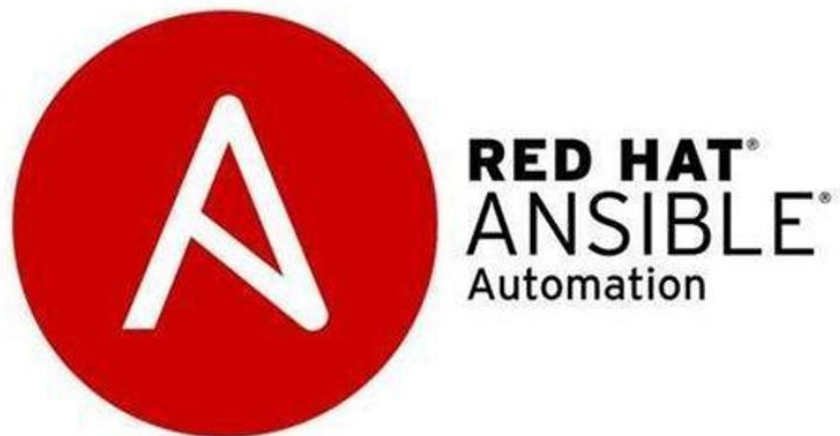


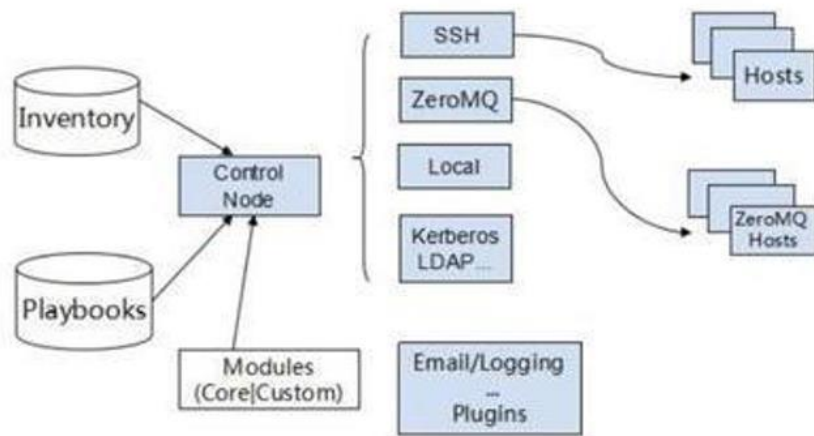
Ansible 自动化入门

我们知道现在自动化工具中，最简单、易于上手，而且最流行的当属 Ansible 莫属了。和 Chef、Puppet 等 Ruby 工具甚至同时 Python 系的 Saltstack 等 CS 架构的自动化工具相比虽然执行性能上可能会稍一点，但是无需客户端，只需 SSH 就可以管理优势还是相当明显的。服务器上架后无需额外操作就可以直接进行操作，比如服务器的初始化标准配置等。本文虫虫给大家介绍就是 Ansible 入门教程，如果此前已有对应的基础可以忽略本文。



概述

Ansible 是一个开源配置管理工具，可以使用它来自动化任务，部署应用程序实现 IT 基础架构。Ansible 可以用来自动化日常任务，比如，服务器的初始化配置、安全基线配置、更新和打补丁系统，安装软件包等。Ansible 架构相对比较简单，仅需通过 SSH 连接客户机执行任务即可：



Ansible 使用过程中会用到一些概念术语，我们先介绍一下。

Ansible 的与节点有关的重要术语包括控制节点，受管节点，清单和主机文件：

控制节点 (Control node)：指安装了 Ansible 的主机，也叫 Ansible 服务器端, 管理机。 Ansible 控制节点主要用于发布运行任务，执行控制命令。 Ansible 的程序都安装在控制节点上，控制节点需要安装 Python 和 Ansible 所需的各种依赖库。注意：目前 Ansible 还不能安装在 Windows 下。

受控节点 (Managed nodes)：也叫客户机，就是想用 Ansible 执行任务的客户服务器。

清单 (Inventory)：受控节点的列表，就是所有要管理的主机列表。

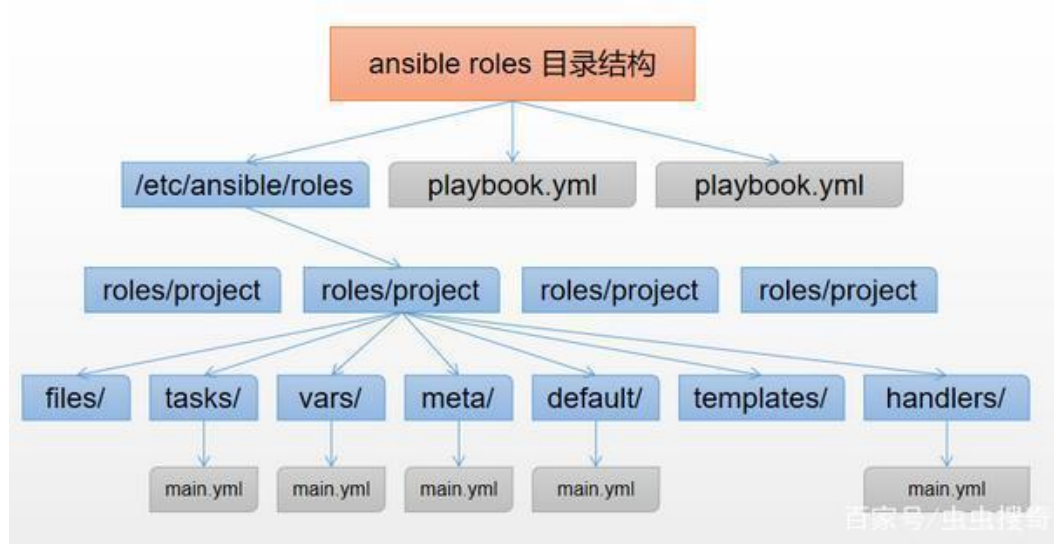
host 文件：清单列表通常保存在一个名为 host 文件中。在 host 文件中，可以使用 IP 地址或者主机名来表示具体的管理主机和认证信息，并可以根据主机的用户进行分组。缺省文件： /etc/ansible/hosts，可以通过 -i 指定自定义的 host 文件。

模块 (Modules)：模块是 Ansible 执行特定任务的代码块。比如：添加用户，上传文件和对客户机执行 ping 操作等。 Ansible 现在默认自带 450 多个模块，， Ansible Galaxy 公共存储库则包含大约 1600 个模块。

任务 (Task)：是 Ansible 客户机上执行的操作。可以使用 ad-hoc 单行命令执行一个任务。

剧本 (Playbook)：是利用 YAML 标记语言编写的可重复执行的任务的列表， playbook 实现任务的更便捷的读写和贡献。比如，在 Github 上有大量的 Ansible playbooks 共享，你要你有一双善于发现的眼睛你就能找到大量的宝藏。

角色 (roles)：角色是 Ansible 1.2 版本引入的新特性，用于层次性、结构化地组织 playbook。 roles 能够根据层次型结构自动装载变量文件、 tasks 以及 handlers 等。



Ansible 的优势

Ansible 作为最受欢迎的自动化配置工具，主要得益于其设计上的优势。

无需客户端

与 Chef、Puppet 以及 Saltstack（现在也支持 Agentless 方式 salt-ssh）不同，Ansible 是无客户端 Agent 的，所以无需在客户机上安装或配置任何程序，就可以运行 Ansible 任务。由于 Ansible 不会在客户机上安装任何软件或运行监听程序，因此消除了许多管理开销，我们可以在即可上手使用 Ansible 管理服务器，同时 Ansible 的更新也不会影响任何客户机。

使用 SSH 进行通讯

默认情况下，Ansible 使用 SSH 协议在管理机和客户机之间进行通信。可以使用 SFTP 与客户机进行安全的文件传输。

并行执行

Ansible 与客户机并行通信，可以更快地运行自动化任务。默认情况下，forks 值为 5，可以按需，在配置文件中增大该值。

安装 Ansible

Ansible 可以运行在任何机器上，但是对管理机有一定要求。管理机应安装 Python 2（2.7）或 Python 3（3.5 或更高版本），另外，管理机不支持 Windows 控制节点。我们可以使用 Linux 发行版包管理器、源码安装或者 Python 包管理器（PIP）来安装 Ansible。

redhat 系

`sudo yum install ansible` 或者 `dnf install ansible`

```
Total
Running transaction check
Running transaction test
Transaction test succeeded
Running transaction
  Installing : python2-jmespath-0.9.0-3.el7.noarch
  Installing : sshpass-1.06-2.el7.x86_64
  Installing : ansible-2.9.1-1.el7.noarch
  Verifying  : sshpass-1.06-2.el7.x86_64
  Verifying  : python2-jmespath-0.9.0-3.el7.noarch
  Verifying  : ansible-2.9.1-1.el7.noarch

Installed:
  ansible.noarch 0:2.9.1-1.el7

Dependency Installed:
  python2-jmespath.noarch 0:0.9.0-3.el7

Complete!
```

百家号/虫虫搜奇

ubuntu

`sudo apt update`

`sudo apt install software-properties-common`

```
sudo apt-add-repository --yes --update ppa:ansible/ansible
```

```
sudo apt install ansible
```

通过 Pip 安装

```
pip install --user ansible
```

或者全局性安装

```
sudo pip install ansible
```

源码安装

```
git clone github.com/ansible/ansible.git
```

```
cd ./ansible
```

```
source ./hacking/env-setup
```

安装成功后，可以使用下面命令检查 Ansible 的安装版本：

```
ansible -version
```

```
ansible 2.9.1
```

```
config file = /etc/ansible/ansible.cfg
```

```
configured module search path = [u'/home/xx/.ansible/plugins/modules', u'/usr/share/ansible/plugins/modules']
```

```
ansible python module location = /usr/lib/python2.7/site-packages/ansible
```

```
executable location = /usr/bin/ansible
```

```
python version = 2.7.5 (default, Oct 30 2018, 23:45:53) [GCC 4.8.5 20150623 (Red Hat 4.8.5-36)]
```

安全配置

为了使 Ansible 与客户端通信，需要使用用户帐户配置管理机和客户机。为了方便快捷安全，一般会配置证书方式连接客户机。

帐号规划

假设我们有四台客户机（node1, node2, node3 和 node4）和管理机（manage）。为了方便，我们创建一个 ansible 用户帐户，将该 ansible 用户添加到 wheel 组，然后配置 SSH 身份验证。在配置新用户帐户时，请在所有节点上创建该帐户（注意可以使用 ansible 批量创建）：

```
sudo useradd ansible
```

然后将 ansible 用户添加到 wheel 组：

```
sudo usermod -aG wheel ansible
```

为用户设置密码：

```
sudo passwd ansible
```

接下来，使用 visudo 配置/etc/sudoers 文件，赋予 ansible 用户使用 sudo 执行特权命令：

```
%wheel ALL=(ALL) NOPASSWD: ALL
```

配置证书登陆

在所有客户机和管理上创建新的 ansible 用户之后，我们在管理机（ansible 用户）生成 SSH 密钥，然后将 SSH 公钥复制到所有客户机。

```
sudo su - ansible
```

```
ssh-keygen
```

```
Created directory '/home/ansible/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/ansible/.ssh/id_rsa.
Your public key has been saved in /home/ansible/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:6AfCSbSd9lfLY2OduqhZDUaU/fJSWQRBbSJxN7QFt7E ansible@node1.yt-cn.com
The key's randomart image is:
+---[RSA 2048]-----+
|
| . o. oBX=
| . o . . . o..o@
| o + . . o E
| o o o . o.o+.
| + o S + B+o
| o . o =.+.
| . . . o.
| . o . .
| o..
+---[SHA256]-----+
```

百家号/虫虫搜奇

现在，将 SSH 公钥复制到所有客户机，这使管理机 ansible 用户无需输入密码即可登录客户机：

```
ssh-copy-id ansible@node1
```

```
/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/home/ansible/.ssh/id_rsa.pub"
/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to fi
/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are p
ansible@node1's password:

Number of key(s) added: 1

Now try logging into the machine, with: "ssh 'ansible@node1'"
and check to make sure that only the key(s) you wanted were added.
```

百家号/虫虫搜奇

第一次使用，完成以后，就可以直接用证书登陆了。

Ansible 配置

默认的配置文件位于/etc/ansible/ansible.cfg 下。可以使用此配置文件来修改绝 Ansible 大多数设置，一般无需额外多配置，默认配置应能满足大多数使用情况。关于 Ansible 配置文件，其执行程序会按照一定顺序搜索配置：

Ansible 按照以下顺序搜索配置文件，优先配置优先使用，而忽略其余配置文件：

\$ANSIBLE_CONFIG 环境变量。

任务当前目录下的：ansible.cfg（如果在当前目录中）。

当前用户下的 ansible.cfg： ~/.ansible.cfg

默认配置文件： /etc/ansible/ansible.cfg。

默认清单配置文件位于/etc/ansible/hosts 中，但是通过 ansible.cfg 配置文件中修改此位置。也可以通过-i 自定义 hosts 清单。

为了安全起见，虫虫建议你，不要直接在/etc/ansible/host 配置清单，尤其是有用户认账等信息时候。对于长期不执行 ansible 可以将 host 文件加密锁定，防止信息泄露，引起安全事故。

清单文件格式如下：

```
[nodes]
```

```
node1
```

```
node2
```

```
node3
```

```
node4
```

```
[web]
```

```
node2
```

```
node3
```

可以对不同用途分组，用[]指定分组名。

Ansible ad-hoc 单行命令执行

ad-hoc 命令行是我们可以随手执行的单个 ansible 任务，是 ansible 任务快速执行方式。对于一些快速获取的任务使用 ad-doc 命令非常简便有效，而且有助于我们学习和熟悉 Ansible 的使用。

命令行选项

常用的 Ansible 命令行选项如下：

-b, --become： 特权方式运行命令。

-m： 要使用的模块名称。

-a, --args： 制定模块所需的参数。

-u： 制定连接的用户名。

-h, --help 显示帮助内容。

-v, --verbose 以详细信息模式运行命令，可以用来调试错误。

更多选项和说明详见 Ansible 官方文档。下面我们介绍几个简单实例示范：

使用 ping 模块检查与客户机的连接性，请执行以下操作：

```
ansible all -m ping
```

在上面的命令中，all 指定 Ansible 应该在所有主机上运行此命令，也可以按照分组比如 nodes 或者主机 node1 等。执行结果如下：

```
node4 | UNREACHABLE! => {
  "changed": false,
  "msg": "Failed to connect to the host via ssh: ssh_exchange_identification: Connection
  "unreachable": true
}
[DEPRECATION WARNING]: Distribution Ubuntu 18.04 on host node1 should be removed in version 2.12. Deprecation warnings can be disabled by setting
compatibility with prior Ansible releases. A future Ansible release will be removed in version 2.12. Deprecation warnings can be disabled by setting
host. See https://docs.ansible.com/ansible/2.9/reference_appendices/compatibility-table.
node1 | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python"
  },
  "changed": false,
  "ping": "pong"
}
node2 | UNREACHABLE! => {
  "changed": false,
  "msg": "Failed to connect to the host via ssh: ssh_exchange_identification: Connection
  "unreachable": true
}
```

可见 node1 是通的，而 node2 和 node4 不通。

使用 ad-hoc 命令管理软件包

使用 Ansible 的 ad-hoc 命令，可以给客户端安装软件包。我们只需执行一个单行命令，然后实现安装。

比如：在分组客户机安装 Apache 只需要执行：

```
ansible web -m yum yum -a "name=httpd state=present" -b
```

使用 ad-hoc 命令管理服务

上面我们给 web 组的客户机批量安装了 Apache 服务器，下面我们来看怎么启动 httpd 服务。

启动 httpd 服务只需要执行：

```
ansible web -b -m service -a "name=httpd enabled=yes"
```

要启动，重新启动和停止该服务，只需 state 参数的值更改为 started 以启动服务，restarted 以重新启动服务，stop 来停止服务：

```
ansible webservers -b -m service -a "name=httpd state=started"
```

Ansible ad-hoc 命令非常出色，适合于运行单个任务。具体任务模块可以参考官方文档。

Ansible Playbook



Playbook 是 Ansible 提供的最强大的任务执行方法。与 ad-hoc 命令不同，Playbooks 配置在文件中，可以重用和共享给其他人。

playbooks 是以 YAML 标记语言来定义的。每个 playbook 由一个或多个 play 组成。play 的目标是将一组主机映射到任务中去。每个 play 包含一个或多个任务，这些任务每次执行一次。

下面是一个简单的 Ansible httpd.yaml playbook，用于给 web 组上安装 Apache 服务器：

```
---
- hosts: web
  remote_user: ansible
  tasks:
  - name: Ensure apache is installed and updated
    yum:
      name: httpd
      state: latest
    become: yes
```

执行 playbook 使用 ansible-playbook 命令，格式如下：

```
ansible-playbook -i <hostfile> <playbook.yaml>
```


ansible-playbook httpd.yaml 结果:

```
PLAY [web] *****

TASK [Gathering Facts] *****
ok: [node2]
ok: [node3]

TASK [Ensure apache is installed and updated] *****
changed: [node2]
changed: [node3]

PLAY RECAP *****
node2 : ok=2 changed=1 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0
node3 : ok=2 changed=1 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0
```

百家号/虫虫搜奇

下面是一个综合多个任务的多个 play 的 playbook，实现安装 apache 服务器。

在 Web 服务器组中启用并启动 httpd 服务。

在所有客户机上安装 git。

其内容如下:

```
- hosts: web
remote_user: ansible
become: yes
tasks:
- name: Installing apache
yum:
name: httpd
state: latest
- name: Enabling httpd service
service:
name: httpd
enabled: yes
notify:
- name: restart httpd
```

```
handlers:
- name: restart httpd
service:
name: httpd
state: restarted
- hosts: all
remote_user: ansible
become: yes
tasks:
- name: Installing git
yum:
name: git
state: latest
```

总结

Ansible 是一个方便快捷又功能强大的自动化执行和配置管理工具。本文只是 Ansible 的入门教程，介绍了 Ansible 的基本概念、优势、安装，配置和简单的使用。师傅领进门修行靠自己，Ansible 更强大的功能和实现自己基于它的全任务自动化需要自己使用探索。当然前面也提到了网络上尤其是 Github 上已经有大量 Ansible 的技术积累 (ansible-role-xxxx) 可供大家学习参考。

作者最新文章

自动化运维工具——【ansible 详解 一】

- [自动化运维工具——【ansible...](#)

[weixin_45555050](#): [code=html] BatchShell 这款国产自动化运维软件更简单易用，类似 Xshell+Ansible,可以去官网 www.batchshell.cn 下载最新版本试试 官网描述的功能特点： 1)无学习成本，支持原生 Shell,也支持 Ansible、SaltStack 脚本 2)内含文件编辑器，支持本地、远程文件在线编辑，替换、保存、对比 3)支持文件本地版本控制，实现文件修改历史回溯 4)支持跨主机文件右键拷贝、复制功能 5)支持跨网络多主机文件同步与命令执行 6)支持本地、远程文件检索功能 7)支持可视化任务调度，包括文件同步、命令执行 [/code]

- [ansible 简介](#)
 - [ansible 是什么?](#)
 - [ansible 特点](#)
 - [ansible 架构图](#)
- [ansible 任务执行](#)
 - [ansible 任务执行模式](#)
 - [ansible 执行流程](#)

- ansible 命令执行过程
- ansible 配置详解
 - ansible 安装方式
 - 使用 pip (python 的包管理模块) 安装
 - 使用 yum 安装
 - ansible 程序结构
 - ansible 配置文件查找顺序
 - ansible 配置文件
 - ansible 主机清单
- ansible 常用命令
 - ansible 命令集
 - ansible-doc 命令
 - ansible 命令详解
 - ansible 配置公私钥
- ansible 常用模块
 - 1) 主机连通性测试
 - 2) command 模块
 - 3) shell 模块
 - 4) copy 模块
 - 5) file 模块
 - 6) fetch 模块
 - 7) cron 模块
 - 8) yum 模块
 - 9) service 模块
 - 10) user 模块
 - 11) group 模块
 - 12) script 模块
 - 13) setup 模块

ansible 简介:
ansible 是什么?

