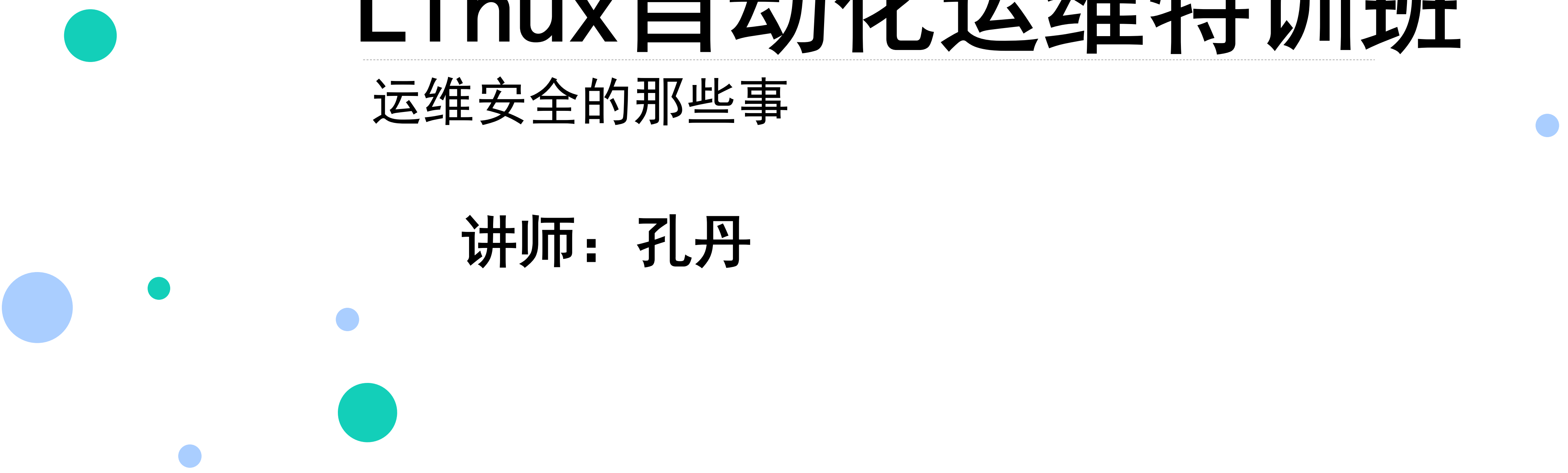


Linux自动化运维特训班

Ansible自动化运维

讲师：孔丹



大纲

- Ansible 架构及特点
- Ansible 安装与配置
- Ansible 组件
- playbook 详解
- playbook 进阶

版权所有，侵权必究

Ansible 架构及特点

▣ Ansible 软件及公司

ansible是个什么东西呢？官方的title是“Ansible is Simple IT Automation”——简单的自动化IT工具。

这个工具的目标有这么几项：让我们自动化部署APP；自动化管理配置项；自动化的持续交付；自动化的（AWS）云服务管理。

AnsibleWorks成立于2012年，由自动化工具Cobbler及Func的开发者Michael DeHaan创建。其Ansible平台是一个开源的配置及计算机管理平台。可实现多节点的软件部署，执行特定任务并进行配置管理。

Ansible跟其他IT自动化技术的区别在于其关注点并非配置管理、应用部署或IT流程工作流，而是提供一个统一的界面来协调所有的IT自动化功能，因此Ansible的系统更加易用，部署更快。受管理的节点无需安装额外的远程控制软件，由平台通过SSH（Secure SHell）对其进行管理，因此十分方便。其模块支持JSON等标准输出格式，可采用任何编程语言重写。

版权所有，侵权必究

Ansible 架构及特点

▣ Ansible 软件及公司

Ansible可以让用户避免编写脚本或代码来管理应用，同时还能搭建工作流实现IT任务的自动化执行。IT自动化可以降低技术门槛及对传统IT的依赖，从而加快项目的交付速度。

目前，Ansible已有30万用户，每月下载量接近3万次。其客户包括Rackspace 、 Twitter 、 Evernote 、 NASA、 GoPro、 Atlassian AppDynamics、 MapR以及金融服务、电信、医疗保健、媒体业的财富500强公司。

<https://www.ansible.com/>

<https://docs.ansible.com/ansible/latest/index.html>

Ansible中文权威指南

<https://ansible-tran.readthedocs.io/en/latest/>

版权所有，侵权必究

Ansible 架构及特点

□ Ansible 设计理念

- 安装部署过程特别简单，学习曲线很平坦。
- 管理主机便捷，支持多台主机并行管理。
- 避免在被管理主机上安装客户代理，打开额外端口，采用无代理方式，只是利用现有的 SSH 后台进程。
- 用于描述基础架构的语言无论对机器还是对人都是友好的。
- 关注安全，很容易对执行的内容进行审计、评估、重写。
- 能够立即管理远程被管理主机，不需要预先安装任何软件。
- 不仅仅支持 Python，可运行使用任何动态语言开发模块。
- 非 root 账户也可以使用。
- 成为最简单、易用的 IT 自动化系统

