mysql/ # 建立role为mysql

httpd/ # 建立role为httpd

nginx/ # 建立role为nginx

memcached/ # 建立role为memcached

**roles模块的路径可以在配置文件/etc/ansible/ansible.cfg里roles\_path定义.**

对于每一个角色，将以特定的层级目录结构进行组织, 例如mysql的角色：

files/ ：存放由copy或script模块等调用的文件；

templates/：template模块查找所需要模板文件的目录；

tasks/：至少应该包含一个名为main.yml的文件；其它的文件需要在此文件中通过include进行包含；

handlers/：至少应该包含一个名为main.yml的文件；其它的文件需要在此文件中通过include进行包含；

vars/：至少应该包含一个名为main.yml的文件；其它的文件需要在此文件中通过include进行包含；

meta/：至少应该包含一个名为main.yml的文件，定义当前角色的特殊设定及其依赖关系；其它的文件需要在此文件中通过include进行包含；注意，角色和角色之间有依赖关系，如安装NT，先装nginx后装tomcat，一般不需定义，执行我们自己定义好安装顺序即可

default/：设定默认变量时使用此目录中的main.yml文件；

**调用使用角色的三种方法:**

方法1：

- hosts: websrvs

Become: yes # 2.6版本以后的参数，之前是sudo，意思为切换用户运行

become\_user: root # 指定sudo用户为root

roles: # 使用mysql, memcached 以及nginx角色

- mysql

- memcached

- nginx

方法2：

- hosts:

remote\_user:

roles:

- {role: nginx, username: nginx} #键role用于指定角色名称；后续的k/v用于传递变量给角色

方法3（基于条件测试实现角色的调用）：

roles:

- {role: nginx, when: "ansible\_distribution\_major\_version == '7' “}

**使用角色时候的查找顺序：**

roles目录，对应的文件夹，先寻找到角色

tasks：先找tasks目录里面的main.yml文件，main.yml文件作为起点，在其中可以看到import\_tasks来导入别的yml文件（也就是其他的任务），这个是必须的

templates： 找这个目录里面的模板文件，在tasks里面是可以用相对路径来写

vars：先找main.yml文件，如果找到就找变量替换

handlers：先找main.yml文件，如果找到，则在tasks里面通过notify来触发

files： 放的是静态文件

🡪

Ansible-playbook定义inventory清单，ansible-playbook可以有多种语法去定义inventory清单。常用的两种是INI语法和YAML语法.

使用INI语法：

mail.example.com

[webservers]

foo.example.com

bar.example.com

[dbservers]

one.example.com

two.example.com

three.example.com

使用YAML语法：

all:

hosts:

mail.example.com:

children:

webservers:

hosts:

foo.example.com:

bar.example.com:

dbservers:

hosts:

one.example.com:

two.example.com:

three.example.com:

🡪

**Ansible-playbook的变量**

Ansible-playbook的变量来源方式有许多，有的是用户自定义的变量，有的是ansible-playbook连接到远程主机以后默认将会收集的变量，有的是ansible-playbook默认提供的变量。

Ansible-playbook中的变量应该只包含 (字母，数字，下划线， 并且以英文字母开头)

**--在Inventory文件中定义变量（对单个host加入变量）：**

在Inventory中使用INI语法：

[atlanta]

host1 http\_port=80 maxRequestsPerChild=808

host2 http\_port=303 maxRequestsPerChild=909

在Inventory中使用YAML语法：

atlanta:

host1:

http\_port: 80

maxRequestsPerChild: 808

host2:

http\_port: 303

maxRequestsPerChild: 909

**--在Inventory文件中定义变量（对成组的host加入变量）：**

在Inventory中使用INI语法：

[atlanta]

host1

host2

[atlanta:vars]

ntp\_server=ntp.atlanta.example.com

proxy=proxy.atlanta.example.com

在Inventory中使用YAML语法：

atlanta:

hosts:

host1:

host2:

vars:

ntp\_server: ntp.atlanta.example.com

proxy: proxy.atlanta.example.com

**--在playbook文件中直接定义：**

- hosts: webservers

vars:

http\_port: 80

**--在playbook文件中引入变量文件：**

- hosts: all

gather\_facts: False

vars\_files:

- /tmp/var.yaml

**--使用Role时，在Role目录下的defaults/main.yml或者vars/main.yml引入变量**

例如main.yml

Sent\_port: 333

Receive\_port: 444

**--Ansible通过Facts机制获取变量**

这一类的变量通过ansible搜集远端主机来进行获取，而不是用户自行定义。搜索的数据放在ansible\_fact字典中,

并且各变量以ansible\_开头

如下：

"ansible\_facts": {

"ansible\_all\_ipv4\_addresses": [

"192.168.238.129"

],

"ansible\_all\_ipv6\_addresses": [

"fe80::20c:29ff:fe21:bd17"

],

"ansible\_architecture": "x86\_64",

"ansible\_bios\_date": "07/02/2015",

"ansible\_bios\_version": "6.00",

"ansible\_cmdline": {

"BOOT\_IMAGE": "vmlinuz0",

"initrd": "initrd0.img",

"liveimg": true,

"nodiskmount": true,

"nolvmmount": true,

"quiet": true,

"rd.dm": "0",

"rd.luks": "0",

"rd.md": "0",

"rhgb": true,

"ro": true,

"root": "live:CDLABEL=CentOS-6.8-x86\_64-LiveCD",

"rootfstype": "auto"

},

"ansible\_python\_version": "2.6.6",

"ansible\_selinux": {

"config\_mode": "enforcing",

"mode": "enforcing",

"policyvers": 24,

"status": "enabled",

"type": "targeted"

},

…..