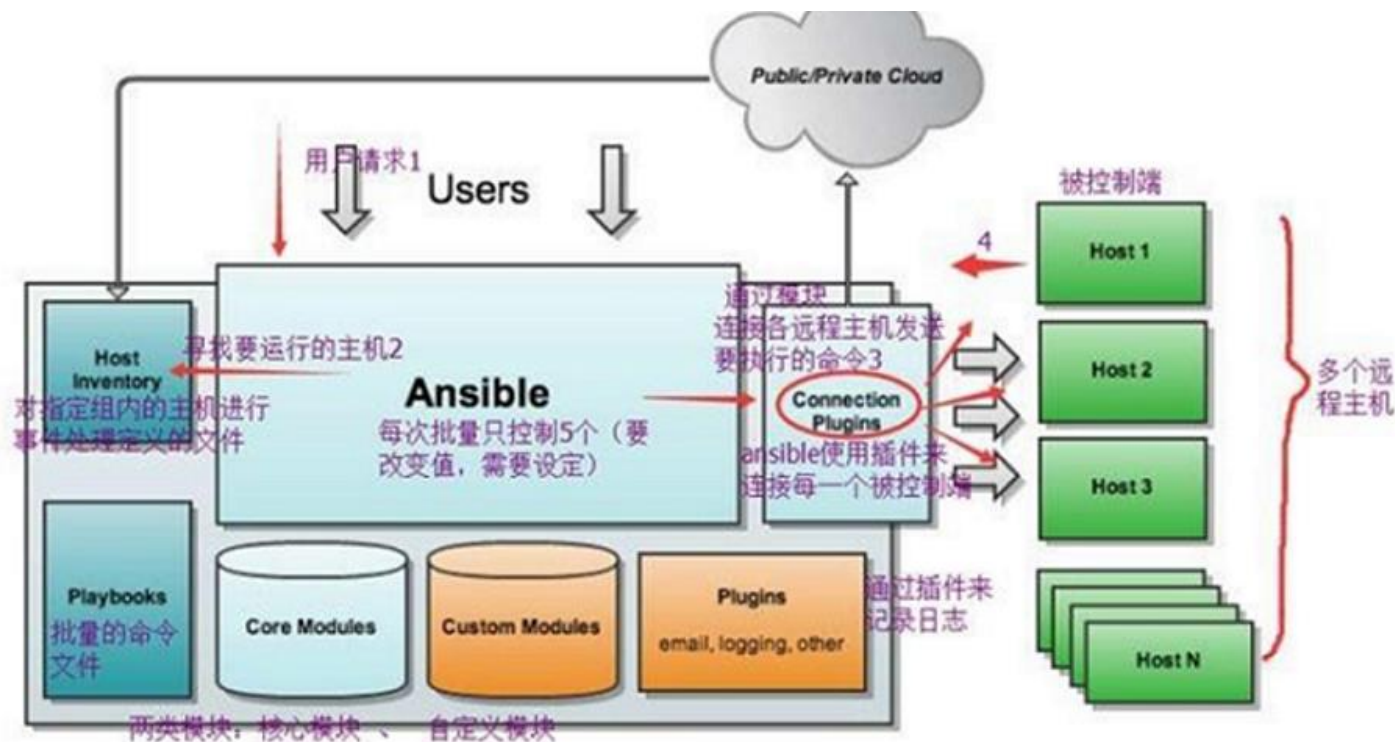
ansible 是新出现的自动化运维工具，基于 Python 开发，集合了众多运维工具（puppet、chef、func、fabric）的优点，实现了批量系统配置、批量程序部署、批量运行命令等功能。

ansible 是基于 paramiko 开发的,并且基于模块化工作，本身没有批量部署的能力。真正具有批量部署的是 ansible 所运行的模块，ansible 只是提供一种框架。ansible 不需要在远程主机上安装 client/agents，因为它们是基于 ssh 来和远程主机通讯的。ansible 目前已经已经被红帽官方收购，是自动化运维工具中大家认可度最高的，并且上手容易，学习简单。是每位运维工程师必须掌握的技能之一。

ansible 特点

1. 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；
2. 默认使用 SSH 协议对设备进行管理；
3. 有大量常规运维操作模块，可实现日常绝大部分操作；
4. 配置简单、功能强大、扩展性强；
5. 支持 API 及自定义模块，可通过 Python 轻松扩展；
6. 通过 Playbooks 来定制强大的配置、状态管理；
7. 轻量级，无需在客户端安装 agent，更新时，只需在操作机上进行一次更新即可；
8. 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。

ansible 架构图



上图中我们看到的主要模块如下：

Ansible: Ansible 核心程序。

HostInventory: 记录由 Ansible 管理的主机信息，包括端口、密码、ip 等。

Playbooks: “剧本” YAML 格式文件，多个任务定义在一个文件中，定义主机需要调用哪些模块来完成的功能。

CoreModules: 核心模块，主要操作是通过调用核心模块来完成管理任务。

CustomModules: 自定义模块，完成核心模块无法完成的功能，支持多种语言。

ConnectionPlugins: 连接插件，Ansible 和 Host 通信使用

ansible 任务执行 ansible 任务执行模式

Ansible 系统由控制主机对被管节点的操作方式可分为两类，即 adhoc 和 playbook：

- ad-hoc 模式(点对点模式)

使用单个模块，支持批量执行单条命令。ad-hoc 命令是一种可以快速输入的命令，而且不需要保存起来的命令。就相当于 bash 中的一句话 shell。

- playbook 模式(剧本模式)

是 Ansible 主要管理方式，也是 Ansible 功能强大的关键所在。playbook 通过多个 task 集合完成一类功能，如 Web 服务的安装部署、数据库服务器的批量备份等。可以简单地把 playbook 理解为通过组合多条 ad-hoc 操作的配置文件。

ansible 执行流程



简单理解就是 Ansible 在运行时，首先读取 `ansible.cfg` 中的配置，根据规则获取 Inventory 中的管理主机列表，并行的在这些主机中执行配置的任务，最后等待执行返回的结果。

ansible 命令执行过程

1. 加载自己的配置文件，默认 `/etc/ansible/ansible.cfg`;
2. 查找对应的主机配置文件，找到要执行的主机或者组;
3. 加载自己对应的模块文件，如 `command`;

4. 通过 ansible 将模块或命令生成对应的临时 py 文件(python 脚本), 并将该文件传输至远程服务器;
5. 对应执行用户的家目录的 .ansible/tmp/XXX/XXX.PY 文件;
6. 给文件 +x 执行权限;
7. 执行并返回结果;
8. 删除临时 py 文件, sleep 0 退出;

ansible 配置详解

ansible 安装方式

ansible 安装常用两种方式, yum 安装和 pip 程序安装。下面我们来详细介绍一下这两种安装方式。

使用 pip (python 的包管理模块) 安装

首先, 我们需要安装一个 python-pip 包, 安装完成以后, 则直接使用 pip 命令来安装我们的包, 具体操作过程如下:

1. yum install python-pip
2. pip install ansible

使用 yum 安装

yum 安装是我们很熟悉的安装方式了。我们需要先安装一个 epel-release 包, 然后再安装我们的 ansible 即可。

1. yum install epel-release -y
2. yum install ansible -y

ansible 程序结构

安装目录如下(yum 安装):

配置文件目录: /etc/ansible/

执行文件目录: /usr/bin/

Lib 库依赖目录: /usr/lib/pythonX.X/site-packages/ansible/

Help 文档目录: /usr/share/doc/ansible-X.X.X/

Man 文档目录: /usr/share/man/man1/

ansible 配置文件查找顺序

ansible 与我们其他的服务在这一点上有很大不同, 这里的配置文件查找是从多个地方找的, 顺序如下:

1. 检查环境变量 ANSIBLE_CONFIG 指向的路径文件(export ANSIBLE_CONFIG=/etc/ansible.cfg);
2. ~/.ansible.cfg, 检查当前目录下的 ansible.cfg 配置文件;
3. /etc/ansible.cfg 检查 etc 目录的配置文件。

ansible 配置文件

ansible 的配置文件为/etc/ansible/ansible.cfg, ansible 有许多参数, 下面我们列出一些常见的参数:

1. `inventory = /etc/ansible/hosts` #这个参数表示资源清单 `inventory` 文件的位置
2. `library = /usr/share/ansible` #指向存放 Ansible 模块的目录, 支持多个目录方式, 只要用冒号 (:) 隔开就可以
3. `forks = 5` #并发连接数, 默认为 5
4. `sudo_user = root` #设置默认执行命令的用户
5. `remote_port = 22` #指定连接被管节点的管理端口, 默认为 22 端口, 建议修改, 能够更加安全
6. `host_key_checking = False` #设置是否检查 SSH 主机的密钥, 值为 True/False。关闭后第一次连接不会提示配置实例
7. `timeout = 60` #设置 SSH 连接的超时时间, 单位为秒
8. `log_path = /var/log/ansible.log` #指定一个存储 ansible 日志的文件 (默认不记录日志)

ansible 主机清单

在配置文件中, 我们提到了资源清单, 这个清单就是我们的主机清单, 里面保存的是一些 ansible 需要连接管理的主机列表。我们可以来看看他的定义方式:

1. 1、 直接指明主机地址或主机名:
2. `## green.example.com#`
3. `# blue.example.com#`
4. `# 192.168.100.1`
5. `# 192.168.100.10`
6. 2、 定义一个主机组[组名]把地址或主机名加进去
7. `[mysql_test]`
8. `192.168.253.159`
9. `192.168.253.160`
10. `192.168.253.153`

需要注意的是, 这里的组成员可以使用通配符来匹配, 这样对于一些标准化的管理来说就很轻松方便了。

我们可以根据实际情况来配置我们的主机列表, 具体操作如下:

1. `[root@server ~]# vim /etc/ansible/hosts`
2. `[web]`
3. `192.168.37.122`
4. `192.168.37.133`

ansible 常用命令

ansible 命令集

```
/usr/bin/ansible      Ansible AD-Hoc 临时命令执行工具，常用于临时命令的执行
/usr/bin/ansible-doc   Ansible 模块功能查看工具
/usr/bin/ansible-galaxy 下载/上传优秀代码或 Roles 模块 的官网平台，基于网络的
/usr/bin/ansible-playbook  Ansible 定制自动化的任务集编排工具
/usr/bin/ansible-pull   Ansible 远程执行命令的工具，拉取配置而非推送配置（使用较少，海量机器时使用，对运维的架构能力要求较高）
/usr/bin/ansible-vault  Ansible 文件加密工具
/usr/bin/ansible-console  Ansible 基于 Linux Console 界面可与用户交互的命令执行工具
```

其中，我们比较常用的是 `/usr/bin/ansible` 和 `/usr/bin/ansible-playbook`。

ansible-doc 命令

`ansible-doc` 命令常用于获取模块信息及其使用帮助，一般用法如下：

1. `ansible-doc -l` #获取全部模块的信息
2. `ansible-doc -s MOD_NAME` #获取指定模块的使用帮助

我们也可以查看一下 `ansible-doc` 的全部用法：

1. `[root@server ~]# ansible-doc`
2. Usage: `ansible-doc [options] [module...]`
- 3.
4. Options:
5. `-h, --help` show this help message and exit # 显示命令参数 API 文档
6. `-l, --list` List available modules #列出可用的模块
7. `-M MODULE_PATH, --module-path=MODULE_PATH` #指定模块的路径
8. specify path(s) to module library (default=None)
9. `-s, --snippet` Show playbook snippet for specified module(s) #显示 playbook 制定模块的用法
10. `-v, --verbose` verbose mode (-vvv for more, -vvvv to enable # 显示 ansible-doc 的版本号查看模块列表:
11. connection debugging)
12. `--version` show program's version number and exit

我们可以来看一下，以 `mysql` 相关的为例：

1. [root@server ~]# ansible-doc -l |grep mysql
2. mysql_db Add or remove MySQL databases from a remote...
3. mysql_replication Manage MySQL replication
4. mysql_user Adds or removes a user from a MySQL databas...
5. mysql_variables Manage MySQL global variables
6. [root@server ~]# ansible-doc -s mysql_user

```
[root@server ~]# ansible-doc -s mysql_user
- name: Adds or removes a user from a MySQL database.
  action: mysql_user
    append_privs          # Append the privileges defined by priv to the
                        # existing ones for
                        # this user instead of
                        # overwriting existing
                        # ones.
    check_implicit_admin  # Check if mysql allows login as root/nopassword
                        # before trying
                        # supplied
                        # credentials.
    config_file           # Specify a config file from which user and password
                        # are to be read
    connect_timeout       # The connection timeout when connecting to the MySQL
                        # server.
    encrypted            # Indicate that the 'password' field is a
                        # `mysql_native_passwo
                        # rd` hash
    host                  # the 'host' part of the MySQL username
    host_all              # override the host option, making ansible apply
                        # changes to all
                        # hostnames for a
: ...skipping...
```

ansible 命令详解

命令的具体格式如下：

```
ansible <host-pattern> [-f forks] [-m module_name] [-a args]
```

也可以通过 ansible -h 来查看帮助，下面我们列出一些比较常用的选项，并解释其含义：

-a MODULE_ARGS #模块的参数，如果执行默认 COMMAND 的模块，即是命令参数，如：“date”，“pwd”等等

-k, --ask-pass #ask for SSH password。登录密码，提示输入 SSH 密码而不是假设基于密钥的验证

--ask-su-pass #ask for su password。su 切换密码

-K, --ask-sudo-pass #ask for sudo password。提示密码使用 sudo, sudo 表示提权操作
--ask-vault-pass #ask for vault password。假设我们设定了加密的密码, 则用该选项进行访问
-B SECONDS #后台运行超时时间
-C #模拟运行环境并进行预运行, 可以进行查错测试
-c CONNECTION #连接类型使用
-f FORKS #并行任务数, 默认为 5
-i INVENTORY #指定主机清单的路径, 默认为/etc/ansible/hosts
--list-hosts #查看有哪些主机组
-m MODULE_NAME #执行模块的名字, 默认使用 command 模块, 所以如果是只执行单一命令可以不用 -m 参数
-o #压缩输出, 尝试将所有结果在一行输出, 一般针对收集工具使用
-S #用 su 命令
-R SU_USER #指定 su 的用户, 默认为 root 用户
-s #用 sudo 命令
-U SUDO_USER #指定 sudo 到哪个用户, 默认为 root 用户
-T TIMEOUT #指定 ssh 默认超时时间, 默认为 10s, 也可在配置文件中修改
-u REMOTE_USER #远程用户, 默认为 root 用户
-v #查看详细信息, 同时支持 -vvv, -vvvv 可查看更详细信息

ansible 配置公私钥

上面我们已经提到过 ansible 是基于 ssh 协议实现的, 所以其配置公私钥的方式与 ssh 协议的方式相同, 具体操作步骤如下:

1. #1.生成私钥
2. [root@server ~]# ssh-keygen
3. #2.向主机分发私钥
4. [root@server ~]# ssh-copy-id root@192.168.37.122
5. [root@server ~]# ssh-copy-id root@192.168.37.133

这样的话, 就可以实现无密码登录, 我们的实验过程也会顺畅很多。

注意, 如果出现了一下报错:

```
-bash: ssh-copy-id: command not found
```

那么就证明我们需要安装一个包:

```
yum -y install openssh-clientsansible
```

把包安装上即可。

[回到顶部](#)

ansible 常用模块

1) 主机连通性测试

我们使用 `ansible web -m ping` 命令来进行主机连通性测试，效果如下：

```
1. [root@server ~]# ansible web -m ping
2. 192.168.37.122 | SUCCESS => {
3. "changed": false,
4. "ping": "pong"
5. }
6. 192.168.37.133 | SUCCESS => {
7. "changed": false,
8. "ping": "pong"
9. }
```

这样就说明我们的主机是连通状态的。接下来的操作才可以正常进行。

2) command 模块

这个模块可以直接在远程主机上执行命令，并将结果返回本主机。

举例如下：

```
1. [root@server ~]# ansible web -m command -a 'ss -ntl'
2. 192.168.37.122 | SUCCESS | rc=0 >>
3. State Recv-Q Send-Q Local Address:Port Peer Address:Port
4. LISTEN 0 128 *:111 *:
5. LISTEN 0 5 192.168.122.1:53 *:
6. LISTEN 0 128 *:22 *:
7. LISTEN 0 128 127.0.0.1:631 *:
8. LISTEN 0 128 *:23000 *:
9. LISTEN 0 100 127.0.0.1:25 *:
10. LISTEN 0 128 :::111 :::
11. LISTEN 0 128 :::22 :::
12. LISTEN 0 128 ::1:631 :::
```

```

13.LISTEN 0 100 ::1:25 :::*
14.
15.192.168.37.133 | SUCCESS | rc=0 >>
16.State Recv-Q Send-Q Local Address:Port Peer Address:Port
17.LISTEN 0 128 *:111 *:*
18.LISTEN 0 128 *:22 *:*
19.LISTEN 0 128 127.0.0.1:631 *:*
20.LISTEN 0 128 *:23000 *:*
21.LISTEN 0 100 127.0.0.1:25 *:*
22.LISTEN 0 128 :::111 :::*
23.LISTEN 0 128 :::22 :::*
24.LISTEN 0 128 ::1:631 :::*
25.LISTEN 0 100 ::1:25 :::*

```

命令模块接受命令名称，后面是空格分隔的列表参数。给定的命令将在所有选定的节点上执行。它不会通过 shell 进行处理，比如\$HOME 和操作如"<", ">", "|", ";", "&" 工作（需要使用（shell）模块实现这些功能）。注意，该命令不支持 | 管道命令。

下面来看一看该模块下常用的几个命令：

```

chdir          # 在执行命令之前，先切换到该目录
executable # 切换 shell 来执行命令，需要使用命令的绝对路径
free_form     # 要执行的 Linux 指令，一般使用 Ansible 的-a 参数代替。
creates      # 一个文件名，当这个文件存在，则该命令不执行,可以用来做判断
removes      # 一个文件名，这个文件不存在，则该命令不执行

```

下面我们来看看这些命令的执行效果：

1. [root@server ~]# ansible web -m command -a 'chdir=/data/ ls' #先切换到/data/ 目录，再执行“ls”命令
2. 192.168.37.122 | SUCCESS | rc=0 >>
3. aaa.jpg
4. fastdfs
5. mogdata
6. tmp
7. web

```
8. wKgleloeYoCAMLtZAAWEekAtkc497.jpg
9.
10.192.168.37.133 | SUCCESS | rc=0 >>
11.aaa.jpg
12.fastdfs
13.mogdata
14.tmp
15.web
16.wKgleloeYoCAMLtZAAWEekAtkc497.jpg
```

```
1. [root@server ~]# ansible web -m command -a 'creates=/data/aaa.jpg ls' #如果/data/aaa.jpg 存在，则不执行“ls”
   命令
2. 192.168.37.122 | SUCCESS | rc=0 >>
3. skipped, since /data/aaa.jpg exists
4.
5. 192.168.37.133 | SUCCESS | rc=0 >>
6. skipped, since /data/aaa.jpg exists
```

```
1. [root@server ~]# ansible web -m command -a 'removes=/data/aaa.jpg cat /data/a' #如果/data/aaa.jpg 存在，则
   执行“cat /data/a”命令
2. 192.168.37.122 | SUCCESS | rc=0 >>
3. hello
4.
5. 192.168.37.133 | SUCCESS | rc=0 >>
6. hello
```

3) shell 模块

shell 模块可以在远程主机上调用 shell 解释器运行命令，支持 shell 的各种功能，例如管道等。

```
1. [root@server ~]# ansible web -m shell -a 'cat /etc/passwd |grep "keer"'
2. 192.168.37.122 | SUCCESS | rc=0 >>
3. keer:x:10001:1000:keer:/home/keer:/bin/sh
```

- 4.
 5. 192.168.37.133 | SUCCESS | rc=0 >>
 6. keer:x:10001:10001:~/home/keer:/bin/sh
- 只要是我们的 shell 命令，都可以通过这个模块在远程主机上运行，这里就不一一举例了。

4) copy 模块

这个模块用于将文件复制到远程主机，同时支持给定内容生成文件和修改权限等。

其相关选项如下：

| | |
|----------------|--|
| src | #被复制到远程主机的本地文件。可以是绝对路径，也可以是相对路径。如果路径是一个目录，则会递归复制，用法类似于"rsync" |
| content | #用于替换"src"，可以直接指定文件的值 |
| dest | #必选项，将源文件复制到的远程主机的绝对路径 |
| backup | #当文件内容发生改变后，在覆盖之前把源文件备份，备份文件包含时间信息 |
| directory_mode | #递归设定目录的权限，默认为系统默认权限 |
| force | #当目标主机包含该文件，但内容不同时，设为"yes"，表示强制覆盖；设为"no"，表示目标主机的目标位置不存在该文件才复制。默认为"yes" |
| others | #所有的 file 模块中的选项可以在这里使用 |

用法举例如下：

① 复制文件：

1. [root@server ~]# ansible web -m copy -a 'src=~/.hello dest=/data/hello'
2. 192.168.37.122 | SUCCESS => {
3. "changed": true,
4. "checksum": "22596363b3de40b06f981fb85d82312e8c0ed511",
5. "dest": "/data/hello",
6. "gid": 0,
7. "group": "root",
8. "md5sum": "6f5902ac237024bdd0c176cb93063dc4",
9. "mode": "0644",
10. "owner": "root",
11. "size": 12,
12. "src": "/root/.ansible/tmp/ansible-tmp-1512437093.55-228281064292921/source",
13. "state": "file",


```
14."uid": 0
15.}
16.192.168.37.133 | SUCCESS => {
17."changed": true,
18."checksum": "22596363b3de40b06f981fb85d82312e8c0ed511",
19."dest": "/data/hello",
20."gid": 0,
21."group": "root",
22."md5sum": "6f5902ac237024bdd0c176cb93063dc4",
23."mode": "0644",
24."owner": "root",
25."size": 12,
26."src": "/root/.ansible/tmp/ansible-tmp-1512437093.74-44694985235189/source",
27."state": "file",
28."uid": 0
29.}
```

② 给定内容生成文件，并制定权限

```
1. [root@server ~]# ansible web -m copy -a 'content="I am keer\n" dest=/data/name mode=666'
2. 192.168.37.122 | SUCCESS => {
3. "changed": true,
4. "checksum": "0421570938940ea784f9d8598dab87f07685b968",
5. "dest": "/data/name",
6. "gid": 0,
7. "group": "root",
8. "md5sum": "497fa8386590a5fc89090725b07f175c",
9. "mode": "0666",
10."owner": "root",
11."size": 10,
12."src": "/root/.ansible/tmp/ansible-tmp-1512437327.37-199512601767687/source",
13."state": "file",
```

```
14."uid": 0
15.}
16.192.168.37.133 | SUCCESS => {
17."changed": true,
18."checksum": "0421570938940ea784f9d8598dab87f07685b968",
19."dest": "/data/name",
20."gid": 0,
21."group": "root",
22."md5sum": "497fa8386590a5fc89090725b07f175c",
23."mode": "0666",
24."owner": "root",
25."size": 10,
26."src": "/root/.ansible/tmp/ansible-tmp-1512437327.55-218104039503110/source",
27."state": "file",
28."uid": 0
29.}
```

我们现在可以去查看一下我们生成的文件及其权限：

```
1. [root@server ~]# ansible web -m shell -a 'ls -l /data/'
2. 192.168.37.122 | SUCCESS | rc=0 >>
3. total 28
4. -rw-rw-rw- 1 root root 12 Dec 6 09:45 name
5.
6. 192.168.37.133 | SUCCESS | rc=0 >>
7. total 40
8. -rw-rw-rw- 1 root root 12 Dec 5 09:45 name
```

可以看出我们的 name 文件已经生成，并且权限为 666。

③ 关于覆盖

我们把文件的内容修改一下，然后选择覆盖备份：

```
1. [root@server ~]# ansible web -m copy -a 'content="I am keerya\n" backup=yes dest=/data/name mode=666'
```

```
2. 192.168.37.122 | SUCCESS => {
3. "backup_file": "/data/name.4394.2017-12-06@09:46:25~",
4. "changed": true,
5. "checksum": "064a68908ab9971ee85dbc08ea038387598e3778",
6. "dest": "/data/name",
7. "gid": 0,
8. "group": "root",
9. "md5sum": "8ca7c11385856155af52e560f608891c",
10."mode": "0666",
11."owner": "root",
12."size": 12,
13."src": "/root/.ansible/tmp/ansible-tmp-1512438383.78-228128616784888/source",
14."state": "file",
15."uid": 0
16.}
17.192.168.37.133 | SUCCESS => {
18."backup_file": "/data/name.5962.2017-12-05@09:46:24~",
19."changed": true,
20."checksum": "064a68908ab9971ee85dbc08ea038387598e3778",
21."dest": "/data/name",
22."gid": 0,
23."group": "root",
24."md5sum": "8ca7c11385856155af52e560f608891c",
25."mode": "0666",
26."owner": "root",
27."size": 12,
28."src": "/root/.ansible/tmp/ansible-tmp-1512438384.0-170718946740009/source",
29."state": "file",
30."uid": 0
31.}
```

现在我们可以去查看一下：

```
1. [root@server ~]# ansible web -m shell -a 'ls -l /data/'
2. 192.168.37.122 | SUCCESS | rc=0 >>
3. total 28
4. -rw-rw-rw- 1 root root 12 Dec 6 09:46 name
5. -rw-rw-rw- 1 root root 10 Dec 6 09:45 name.4394.2017-12-06@09:46:25~
6.
7. 192.168.37.133 | SUCCESS | rc=0 >>
8. total 40
9. -rw-rw-rw- 1 root root 12 Dec 5 09:46 name
10.-rw-rw-rw- 1 root root 10 Dec 5 09:45 name.5962.2017-12-05@09:46:24~
```