版权所有，侵权必究

Ansible 架构及特点

□ Ansible 优缺点

使用ansible这样的自动化配置管理工具，主要的优点是：

- ∅ 轻量级，他不需要去客户端安装agent，更新时，只需要在操作机上进行一次更新即可
- ∅ 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
- ∅ 使用python编写的，维护更简单
- ∅ 支持sudo

缺点：

- ∅ 对于几千台、上万台机器的操作，还不清楚性能、效率情况如何，需要进一步了解

版权所有，侵权必究

Ansible 架构及特点

▣ Ansible 应用领域

Ansible 的编排引擎可以出色地完成配置管理 、 流程控制 、 资源部署等多方面工作 。

1. 配置管理

2. 服务即时开通

这个领域的工具主要是在数据中心 、 虚拟化环境 、 云计算中快速开通新的主机。

3. 应用部署

这个领域的工具重点关注如何尽可能地零停机部署应用。

4. 流程编排

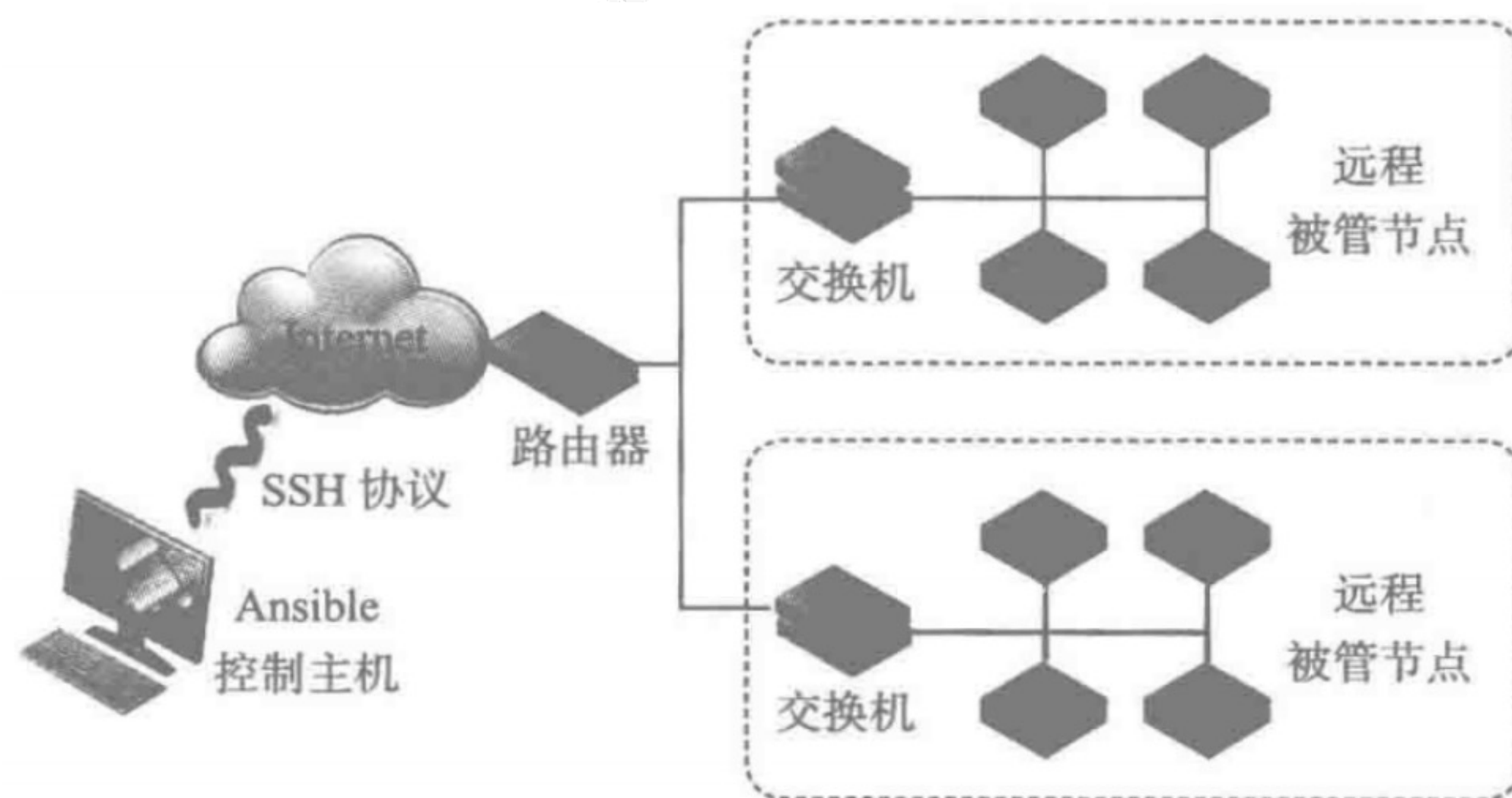
流程编排主要是进行部署时候如何保证基础架构中的各种组件协调一致。

版权所有，侵权必究

Ansible 架构及特点

□ Ansible 管理方式

Ansible 是一个模型驱动的配置管理器，支持多节点发布、远程任务执行。默认使用SSH 进行远程连接。无需在被管节点上安装附加软件，可使用各种编程语言进行扩展。