}

在playbook中使用变量时，使用{{ ansible\_python\_version }}便为2.6.6

如果我们想禁用ansible的自动收集数据的功能则：

- hosts: whatever

gather\_facts: no # ansible便不再自动收集远程主机的数据

**Ansible-Playbook的魔法变量**

即便我们不在ansible-playbook中定义任何变量，禁用ansible的facts搜集特性， ansible仍然会为我们提供一些特殊的特定变量（magic variable）, 我们可以直接在playbook中直接使用。例如hostvars, groups, group\_names, inventory\_hostname.

inventory\_hostname的值将会是在inventory文件中定义的主机名。

🡪Ansible的Play应该被设计成具有幂等性。即是多次执行将会产生相同的结果，实现的方法可以为在执行tasks前检查系统状态是否已经达到所要求状态，如果达到退出否则才执行tasks.

🡪检验playbook合法性

ansible-lint [playbook名] 用于检查playbook的合法性，如果有问题返回问题代码，可以查看文档查询问题代码详细信息。

执行ansible-playbook时--syntax-check标记用于检查playbook以及背后的roles, files, 的语法问题。

执行ansible-playbook [playbook名]时--list-hosts标记用于在playbook执行前展示将影响的机器。

🡪 Playbook的执行顺序。

默认情况下Ansible从上到下执行Playbook中的每一个play. 对Play中的Tasks,从上到下执行Tasks中的每一个Task,并且对所有目标Hosts执行完一个Task1再对所有目标Hosts执行下一个Task.

默认情况下，如果一个Host在执行其中一个Task时发生错误，则该Ansible会停止在该Host上接下来的所有Task运行。并无法自动回滚已经对该Host执行的操作。

🡪ansible-playbook在执行我们定义的tasks前，还会执行自带的Gathering Facts任务。用于收集被管理主机的基本信息。

🡪ansible-playbook使用with\_items来表示循环。with\_items和任务模块处于同一层级

例如：

- hosts: web #指定web主机组, 配置见/etc/ansible/hosts文件

remote\_user: root #指定远端服务器使用root用户执行

tasks:

- name: create\_user

user: name={{ item }} #变量item

with\_items: #循环三个用户名

- user1

- user2

- user3

🡪ansible-playbook的when关键字。在使用ansible做自动化运维的时候，大多数情况下都执行某些任务的时候都需要依赖某个变量的值或者是上一个任务的执行结果。如，根据facts信息中的系统版本相关的信息来确定使用哪种包管理器安装软件。Ansible提供when关键字，可以控制任务的执行流程。

例如被管理主机为Debian的时候，执行关机操作

    tasks:

      - name: "shutdown Debian flavored systems"

        command: /sbin/shutdown -t now

        when: ansible\_os\_family == "Debian”

当`when`和`with\_items`一起使用的时候，每个项都会单独被`when`关键字处理：

tasks:

* name: “looping printing”

        command: echo {{item}}

        with\_items: [ 0, 2, 4, 6, 8, 10]

        when: item > 5

ansible-playbook只会对item=6, item=8, item=10执行

🡪ansible-playbook的failed\_when关键字。遇到需要在变量不符合的情况下，中断ansible的剧本。

例如：

- name: shell

shell: ps -ef | grep kernel | wc -l

register: kernel.status

- name: update

shell: kill -9 kernel

failed\_when: kernel.status.stdout != "1"

P.s 是否使用failed\_when关键字需要考虑task是否会因为执行不成功而自行失败，从而无需使用failed\_when关键字。例如我们要判断服务是否存在，文件是否存在时候，将执行command模块。当ls文件，执行服务时候如果他们不存在则task会自行failed,无需使用failed\_when。 而判断输出内容状态时候，命令肯定执行成功，但我们要根据状态内容判断时候，则需要使用failed\_when关键字。

🡪ansible-playbook注册变量Register关键字。用于在playbook中将某个命令运行的结果保存起来，提供给后续任务使用。如，通过command模块来判断远程节点上某个文件是否存在或者通过执行某个命令的获取其返回结果，并保存起来，下个任务根据获取的变量值来决定执行的具体操作。

例如：

- name: test play

      hosts: all

      tasks:

        - shell: cat /etc/motd

          register: motd\_contents # 将此task返回的结果放入申明的motd\_contents变量中

        - shell: echo "motd contains the word hi"

          when: motd\_contents.stdout.find('hi') != -1

# 🡪ansible playbook中的过滤器|

# 过滤器是一种能够帮助我们处理数据的工具，其实，ansible中的过滤器功能来自于jinja2模板引擎，我们可以借助jinja2的过滤器功能在ansible中对数据进行各种处理，例如|upper就是一种过滤器，这个过滤器的作用就是将小写字母变成大写，我们可以发现，当我们想要通过过滤器处理数据时，只需要将数据通过管道符传递给对应的过滤器即可，当然，过滤器不只有upper，还有很多其他的过滤器，这些过滤器有些是jinja2内置的，有些是ansible特有的，如果这些过滤器都不能满足你的需求，jinja2也支持自定义过滤器。

例如：

- hosts: testB

remote\_user: root

vars:

testvar: "abc123ABC 666"

tasks:

- debug:

#将字符串转换成纯大写

msg: "{{ testvar | upper }}"

🡪