可以看出，我们的源文件已经被备份，我们还可以查看一下 name 文件的内容：

```
1. [root@server ~]# ansible web -m shell -a 'cat /data/name'
2. 192.168.37.122 | SUCCESS | rc=0 >>
3. I am keerya
4.
5. 192.168.37.133 | SUCCESS | rc=0 >>
6. I am keerya
```

证明，这正是我们新导入的文件的内容。

5) file 模块

该模块主要用于设置文件的属性，比如创建文件、创建链接文件、删除文件等。

下面是一些常见的命令：

```
force      #需要在两种情况下强制创建软链接，一种是源文件不存在，但之后会建立的情况下；另一种是目标软链接已存在，需要先取消之前的软链，然后创建新的软链，有两个选项：yes|no
group      #定义文件/目录的属组。后面可以加上 mode：定义文件/目录的权限
owner      #定义文件/目录的属主。后面必须跟上 path：定义文件/目录的路径
recurse    #递归设置文件的属性，只对目录有效，后面跟上 src：被链接的源文件路径，只应用于 state=link 的情况
dest       #被链接到的路径，只应用于 state=link 的情况
state      #状态，有以下选项：
```

directory: 如果目录不存在, 就创建目录

file: 即使文件不存在, 也不会被创建

link: 创建软链接

hard: 创建硬链接

touch: 如果文件不存在, 则会创建一个新的文件, 如果文件或目录已存在, 则更新其最后修改时间

absent: 删除目录、文件或者取消链接文件

用法举例如下:

① 创建目录:

```
1. [root@server ~]# ansible web -m file -a 'path=/data/app state=directory'
2. 192.168.37.122 | SUCCESS => {
3. "changed": true,
4. "gid": 0,
5. "group": "root",
6. "mode": "0755",
7. "owner": "root",
8. "path": "/data/app",
9. "size": 6,
10."state": "directory",
11."uid": 0
12.}
13.192.168.37.133 | SUCCESS => {
14."changed": true,
15."gid": 0,
16."group": "root",
17."mode": "0755",
18."owner": "root",
19."path": "/data/app",
20."size": 4096,
21."state": "directory",
22."uid": 0
```


23.}

我们可以查看一下：

```
1. [root@server ~]# ansible web -m shell -a 'ls -l /data'
2. 192.168.37.122 | SUCCESS | rc=0 >>
3. total 28
4. drwxr-xr-x 2 root root 6 Dec 6 10:21 app
5.
6. 192.168.37.133 | SUCCESS | rc=0 >>
7. total 44
8. drwxr-xr-x 2 root root 4096 Dec 5 10:21 app
```

可以看出，我们的目录已经创建完成。

② 创建链接文件

```
1. [root@server ~]# ansible web -m file -a 'path=/data/bbb.jpg src=aaa.jpg state=link'
2. 192.168.37.122 | SUCCESS => {
3. "changed": true,
4. "dest": "/data/bbb.jpg",
5. "gid": 0,
6. "group": "root",
7. "mode": "0777",
8. "owner": "root",
9. "size": 7,
10. "src": "aaa.jpg",
11. "state": "link",
12. "uid": 0
13.}
14. 192.168.37.133 | SUCCESS => {
15. "changed": true,
16. "dest": "/data/bbb.jpg",
17. "gid": 0,
```

```
18."group": "root",
19."mode": "0777",
20."owner": "root",
21."size": 7,
22."src": "aaa.jpg",
23."state": "link",
24."uid": 0
25.}
```

我们可以去查看一下:

```
1. [root@server ~]# ansible web -m shell -a 'ls -l /data'
2. 192.168.37.122 | SUCCESS | rc=0 >>
3. total 28
4. -rw-r--r-- 1 root root 5649 Dec 5 13:49 aaa.jpg
5. lrwxrwxrwx 1 root root 7 Dec 6 10:25 bbb.jpg -> aaa.jpg
6.
7. 192.168.37.133 | SUCCESS | rc=0 >>
8. total 44
9. -rw-r--r-- 1 root root 5649 Dec 4 14:44 aaa.jpg
10.lrw-rwxrwx 1 root root 7 Dec 5 10:25 bbb.jpg -> aaa.jpg
```

我们的链接文件已经创建成功。

③ 删除文件

```
1. [root@server ~]# ansible web -m file -a 'path=/data/a state=absent'
2. 192.168.37.122 | SUCCESS => {
3. "changed": true,
4. "path": "/data/a",
5. "state": "absent"
6. }
7. 192.168.37.133 | SUCCESS => {
8. "changed": true,
```

```
9. "path": "/data/a",
10."state": "absent"
11.}
```

我们可以查看一下：

```
1. [root@server ~]# ansible web -m shell -a 'ls /data/a'
2. 192.168.37.122 | FAILED | rc=2 >>
3. ls: cannot access /data/a: No such file or directory
4.
5. 192.168.37.133 | FAILED | rc=2 >>
6. ls: cannot access /data/a: No such file or directory
```

发现已经没有这个文件了。

6) fetch 模块

该模块用于从远程某主机获取（复制）文件到本地。

有两个选项：

dest：用来存放文件的目录

src：在远程拉取的文件，并且必须是一个 file，不能是目录

具体举例如下：

```
1. [root@server ~]# ansible web -m fetch -a 'src=/data/hello dest=/data'
2. 192.168.37.122 | SUCCESS => {
3. "changed": true,
4. "checksum": "22596363b3de40b06f981fb85d82312e8c0ed511",
5. "dest": "/data/192.168.37.122/data/hello",
6. "md5sum": "6f5902ac237024bdd0c176cb93063dc4",
7. "remote_checksum": "22596363b3de40b06f981fb85d82312e8c0ed511",
8. "remote_md5sum": null
9. }
10.192.168.37.133 | SUCCESS => {
11."changed": true,
```

```
12."checksum": "22596363b3de40b06f981fb85d82312e8c0ed511",
13."dest": "/data/192.168.37.133/data/hello",
14."md5sum": "6f5902ac237024bdd0c176cb93063dc4",
15."remote_checksum": "22596363b3de40b06f981fb85d82312e8c0ed511",
16."remote_md5sum": null
17.}
```

我们可以在本机上查看一下文件是否复制成功。要注意，文件保存的路径是我们设置的接收目录下的被管制主机 ip 目录下：

```
1. [root@server ~]# cd /data/
2. [root@server data]# ls
3. 1 192.168.37.122 192.168.37.133 fastdfs web
4. [root@server data]# cd 192.168.37.122
5. [root@server 192.168.37.122]# ls
6. data
7. [root@server 192.168.37.122]# cd data/
8. [root@server data]# ls
9. hello
10.[root@server data]# pwd
11./data/192.168.37.122/data
```

7) cron 模块

该模块适用于管理 cron 计划任务的。

其使用的语法跟我们的 crontab 文件中的语法一致，同时，可以指定以下选项：

```
day= #日应该运行的工作( 1-31, , /2, )
hour= # 小时 ( 0-23, , /2, )
minute= #分钟( 0-59, , /2, )
month= # 月( 1-12, *, /2, )
weekday= # 周 ( 0-6 for Sunday-Saturday,, )
job= #指明运行的命令是什么
name= #定时任务描述
reboot # 任务在重启时运行，不建议使用，建议使用 special_time
special_time #特殊的时间范围，参数：reboot（重启时），annually（每年），monthly（每月），weekly（每周），daily（每天），hourly
```

(每小时)

state #指定状态, present 表示添加定时任务, 也是默认设置, absent 表示删除定时任务

user # 以哪个用户的身份执行

举例如下:

① 添加计划任务

```
1. [root@server ~]# ansible web -m cron -a 'name="ntp update every 5 min" minute=*/5 job="/sbin/ntpdate 172.17.0.1 &> /dev/null"'
2. 192.168.37.122 | SUCCESS => {
3. "changed": true,
4. "envs": [],
5. "jobs": [
6. "ntp update every 5 min"
7. ]
8. }
9. 192.168.37.133 | SUCCESS => {
10."changed": true,
11."envs": [],
12."jobs": [
13."ntp update every 5 min"
14.]
15.}
```

我们可以去查看一下:

```
1. [root@server ~]# ansible web -m shell -a 'crontab -l'
2. 192.168.37.122 | SUCCESS | rc=0 >>
3. #Ansible: ntp update every 5 min
4. */5 * * * * /sbin/ntpdate 172.17.0.1 &> /dev/null
5.
6. 192.168.37.133 | SUCCESS | rc=0 >>
7. #Ansible: ntp update every 5 min
```

```
8. */5 * * * * /sbin/ntpdate 172.17.0.1 &> /dev/null
```

可以看出，我们的计划任务已经设置成功了。

② 删除计划任务

如果我们的计划任务添加错误，想要删除的话，则执行以下操作：

首先我们查看一下现有的计划任务：

```
1. [root@server ~]# ansible web -m shell -a 'crontab -l'
2. 192.168.37.122 | SUCCESS | rc=0 >>
3. #Ansible: ntp update every 5 min
4. */5 * * * * /sbin/ntpdate 172.17.0.1 &> /dev/null
5. #Ansible: df everyday
6. * 15 * * * df -lh >> /tmp/disk_total &> /dev/null
7.
8. 192.168.37.133 | SUCCESS | rc=0 >>
9. #Ansible: ntp update every 5 min
10.*/5 * * * * /sbin/ntpdate 172.17.0.1 &> /dev/null
11.#Ansible: df everyday
12.* 15 * * * df -lh >> /tmp/disk_total &> /dev/null
```

然后执行删除操作：

```
1. [root@server ~]# ansible web -m cron -a 'name="df everyday" hour=15 job="df -lh >> /tmp/disk_total &> /dev/null" state=absent'
2. 192.168.37.122 | SUCCESS => {
3. "changed": true,
4. "envs": [],
5. "jobs": [
6. "ntp update every 5 min"
7. ]
8. }
9. 192.168.37.133 | SUCCESS => {
10."changed": true,
```

```
11."envs": [],
12."jobs": [
13."ntp update every 5 min"
14.]
15.}
```

删除完成后，我们再查看一下现有的计划任务确认一下：

```
1. [root@server ~]# ansible web -m shell -a 'crontab -l'
2. 192.168.37.122 | SUCCESS | rc=0 >>
3. #Ansible: ntp update every 5 min
4. */5 * * * * /sbin/ntpdate 172.17.0.1 &> /dev/null
5.
6. 192.168.37.133 | SUCCESS | rc=0 >>
7. #Ansible: ntp update every 5 min
8. */5 * * * * /sbin/ntpdate 172.17.0.1 &> /dev/null
```

我们的删除操作已经成功。

8) yum 模块

顾名思义，该模块主要用于软件的安装。

其选项如下：

```
name=      #所安装的包的名称
state=     #present--->安装, latest--->安装最新的, absent---> 卸载软件。
update_cache    #强制更新 yum 的缓存
conf_file      #指定远程 yum 安装时所依赖的配置文件（安装本地已有的包）。
disable_pgp_check  #是否禁止 GPG checking, 只用于 presentor latest。
disablerepo    #临时禁止使用 yum 库。 只用于安装或更新时。
enablerepo     #临时使用的 yum 库。只用于安装或更新时。
```

下面我们就来安装一个包试试看：

```
1. [root@server ~]# ansible web -m yum -a 'name=htop state=present'
2. 192.168.37.122 | SUCCESS => {
3. "changed": true,
```



```
4. "msg": "",
5. "rc": 0,
6. "results": [
7. "Loaded plugins: fastestmirror, langpacks\nLoading mirror speeds from cached hostfile\nResolving
Dependencies\n--> Running transaction check\n--> Package htop.x86_64 0:2.0.2-1.el7 will be installed\n-
-> Finished Dependency Resolution\n\nDependencies
Resolved\n\n=====
Arch Version Repository
Size\n\n=====
Installing:\n
htop x86_64 2.0.2-1.el7 epel 98 k\n\nTransaction
Summary\n\n=====
Install 1
Package\n\nTotal download size: 98 k\nInstalled size: 207 k\nDownloading packages:\nRunning transaction
check\nRunning transaction test\nTransaction test succeeded\nRunning transaction\n Installing : htop-
2.0.2-1.el7.x86_64 1/1 \n Verifying : htop-2.0.2-1.el7.x86_64 1/1 \n\nInstalled:\n htop.x86_64 0:2.0.2-
1.el7 \n\nComplete!\n"
8. ]
9. }
10.192.168.37.133 | SUCCESS => {
11."changed": true,
12."msg": "Warning: RPMDB altered outside of yum.\n** Found 3 pre-existing rpmdb problem(s), 'yum check'
output follows:\nnipa-client-4.4.0-12.el7.centos.x86_64 has installed conflicts freeipa-client: ipa-
client-4.4.0-12.el7.centos.x86_64\nipa-client-common-4.4.0-12.el7.centos.noarch has installed conflicts
freeipa-client-common: ipa-client-common-4.4.0-12.el7.centos.noarch\nipa-common-4.4.0-
12.el7.centos.noarch has installed conflicts freeipa-common: ipa-common-4.4.0-12.el7.centos.noarch\n",
13."rc": 0,
14."results": [
15."Loaded plugins: fastestmirror, langpacks\nLoading mirror speeds from cached hostfile\nResolving
Dependencies\n--> Running transaction check\n--> Package htop.x86_64 0:2.0.2-1.el7 will be installed\n-
-> Finished Dependency Resolution\n\nDependencies
Resolved\n\n=====
Arch Version Repository
```

```
Size\n=====\\nInstalling:\\n
htop x86_64 2.0.2-1.el7 epel 98 k\\n\\nTransaction
Summary\n=====\\nInstall 1
Package\\n\\nTotal download size: 98 k\\nInstalled size: 207 k\\nDownloading packages:\\nRunning transaction
check\\nRunning transaction test\\nTransaction test succeeded\\nRunning transaction\\n Installing : htop-
2.0.2-1.el7.x86_64 1/1 \\n Verifying : htop-2.0.2-1.el7.x86_64 1/1 \\n\\nInstalled:\\n htop.x86_64 0:2.0.2-
1.el7 \\n\\nComplete!\\n"
```

16.]

17.}

安装成功。

9) service 模块

该模块用于服务程序的管理。

其主要选项如下：

arguments #命令行提供额外的参数

enabled #设置开机启动。

name= #服务名称

runlevel #开机启动的级别，一般不用指定。

sleep #在重启服务的过程中，是否等待。如在服务关闭以后等待 2 秒再启动。（定义在剧本中。）

state #有四种状态，分别为：started--->启动服务， stopped--->停止服务， restarted--->重启服务， reloaded--->重载配置

下面是一些例子：

① 开启服务并设置自启动

```
1. [root@server ~]# ansible web -m service -a 'name=nginx state=started enabled=true'
2. 192.168.37.122 | SUCCESS => {
3. "changed": true,
4. "enabled": true,
5. "name": "nginx",
6. "state": "started",
7. .....
8. }
9. 192.168.37.133 | SUCCESS => {
```

```
10."changed": true,  
11."enabled": true,  
12."name": "nginx",  
13."state": "started",  
14.....  
15.}
```

我们可以去查看一下端口是否打开：

```
1. [root@server ~]# ansible web -m shell -a 'ss -ntl'  
2. 192.168.37.122 | SUCCESS | rc=0 >>  
3. State Recv-Q Send-Q Local Address:Port Peer Address:Port  
4. LISTEN 0 128 *:80 *: *  
5.  
6. 192.168.37.133 | SUCCESS | rc=0 >>  
7. State Recv-Q Send-Q Local Address:Port Peer Address:Port  
8. LISTEN 0 128 *:80 *: *
```

可以看出我们的 80 端口已经打开。

② 关闭服务

我们也可以通过该模块来关闭我们的服务：

```
1. [root@server ~]# ansible web -m service -a 'name=nginx state=stopped'  
2. 192.168.37.122 | SUCCESS => {  
3. "changed": true,  
4. "name": "nginx",  
5. "state": "stopped",  
6. ....  
7. }  
8. 192.168.37.133 | SUCCESS => {  
9. "changed": true,  
10."name": "nginx",  
11."state": "stopped",
```

12.....

13.}

一样的，我们来查看一下端口：

```
1. [root@server ~]# ansible web -m shell -a 'ss -ntl | grep 80'
```

```
2. 192.168.37.122 | FAILED | rc=1 >>
```

```
3.
```

```
4. 192.168.37.133 | FAILED | rc=1 >>
```

可以看出，我们已经没有 80 端口了，说明我们的 nginx 服务已经关闭了。

10) user 模块

该模块主要是用来管理用户账号。

其主要选项如下：

```
comment      # 用户的描述信息
createhome    # 是否创建家目录
force         # 在使用 state=absent 时，行为与 userdel -force 一致。
group         # 指定基本组
groups        # 指定附加组，如果指定为(groups=)表示删除所有组
home          # 指定用户家目录
move_home     # 如果设置为 home=时，试图将用户主目录移动到指定的目录
name          # 指定用户名
non_unique    # 该选项允许改变非唯一的用户 ID 值
password      # 指定用户密码
remove        # 在使用 state=absent 时，行为是与 userdel -remove 一致
shell         # 指定默认 shell
state         # 设置帐号状态，不指定为创建，指定值为 absent 表示删除
system        # 当创建一个用户，设置这个用户是系统用户。这个设置不能更改现有用户
uid           # 指定用户的 uid
```

举例如下：

① 添加一个用户并指定其 uid

```
1. [root@server ~]# ansible web -m user -a 'name=keer uid=11111'
```

```
2. 192.168.37.122 | SUCCESS => {
3. "changed": true,
4. "comment": "",
5. "createhome": true,
6. "group": 11111,
7. "home": "/home/keer",
8. "name": "keer",
9. "shell": "/bin/bash",
10."state": "present",
11."stderr": "useradd: warning: the home directory already exists.\nNot copying any file from skel
    directory into it.\nCreating mailbox file: File exists\n",
12."system": false,
13."uid": 11111
14.}
15.192.168.37.133 | SUCCESS => {
16."changed": true,
17."comment": "",
18."createhome": true,
19."group": 11111,
20."home": "/home/keer",
21."name": "keer",
22."shell": "/bin/bash",
23."state": "present",
24."stderr": "useradd: warning: the home directory already exists.\nNot copying any file from skel
    directory into it.\nCreating mailbox file: File exists\n",
25."system": false,
26."uid": 11111
27.}
```

添加完成，我们可以去查看一下：

```
1. [root@server ~]# ansible web -m shell -a 'cat /etc/passwd |grep keer'
```

```
2. 192.168.37.122 | SUCCESS | rc=0 >>
3. keer:x:11111:11111::/home/keer:/bin/bash
4.
5. 192.168.37.133 | SUCCESS | rc=0 >>
6. keer:x:11111:11111::/home/keer:/bin/bash
```

② 删除用户

```
1. [root@server ~]# ansible web -m user -a 'name=keer state=absent'
2. 192.168.37.122 | SUCCESS => {
3. "changed": true,
4. "force": false,
5. "name": "keer",
6. "remove": false,
7. "state": "absent"
8. }
9. 192.168.37.133 | SUCCESS => {
10."changed": true,
11."force": false,
12."name": "keer",
13."remove": false,
14."state": "absent"
15.}
```

一样的，删除之后，我们去看一下：

```
1. [root@server ~]# ansible web -m shell -a 'cat /etc/passwd |grep keer'
2. 192.168.37.122 | FAILED | rc=1 >>
3.
4. 192.168.37.133 | FAILED | rc=1 >>
```

发现已经没有这个用户了。

11) group 模块

该模块主要用于添加或删除组。

常用的选项如下：

```
gid=      #设置组的 GID 号
name=     #指定组的名称
state=    #指定组的状态，默认为创建，设置值为 absent 为删除
system=   #设置值为 yes，表示创建为系统组
```

举例如下：

① 创建组

```
1. [root@server ~]# ansible web -m group -a 'name=sanguo gid=12222'
2. 192.168.37.122 | SUCCESS => {
3. "changed": true,
4. "gid": 12222,
5. "name": "sanguo",
6. "state": "present",
7. "system": false
8. }
9. 192.168.37.133 | SUCCESS => {
10."changed": true,
11."gid": 12222,
12."name": "sanguo",
13."state": "present",
14."system": false
15.}
```

创建过后，我们来查看一下：

```
1. [root@server ~]# ansible web -m shell -a 'cat /etc/group | grep 12222'
2. 192.168.37.122 | SUCCESS | rc=0 >>
3. sanguo:x:12222:
4.
5. 192.168.37.133 | SUCCESS | rc=0 >>
```


6. sanguo:x:12222:

可以看出，我们的组已经创建成功了。

② 删除组

```
1. [root@server ~]# ansible web -m group -a 'name=sanguo state=absent'
2. 192.168.37.122 | SUCCESS => {
3. "changed": true,
4. "name": "sanguo",
5. "state": "absent"
6. }
7. 192.168.37.133 | SUCCESS => {
8. "changed": true,
9. "name": "sanguo",
10."state": "absent"
11.}
```

照例查看一下：

```
1. [root@server ~]# ansible web -m shell -a 'cat /etc/group | grep 12222'
2. 192.168.37.122 | FAILED | rc=1 >>
3.
4. 192.168.37.133 | FAILED | rc=1 >>
```

已经没有这个组的相关信息了。

12) script 模块

该模块用于将本机的脚本在被管理端的机器上运行。

该模块直接指定脚本的路径即可，我们通过例子来看一看到底如何使用的：

首先，我们写一个脚本，并给其加上执行权限：

```
1. [root@server ~]# vim /tmp/df.sh
2. #!/bin/bash
3.
4. date >> /tmp/disk_total.log
```

```
5. df -lh >> /tmp/disk_total.log
```

```
6. [root@server ~]# chmod +x /tmp/df.sh
```

然后，我们直接运行命令来实现在被管理端执行该脚本：

```
1. [root@server ~]# ansible web -m script -a '/tmp/df.sh'
```

```
2. 192.168.37.122 | SUCCESS => {
```

```
3. "changed": true,
```

```
4. "rc": 0,
```

```
5. "stderr": "Shared connection to 192.168.37.122 closed.\r\n",
```

```
6. "stdout": "",
```

```
7. "stdout_lines": []
```

```
8. }
```

```
9. 192.168.37.133 | SUCCESS => {
```

```
10."changed": true,
```

```
11."rc": 0,
```

```
12."stderr": "Shared connection to 192.168.37.133 closed.\r\n",
```

```
13."stdout": "",
```

```
14."stdout_lines": []
```

```
15.}
```