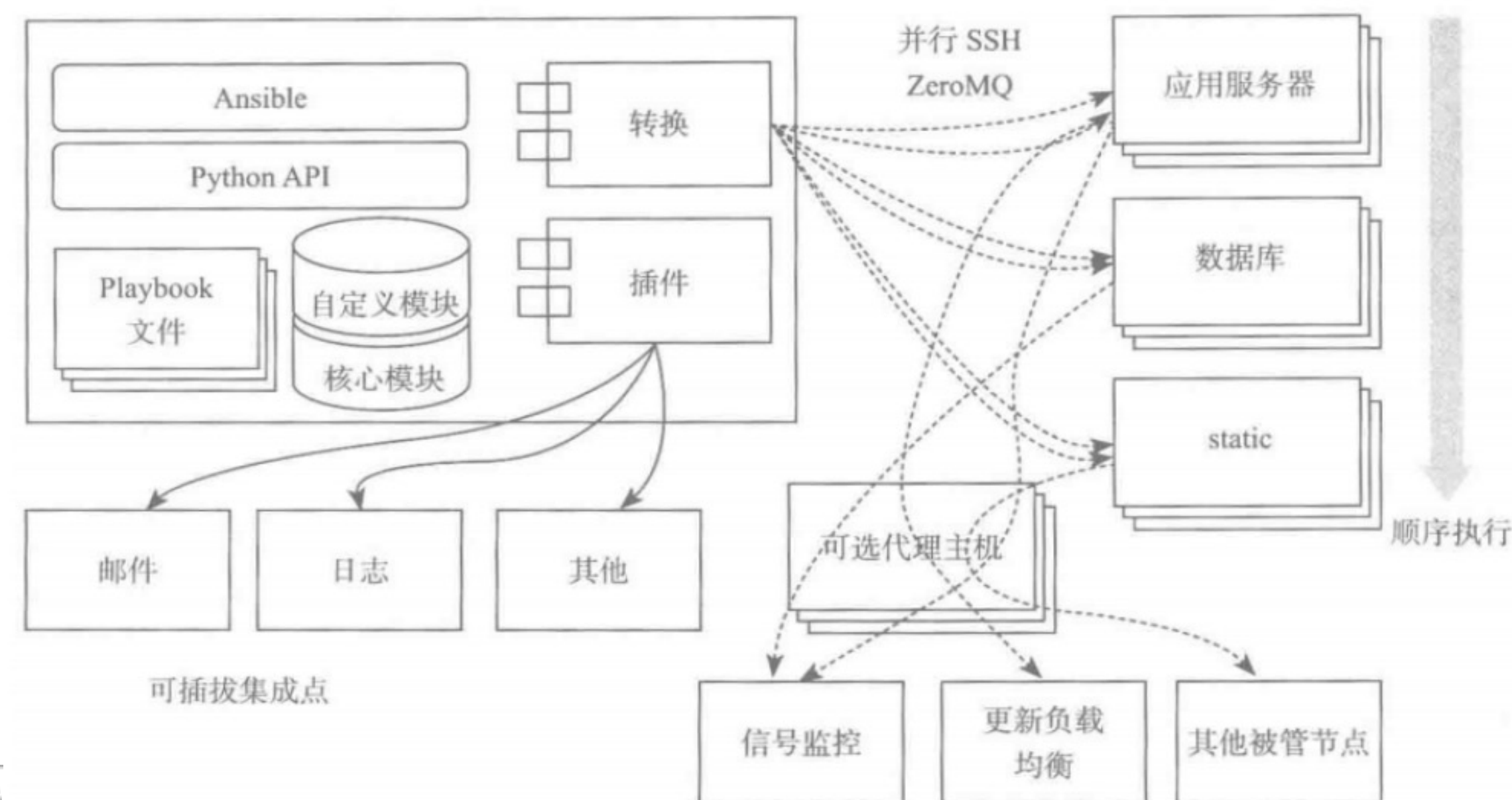


版权所有，侵权必究

Ansible 架构及特点

□ Ansible 系统架构

Ansible 是基于模块工作的，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 由以下各部分组成：



版权所有，仅供学习

Ansible 架构及特点

□ Ansible 系统架构

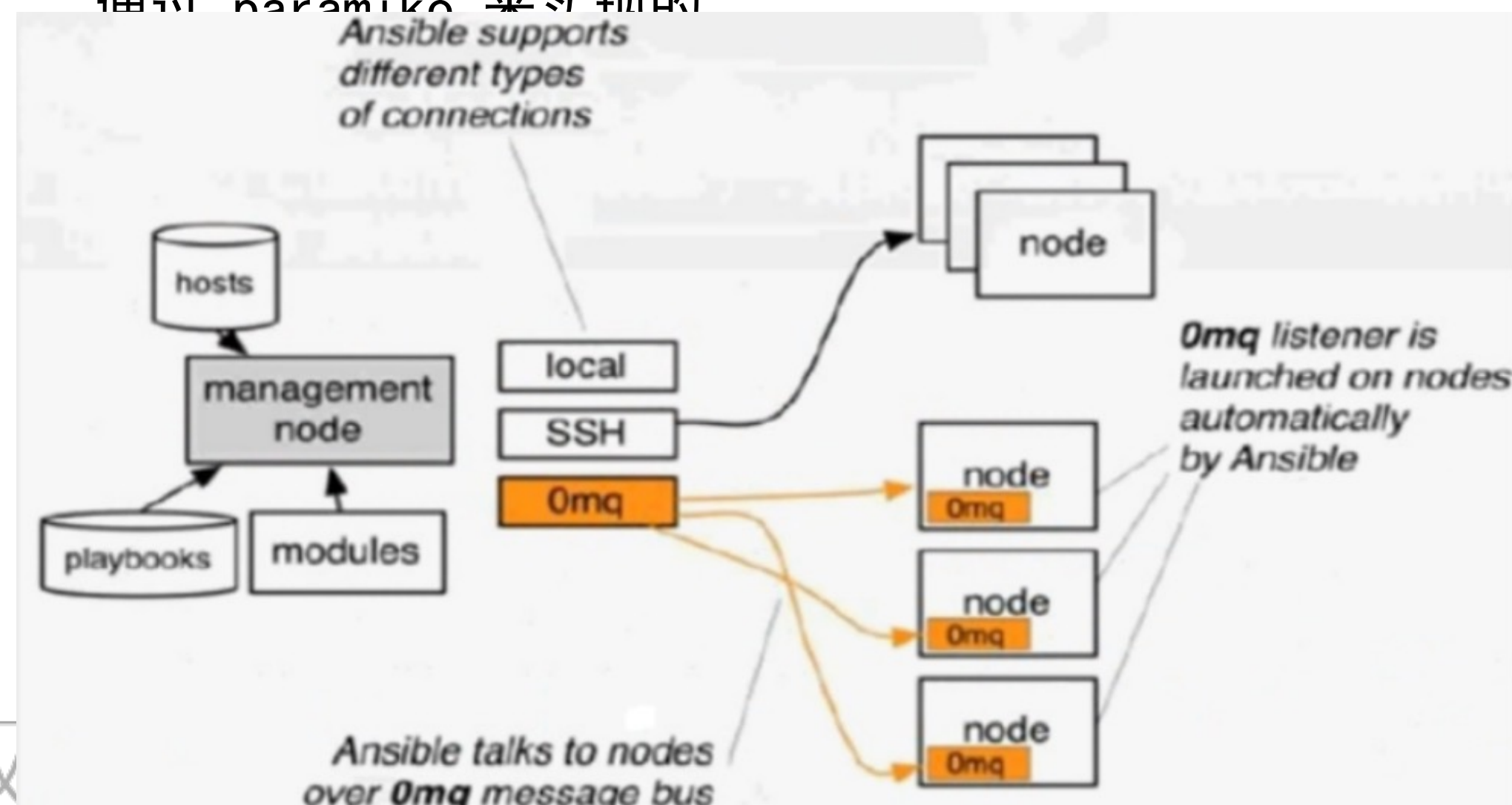
- 核心引擎：即 Ansible。
- 核心模块（core modules）：这些都是Ansible自带的模块。
- 自定义模块（custom modules）：如果核心模块不足以完成某种功能，可以添加自定义模块。
- 插件（plugins）：完成模块功能的补充，借助于插件完成记录日志、邮件等功能。
- 剧本（playbook）：定义Ansible任务的配置文件，可以将多个任务定义在一个剧本中，由 Ansible 自动执行，剧本执行支持多个任务，可以由控制主机运行多个任务，同时对多台远程主机进行管理。
- playbook是Ansible 的配置、部署和编排语言，可以描述一个你想要的远程系统执行策略，或一组步骤的一般过程。
- 连接插件（connector plugins）：Ansible 基于连接插件连接到各个主机上，负责和被管节点实现通信。
- 主机清单（host inventory）：定义Ansible管理的主机策略，默认是在Ansible的hosts配置文件中定义被管节点