照例查看一下文件内容：

```
1. [root@server ~]# ansible web -m shell -a 'cat /tmp/disk_total.log'
```

```
2. 192.168.37.122 | SUCCESS | rc=0 >>
```

```
3. Tue Dec 5 15:58:21 CST 2017
```

```
4. Filesystem Size Used Avail Use% Mounted on
```

```
5. /dev/sda2 47G 4.4G 43G 10% /
```

```
6. devtmpfs 978M 0 978M 0% /dev
```

```
7. tmpfs 993M 84K 993M 1% /dev/shm
```

```
8. tmpfs 993M 9.1M 984M 1% /run
```

```
9. tmpfs 993M 0 993M 0% /sys/fs/cgroup
```

```
10./dev/sda3 47G 33M 47G 1% /app
```

```
11./dev/sda1 950M 153M 798M 17% /boot
12.tmpfs 199M 16K 199M 1% /run/user/42
13.tmpfs 199M 0 199M 0% /run/user/0
14.
15.192.168.37.133 | SUCCESS | rc=0 >>
16.Tue Dec 5 15:58:21 CST 2017
17.Filesystem Size Used Avail Use% Mounted on
18./dev/sda2 46G 4.1G 40G 10% /
19.devtmpfs 898M 0 898M 0% /dev
20.tmpfs 912M 84K 912M 1% /dev/shm
21.tmpfs 912M 9.0M 903M 1% /run
22.tmpfs 912M 0 912M 0% /sys/fs/cgroup
23./dev/sda3 3.7G 15M 3.4G 1% /app
24./dev/sda1 1.9G 141M 1.6G 9% /boot
25.tmpfs 183M 16K 183M 1% /run/user/42
26.tmpfs 183M 0 183M 0% /run/user/0
```

可以看出已经执行成功了。

13) setup 模块

该模块主要用于收集信息，是通过调用 facts 组件来实现的。

facts 组件是 Ansible 用于采集被管机器设备信息的一个功能，我们可以使用 setup 模块查机器的所有 facts 信息，可以使用 filter 来查看指定信息。整个 facts 信息被包装在一个 JSON 格式的数据结构中，ansible_facts 是最上层的值。

facts 就是变量，内建变量。每个主机的各种信息，cpu 颗数、内存大小等。会存在 facts 中的某个变量中。调用后返回很多对应主机的信息，在后面的操作中可以根据不同的信息来做不同的操作。如 redhat 系列用 yum 安装，而 debian 系列用 apt 来安装软件。

① 查看信息

我们可以直接用命令获取到变量的值，具体我们来看看例子：

```
1. [root@server ~]# ansible web -m setup -a 'filter="*mem*"' #查看内存
2. 192.168.37.122 | SUCCESS => {
3. "ansible_facts": {
4. "ansible_memfree_mb": 1116,
5. "ansible_memory_mb": {
```

```
6. "nocache": {
7. "free": 1397,
8. "used": 587
9. },
10."real": {
11."free": 1116,
12."total": 1984,
13."used": 868
14.},
15."swap": {
16."cached": 0,
17."free": 3813,
18."total": 3813,
19."used": 0
20.}
21.},
22."ansible_memtotal_mb": 1984
23.},
24."changed": false
25.}
26.192.168.37.133 | SUCCESS => {
27."ansible_facts": {
28."ansible_memfree_mb": 1203,
29."ansible_memory_mb": {
30."nocache": {
31."free": 1470,
32."used": 353
33.},
34."real": {
35."free": 1203,
36."total": 1823,
```

```
37."used": 620
38.},
39."swap": {
40."cached": 0,
41."free": 3813,
42."total": 3813,
43."used": 0
44.}
45.},
46."ansible_memtotal_mb": 1823
47.},
48."changed": false
49.}
```

我们可以通过命令查看一下内存的大小以确认一下是否一致：

```
1. [root@server ~]# ansible web -m shell -a 'free -m'
2. 192.168.37.122 | SUCCESS | rc=0 >>
3. total used free shared buff/cache available
4. Mem: 1984 404 1122 9 457 1346
5. Swap: 3813 0 3813
6.
7. 192.168.37.133 | SUCCESS | rc=0 >>
8. total used free shared buff/cache available
9. Mem: 1823 292 1207 9 323 1351
10.Swap: 3813 0 3813
```

可以看出信息是一致的。

② 保存信息

我们的 setup 模块还有一个很好用的功能就是可以保存我们所筛选的信息至我们的主机上，同时，文件名为我们被管制的主机的 IP，这样方便我们知道是哪台机器出的问题。

我们可以看一看例子：

```
1. [root@server tmp]# ansible web -m setup -a 'filter="*mem*" --tree /tmp/facts
2. 192.168.37.122 | SUCCESS => {
3. "ansible_facts": {
4. "ansible_memfree_mb": 1115,
5. "ansible_memory_mb": {
6. "nocache": {
7. "free": 1396,
8. "used": 588
9. },
10. "real": {
11. "free": 1115,
12. "total": 1984,
13. "used": 869
14. },
15. "swap": {
16. "cached": 0,
17. "free": 3813,
18. "total": 3813,
19. "used": 0
20. }
21. },
22. "ansible_memtotal_mb": 1984
23. },
24. "changed": false
25. }
26. 192.168.37.133 | SUCCESS => {
27. "ansible_facts": {
28. "ansible_memfree_mb": 1199,
29. "ansible_memory_mb": {
30. "nocache": {
31. "free": 1467,
```

```
32."used": 356
33.},
34."real": {
35."free": 1199,
36."total": 1823,
37."used": 624
38.},
39."swap": {
40."cached": 0,
41."free": 3813,
42."total": 3813,
43."used": 0
44.}
45.},
46."ansible_memtotal_mb": 1823
47.},
48."changed": false
49.}
```

自动化运维工具——ansible 详解（二）

Ansible playbook 简介

playbook 是 ansible 用于配置，部署，和管理被控节点的剧本。

- 通过 playbook 的详细描述，执行其中的一系列 tasks，可以让远端主机达到预期的状态。playbook 就像 Ansible 控制器给被控节点列出的的一系列 to-do-list，而被控节点必须要完成。
- 也可以这么理解，playbook 字面意思，即剧本，现实中由演员按照剧本表演，在 Ansible 中，这次由计算机进行表演，由计算机安装，部署应用，提供对外服务，以及组织计算机处理各种各样的事情。

Ansible playbook 使用场景

- 执行一些简单的任务，使用 ad-hoc 命令可以方便的解决问题，但是有时一个设施过于复杂，需要大量的操作时候，执行的 ad-hoc 命令是不适合的，这时最好使用 playbook。

- 就像执行 shell 命令与写 shell 脚本一样，也可以理解为批处理任务，不过 playbook 有自己的语法格式。
- 使用 playbook 你可以方便的重用这些代码，可以移植到不同的机器上面，像函数一样，最大化的利用代码。在你使用 Ansible 的过程中，你也会发现，你所处理的大部分操作都是编写 playbook。可以把常见的应用都编写成 playbook，之后管理服务器会变得十分简单。

Ansible playbook 格式

1) 格式简介

playbook 由 YAML 语言编写。YAML(/'jæmə/)参考了其他多种语言，包括：XML、C 语言、Python、Perl 以及电子邮件格式 RFC2822，Clark Evans 在 2001 年 5 月在首次发表了这种语言，另外 Ingy döt Net 与 OrenBen-Kiki 也是这语言的共同设计者。

YAML 格式是类似于 JSON 的文件格式，便于人理解和阅读，同时便于书写。首先学习了解一下 YAML 的格式，对我们后面书写 playbook 很有帮助。以下为 playbook 常用到的 YAML 格式：

- 1、文件的第一行应该以 “—” (三个连字符)开始，表明 YAML 文件的开始。
 - 2、在同一行中，#之后的内容表示注释，类似于 shell，python 和 ruby。
 - 3、YAML 中的列表元素以“ -” 开头然后紧跟着一个空格，后面为元素内容。
 - 4、同一个列表中的元素应该保持相同的缩进。否则会被当做错误处理。
 - 5、play 中 hosts, variables, roles, tasks 等对象的表示方法都是键值中间以": “分隔表示，” :”后面还要增加一个空格。
- 下面是一个举例：

#安装与运行 mysql 服务

```
- hosts: node1
  remote_user: root
  tasks:

    - name: install mysql-server package
      yum: name=mysql-server state=present
    - name: starting mysqld service
      service: name=mysql state=started
```

我们的文件名称应该以.yml 结尾，像我们上面的例子就是 mysql.yml。其中，有三个部分组成：

host 部分：使用 hosts 指示使用哪个主机或主机组来运行下面的 tasks，每个 playbook 都必须指定 hosts，hosts 也可以使用通配符格式。主机或主机组在 inventory 清单中指定，可以使用系统默认的/etc/ansible/hosts，也可以自己编辑，在运行的时候加上-i 选项，指定清单的位置即可。在运行清单文件的时候，-list-hosts 选项会显示那些主机将会参与执行 task 的过程中。

remote_user：指定远端主机中的哪个用户来登录远端系统，在远端系统执行 task 的用户，可以任意指定，也可以使用 sudo，但是用户必须要有执行相应 task 的权限。

tasks: 指定远端主机将要执行的一系列动作。tasks 的核心为 ansible 的模块，前面已经提到模块的用法。tasks 包含 name 和要执行的模块，name 是可选的，只是为了便于用户阅读，不过还是建议加上去，模块是必须的，同时也要给予模块相应的参数。

使用 ansible-playbook 运行 playbook 文件，得到如下输出信息，输出内容为 JSON 格式。并且由不同颜色组成，便于识别。一般而言

- | 绿色代表执行成功，系统保持原样
- | 黄色代表系统代表系统状态发生改变
- | 红色代表执行失败，显示错误输出

执行有三个步骤：1、收集 facts 2、执行 tasks 3、报告结果

2) 核心元素

Playbook 的核心元素：

Hosts: 主机组；

Tasks: 任务列表；

Variables: 变量，设置方式有四种；

Templates: 包含了模板语法的文本文件；

Handlers: 由特定条件触发的任务；

3) 基本组件

Playbooks 配置文件的基础组件：

Hosts: 运行指定任务的目标主机

remote_user: 在远程主机上执行任务的用户；

sudo_user:

tasks: 任务列表

格式：

tasks:

- ```
- name: TASK_NAME
 module: arguments
 notify: HANDLER_NAME
 handlers:
- name: HANDLER_NAME
 module: arguments
```

模块，模块参数：

格式：

(1) action: module arguments

## (2) module: arguments

注意：shell 和 command 模块后面直接跟命令，而非 key=value 类的参数列表；

handlers：任务，在特定条件下触发；接收到其它任务的通知时被触发；

(1) 某任务的状态在运行后为 changed 时，可通过 “notify” 通知给相应的 handlers；

(2) 任务可以通过 “tags” 打标签，而后可在 ansible-playbook 命令上使用 -t 指定进行调用；

举例

### ① 定义 playbook

```
[root@server ~]# cd /etc/ansible
[root@server ansible]# vim nginx.yml

- hosts: web
 remote_user: root
 tasks:

 - name: install nginx
 yum: name=nginx state=present
 - name: copy nginx.conf
 copy: src=/tmp/nginx.conf dest=/etc/nginx/nginx.conf backup=yes
 notify: reload #当 nginx.conf 发生改变时，通知给相应的 handlers
 tags: reloadnginx #打标签
 - name: start nginx service
 service: name=nginx state=started
 tags: startnginx #打标签

handlers: #注意，前面没有-，是两个空格
 - name: reload
 service: name=nginx state=restarted #为了在进程中能看出来
```

### ② 测试运行结果

写完了以后，我们就可以运行了：

```
[root@server ansible]# ansible-playbook nginx.yml
```

```

[root@server ansible]# ansible-playbook nginx.yml

PLAY [web] *****

TASK [setup] *****
ok: [192.168.37.122]
ok: [192.168.37.133]

TASK [install nginx] *****
ok: [192.168.37.122]
ok: [192.168.37.133] 提示ok表明之前安装过nginx服务

TASK [copy nginx.conf] *****
ok: [192.168.37.122]
ok: [192.168.37.133]

TASK [start nginx service] *****
ok: [192.168.37.122]
ok: [192.168.37.133]

PLAY RECAP *****
192.168.37.122 : ok=4 changed=0 unreachable=0 failed=0
192.168.37.133 : ok=4 changed=0 unreachable=0 failed=0

```

现在我们可以看看两台机器的端口是否开启：

```

[root@server ansible]# ansible web -m shell -a 'ss -nutlp |grep nginx'
192.168.37.122 | SUCCESS | rc=0 >>
tcp LISTEN 0 128 *:80 *:~
users: (("nginx",pid=8304,fd=6),("nginx",pid=8303,fd=6))

192.168.37.133 | SUCCESS | rc=0 >>
tcp LISTEN 0 128 *:80 *:~
users: (("nginx",pid=9671,fd=6),("nginx",pid=9670,fd=6))

```

### ③ 测试标签

我们在里面已经打上了一个标签，所以可以直接引用标签。但是我们需要先把服务关闭，再来运行剧本并引用标签：

```
[root@server ansible]# ansible web -m shell -a 'systemctl stop nginx'
```

```
[root@server ansible]# ansible-playbook nginx.yml -t startnginx
```

```

[root@server ansible]# ansible-playbook nginx.yml -t startnginx

PLAY [web] *****

TASK [setup] *****
ok: [192.168.37.122]
ok: [192.168.37.133]

TASK [start nginx service] *****
ok: [192.168.37.122]
ok: [192.168.37.133]

PLAY RECAP *****
192.168.37.122 : ok=2 changed=0 unreachable=0 failed=0
192.168.37.133 : ok=2 changed=0 unreachable=0 failed=0

```

直接到了开启服务这里

#### ④ 测试 notify

我们还做了一个 notify，来测试一下：

首先，它的触发条件是配置文件被改变，所以我们去把配置文件中的端口改一下：

```
[root@server ansible]# vim /tmp/nginx.conf
```

```
listen 8080;
```

然后我们重新加载一下这个剧本：

```

[root@server ansible]# ansible-playbook nginx.yml -t reloadnginx

PLAY [web] *****

TASK [setup] *****
ok: [192.168.37.122]
ok: [192.168.37.133]

TASK [copy nginx.conf] *****
changed: [192.168.37.122]
changed: [192.168.37.133]

RUNNING HANDLER [reload] *****
changed: [192.168.37.122]
changed: [192.168.37.133]

PLAY RECAP *****
192.168.37.122 : ok=3 changed=2 unreachable=0 failed=0
192.168.37.133 : ok=3 changed=2 unreachable=0 failed=0

```

发现我们执行的就是 reload 段以及我们定义的 notify 部分。

我们来看一看我们的端口号：

```

[root@server ansible]# ansible web -m shell -a 'ss -ntlp | grep nginx'
192.168.37.122 | SUCCESS | rc=0 >>
LISTEN 0 128 *:8080 *:8080
users:(("nginx",pid=2097,fd=6),("nginx",pid=2096,fd=6))

192.168.37.133 | SUCCESS | rc=0 >>
LISTEN 0 128 *:8080 *:8080
users:(("nginx",pid=3061,fd=6),("nginx",pid=3060,fd=6))

```

可以看出，我们的 nginx 端口已经变成了 8080。

#### 4) variables 部分

上文中，我们说到了 variables 是变量，有四种定义方法，现在我们就来说说这四种定义方法：

① facts : 可直接调用

上一篇中，我们有说到 setup 这个模块，这个模块就是通过调用 facts 组件来实现的。我们这里的 variables 也可以直接调用 facts 组件。具体的 facts 我们可以使用 setup 模块来获取，然后直接放入我们的剧本中调用即可。

## ② 用户自定义变量

我们也可以直接使用用户自定义变量，想要自定义变量有以下两种方式：

## 通过命令行传入

ansible-playbook 命令的命令行中的-e VARS, --extra-vars=VARS, 这样就可以直接把自定义的变量传入。

## 在 playbook 中定义变量

我们也可以直接在 `playbook` 中定义我们的变量:

vars:

- var1: value1
- - var2: value2

## 举例

### ① 定义剧本

我们就使用全局替换把我们刚刚编辑的文件修改一下：

```
[root@server ansible]# vim nginx.yml
```

```
--
- hosts: web
 remote_user: root
 tasks:
 - name: install nginx
 yum: name=nginx state=present
 - name: copy nginx.conf
 copy: src=/tmp/nginx.conf dest=/etc/nginx/nginx.conf backup=yes
 notify: reload
 tags: reloadnginx
 - name: start nginx service
 service: name=nginx state=started
 tags: startnginx

 handlers:
 - name: reload
 service: name=nginx state=restarted
~
~
~
~
~
~
```

使用全局替换，把所有的nginx替换为变量

```
:% s/nginx/\{\{ rpmname\ }\}/g
```

<https://blog.csdn.net/wq962464>



```

- hosts: web
 remote_user: root
 tasks:
 - name: install {{ rpmname }}
 yum: name={{ rpmname }} state=present
 - name: copy {{ rpmname }}.conf
 copy: src=/tmp/{{ rpmname }}.conf dest=/etc/{{ rpmname }}/{{ rpmname }}.conf
 backup=yes
 notify: reload
 tags: reload{{ rpmname }}
 - name: start {{ rpmname }} service
 service: name={{ rpmname }} state=started
 tags: start{{ rpmname }}

 handlers:
 - name: reload
 service: name={{ rpmname }} state=restarted
~
~
~
~
11 substitutions on 9 lines

```

替换过后的效果

<https://blog.csdn.net/wq962464>  
 17,7 A11

## ② 拷贝配置文件

我们想要在被监管的机器上安装什么服务的话，就直接在我们的 server 端上把该服务的配置文件拷贝到我们的/tmp/目录下。这样我们的脚本才能正常运行。

我们就以 keepalived 服务为例：

```
[root@server ansible]# cp /etc/keepalived/keepalived.conf /tmp/keepalived.conf
```

## ③ 运行剧本，变量由命令行传入

```
[root@server ansible]# ansible-playbook nginx.yml -e rpmname=keepalived
```

```

[root@server ansible]# ansible-playbook nginx.yml -e rpmname=keepalived
PLAY [web] *****
TASK [setup] *****
ok: [192.168.37.122]
ok: [192.168.37.133]
TASK [install keepalived] *****
ok: [192.168.37.122]
changed: [192.168.37.133]
TASK [copy keepalived.conf] *****
changed: [192.168.37.122]
ok: [192.168.37.133]
TASK [start keepalived service] *****
changed: [192.168.37.122]
changed: [192.168.37.133]
RUNNING HANDLER [reload] *****
changed: [192.168.37.122]
PLAY RECAP *****
192.168.37.122 : ok=5 changed=3 unreachable=0 failed=0
192.168.37.133 : ok=4 changed=2 unreachable=0 failed=0

```

运行脚本并传入变量

执行成功

#### ④ 修改剧本，直接定义变量

同样的，我们可以直接在剧本中把变量定义好，这样就不需要在通过命令行传入了。以后想要安装不同的服务，直接在剧本里把变量修改一下即可。

```
[root@server ansible]# vim nginx.yml
```

```

- hosts: web
 remote_user: root
 vars:
 - rpmname: keepalived 直接定义变量
 tasks:
 - name: install {{ rpmname }}
 yum: name={{ rpmname }} state=present
 - name: copy {{ rpmname }}.conf
 copy: src=/tmp/{{ rpmname }}.conf dest=/etc/{{ rpmname }}/{{ rpmname }}.conf
 backup=yes
 notify: reload
 tags: reload{{ rpmname }}
 - name: start {{ rpmname }} service
 service: name={{ rpmname }} state=started
 tags: start{{ rpmname }}

 handlers:
 - name: reload
 service: name={{ rpmname }} state=restarted
```

<https://blog.csdn.net/wq962464>

#### ⑤ 运行定义过变量的剧本

我们刚刚已经把变量定义在剧本里面了。现在我们来运行一下试试看：

```
[root@server ansible]# ansible-playbook nginx.yml
```

```

[root@server ansible]# ansible-playbook nginx.yml

PLAY [web] *****

TASK [setup] *****
ok: [192.168.37.122]
ok: [192.168.37.133]

TASK [install keepalived] *****
ok: [192.168.37.122]
ok: [192.168.37.133]

TASK [copy keepalived.conf] *****
ok: [192.168.37.122]
ok: [192.168.37.133]

TASK [start keepalived service] *****
ok: [192.168.37.122]
ok: [192.168.37.133]

PLAY RECAP *****
192.168.37.122 : ok=4 changed=0 unreachable=0 failed=0
192.168.37.133 : ok=4 changed=0 unreachable=0 failed=0

```

<https://blog.csdn.net/wq962464>

### ③ 通过 roles 传递变量

具体的，我们下文中说到 roles 的时候再详细说明。这里是传送带

### ④ Host Inventory

我们也可以直接在主机清单中定义。

定义的方法如下：

向不同的主机传递不同的变量：

IP/HOSTNAME variable=value var2=value2

向组中的主机传递相同的变量：

[groupname:vars]

variable=value

5) 模板 templates

模板是一个文本文件，嵌套有脚本（使用模板编程语言编写）。

Jinja2: Jinja2 是 python 的一种模板语言，以 Django 的模板语言为原本。

模板支持：

字符串：使用单引号或双引号；

数字：整数，浮点数；

列表：[item1, item2, ...]

元组：(item1, item2, ...)