Ansible 架构及特点

□ Ansible 工作原理

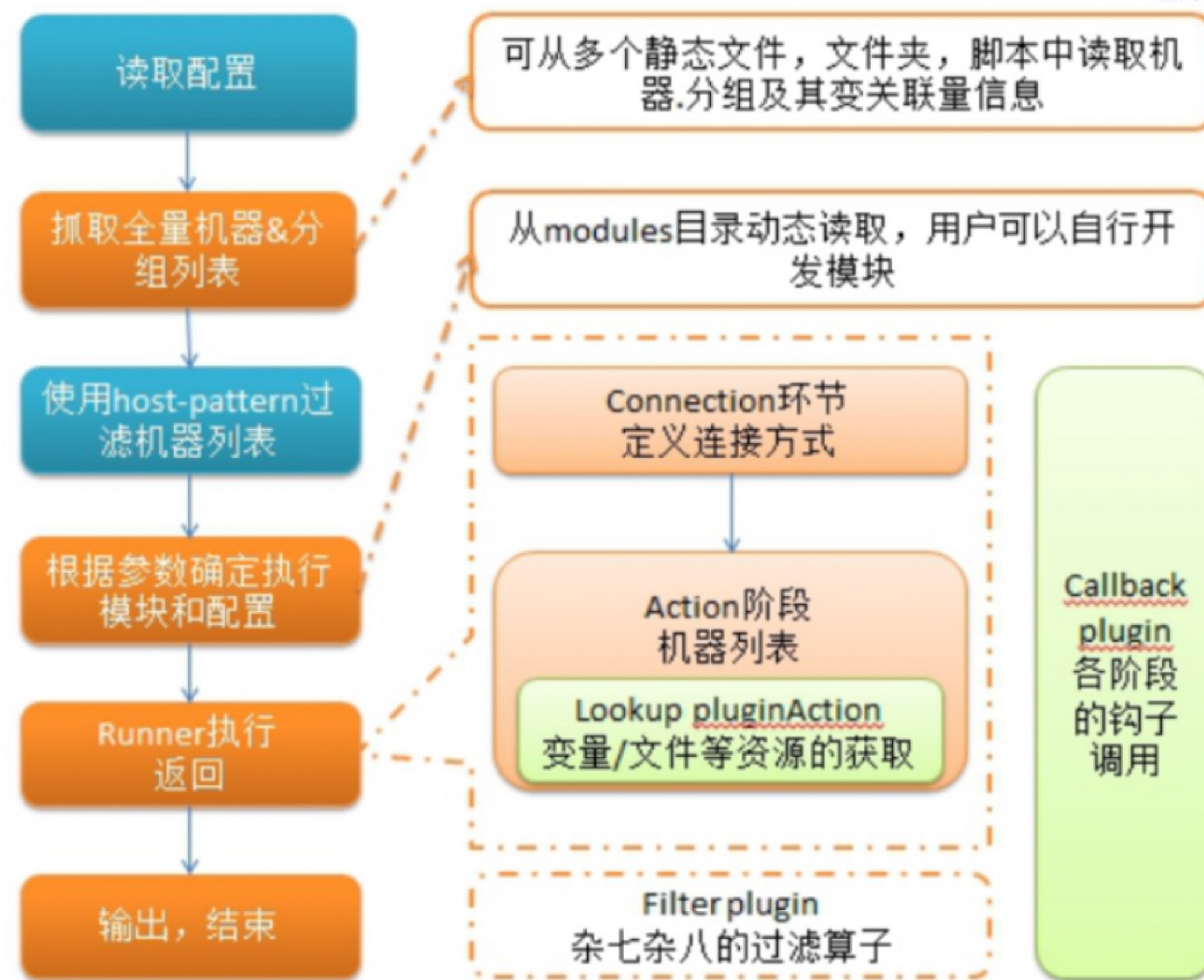
- 1、管理端支持 local 、 ssh、 zeromq 三种方式连接被管理端，默认使用基于ssh的连接——这部分对应基本架构图中的连接模块；
- 2、可以按应用类型等方式进行Host Inventory（主机群）分类，管理节点通过各类模块实现相应的操作——单个模块，单条命令的批量执行，我们可以称之为ad-hoc；
- 3、管理节点可以通过playbooks 实现多个task的集合实现一类功能，如web服务的安装部署、数据库服务器的批量备份等。playbooks我们可以简单的理解为，系统通过组合多条ad-hoc操作的配置文件。

Ansible默认是通过SSH通道来管理的，也就是它所说的免客户端方式管理，它底层是通过 paramiko 来实现的



Ansible 架构及特点

Ansible 执行过程



版权所有，侵权必究

Ansible 架构及特点

▣ Ansible、Puppet、SaltStack 技术特性比较

项目	Puppet	SaltStack	Ansible
开发语言	Ruby	Python	Python
是否有客户端	有	有	无
是否支持二次开发	不支持	支持	支持
服务器与远程机器是否相互验证	是	是	是
服务器与远程机器通信是否加密	是，标准SSL	是，使用AES	是，使用OpenSSH
平台支持	支持 AIX 、 BSD 、 HP-UX 、 Linux 、 Mac OS X 、 Solaris、 Windows	支持 BSD 、 Linux 、 MacOS X、 Solaris、 Windows	支持 AIX、 BSD、 HP-UX 、 Linux 、 Mac OS X 、 Solaris
是否提供 Web UI	提供	提供	提供，不过是商业版本
配置文件格式	Ruby	YAML	YAML
命令行执行	不支持，但可以通过配置模块实现	支持	支持