字典：{key1:value1, key2:value2, ...}

布尔型：true/false

算术运算：

+, -, \*, /, //, %, \*\*

比较操作：

==, !=, >, >=, <, <=

逻辑运算：

and, or, not

通常来说，模板都是通过引用变量来运用的。

举例

### ① 定义模板

我们直接把之前定义的/tmp/nginx.conf 改个名，然后编辑一下，就可以定义成我们的模板文件了：

```
[root@server ansible]# cd /tmp
[root@server tmp]# mv nginx.conf nginx.conf.j2
[root@server tmp]# vim nginx.conf.j2
worker_processes {{ ansible_processor_vcpus }};
listen {{ nginxport }};
```

### ② 修改剧本

我们现在需要去修改剧本来定义变量：

```
[root@server ansible]# vim nginx.yml
```

```

- hosts: web
 remote_user: root
 vars:
 - rpmname: nginx
 nginxport: 8888
 tasks:
 - name: install {{ rpmname }}
 yum: name={{ rpmname }} state=present
 - name: copy {{ rpmname }}.conf
 copy: src=/tmp/{{ rpmname }}.conf.j2 dest=/etc/{{ rpmname }}/{{ rpmname }}.co
nf backup=yes
 notify: reload
 tags: reload{{ rpmname }}
 - name: start {{ rpmname }} service
 service: name={{ rpmname }} state=started
 tags: start{{ rpmname }}
 handlers:
 - name: reload
 service: name={{ rpmname }} state=restarted

```

需要修改的部分都圈起来了

<https://blog.csdn.net/wq962464>

### ③ 运行剧本

上面的准备工作完成后，我们就可以去运行剧本了：

```
[root@server ansible]# ansible-playbook nginx.yml -t reloadnginx
```

```
PLAY [web] *****
```

```
TASK [setup] *****
```

```
ok: [192.168.37.122]
```

```
ok: [192.168.37.133]
```

```
TASK [copy nginx.conf] *****
```

```
ok: [192.168.37.122]
```

```
ok: [192.168.37.133]
```

```
PLAY RECAP *****
```

```
192.168.37.122 : ok=2 changed=0 unreachable=0 failed=0
```

```
192.168.37.133 : ok=2 changed=0 unreachable=0 failed=0
```

## 6) 条件测试

when 语句：在 task 中使用，jinja2 的语法格式。

举例如下：

```
> tasks:
```

```
> - name: install conf file to centos7 template: src=files/nginx.conf.c7.j2 when: ansible_distribution_major_version
```

```
> == "7"
```

```
> - name: install conf file to centos6 template: src=files/nginx.conf.c6.j2 when: ansible_distribution_major_version
```

```
> == "6"
```

循环：迭代，需要重复执行的任务；

对迭代项的引用，固定变量名为"item"，而后，要在 task 中使用 with\_items 给定要迭代的元素列表；

举例如下：

```
tasks:
```

```
- name: unsta1l web packages
```

```
 yum: name={{ item }} state=absent
```

```
 with_items:
```

```
 - httpd
```

```
 - php
```

```
 - php-mysql
```

## 7) 字典

ansible playbook 还支持字典功能。举例如下：

```
- name: install some packages
```

```
 yum: name={{ item }} state=present
```

```
 with_items:
```

```
 - nginx
```

```
 - memcached
```

```
 - php-fpm
```

```
- name: add some groups
```

```
 group: name={{ item }} state=present
```

```

with_items:
 - group11
 - group12
 - group13
- name: add some users
 user: name={{ item.name }} group={{ item.group }} state=present
 with_items:
 - { name: 'user11', group: 'group11' }
 - { name: 'user12', group: 'group12' }
 - { name: 'user13', group: 'group13' }

```

## 8) 角色订制: roles

### ① 简介

对于以上所有的方式有个弊端就是无法实现复用假设在同时部署 Web、db、ha 时或不同服务器组合不同的应用就需要写多个 yml 文件。很难实现灵活的调用。

roles 用于层次性、结构化地组织 playbook。roles 能够根据层次型结构自动装载变量文件、tasks 以及 handlers 等。要使用 roles 只需要在 playbook 中使用 include 指令即可。简单来讲，roles 就是通过分别将变量(vars)、文件(file)、任务(tasks)、模块(modules)及处理器(handlers)放置于单独的目录中，并可以便捷地 include 它们的一种机制。角色一般用于基于主机构建服务的场景中，但也可以是用于构建守护进程等场景中。

### ② 角色集合

角色集合: roles/

mysql/ httpd/ nginx/ files/: 存储由 copy 或 script 等模块调用的文件;

tasks/: 此目录中至少应该有一个名为 main.yml 的文件，用于定义各 task; 其它的文件需要由 main.yml 进行“包含”调用;

handlers/: 此目录中至少应该有一个名为 main.yml 的文件，用于定义各 handler; 其它的文件需要由 main.yml 进行“包含”调用;

vars/: 此目录中至少应该有一个名为 main.yml 的文件，用于定义各 variable; 其它的文件需要由 main.yml 进行“包含”调用;

templates/: 存储由 template 模块调用的模板文本;

meta/: 此目录中至少应该有一个名为 main.yml 的文件，定义当前角色的特殊设定及其依赖关系; 其它的文件需要由 main.yml 进行“包含”调用;

default/: 此目录中至少应该有一个名为 main.yml 的文件，用于设定默认变量;

### ③ 角色定制实例

#### 1. 在 roles 目录下生成对应的目录结构

```
[root@server ansible]# cd roles/
```

```
[root@server roles]# ls
```

```
[root@server roles]# mkdir -pv ./{nginx,mysql,httpd}/{files,templates,vars,tasks,handlers,meta,default}
```



```
[root@server roles]# tree
```

```
.
├── httpd
│ ├── default
│ ├── files
│ ├── handlers
│ ├── meta
│ ├── tasks
│ ├── templates
│ └── vars
├── mysql
│ ├── default
│ ├── files
│ ├── handlers
│ ├── meta
│ ├── tasks
│ ├── templates
│ └── vars
└── nginx
 ├── default
 ├── files
 ├── handlers
 ├── meta
 ├── tasks
 ├── templates
 └── vars
```

24 directories, 0 files

## 2. 定义配置文件

我们需要修改的配置文件为/tasks/main.yml，下面，我们就来修改一下：

```
[root@server roles]# vim nginx/tasks/main.yml
```

```
- name: cp
 copy: src=nginx-1.10.2-1.el7ngx.x86_64.rpm dest=/tmp/nginx-1.10.2-1.el7ngx.x86_64.rpm
- name: install
 yum: name=/tmp/nginx-1.10.2-1.el7ngx.x86_64.rpm state=latest
- name: conf
 template: src=nginx.conf.j2 dest=/etc/nginx/nginx.conf
 tags: nginxconf
 notify: new conf to reload
- name: start service
 service: name=nginx state=started enabled=true
```

- 1 放置我们所需要的文件到指定目录

因为我们定义的角色已经有了新的组成方式，所以我们需要把文件都放到指定的位置，这样，才能让配置文件找到这些并进行加载。

rpm 包放在 files 目录下，模板放在 templates 目录下：

```
[root@server nginx]# cp /tmp/nginx-1.10.2-1.el7ngx.x86_64.rpm
./files/
[root@server nginx]# cp /tmp/nginx.conf.j2 ./templates/
[root@server nginx]# tree
.
├── default
├── files
│ └── nginx-1.10.2-1.el7ngx.x86_64.rpm
├── handlers
├── meta
├── tasks
│ └── main.yml
├── templates
│ └── nginx.conf.j2
└── vars
```

7 directories, 3 files

#### 4. 修改变量文件

我们在模板中定义的变量，也要去配置文件中加上：

```
[root@server nginx]# vim vars/main.yml
```

```
nginxprot: 9999
```

## 5. 定义 handlers 文件

我们在配置文件中定义了 notify，所以我么也需要定义 handlers，我们来修改配置文件：

```
[root@server nginx]# vim handlers/main.yml
```

### 1. name: new conf to reload

```
service: name=nginx state=restarted
```

## 6. 定义剧本文件

接下来，我们就来定义剧本文件，由于大部分设置我们都单独配置在了 roles 里面，所以，接下来剧本就只需要写一点点内容即可：

```
[root@server ansible]# vim roles.yml
```

- hosts: web
- remote\_user: root
- roles:
  - nginx

## 7. 启动服务

剧本定义完成以后，我们就可以来启动服务了：

```
[root@server ansible]# ansible-playbook roles.yml
```

```
PLAY [web] *****
```

```
TASK [setup] *****
```

```
ok: [192.168.37.122]
```

```
ok: [192.168.37.133]
```

```
TASK [nginx : cp] *****
```

```
ok: [192.168.37.122]
```

```
ok: [192.168.37.133]
```

```
TASK [nginx : install] *****
```

```
changed: [192.168.37.122]
```

```
changed: [192.168.37.133]
```

```
TASK [nginx : conf] *****
```

```
changed: [192.168.37.122]
```

```
changed: [192.168.37.133]
```

```
TASK [nginx : start service] *****
changed: [192.168.37.122]
changed: [192.168.37.133]
RUNNING HANDLER [nginx : new conf to reload] *****
changed: [192.168.37.122]
changed: [192.168.37.133]
PLAY RECAP *****
192.168.37.122 : ok=6 changed=4 unreachable=0 failed=0
192.168.37.133 : ok=6 changed=4 unreachable=0 failed=0
```

启动过后照例查看端口号：

```
[root@server ansible]# ansible web -m shell -a "ss -ntulp |grep 9999"
192.168.37.122 | SUCCESS | rc=0 >>
tcp LISTEN 0 128 *:9999 *:*
users: (("nginx",pid=7831,fd=6), ("nginx",pid=7830,fd=6), ("nginx",pid=7829,fd=6))

192.168.37.133 | SUCCESS | rc=0 >>
tcp LISTEN 0 128 *:9999 *:*
users: (("nginx",pid=9654,fd=6), ("nginx",pid=9653,fd=6), ("nginx",pid=9652,fd=6))
```

## Ansible 详细用法说明(一)

### 一、概述

运维工具按需不需要有代理程序来划分的话分两类：

- agent（需要有代理工具）：基于专用的 agent 程序完成管理功能，puppet, func, zabbix
- agentless（无须代理工具）：基于 ssh 服务完成管理，ansible, fabric

### 二、简介

Ansible 是一个简单的自动化运维管理工具，基于 Python 语言实现，由 Paramiko 和 PyYAML 两个关键模块构建，可用于自动化部署应用、配置、编排 task(持续交付、无宕机更新等)。主版本大概每 2 个月发布一次。

**Ansible 与 Saltstack** 最大的区别是 Ansible 无需在被控主机部署任何客户端代理，默认直接通过 SSH 通道进行远程命令执行或下发配置：相同点是都具备功能强大、灵活的系统管理、状态配置，两者都提供丰富的模板及 API，对云计算平台、大数据都有很好的支持。

#### 1、特点：

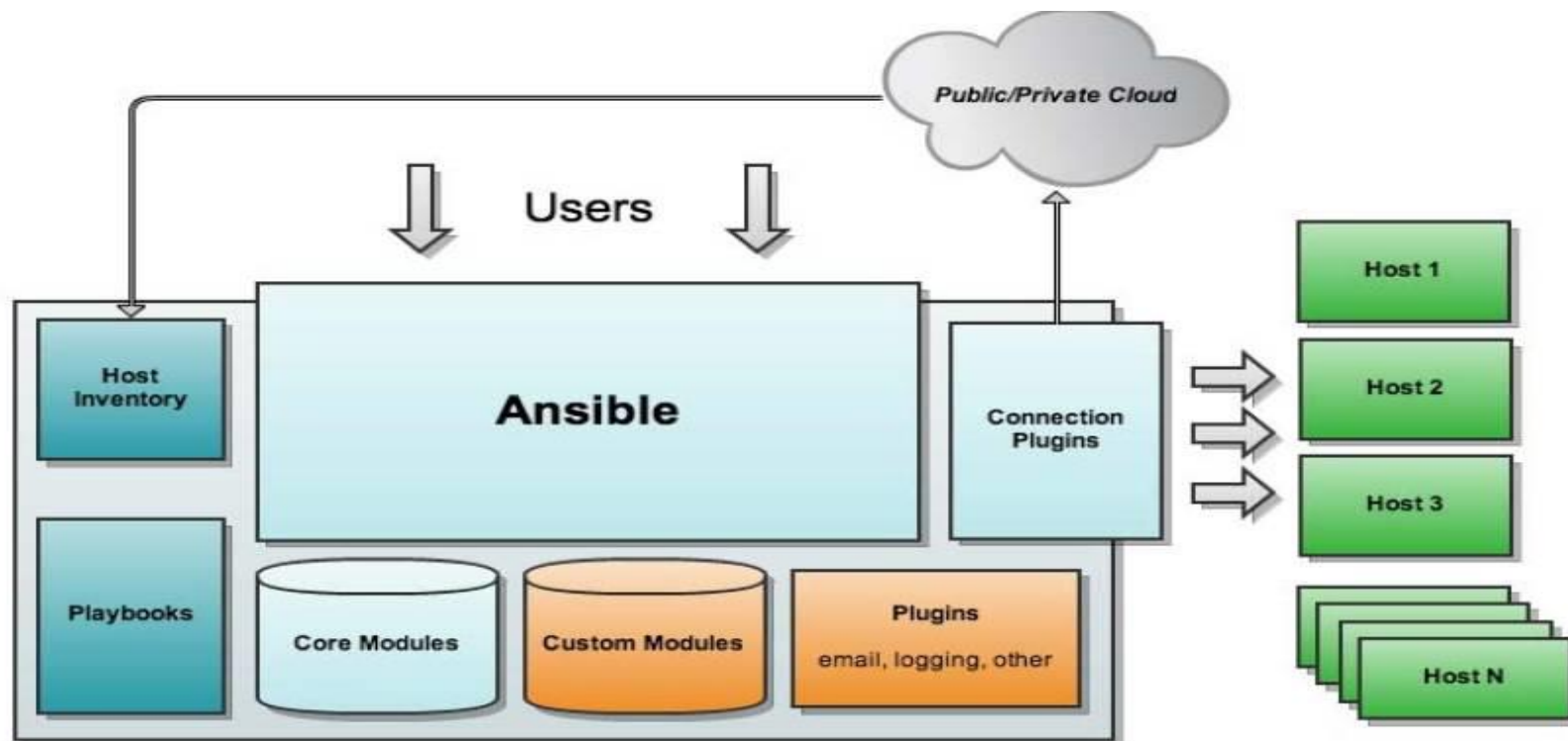
- 部署简单，只需在主控端部署 Ansible 环境，被控端无需做任何操作；

- 默认使用 SSH 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。
- 幂等性：一种操作重复多次结果相同

### 简评：

- (1)、轻量级，无需在客户端安装 agent，更新时，只需在操作机上进行一次更新即可；
- (2)、批量任务执行可以写成脚本，而且不用分发到远程就可以执行；
- (3)、使用 python 编写，维护更简单，ruby 语法过于复杂；
- (4)、支持 sudo。

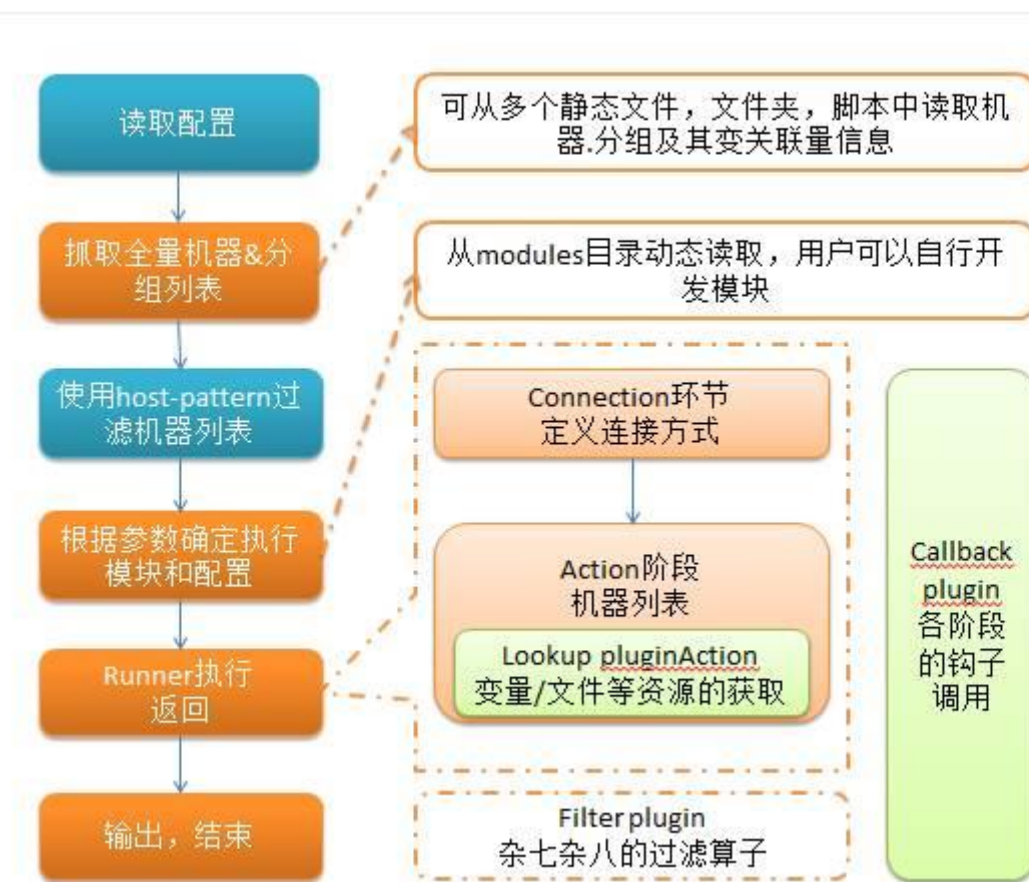
## 2、Ansible 架构图



## Ansible 核心组件说明：

- Ansible: Ansible 的核心程序
- Host Inventory: 记录了每一个由 Ansible 管理的主机信息，信息包括 ssh 端口，root 帐号密码，ip 地址等等。可以通过 file 来加载，可以通过 CMDB 加载
- Playbooks: YAML 格式文件，多个任务定义在一个文件中，使用时可以统一调用，“剧本”用来定义那些主机需要调用那些模块来完成的功能。
- Core Modules: Ansible 执行任何管理任务都不是由 Ansible 自己完成，而是由核心模块完成；Ansible 管理主机之前，先调用 core Modules 中的模块，然后指明管理 Host Inventory 中的主机，就可以完成管理主机。
- Custom Modules: 自定义模块，完成 Ansible 核心模块无法完成的功能，此模块支持任何语言编写。
- Connection Plugins: 连接插件，Ansible 和 Host 通信使用

## 3、ansible 执行过程，其中暖色调的代表已经模块化。



### 三、基础环境的安装与配置

#### 1、ssh 免密登录 配置

##### (1)、SSH 免密钥登录设置

- ansiblemaster: 10.1.6.172 **Centos7.2**
- ansibleslave1: 10.1.6.72 Centos7.2
- ansibleslave2: 10.1.6.73 Centos7.2
- ansibleslave3: 10.1.6.68 **Centos6.8**

##### 生成公钥/私钥

```
ssh-keygen -N ''
```

```
[root@localhost .ssh]# ssh-keygen -N '' 这里 -N -P都行 后面直接是两个单引号
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa): 默认就行
Your identification has been saved in /root/.ssh/id_rsa. 要不分发时就要用 -i 指明
Your public key has been saved in /root/.ssh/id_rsa.pub. 公钥
The key fingerprint is:
33:3e:89:12:24:bf:17:0d:dc:63:1e:25:62:c6:e4:93 root@localhost.localdomain
The key's randomart image is:
+--[RSA 2048]-----+
| o= . . |
| =.+ o |
| . . E = |
| + * o |
| o . S |
| o + + |
| o o + |
| o . |
+-----+
[root@localhost .ssh]# ls
id_rsa id_rsa.pub
```

##### 分发密钥

```
ssh-copy-id root@10.1.6.73
ssh-copy-id root@10.1.6.72
```



```
ssh-copy-id root@10.1.6.68
```

## 测试

```
[root@localhost .ssh]# ls
id_rsa id_rsa.pub known_hosts
[root@localhost .ssh]# cat known_hosts
10.1.6.72 ecdsa-sha2-nistp256 AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBMD+YWM/HtFI
XpyNy7hVqD9j9k4U09Jbk5HmMSoBk8b4WuB5IiUx3otAo=
10.1.6.73 ecdsa-sha2-nistp256 AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBBC6a/jlUJ5o
6eJfFjaiAIltfpIX30yKbHKJCHAIHZD2/rDnbUxSnBjkk=
10.1.6.68 ssh-rsa AAAAB3NzaC1yc2EAAAABIWAAAQEAlm8UA0VNZZUrMEUUhK8PyVU1CIq1sR+WlXyq/e1TkxeXnzrj
mt+lYT4dnfbnPq0jBw0Qc0JxQS4gzBt9cbn0s4F9FtbQzJ100Mt3c4G3yTeNBTEc7gz50WB1zQIXbV8fJ25ps2mVjkV7b
VKq4eCHf+QsplrBnuw337GJ93UyX6HiXEXCD26Kjc02BVn3ESWnLkdveaMum/GboxxlzzkamknTf3bWvr49pJs0SV0uRUl
X4lV4gkEM32kRlsw==
[root@localhost .ssh]# ssh 10.1.6.68
Last login: Sat Oct 15 00:31:56 2016 from 10.1.6.172
[root@localhost ~]# logout
Connection to 10.1.6.68 closed. 测试登录 无密码登录成功
```

```
[root@localhost .ssh]# ls
authorized_keys → 会在远端主机生成这个文件
[root@localhost .ssh]# cat authorized_keys
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQDet7aKWd3eLhKZP8tl+05B5WCFM1+l
1EznSoLDo4/3jrDmvrI/7Ssq0A0/5ogYgDzzXZ/G9J7F8eHklkTtCV0VNXqNSbRpYgpl
zNC0Dy7W0LkWz1PAL/4P/gzsKVDtb7cZXW0wcrDj4EMrhau4QJ6QsBffX21SlkTo0c2
LMXsPc0V root@localhost.localdomain
[root@localhost .ssh]#
```

>- 仅将文本发送到当前选项卡

ssh://10.1.6.72:22



注意

```
[root@ansible_master ~]# ssh-copy-id -i ~/.ssh/id_rsa.pub 10.1.6.72
-bash: ssh-copy-id: command not found
```

报错了

解决方法:

```
yum -y install openssh-clients
```

## 2、ansible 安装及程序环境:

### 安装

```
yum install ansible
```

### 程序:

```
ansible
ansible-playbook
ansible-doc
```

### 配置文件:

```
/etc/ansible/ansible.cfg
host_key_checking = False
```

### 主机清单:

```
/etc/ansible/hosts
```

### 插件目录:

```
/usr/share/ansible_plugins/
```

## 3.ansible 命令的使用说明

### 常用选项

**ansible -m MOD\_NAME -a MOD\_ARGS**

表示调用什么模块, 使用模块的那些参数

**ansible -h**

```
[root@localhost ~]# ansible -h
Usage: ansible <host-pattern> [options]
```

## Options:

`-a MODULE_ARGS, --args=MODULE_ARGS` 模块的参数,如果执行默认 COMMAND 的模块,即是命令参数,如: "date","pwd"等等  
module arguments 模块参数

`-k, --ask-pass` ask for SSH password 登录密码,提示输入 SSH 密码而不是假设基于密钥的验证

`--ask-su-pass` ask for su password su 切换密码

`-K, --ask-sudo-pass` ask for sudo password 提示密码使用 sudo,sudo 表示提权操作

`--ask-vault-pass` ask for vault password

`-B SECONDS, --background=SECONDS` 后台运行超时时间  
run asynchronously, failing after X seconds  
(default=N/A)

`-C, --check` don't make any changes; instead, try to predict some 只是测试一下会改变什么内容,不会真正去执行;相反,试图预测一些可能发生的变化  
of the changes that may occur

`-c CONNECTION, --connection=CONNECTION` 连接类型使用。可能的选项是 paramiko (SSH),SSH 和地方。当地主要是用于 crontab 或启动。  
connection type to use (default=smart)

`-f FORKS, --forks=FORKS` 并行任务数。NUM 被指定为一个整数,默认是 5  
specify number of parallel processes to use  
(default=5)

`-h, --help` show this help message and exit 打开帮助文档 API

`-i INVENTORY, --inventory-file=INVENTORY` 指定库存主机文件的路径,默认为/etc/ansible/hosts  
specify inventory host file  
(default=/etc/ansible/hosts)

`-l SUBSET, --limit=SUBSET` 进一步限制所选主机/组模式 --limit=192.168.91.135 只对这个 ip 执行  
further limit selected hosts to an additional pattern

`--list-hosts` outputs a list of matching hosts; does not execute  
anything else

`-m MODULE_NAME, --module-name=MODULE_NAME` 执行模块的名字,默认使用 command 模块,所以如果是只执行单一命令可以不用 -m 参数  
module name to execute (default=command)

`-M MODULE_PATH, --module-path=MODULE_PATH` 要执行的模块的路径,默认为/usr/share/ansible/  
specify path(s) to module library  
(default=/usr/share/ansible/)