Ansible 安装配置

▣ Ansible 环境

角色	主机名	IP地址	组名	CPU（核数）	Web根目录
控制主机	ansible	192.168.150.11	---	---	---
被管节点	web1	192.168.150.12	webservers	2	/website
被管节点	web2	192.168.150.13	webservers	2	/website

版权所有，侵权必究

Ansible 安装配置

□ 安装前提

对管理主机要求

Currently Ansible can be run from any machine with Python 2.6 installed (Windows isn't supported for the control machine). This includes Red Hat, Debian, CentOS, OS X, any of the BSDs, and so on.

目前，只要机器上安装了 Python 2.6 (windows系统不可以做控制主机)，都可以运行Ansible.

主机的系统可以是 Red Hat, Debian, CentOS, OS X, BSD的各种版本，等等。

对托管主机要求

On the managed nodes, you only need Python 2.4 or later, but if you are running less than Python 2.5 on the remotes, you will also need:

托管节点上需要安装 Python 2.4 及以上的版本。但如果版本低于 Python 2.5，则需要额外安装一个模块：

python-simplejson

版权所有，侵权必究

Ansible 安装配置

□ 安装Ansible

源码安装

源码安装需要python2.6以上版本，其依赖模块

paramiko、PyYAML、Jinja2、httplib2、simplejson、
pycrypto模块，以上模块可以通过pip或easy_install 进行安装

pip安装

pip是专门用来管理Python模块的工具，Ansible会将每次正式发布都更新到pip仓库中。所以通过pip安装或更新Ansible，会比较稳妥的拿到最新稳定版。

Ansible 安装配置

□ 安装Ansible yum安装

首先，安装使用EPEL YUM源

// 安装清华的镜像EPEL源

```
[root@ansiblecontrol ~]# yum install -y  
https://mirrors.tuna.tsinghua.edu.cn/epel/epel-release-latest-  
7.noarch.rpm
```

// 安装ansible

```
[root@ansiblecontrol ~]# yum install -y ansible
```

```
[root@ansiblecontrol ~]# ansible --version
```

版权所有，侵权必究

Ansible 安装配置

□ 配置运行环境

配置 Ansible 环境

Ansible 配置文件是以 ini 格式存储配置数据的，在 Ansible 中，几乎所有的配置项都可以通过 Ansible 的 playbook 或环境变量来重新赋值。在运行 Ansible 命令时，命令将会按照预先设定的顺序查找配置文件，如下所示：

- 1) `ANSIBLE_CONFIG` : 首先，Ansible 命令会检查环境变量，及这个环境变量将指向的配置文件。
- 2) `./ansible.cfg`: 其次，将会检查当前目录下的 `ansible.cfg` 配置文件。
- 3) `~/.ansible.cfg`: 再次，将会检查当前用户 home 目录下的 `.ansible.cfg` 配置文件。
- 4) `/etc/ansible/ansible.cfg` : 最后，将会检查在用软件包管理工具安装 Ansible 时自动产生的配置文件。

Ansible 安装配置

□ 配置运行环境

配置 Ansible 环境

1. 使用环境变量方式来配置

大多数的 Ansible 参数可以通过设置带有 `ANSIBLE_` 开头的环境变量进行配置，参数名称必须都是大写字母，如下配置项：

```
export ANSIBLE_SUDO_USER=root
```

设置了环境变量之后，`ANSIBLE_SUDO_USER` 就可以在 `playbook` 中直接引用。

2. 设置 `ansible.cfg` 配置参数

- `inventory` : 这个参数表示资源清单 `inventory` 文件的位置
- `library` : 这个 `library` 参数就是指向存放 Ansible 模块的目录
- `forks` : 设置默认情况下 Ansible 最多能有多少个进程同时工作，默认设置最多5 个进程并行处理。
- `sudo_user` : 设置默认执行命令的用户
- `remote_port` : 指定连接被管节点的管理端口，默认是 22
- `host_key_checking` : 这是设置是否检查 SSH 主机的密钥。
- `timeout` : 这是设置 SSH 连接的超时间隔，单位是秒
- `log_path` : Ansible 系统默认是不记录日志的，如果要把 Ansible 系统的输出记录到日志文件中，需要设置 `_` 来指定一个存储 日志的文件

Ansible 安装配置

□ 配置运行环境 使用公钥认证

现在运维对安全要求都是比较重视的，一般会采用密钥验证方式来登录，Ansible 1.2.1之后的版本都默认启用公钥认证

如果有台被管节点重新安装系统并在 known hosts 中有了与之前不同的密钥信息，就会提示一个密钥不匹配的错误信息，直到被纠正为止。在使用 Ansible 时，如果有台被管节点没有在 known_hosts 中被初始化，将会在使用 Ansible 或定时执行 Ansible 时提示对 key 信息的确认。

如果你不想出现这种情况，并且你明白禁用此项行为的含义，只要修改 home 目录下 ~/.ansible.cfg 或 /etc/ansible/ansible.cfg 的配置项：

```
[defaults]
host_key_checking = False
```

或者直接在控制主机的操作系统中设置环境变量，如下所示：

```
$export ANSIBLE_HOST_KEY_CHECKING=False
```

Ansible 安装配置

□ 配置运行环境

配置 Linux 主机 SSH 无密码访问

为了避免 Ansible 下发指令时输入目标主机密码，通过证书签名达到 SSH 无密码是一个好的方案。推荐使用 `ssh-keygen` 与 `ssh-copy-id` 来实现快速证书的生成及公钥下发，其中 `ssh-keygen` 生产一对密钥，使用 `ssh-copy-id` 来下发生成的公钥。

1、在控制主机（ansible）上创建密钥

```
[root@ansible ~]# ssh-keygen -f ~/.ssh/id_rsa -P '' -q
```

2、下发密钥到被管理节点

```
[root@ansible ~]# ssh-copy-id ansiblecontrol
```

```
[root@ansible ~]# ssh-copy-id web1
```

```
[root@ansible ~]# ssh-copy-id web2
```

版权所有，侵权必究

Ansible 安装配置

▣ Ansible初体验 主机联通性测试

1、修改主机与组配置

```
[root@ansible ~]# cat >> /etc/ansible/hosts << EOF
> [webservers]
> 192.168.150.12
> 192.168.150.13
> EOF
```

2. 使用ping模块测试

测试单台机器

```
[root@ansible ~]# ansible 192.168.150.12 -m ping
```

测试一组机器

```
[root@ansible ~]# ansible webservers -m ping
```

版权所有，侵权必究

Ansible 安装配置

▣ Ansible初体验 批量执行命令

使用ansible的shell模块在远程机器输出 “hello ansible”

```
[root@ansible ~]# ansible webservers -m shell -a '/bin/echo hello ansible!'
```

也可以使用command模块或raw模块

```
[root@ansible ~]# ansible webservers -m command -a '/bin/echo hello ansible!'
```

```
[root@ansible ~]# ansible webservers -m raw -a '/bin/echo hello ansible!'
```

版权所有，侵权必究

Ansible 安装配置

▣ Ansible初体验 获取帮助信息

在 Ansible 中有 8 个主要的 Ansible 管理工具，每个管理工具都是一系列的模块、参数支持。随时可获取的帮助信息对了解掌握 Ansible 系统非常重要。对于 Ansible 每个工具，都可以简单地在命令后面加上 `-h` 或 `-help` 直接获取帮助。

1 ansible-doc

该指令用于查看模块信息，常用参数有两个 `-l` 和 `-s`

//参数 `-l` 列出可使用的模块

```
[root@ansible ~]# ansible-doc -l
```

//`-s`列出某个模块支持的动作

```
[root@ansible ~]# ansible-doc -s shell
```

版权所有，侵权必究

Ansible 安装配置

▣ Ansible初体验 获取帮助信息

ansible

ansible是指令核心部分，其主要用于执行ad-hoc命令，即单条命令。默认后面需要跟主机和选项部分，默认不指定模块时，使用的是command模块。

```
# ansible -h
```

```
Usage: ansible <host-pattern> [options]
```

Define and run a single task 'playbook' against a set of hosts

Options:

```
-a MODULE_ARGS, --args=MODULE_ARGS
```

module arguments

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

```
-B SECONDS, --background=SECONDS
```

...

另外，在 Ansible 调试自动化脚本时候经常需要获取执行过程的详细信息，可以在命令后面添加 `-v` 或 `-vvv` 得到详细的输出结果。

Ansible 组件

□ Ansible Inventory

1. 定义主机和主机组

下面是一个具体的定义示例

```
192.168.150.11 ansible_ssh_pass='123456'  
192.168.150.12 ansible_ssh_pass='123456'  
[docker]  
192.168.150.1[1:3]  
[docker:vars]  
ansible_ssh_pass='123456'  
[ansible:children]  
docker
```

第1行和第2行各定义了一个主机是 192.168.150.11/12，然后使用

Inventory 内置变量定义了SSH 登录密码

第3行定义了一个组叫 docker

第4行定义了 docker 组下面 4 台主机从 192.168.150.11 到

192.168.150.13

第5行到第6行针对 docker组使用 Inventory内置变量定义了 SSH 登录密码

第7行到第 8 行定义了一个组叫 ansible，这个组下面包含 docker 组

版权所有，侵权必究

Ansible 组件

□ Ansible Inventory

1. 定义主机和主机组

下面是分别针对不同的主机和主机组进行 Ansible 的 ping 模块检测

```
ansible 192.168.150.11:192.168.150.12 -m ping -o
```

```
ansible docker -m ping -o
```

```
ansible ansible -m ping -o
```