`-o, --one-line` condense output 压缩输出,摘要输出.尝试一切都在一行上输出。

```

-P POLL_INTERVAL, --poll=POLL_INTERVAL 调查背景工作每隔数秒。需要- b
 set the poll interval if using -B (default=15)
--private-key=PRIVATE_KEY_FILE 私钥路径, 使用这个文件来验证连接
 use this file to authenticate the connection
-S, --su run operations with su 用 su 命令
-R SU_USER, --su-user=SU_USER 指定 SU 的用户, 默认是 root 用户
 run operations with su as this user (default=root)
-s, --sudo run operations with sudo (nopasswd)
-U SUDO_USER, --sudo-user=SUDO_USER sudo 到哪个用户, 默认为 root
 desired sudo user (default=root)
-T TIMEOUT, --timeout=TIMEOUT 指定 SSH 默认超时时间, 默认是 10S
 override the SSH timeout in seconds (default=10)
-t TREE, --tree=TREE log output to this directory 将日志内容保存在该输出目录, 结果保存在一个文件中在每台主机上。
-u REMOTE_USER, --user=REMOTE_USER 远程用户, 默认是 root 用户
 connect as this user (default=root)
--vault-password-file=VAULT_PASSWORD_FILE
 vault password file
-v, --verbose verbose mode (-vvv for more, -vvvv to enable 详细信息
 connection debugging)
--version show program's version number and exit 输出 ansible 的版本

```

## 4、ansible 主机清单的配置

### vim /etc/ansible/hosts

#### 定义方式:

- 直接指明主机地址或主机名
- blue.example.com  
192.168.100.1
- 定义一个主机组 [组名] 把地址或主机名加进去

- [webservers]
- alpha.example.org
- beta.example.org
- 192.168.1.100



## 测试：成功

```
no cronjob for root
[root@localhost tmp]# crontab -l
#Ansible: sync time 指明ansible设置的
*/5 * * * * /usr/sbin/ntpdate 10.1.0.1 &> /dev/null
[root@localhost tmp]#
```

➤ 仅将文本发送到当前选项卡

ssh://root@10.1.6.72:22

```
[root@localhost ansible]# ansible web -m command -a "useradd centos"
10.1.6.73 | success | rc=0 >>

10.1.6.72 | success | rc=0 >>

[root@localhost ansible]# ansible web -m command -a "id centos"
10.1.6.73 | success | rc=0 >>
uid=1001(centos) gid=1001(centos) groups=1001(centos)

10.1.6.72 | success | rc=0 >>
uid=1001(centos) gid=1001(centos) groups=1001(centos)
```

ansible 10.6.0.152 -m command -a "id root"

10.6.0.152 | SUCCESS | rc=0 >>

uid=0(root) gid=0(root) groups=0(root)

ansible 10.6.0.153 -m command -a "id root"

10.6.0.153 | SUCCESS | rc=0 >>

uid=0(root) gid=0(root) groups=0(root)

ansible testhost -m command -a "id root"

10.6.0.152 | SUCCESS | rc=0 >>

uid=0(root) gid=0(root) groups=0(root)

```
10.6.0.153 | SUCCESS | rc=0 >>
uid=0(root) gid=0(root) groups=0(root)
```

```
[root@master ~]# ansible proxy -m command -a "ls -lh /var/log/squid/access.log"
10.6.0.100 | SUCCESS | rc=0 >>
-rw-r----- 1 squid nogroup 0 Nov 17 11:02 /var/log/squid/access.log
[root@master ~]# ansible proxy -m command -a "ls -lh /var/log/squid/store.log"
10.6.0.100 | SUCCESS | rc=0 >>
-rw-r----- 1 squid nogroup 320 Nov 17 11:03 /var/log/squid/store.log
[root@master ~]#
```

四、ansible 使用 之一 “命令管理方式”

常用模块：

```
=====
ping: 探测目标主机是否存活;
=====
command: 在远程主机执行命令；不支持管道命令
=====
```

```
ansible storm_cluster -m command -a "ls -al /tmp/resolv.conf"
```

```
-
```

```
相关选项如下：
creates: 一个文件名，当该文件存在，则该命令不执行
free_form: 要执行的 linux 指令
chdir: 在执行指令之前，先切换到该目录
removes: 一个文件名，当该文件不存在，则该选项不执行
executable: 切换 shell 来执行指令，该执行路径必须是一个绝对路径
```

shell: 在远程主机上调用 shell 解释器运行命令，支持 shell 的各种功能，例如管道等；  
注意：command 和 shell 模块的核心参数直接为命令本身；而其它模块的参数通常为 “key=value” 格式；

```
=====
copy: 复制文件到远程主机，可以改权限等
=====
```

```
用法：
```

(1) 复制文件

```
-a "src= dest= "
```

(2) 给定内容生成文件

```
-a "content= dest= "
```

- 常用用法

第一种：直接复制

```
[root@localhost ansible]# ansible all -m copy -a "src=/etc/fstab dest=/tmp/ansiblecopy mode=640"
10.1.6.68 | success >> {
 "changed": true,
 "checksum": "367ec79623fef145d695af6de2ba590ed22f8014",
 "dest": "/tmp/ansiblecopy",
 "gid": 0,
 "group": "root",
 "mode": "0640",
 "owner": "root",
 "path": "/tmp/ansiblecopy",
 "size": 595,
 "state": "file",
 "uid": 0
}
10.1.6.73 | success >> {
 "changed": true,
```

表示所有主机

设置权限

第二种：生成内容的复制

```
[root@localhost ansible]# ansible all -m copy -a "content='helo\nworld' dest=/tmp/test.ansible mode=640"
10.1.6.68 | success >> {
 "changed": true,
 "checksum": "cb327892b41d32de1842f4caeb22a17394fb580a",
 "dest": "/tmp/test.ansible",
 "gid": 0,
 "group": "root",
 "md5sum": "3004e7367bdf1d8ba1f4c6da9d5dbaca",
 "mode": "0640",
 "owner": "root",
 "size": 10,
 "src": "/root/.ansible/tmp/ansible-tmp-1478433242.24-151738880106271/source",
 "state": "file",
}
```

```
[root@localhost tmp]# cat test.ansible
helo
world[root@localhost tmp]#
```

- `ansible test -m copy -a "src=/root/test dest=/home/testcopy mode=640"`

```
10.6.0.152 | SUCCESS => {
 "changed": true,
 "checksum": "da39a3ee5e6b4b0d3255bfef95601890afd80709",
 "dest": "/home/testcopy",
 "gid": 0,
 "group": "root",
 "md5sum": "d41d8cd98f00b204e9800998ecf8427e",
 "mode": "0640",
 "owner": "root",
 "size": 0,
 "src": "/root/.ansible/tmp/ansible-tmp-1479352868.18-83082559134161/source",
 "state": "file",
 "uid": 0
}
```

相关选项如下:



backup: 在覆盖之前，将源文件备份，备份文件包含时间信息。有两个选项：yes|no

content: 用于替代“src”，可以直接设定指定文件的值

dest: 必选项。要将源文件复制到的远程主机的绝对路径，如果源文件是一个目录，那么该路径也必须是个目录

directory\_mode: 递归设定目录的权限，默认为系统默认权限

force: 如果目标主机包含该文件，但内容不同，如果设置为 yes，则强制覆盖，如果为 no，则只有当目标主机的目标位置不存在该文件时，才复制。默认为 yes

others: 所有的 file 模块里的选项都可以在这里使用

src: 被复制到远程主机的本地文件，可以是绝对路径，也可以是相对路径。如果路径是一个目录，它将递归复制。在这种情况下，如果路径使用“/”来结尾，则只复制目录里的内容，如果没有使用“/”来结尾，则包含目录在内的整个内容全部复制，类似于 rsync。

=====

## file：设置文件属性。

=====

用法:

(1) 创建目录:

```
-a "path=。。。 state=directory"
```

(2) 创建链接文件:

```
-a "path=。。。 src=... state=link"
```

(3) 删除文件:

```
-a "path=。。。 state=absent"
```

```
no need to mention
[root@localhost ansible]# ansible web -m file -a "path=/tmp/test.ansible owner=root"
10.1.6.72 | success >> {
 "changed": true,
 "gid": 0,
 "group": "root",
 "mode": "0640",
 "owner": "root",
 "path": "/tmp/test.ansible",
 "size": 10,
 "state": "file",
 "uid": 0
}

10.1.6.73 | success >> {
 "changed": true,
 "gid": 0,
 "group": "root",
 "mode": "0640"
```

改属主为root

前提这个文件必须事先存在

相关选项如下：

**force:** 需要在两种情况下强制创建软链接，一种是源文件不存在，但之后会建立的情况下；另一种是目标软链接已存在，需要先取消之前的软链，然后创建新的软链，有两个选项：yes|no

**group:** 定义文件/目录的属组

**mode:** 定义文件/目录的权限

**owner:** 定义文件/目录的属主

**path:** 必选项，定义文件/目录的路径

**recurse:** 递归设置文件的属性，只对目录有效

**src:** 被链接的源文件路径，只应用于 state=link 的情况

**dest:** 被链接到的路径，只应用于 state=link 的情况

**state:**

**directory:** 如果目录不存在，就创建目录

**file:** 即使文件不存在，也不会被创建

**link:** 创建软链接

**hard:** 创建硬链接

**touch:** 如果文件不存在，则会创建一个新的文件，如果文件或目录已存在，则更新其最后修改时间

**absent:** 删除目录、文件或者取消链接文件

=====

**fetch:** 从远程某一个主机获取文件到本地

=====

**cron:** 管理 cron 计划任务

=====

```
[root@localhost ansible]# ansible all -m cron -a "minute='*/5' job='/usr/sbin/ntpdate 10.1.0.1 &> /dev/null' name='sync time'"
10.1.6.68 | success >> {
 "changed": true,
 "jobs": [
 "sync time"
]
}
10.1.6.72 | success >> {
 "changed": true,
 "jobs": [
```

表示每5分钟同步一下时间

这一项必须要有，指明名称

```
no cron tab for root
[root@localhost tmp]# crontab -l
#Ansible: sync time
*/5 * * * * /usr/sbin/ntpdate 10.1.0.1 &> /dev/null
[root@localhost tmp]#
```

指明ansible设置的

ansible all -m cron -a "name='sync time' state=absent" 表示删除此任务

- a ""：设置管理节点生成定时任务

action: cron

backup= # 如果设置，创建一个 crontab 备份

cron\_file= #如果指定，使用这个文件 cron.d，而不是单个用户 crontab

day= # 日应该运行的工作 ( 1-31, \*, \*/2, etc )

hour= # 小时 ( 0-23, \*, \*/2, etc )

job= #指明运行的命令是什么

```
minute= #分钟(0-59, *, */2, etc)
month= # 月(1-12, *, */2, etc)
name= #定时任务描述
reboot # 任务在重启时运行, 不建议使用, 建议使用 special_time
special_time # 特殊的时间范围, 参数: reboot (重启时),annually (每年),monthly (每月),weekly (每周),daily (每天),hourly (每小时)

state #指定状态, prsent 表示添加定时任务, 也是默认设置, absent 表示删除定时任务

user # 以哪个用户的身份执行
weekday # 周 (0-6 for Sunday-Saturday, *, etc)
```

=====

## yum: yum 安装软件, 也有 apt,zypper

=====

```
conf_file #设定远程 yum 安装时所依赖的配置文件。如配置文件没有在默认的位置。
disable_gpg_check #是否禁止 GPG checking, 只用于`present' or `latest'。
disablerepo #临时禁止使用 yum 库。 只用于安装或更新时。
enablerepo #临时使用的 yum 库。只用于安装或更新时。
name= #所安装的包的名称
state #present 安装, latest 安装最新的, absent 卸载软件。
update_cache #强制更新 yum 的缓存。
```

=====

## service: 服务程序管理

=====

```
arguments #命令行提供额外的参数
enabled #设置开机启动。
name= #服务名称
runlevel #开机启动的级别, 一般不用指定。
sleep #在重启服务的过程中, 是否等待。如在服务关闭以后等待 2 秒再启动。
state #started 启动服务, stopped 停止服务, restarted 重启服务, reloaded 重载配置
```

启动 httpd 服务:

```
ansible all -m service -a 'name=httpd state=started'
```

## 开机自启动

```
ansible all -m service -a "name=httpd enabled=true"
```

=====

## group: 组管理

=====

```
[root@node1 ~]# ansible-doc -s group
```

```
- name: 添加或删除组
```

```
action: group
```

```
gid # 设置组的 GID 号
```

```
name= # 管理组的名称
```

```
state # 指定组状态，默认为创建，设置值为 absent 为删除
```

```
system # 设置值为 yes，表示为创建系统组
```

=====

## User:用户管理

=====

```
-a ""
```

```
action: user
```

```
comment # 用户的描述信息
```

```
createhom # 是否创建家目录
```

```
force # 在使用`state=absent`是，行为与`userdel --force`一致。
```

```
group # 指定基本组
```

```
groups # 指定附加组，如果指定为('groups=')表示删除所有组
```

```
home # 指定用户家目录
```

```
login_class #可以设置用户的登录类 FreeBSD, OpenBSD and NetBSD 系统。
```

```
move_home # 如果设置为`home=`时，试图将用户主目录移动到指定的目录
```

```
name= # 指定用户名
```

```
non_unique # 该选项允许改变非唯一的用户 ID 值
```

```
password # 指定用户密码
```

```
remove # 在使用 `state=absent`时，行为是与 `userdel --remove`一致。
```

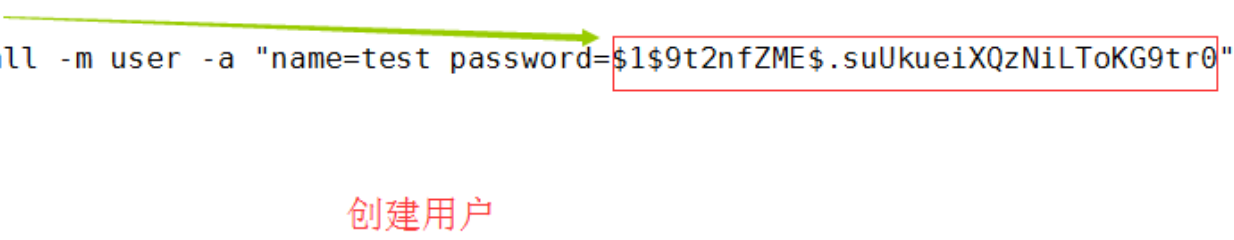
```
shell # 指定默认 shell
```

```
state #设置帐号状态，不指定为创建，指定值为 absent 表示删除
```

```
system # 当创建一个用户，设置这个用户是系统用户。这个设置不能更改现有用户。
uid #指定用户的 uid
update_password # 更新用户密码
expires #指明密码的过期时间
```

例：创建用户并添加密码

```
[root@localhost ansible]# openssl passwd -1
Password:
Verifying - Password:
$1$9t2nfZME$.suUkueiXQzNiLToKG9tr0
[root@localhost ansible]# ansible all -m user -a "name=test password=$1$9t2nfZME$.suUkueiXQzNiLToKG9tr0"
10.1.6.68 | success >> {
 "changed": true,
 "comment": "",
 "createhome": true,
 "group": 500,
 "home": "/home/test",
 "name": "test",
 "password": "NOT_LOGGING_PASSWORD",
 "shell": "/bin/bash",
 "state": "present",
 "system": false,
 "uid": 500
```



=====

setup: 获取指定主机的 facts。

## Ansible 详细用法说明(二)

setup: 获取指定主机的 facts。

=====

facts 就是变量，**内建变量**。每个主机的各种信息，cpu 颗数、内存大小等。会存在 facts 中的某个变量中。调用后返回很多对应主机的信息，在后面的操作中可以根据不同的信息来做不同的操作。如 redhat 系列用 yum 安装，而 debian 系列用 apt 来安装软件。

例：获取某台主机的变量

```
ansible 10.1.6.68 -m setup
```

=====

**script: 发送脚本到各被管理节点，并执行。不需要参数**

=====

```
ansible all -m script -a 'test.sh'
```

直接在-a 后面指定脚本即可。

=====

**selinux: 管理 selinux。**

=====

```
conf #指定应用 selinux 的配置文件。
```

```
state=enforcing|permissive|disabled #对应于 selinux 配置文件的 SELINUX。
```

```
policy=targeted|minimum|mls #对应于 selinux 配置文件的 SELINUXTYPE
```

关闭 selinux:

```
ansible all -m selinux -a 'state=disabled'
```

=====

**template:使用了 Jinja2 格式作为文件模版，进行文档内变量的替换的模块。**

=====

## 五、playbook: “跑剧本”

**playbook** 就是一个用 `yaml 语法` 把多个模块堆起来的一个文件而已。

### 1.简介

YAML 是一个可读性高的用来表达资料序列的格式。YAML 参考了其他多种语言，包括：XML、C 语言、Python、Perl 以及电子邮件格式 RFC2822 等。Clark Evans 在 2001 年在首次发表了这种语言，另外 Ingy döt Net 与 Oren Ben-Kiki 也是这语言的共同设计者。

### 2、特点

- YAML 的可读性好
- YAML 和脚本语言的交互性好
- YAML 使用实现语言的数据类型
- YAML 有一个一致的信息模型
- YAML 易于实现
- YAML 可以基于流来处理
- YAML 表达能力强，扩展性好

YAML 的语法和其他高阶语言类似，并且可以简单表达清单、散列表、标量等数据结构。其结构（Structure）通过空格来展示，序列（Sequence）里的项用"-"来代表，Map 里的键值对用":"分隔。下面是一个示例。

```
- hosts: 10.1.0.1 #定义主机
 vars: #定义变量
 var1: value
 var2: value
 tasks: #定义任务
 - name: #任务名称。
 #这里就可以开始用模块来执行具体的任务了。

 handlers: #定义触发通知所作的操作。里面也是跟 tasks 一样，用模块定义任务。
 - name:

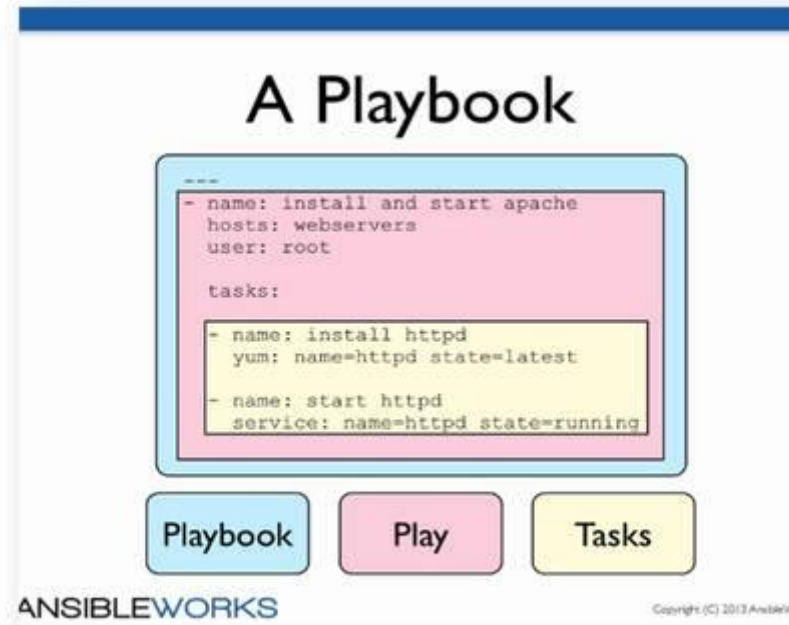
 remote_user: #远程主机执行任务时的用户。一般都是 root，一般也不用指定。

- hosts: web
 vars:
 tasks:
 handlers:
 remote_user:
```

YAML 文件扩展名通常为.yaml，如 example.yaml

## Playbooks





## 1.核心元素：

- **Tasks:** 任务，由模块定义的操作的列表；
- **Variables:** 变量
- **Templates:** 模板，即使用了模板语法的文本文件；
- **Handlers:** 由特定条件触发的 Tasks；
- **Roles:** 角色；

## 2.playbook 的基础组件：

**Hosts:** 运行指定任务的目标主机；

**remote\_user:** 在远程主机以哪个用户身份执行；

**sudo\_user:** 非管理员需要拥有 sudo 权限；

**tasks:** 任务列表

模块，模块参数：

格式：

(1) action: module arguments

(2) module: arguments

## 示例 1:

vim test.yaml 也可以是 .yaml

```
- hosts: all
remote_user: root
tasks:
- name: install a group
group: name=mygrp system=true
- name: install a user
user: name=user1 group=mygrp system=true
- hosts: webservers
 remote_user: root
 tasks:
 - name: install httpd package
 yum: name=httpd
 - name: start httpd service
 service: name=httpd state=started
```

注意空格

注意左对齐

注意：后要有空格

```
- hosts: all
 remote_user: root
 tasks:
 - name: install a group
 group: name=mygrp system=true
 - name: install a user
 user: name=user1 group=mygrp system=true

- hosts: webservers
 remote_user: root
 tasks:
 - name: install httpd package
 yum: name=httpd
 - name: start httpd service
 service: name=httpd state=started
```

### 3.运行 playbook，使用 ansible-playbook 命令

#### (1) 检测语法

```
ansible-playbook --syntax-check /path/to/playbook.yaml
```

#### (2) 测试运行

```
ansible-playbook -C /path/to/playbook.yaml
```

```
--list-hosts
```

```
--list-tasks
```

```
--list-tags
```

```
ansible-playbook --check /path/to/playbook.yaml
```

#### (3) 运行

```
ansible-playbook /path/to/playbook.yaml
```

```
-t TAGS, --tags=TAGS
```

```
--skip-tags=SKIP_TAGS 跳过指定的标签
```

```
--start-at-task=START_AT 从哪个任务后执行
```

**tags:** 给指定的任务定义一个调用标识；

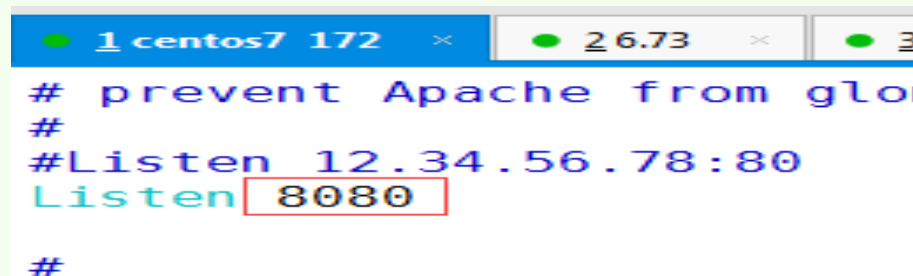
```
- name: NAME
```

```
 module: arguments
```

```
 tags: TAG_ID
```

可以一次调用多个名称相同的标签。也可以调用不同的标签用 “,” 分割。

改变监听端口



```
prevent Apache from glo
#
#Listen 12.34.56.78:80
Listen 8080
#
```

写剧本把此文件传过去

```
- hosts: web
 remote_user: root
 tasks:
 - name: install httpd package
 yum: name=httpd state=latest
 - name: install conf file
 copy: src=/root/httpd.conf dest=/etc/httpd/conf/httpd.conf
 tags: instconf
 notify: restart httpd service
 - name: start httpd service
 service: name=httpd state=started

 handlers:
 - name: restart httpd service
 service: name=httpd state=restarted
~
```

这里就是表示一个标签

指明标签的检查

```
[root@localhost ~]# ansible-playbook -C -t instconf aaa.yaml
```

表示直接跳过标签前的步骤，直接从标签位置开始

```
PLAY [web] *****
```

```
GATHERING FACTS *****
```

```
ok: [10.1.6.72]
```

```
ok: [10.1.6.73]
```

```
TASK: [install conf file] *****
```

```
changed: [10.1.6.73]
```

```
changed: [10.1.6.72]
```

直接跳过了标签之前的步骤

查看标签

```
[root@localhost ~]# ansible-playbook --list-tags aaa.yaml
```

```
playbook: aaa.yaml
```

此命令可以查看剧本的 标签

```
play #1 (web): TAGS: []
```

```
TASK TAGS: [instconf]
```

## playbook 执行过程

```
[root@localhost ~]# ansible-playbook -t instconf aaa.yaml
```

PLAY [web] \*\*\*\*\* 指明从哪个标签位置开始运行 \*\*\*\*\*

GATHERING FACTS \*\*\*\*\*

```
ok: [10.1.6.72]
ok: [10.1.6.73]
```

TASK: [install conf file] \*\*\*\*\*

```
changed: [10.1.6.73]
changed: [10.1.6.72]
```

NOTIFIED: [restart httpd service] \*\*\*\*\*

```
changed: [10.1.6.73]
changed: [10.1.6.72]
```

直接调用了NOTIFIED 就不会运行start了

PLAY RECAP \*\*\*\*\*

|           |        |           |               |          |
|-----------|--------|-----------|---------------|----------|
| 10.1.6.72 | : ok=3 | changed=2 | unreachable=0 | failed=0 |
| 10.1.6.73 | : ok=3 | changed=2 | unreachable=0 | failed=0 |

## 跳过标签的事件

```
[root@localhost ~]# ansible-playbook --skip-tags=instconf aaa.yaml
```

PLAY [web] \*\*\*\*\* 跳过这个标签 \*\*\*\*\*

GATHERING FACTS \*\*\*\*\*

```
ok: [10.1.6.72]
ok: [10.1.6.73]
```

TASK: [install httpd package] \*\*\*\*\*

```
ok: [10.1.6.73]
ok: [10.1.6.72]
```

TASK: [start httpd service] \*\*\*\*\*

```
ok: [10.1.6.73]
ok: [10.1.6.72]
```

就不会执行NOTIFIED了

PLAY RECAP \*\*\*\*\*

|           |        |           |               |          |
|-----------|--------|-----------|---------------|----------|
| 10.1.6.72 | : ok=3 | changed=0 | unreachable=0 | failed=0 |
| 10.1.6.73 | : ok=3 | changed=0 | unreachable=0 | failed=0 |

## handlers: 由特定条件触发的 Tasks;

```
- name: TASK_NAME
 module: arguments
 notify: HANDLER_NAME
handlers:
- name: HANDLER_NAME
 module: arguments
```

```
- hosts: webservs
 remote_user: root
 tasks:
 - name: install httpd package
 yum: name=httpd state=latest
 - name: install conf file
 copy: src=/root/httpd.conf dest=/etc/httpd/conf/httpd.conf
 notify: reload httpd service
 - name: start httpd service
 service: name=httpd state=started
 handlers:
 - name: reload httpd service
 shell: systemctl reload httpd.service
```

通知机制

触发执行器，这里的名字必须一致

触发器

第一次的话都会运行，后边如果文件内容发生改变就会触发 `notify`，然后会直接执行 `handlers` 的内容（这里 `notify` 后边的事件就都不会执行了）。估计是 md5 那种的校验。删了个 # 号竟然也会通知。

## 六、Variables: 变量

内建:

```
(1) facts
```

自定义:

(1) 命令行传递; 这个优先级最高

(2) 在 hosts Inventory (**/etc/ansible/hosts**) 中为每个主机定义专用变量值;

```
[web]
10.1.6.72 QQQ=nginx
10.1.6.73 QQQ=httpd
```

```
- hosts: web
 remote_user: root
 tasks:
 - name: install httpd package
 yum: name={{QQQ}} state=latest
```

(a) 向不同的主机传递不同的变量;

```
IP/HOSTNAME variable_name=value

[web]
10.1.6.72 qzx=httpd
10.1.6.73 qzx=nginx
```

(b) 向组内的所有主机传递相同的变量 ;

```
[groupname:vars]
variable_name=value

[web:qzx]
qzx=httpd
```

```
[web]

10.1.6.72

10.1.6.73
```

### (3) 在 **playbook** 中定义,建议使用这个!

```
vars:

- var_name: value

- var_name: value
```

### (4) Inventory 还可以使用参数:

```
用于定义 ansible 远程连接目标主机时使用的属性，而非传递给 playbook 的变量：

ansible_ssh_host

ansible_ssh_port

ansible_ssh_user

ansible_ssh_pass

ansible_sudo_pass

...
```

### (5) 在角色调用时传递

```
roles:

- { role: ROLE_NAME, var: value, ...}
```

**变量调用：**有空格

`{{ var_name }}`

## 七、Templates：模板

文本文件，内部嵌套有模板语言脚本（使用模板语言编写）

Jinja2 是由 python 编写的。 在我们打算使用基于文本的模板语言时，jinja2 是很好的解决方案。`yeml` 是写 `playbook`，`jinja2` 是写 `配置文件模板` 的 **功用**

将模板的文件的变量值转换成对应的本地主机的确定值。例如：ansible 端写一个内建变量`{{ ansible_processor_vcpus }}`，当这个文件被复制到对应主机时会自动生成对应主机 cpu 的颗数的结果替换之。

**Jinja2 语法：**

字面量：

```
字符串：使用单引号或双引号；

数字：整数、浮点数；

列表：[item1, item2, ...]
```



```
元组: (item1, item2, ...)

字典: {key1:value1, key2:value2, ...}

布尔型: true/false
```

算术运算:

```
+, -, *, /, //, %, **
```

比较操作:

```
==, !=, >, <, >=, <=
```

逻辑运算: and, or, not

执行模板文件中的脚本，并生成结果数据流，需要使用 `template` 模块；

=====

**template:使用了 Jinja2 格式作为文件模版，进行文档内变量的替换的模块。**相当于 copy

=====

将 jinja2 的文件模板理解并执行，转化为各个主机间的对应值

|        |                                             |
|--------|---------------------------------------------|
| backup | 建立个包括 timestamp 在内的文件备份，以备不时之需。             |
| dest   | 远程节点上的绝对路径，用于放置 template 文件。                |
| group  | 设置远程节点上的的 template 文件的所属用户组                 |
| mode   | 设置远程节点上的 template 文件权限。类似 Linux 中 chmod 的用法 |
| owner  | 设置远程节点上的 template 文件所属用户                    |
| src    | 本地 Jinja2 模版的 template 文件位置。                |

注意：此模板不能在命令行使用，而只能用于 playbook；**用法同 copy**

**1、普通示例：**

这里/root/nginx.conf 内容发生了改变。

```
- hosts: ngxsrvs
 remote_user: root
 tasks:
 - name: install nginx package
 yum: name=nginx state=latest
 - name: install conf file
```

```
template: src=/root/nginx.conf.j2 dest=/etc/nginx/nginx.conf

tags: ngxconf

notify: reload nginx service

- name: start nginx service

 service: name=nginx state=started enabled=true

handlers:

- name: reload nginx service

 shell: /usr/sbin/nginx -s reload
```

## 2、条件测试:

when 语句: 在 tasks 中使用, Jinja2 的语法格式;

```
- hosts: all

 remote_user: root

 tasks:

 - name: install nginx package

 yum: name=nginx state=latest

 - name: start nginx service on CentOS6

 shell: service nginx start

 when: ansible_distribution == "CentOS" and ansible_distribution_major_version == "6"

 - name: start nginx service

 shell: systemctl start nginx.service

 when: ansible_distribution == "CentOS" and ansible_distribution_major_version == "7"
```

## 3、循环: 迭代, 需要重复执行的任务;

对迭代项的引用, 固定变量名为"item", 使用 with\_item 属性给定要迭代的元素; **这个是以任务为中心, 围绕每个任务来跑主机, 如果中间某个任务中断, 那么所有主机以后的任务就无法安装。**

元素:

- 列表
- 字符串
- 字典

基于字符串列表给出元素示例:

```
- hosts: webservs

 remote_user: root
```

```

tasks:
- name: install packages
 yum: name={{ item }} state=latest
 with_items:
 - httpd
 - php
 - php-mysql
 - php-mbstring
 - php-gd

```

基于字典列表给元素示例：item.name 后边的表示键

```

- hosts: all
 remote_user: root
 tasks:
- name: create groups
 group: name={{ item }} state=present
 with_items:
 - groupx1
 - groupx2
 - groupx3
- name: create users
 user: name={{ item.name }} group={{ item.group }} state=present
 with_items:
 - {name: 'userx1', group: 'groupx1'}
 - {name: 'userx2', group: 'groupx2'}
 - {name: 'userx3', group: 'groupx3'}

```

## 八、角色：roles

以特定的层级目录结构进行组织的 tasks、variables、handlers、templates、files 等；相当于函数的调用把各个事件切割成片段来执行。

```
mkdir ./{nginx,memcached,httpd,mysql}/{file,templates,vars,handlers,meta,default,tasks} -pv
```

### role\_name/

files/: 存储由 copy 或 script 等模块调用的文件:

tasks/: 此目录中至少应该有一个名为 main.yml 的文件，用于定义各 task； 其它的文件需要由 main.yml 进行“包含”调用；

handlers/: 此目录中至少应该有一个名为 main.yml 的文件，用于定义各 handler； 其它的文件需要由 main.yml 进行“包含”调用；

vars/: 此目录中至少应该有一个名为 main.yml 的文件，用于定义各 variable； 其它的文件需要由 main.yml 进行“包含”调用；

templates/: 存储由 template 模块调用的模板文本；

meta/: 此目录中至少应该有一个名为 main.yml 的文件，定义当前角色的特殊设定及其依赖关系； 其它的文件需要由 main.yml 进行“包含”调用；

default/: 此目录中至少应该有一个名为 main.yml 的文件，用于设定默认变量；

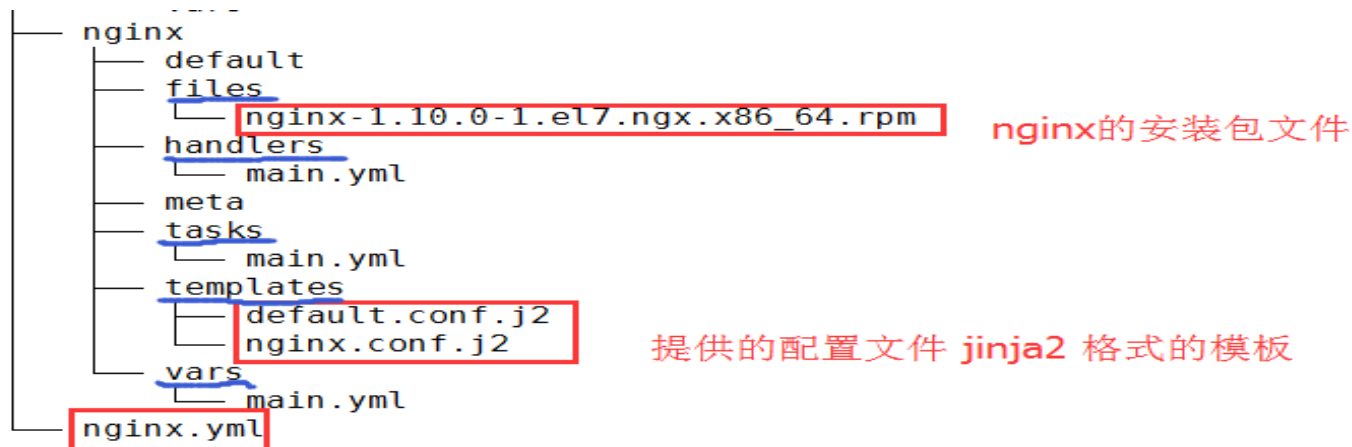
## 在 playbook 中调用角色的方法：

```
- hosts: HOSTS
 remote_user: USERNAME
 roles:
 - ROLE1
 - ROLE2
 - { role: ROLE3, VARIABLE: VALUE, ...}
 - { role: ROLE4, when: CONDITION }
```

## 事例： 基于角色的方式安装 nginx

### 1、创建需要的文件

```
mkdir ./{nginx,memcached,httpd,mysql}/{files,templates,vars,handlers,meta,default,tasks} -pv
```



33 directories, 10 files

[root@localhost ansible]# `tree /etc/ansible/` 这些文件的所在位置

### 3、写 tasks/下的主 main.yml

```

- name: copy nginx package 复制安装包文件
 copy: src=nginx-1.10.0-1.el7.ngx.x86_64.rpm dest=/tmp/nginx-1.10.0-1.el7.ngx.x86_64.rpm

- name: install nginx package
 yum: name=/tmp/nginx-1.10.0-1.el7.ngx.x86_64.rpm state=present

- name: install nginx.conf file
 template: src=nginx.conf.j2 dest=/etc/nginx/nginx.conf
 tags: ngxconf
 notify: reload nginx service

- name: install default.conf file
 template: src=default.conf.j2 dest=/etc/nginx/conf.d/default.conf
 tags: ngxconf
 notify: reload nginx service

- name: start nginx service
 service: name=nginx enabled=true state=started

```

```

- name: copy nginx package
 copy: src=nginx-1.10.0-1.el7.ngx.x86_64.rpm dest=/tmp/nginx-1.10.0-1.el7.ngx.x86_64.rpm

```

```

- name: install nginx package
 yum: name=/tmp/nginx-1.10.0-1.el7ngx.x86_64.rpm state=present

- name: install nginx.conf file
 template: src=nginx.conf.j2 dest=/etc/nginx/nginx.conf
 tags: ngxconf
 notify: reload nginx service

- name: install default.conf file
 template: src=default.conf.j2 dest=/etc/nginx/conf.d/default.conf
 tags: ngxconf
 notify: reload nginx service

- name: start nginx service
 service: name=nginx enabled=true state=started

```

## 2.复制相应的安装包和模板到对应目录下

3、根据需要修改 nginx 的配置文件模板。（这里改的是 work 进程生成数和监听的端口）

```

default.conf.j2 nginx.conf.j2
[root@localhost ansible]# vim /etc/ansible/roles/nginx/templates/default.conf.j2

server {
 listen {{ ngxprot }};
 server_name localhost;

 #charset koi8-r;
 #access_log /var/log/nginx/log/host.access.log main;

 location / {

```

仅仅改了端口为变量

```

[...]- nginx.conf -
[root@localhost ansible]# vim /etc/ansible/roles/nginx/templates/nginx.conf.j2
user nginx;
worker_processes {{ ansible_processor_vcpus }};
error_log /var/log/nginx/error.log warn;
pid /var/run/nginx.pid

```

仅改了work进程启动数为cpu核心数

#### 4、写 handlers 目录和 vars/下的 main.yml 文件

```

[root@localhost ansible]# vim /etc/ansible/roles/nginx/handlers/main.yml
- name: reload nginx service
 service: name=nginx state=restarted
~

```

定义触发的事件

```

[root@localhost ansible]# vim /etc/ansible/roles/nginx/vars/main.yml
ngxprot: "8088"
~
~

```

用来定义变量的值

#### 5、写需要运行的主 yml 文件

```

- hosts: web
 remote_user: root
 roles:
 - nginx
- { role: nginx, ngxprot: "8080" }
~
~

```

这里的意思必须是roles下的nginx目录，这个定义在，主配置文件有介绍

这里表示如果想改端口可以直接这样改

## 7、测试

```
[root@localhost roles]# ansible-playbook --check nginx.yml
```

```
PLAY [web] *****
```

```
GATHERING FACTS *****
```

```
ok: [10.1.6.72]
```

```
ok: [10.1.6.73]
```

```
TASK: [nginx | copy nginx package] *****
```

```
changed: [10.1.6.73]
```

```
changed: [10.1.6.72]
```

```
TASK: [nginx | install nginx package] *****
```

```
failed: [10.1.6.73] => {"changed": false, "failed": true, "rc": 0, "results": []}
```

```
msg: No Package file matching '/tmp/nginx-1.10.0-1.el7ngx.x86_64.rpm' found on system
```

```
failed: [10.1.6.72] => {"changed": false, "failed": true, "rc": 0, "results": []}
```

```
msg: No Package file matching '/tmp/nginx-1.10.0-1.el7ngx.x86_64.rpm' found on system
```

```
FATAL: all hosts have already failed -- aborting
```

```
PLAY RECAP *****
```

```
to retry, use: --limit @/root/nginx.retry
```

```
10.1.6.72 : ok=2 changed=1 unreachable=0 failed=1
```

```
10.1.6.73 : ok=2 changed=1 unreachable=0 failed=1
```

这里由于是测试并没有真正安装，所以出现了错误

## 8、运行



```
[root@localhost roles]# ansible-playbook nginx.yml

PLAY [web] *****

GATHERING FACTS *****
ok: [10.1.6.72]
ok: [10.1.6.73]

TASK: [nginx | copy nginx package] *****
changed: [10.1.6.73]
changed: [10.1.6.72]

TASK: [nginx | install nginx package] *****
changed: [10.1.6.72]
changed: [10.1.6.73]

TASK: [nginx | install nginx.conf file] *****
changed: [10.1.6.73]
ok: [10.1.6.72]

TASK: [nginx | install default.conf file] *****
changed: [10.1.6.73]
changed: [10.1.6.72]
```

成功

9、该端口测试、传递参数方式

```
[root@localhost roles]# ansible-playbook nginx.yml -t ngxconf -e ngxprot=80
```

```
PLAY [web] *****
```

```
GATHERING FACTS *****
```

```
ok: [10.1.6.72]
```

```
ok: [10.1.6.73]
```

```
TASK: [nginx | install nginx.conf file] *****测试后是成功的*****
```

```
ok: [10.1.6.73]
```

```
ok: [10.1.6.72]
```

```
TASK: [nginx | install default.conf file] *****
```

```
changed: [10.1.6.73]
```

```
changed: [10.1.6.72]
```

```
[root@qzx ~]# ss -tnl
```

| State  | Recv-Q | Send-Q | Local Address:Port |
|--------|--------|--------|--------------------|
| LISTEN | 0      | 5      | 192.168.122.1:53   |
| LISTEN | 0      | 128    | *:22               |
| LISTEN | 0      | 128    | 127.0.0.1:631      |
| LISTEN | 0      | 128    | *:8088             |
| LISTEN | 0      | 100    | 127.0.0.1:25       |
| LISTEN | 0      | 128    | 127.0.0.1:6010     |
| LISTEN | 0      | 128    | :::22              |
| LISTEN | 0      | 128    | :::1:631           |
| LISTEN | 0      | 100    | :::1:25            |
| LISTEN | 0      | 128    | :::1:6010          |

```
[root@qzx ~]# ss -tnl
```

| State  | Recv-Q | Send-Q | Local Address:Port |
|--------|--------|--------|--------------------|
| LISTEN | 0      | 128    | *:80               |
| LISTEN | 0      | 5      | 192.168.122.1:53   |

## 运维自动化的最佳实践探索

运维自动化的一些认识和实践，包括如下八点：

1. 自动化需要整体规划
2. 自动化的基础是标准化
3. 首先从持续交付开始
4. DevOps 的四观
5. 善于借助研测的力量
6. 不一定强依赖 CMDB
7. 以 NO OPS 为最终目标
8. Docker 等不是干掉运维

以下为详细内容，敬请欣赏。

### 1. 自动化需要整体规划

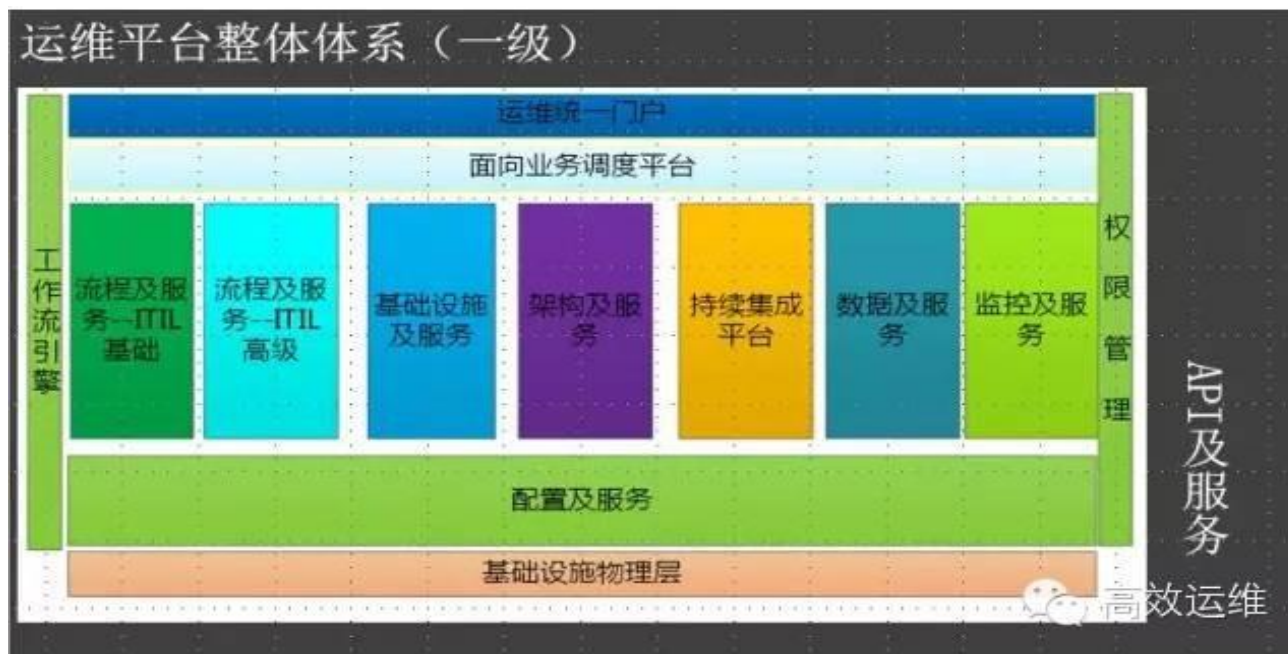
**认识1：自动化是需要有整体的规划、边界和运维场景的识别**

没有整体的规划始终觉得运维是在建一个个的工具，没法形成递进式的实现策略。

边界的识别是通过分层体系来构建 DevOps 自动化工具栈，而不是用一个工具解决所有问题，和智锦的观点类似：

千万不要以为 puppet/salt/ansible 所管理的配置工具能够解决所有运维自动化的问题（不过小企业运维另论哈）。

运维场景是寻找自动化平台实现的驱动力，可以衡量成本和收益比，不要为了自动化而自动化。



我把其中的每一块都抽象成服务，比如说基础设施及服务（IAAS 部分）、配置及服务、流程及服务（ITIL 部分）、架构及服务（PAAS 部分）、数据及服务、监控及服务等等。

**持续集成平台，我把他单独提出来，它很特别：**是一个应用交付的主线，他涉及的点很多，自动化编译、自动化测试、自动化部署等等，另外横跨了多个团队，带来的收益很高。

**监控及服务也很特别，对于我来说，一切数据都应该有监控的能力，所以我更多觉得监控是一种数据化的应用，和数据分析一样，个人监控观点是“先数据、后监控”。**

## 自动化平台分层体系



我习惯把它们映射到对应的层次上，对每一层的自动化手段都不一样，其实我觉得底层的资源及服务和服务层应该越简单越好，最好不要在系统层面上有依赖，比如说特殊的网络设置，特殊的 DNS 设置。

严格禁止系统内部调用通过运维系统路径，比如说 DNS、LVS，目的就是为了简化服务间的依赖，对于运维来说部署一个完整的服务，就要做到部署一个包这么简单。

## 自动化平台场景化分类



另外一个维度就是运维场景的识别，业务形态不一样，场景就不一样，逐步挑选对运维收益最大的部分自动化实现它。

### 2. 自动化的基础是标准化

#### 认识2：自动化必须以标准化为基础

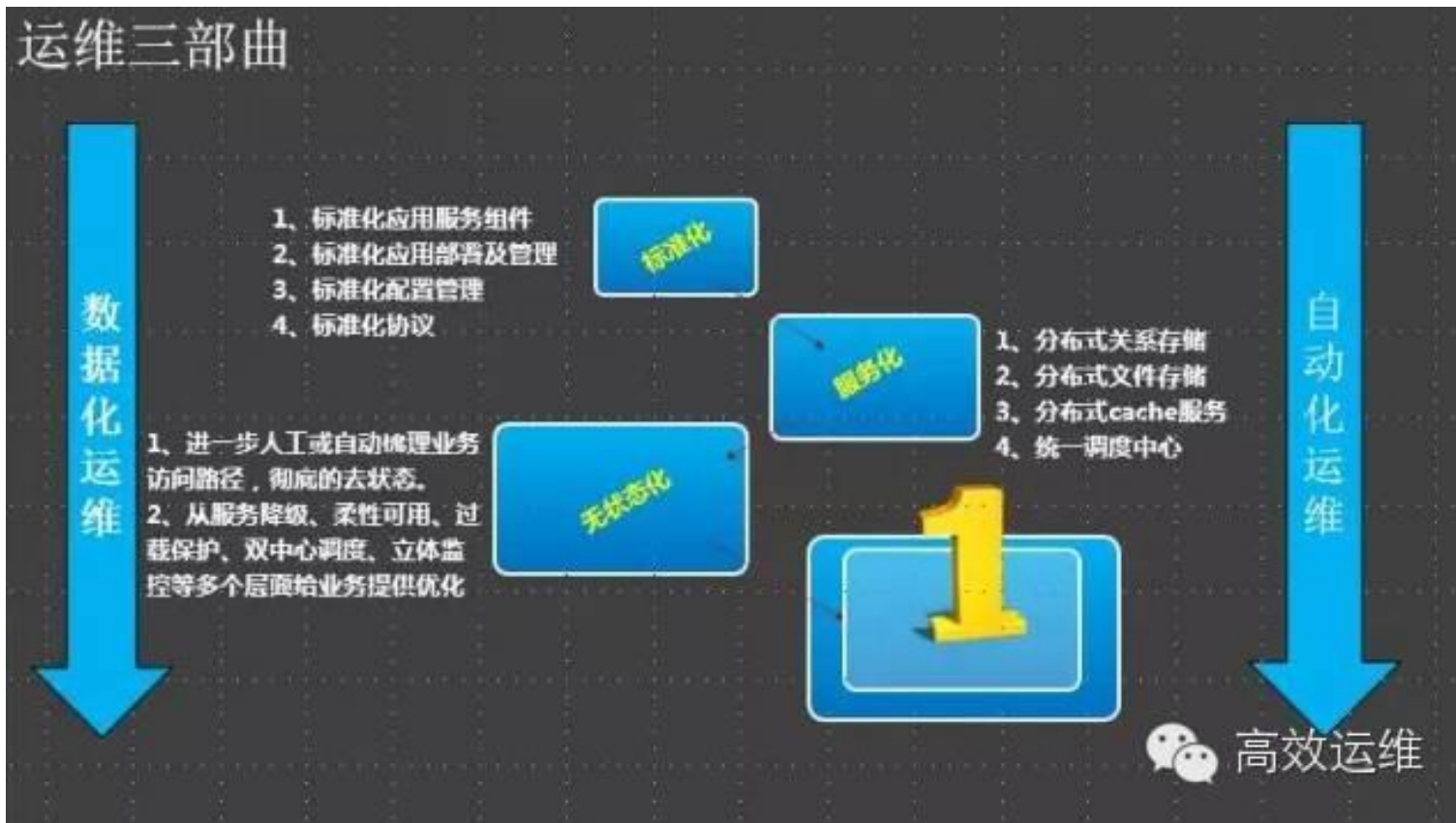
自动化平台必须是经验交付平台，而非技术平台。

技术是经验的一部分，而我想强调的是，做每一个自动化平台都需要搞清楚：

1. 自己的自动化平台要解决的问题；
2. 能达到的收益；
3. 使用 user 是谁等。

我将在后面持续交付平台中继续体现这个思路。





我个人认为标准化能体现你对运维理解的**精准度**及**勇气**。标准化的推进很需要运维的勇气，否则没法影响研发按照自己的节奏走：标准化是让人和系统更有效率和效力的做事：效率是快速的做事、效力是正确的做事。

在 UC，我完全是按照这套方法论和节奏去推进运维工作，自动化一定是随着业务发展而发展的。

更需要指出的是，越到后续阶段，运维工作更是需要和研发、测试深度合作完成，所以运维自动化不能忽略研发、测试。另外：

- 各个部分对自动化都有影响，比如说标准化部分的配置标准化（让配置管理更简单）；
- 服务化部分的组件向公共服务组件迁移（服务通过 **API** 暴露，让服务的管理更简单）；
- 无状态化的服务双中心改造，服务高可用也是依赖技术而非人来解决。

接下来举两个小例子。

## 标准化之应用环境管理



高效运维

这个图中两个重要的标准化工作就是配置管理标准化和应用包的标准化，基于此构造的持续部署系统非常简单，不用兼容业务的个性化特点。

## 标准化之容器标准化



基于netty的java  
容器框架

高效运维

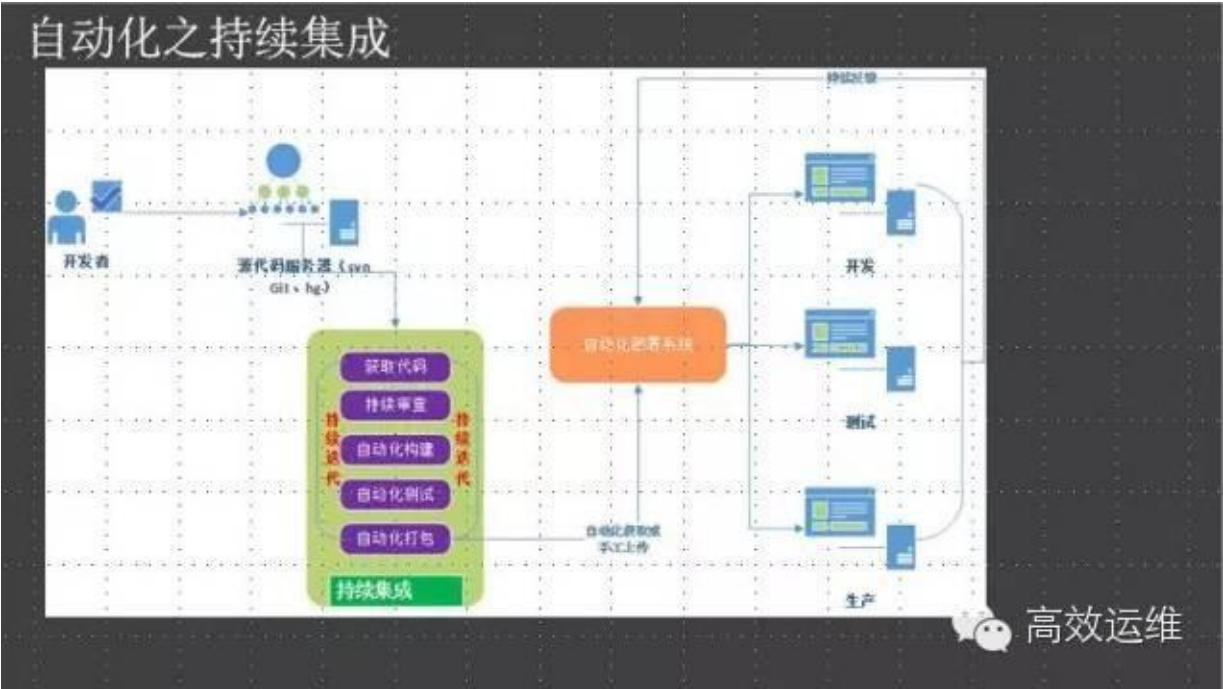


很庆幸，我们把所有的 java 容器全部迁移到自研的容器上，我们的容器接管了所有共性的研发服务，比如 http 服务，缓存服务、配置服务...完全插件化的服务容器。这是可运维性一个很好的例子。

3. 首先从持续交付开始

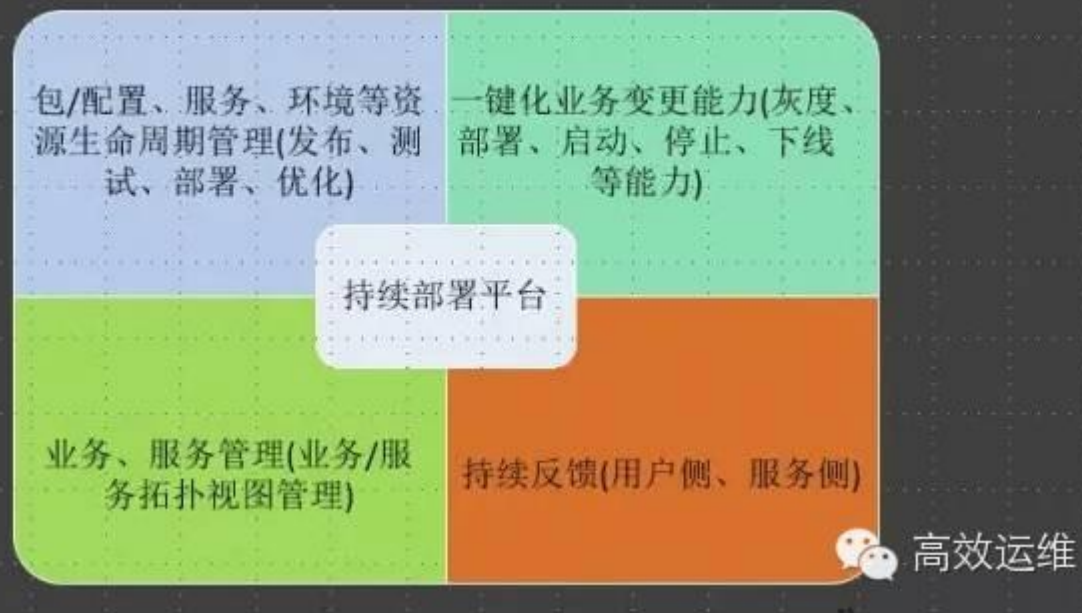
认识3：持续交付可以作为运维自动化首选

有些人问，运维自动化该如何开始？  
我的一般建议都是把持续集成或者 **CMDB** 作为开始主线。持续交付带来的是多个团队的利益和价值，这个工作可以由运维牵头来解决，运维解决的方法可以先从持续部署开始，然后在对接上面的持续集成：  
持续部署系统的建设可以联合研发、测试和运维来建设，方便推广，降低噪音（反对声）。  
另外，持续交付系统会反向推动运维内部的一些标准化工作，比如说环境标准化、应用标准化等等，因为你的部署要越来越简单。



持续集成+持续部署等于持续交付，实现面向用户产品功能的持续交付。  
但持续集成仅仅是面向应用交付的能力封装，还不能够成为运维自动化能力的全部，往上有面向业务的全流程自动化（调度平台，实现任务封装），往下有 **OS** 级的自动化（配置管理）等等。

## 持续部署的业务目标



这是我做持续集成系统定的一个业务目标（供参考）。持续部署必须要包含对要管理对象的生命周期的管理、一键化变更管理能力，同时还需要对变更后的结果做到持续反馈。业务和服务拓扑是基于之前配置标准化的一个能力实现，没有放到 CMDB 中。

## 持续部署平台的收益



当前我们实现持续部署能力有有两套方案，目前 UC 使用的基于 Cloud Foundry 封装的 UAE 平台。

这种对应用有改造要求的 **PAAS** 平台不是太好：

一则太复杂，二则底层服务有依赖（比如 **agent** 重启，业务进程也要重启）

第二种方案，就是无侵入的运维方案，把运维对持续部署的控制，封装成标准的事件，大家看看 **RPM** 包里面的能力就好理解了，再结合持续集成和持续交付的理念，把他们做成可视化。

#### 4. DevOps 的四观

**认识4:DevOps没有具体的形态，而是一种四观问题  
(文化观、价值观、思维观、工具观)**

我自己对 **DevOps** 的一点认识：

文化观：突破部门墙、深度合作的文化。

价值观：持续优化、共享责任、杜绝浪费、关注用户。

共享责任是合作的内在细化，谈合作太虚无缥缈。

思维观：精益、价值、跨界、敏捷

工具观：自动化平台集（实现价值的交付）+数据化平台集（实现交付价值的衡量）

我眼中的DevOps

**DevOps是Dev用D的能力延伸到Ops(Ops-friendly)，而Ops则把O的能力传递到Dev(Dev-friendly)，确保高质量、持续、快速向用户的交付价值 (user-friendly)。**

为什么不是 **DevTest**？为什么不是 **TestOps**？为什么不是 **DTO**？

我自己的理解是 **D** 和 **O** 代表面向用户服务能力的两端，两端能力的对接是优化的极致。

运维人很多时候都和开发提运维友好，却忽略我们自己要做的东西也要研发友好。所以很多时候不要站在“我”的需求立场上，而是“我们”。

14 年 **DevOps** 调查报告，指出要“自动化、自动化还是自动化”。



运维自动化就是要解决运维团队服务能力的吞吐率和延时问题，也即如何更多、更快的提供运维服务，其实是和线上的服务能力一样的。

需要思考的问题是，制约我们服务能力的关键因素，是资源约束（服务器不够）、还是架构约束（架构公共化能力不强）、还是运维服务约束（运维基础平台 DNS/LVS 服务能力）？



这个可以不断的驱动我们思考 DevOps 的产品形态和状态了。当然人的意识很重要哈，D/O 分离应该走向 DO 合作。

## 5. 善于借助研测的力量

### 认识5：运维自动化不要忽略研测的力量

运维自动化的工作少不了研发测试的支持，有时候运维复杂的系统封装，结果在研发侧做一些小小的改造就可以了，然后部门墙和彼此的强势文化导致 DO 是分离，而不是合作。

我举两个例子，配置管理和名字服务。

# 配置管理标准化

| 功能模块       | 配置文件名称                    | 备注                              |
|------------|---------------------------|---------------------------------|
| 主控配置       | app[redacted].conf        | 不包含端口、pool信息。                   |
| 项目jar包依赖配置 | deper[redacted].yaml      |                                 |
| 信息提示配置     | mess[redacted]            |                                 |
| 路由配置       | routes                    |                                 |
| 环境变量配置     | define.conf               | 各环境唯一不同的配置文件                    |
| 业务类配置biz   | business.conf caches.xml等 | 由研发管理，无需运维关注                    |
| 环境类配置env   | 账号安全配置                    | account-sdk.conf                |
|            | 白名单配置                     | acl.conf                        |
|            | 缓存类配置（必选）                 | cache-servers.xml               |
|            | 计划任务配置                    | cron.conf                       |
|            | 数据库配置（必选）                 | database-servers.conf           |
|            | http配置（必选）                | http.conf                       |
|            | 日志配置（必选）                  | log4j.properties                |
|            | 调用外部系统配置                  | out-system.conf                 |
|            | [redacted]                | [redacted]                      |
|            | 基础支撑服务配置                  | fdfs.conf/ucmq.conf/sphinx.conf |
|            | 邮件配置                      | smtp.conf                       |

奥妙在于此？

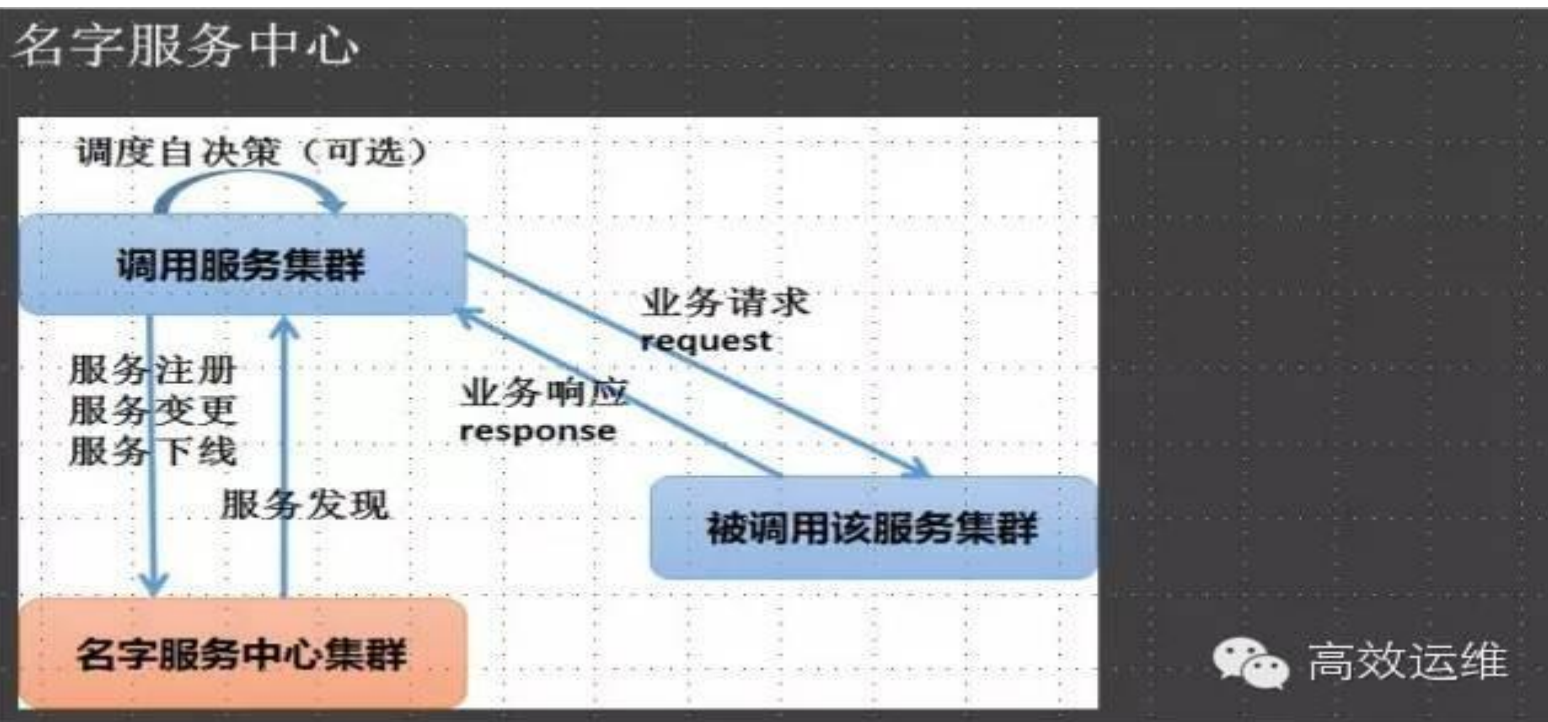
各环境唯一不同的配置文件

由研发管理，无需运维关注

端口、模板、压缩、解密文件等配置

高效运维

define.conf 是把其他底层配置在研发、测试和生产环境的差异消除掉，底层配置文件中采用变量配置方法，通过 define.conf 在三个环境中定义具体的值来简化配置管理，持续部署系统就变得极度简单，因为只需要管理一个 define.conf 配置即可。因为我们业务规模很小，所以没有提配置中心，配置中心在规模服务化运维下，必须要构造的一个基础服务。需要研发深度支持配合！



名字服务中心是我的个人情节，终于在 UC 变成了现实。

我相信很多中小企业的服务调用都是通过 DNS 获取服务地址来实现调用的，这个缺点就是故障的时候，需要运维深度参与故障处理（TTL、JDK 缓存等等）

我们为此特意建立一个名字服务中心，通过它来实现服务名到实例的具体映射，同时调用端统一实现服务的调度、容错。

现在内部业务故障，基本上不需要运维参与切换和处理了，大大简化运维故障处理系统的复杂度，还有服务扩容的时候，服务自动注册不需要人为修改配置文件，更加的自动化。需要研发深度支持配合！

## 6. 不一定强依赖 CMDB

不要觉得 CMDB 是自动化的前提。cmdb 和自动化平台的关系有两种：

1. 自动化平台与 CMDB 的关联发生在某些场景下的某些流程片段，比如说业务上线流程中的资源自动化申请，从 CMDB 获取信息。
2. 自动化平台把变更的信息回写到 CMDB，比如说应用部署系统 / DNS 系统 / LVS 系统等信息，目标是把 CMDB 作为元数据管理的平台，它可以收敛之上服务之间的数据获取接口。

当前我们是这么干的，但是发挥作用还不大。特别是业务规模不大，而业务变化不是非常频繁时，CMDB 的管理仅仅只需要记录资源被谁，用在哪个业务上即可。在公有云 IaaS 平台下，CMDB 的形态就更简单了。

## 7. 以 NO OPS 为最终目标

认识7：运维自动化以“no ops”为目标，以“可视化”为要求

运维自动化以挑战每个运维职责部分的“no ops”为目标，比如说服务器交付 / 应用交付 / 组件服务交付等等。最终这种“no ops”的运维平台可以让研发自己去完成（持续部署交给研发），也可以让用户来帮我们决策完成（容量驱动变更）把这些专业的服务能力可视化封装起来，提供 **API**，供其他关联服务调用，体现自服务能力。每个人运维人应该带着可视化管理的要求去面对自己的日常工作，这样可以确保自动化的能力每个人都能获取，并且执行结果是一致的。

## 8. Docker 等不是干掉运维

认识8：Docker/微服务/名字服务都是技术最佳实践，他们是在简化运维管理，而不是让运维消失

个人认为对运维的影响不是这些技术，而是基于该技术之上的产品化能力。但是不同服务能力的结合，爆发出来的能量需要运维人注意：

比如说 IaaS 平台和 Docker 技术结合，服务调度和名字服务中心结合，微服务对技术架构标准化影响。

因此 DevOps 不仅仅是对 D 有 Ops 能力的要求，对 O 来说，也要有 Dev 能力的要求。另外大家可以看看运维还有哪些工作，就知道这些技术对运维的影响了。