版权所有，侵权必究

Ansible 组件

▣ Ansible Inventory

2. 多个 Inventory 列表

- 1、首先需要修改 `ansible.cfg` 中 `hosts` 文件的定义，或者使用 `ANSIBLE_HOSTS` 环境变量定义。这里我们准备一个文件夹，里面将存放多个 Inventory 文件，如以下目录所示：

```
# mkdir inventory
# touch inventory/{docker, hosts}
# tree inventory/
inventory/
├── docker
└── hosts
```

不同的文件可以存放不同的主机，内容如下：

```
# cat inventory/hosts
192.168.150.11 ansible_ssh_pass='123456'
192.168.150.12 ansible_ssh_pass='123456'
# cat inventory/docker
[docker]
192.168.150.1[1:3]
[docker:vars]
ansible_ssh_pass='123456'
[ansible:children]
docker
```

Ansible 组件

□ Ansible Inventory

2. 多个 Inventory 列表

2、修改了 ansible.cfg 文件中 inventory 的值

```
# vim /etc/ansible/ansible.cfg
```

```
[defaults]
```

```
inventory = /root/inventory/
```

3、使用 Ansible 的 list-hosts 参数来进行如下验证

```
# ansible 192.168.150.11:192.168.150.12 --list-hosts
```

```
# ansible docker --list-hosts
```

```
# ansible ansible --list-hosts
```

版权所有，侵权必究

Ansible 组件

□ Ansible Inventory

3. 动态 Inventory

在实际应用部署中会有大量的主机列表。如果手动维护这些列表将是一个非常繁琐的事情。其实 Ansible 还支持动态的 Inventory，动态 Inventory 就是 Ansible 所有的 Inventory 文件里面的主机列表和变量信息都支持从外部拉取。

关于引用动态 Inventory 的功能配置只需要把 `ansible.cfg` 文件中 `inventory` 的定义值改成一个执行脚本即可。这个脚本的内容不受任何编程语言限制，但是这个脚本使用参数时有一定的规范并且对脚本执行的结果也有要求。这个脚本需要支持两个参数：

- `--list` 或者 `-l`，这个参数运行后会显示所有的主机以及主机组的信息（JSON 格式）。
- `--host` 或者 `-H`，这个参数后面需要指定一个 `host`，运行结果会返回这台主机的所有信息（包括认证信息、主机变量等），也是 JSON 格式。

版权所有，侵权必究

Ansible 组件

□ Ansible Inventory

3. 动态 Inventory

这个hosts.py脚本定义了两个函数，lists在指定--list时执行，hosts在指定--host时执行

```
[root@ansible ~]# python hosts.py -l
{
    "docker": {
        "hosts": [
            "192.168.150.11",
            "192.168.150.12",
            "192.168.150.13"
        ]
    }
}
[root@ansible ~]# python hosts.py -H 192.168.150.12
{"ansible_ssh_pass": "123456"}
```

测试

```
[root@ansible ~]# chmod +x hosts.py
```

```
[root@ansible ~]# ansible -i hosts.py
```

```
192.168.150.12:192.168.150.13 -m ping -o
```

版权所有，侵权必究

Ansible 组件

□ Ansible Inventory

3. 动态 Inventory

测试:

```
# chmod +x hosts.py
```

```
ansible -i hosts.py 192.168.150.12:192.168.150.13 -m ping -o
```

版权所有，侵权必究

Ansible 组件

▣ Ansible Inventory

4. Inventory内置参数

参数	解释	例子
ansible_ssh_host	定义 host ssh 地址	ansible_ssh_host=192.168.1.117
ansible_ssh_port	定义 hosts ssh 端口	ansible_ssh_port=5000
ansible_ssh_user	定义 hosts ssh 认证用户	ansible_ssh_user=yadmin
ansible_ssh_pass	定义 hosts ssh 认证密码	ansible_ssh_user='123456'
ansible_sudo	定义 hosts sudo 用户	ansible_sudo=yadmin
ansible_sudo_pass	定义 hosts sudo 密码	ansible_sudo_pass='123456'
ansible_sudo_exe	定义 hosts sudo 路径	ansible_sudo_exe=/usr/bin/sudo
ansible_connection	定义 hosts 连接方式	ansible_connection=local
ansible_ssh_private_key_file	定义 hosts 私钥	ansible_ssh_private_key_file=/root/key
ansible_shell_type	定义 hosts shell 类型	ansible_shell_type=zsh
ansible_python_interpreter	定义 hosts 任务执行 python 路径	ansible_python_interpreter=/usr/bin/python2.6
ansible_*_interpreter	定义 hosts 其他语言解析器路径	ansible_ruby_interpreter=/usr/bin/ruby

版权所有，侵权必究

Ansible 组件

▣ Ansible Ad-Hoc 命令

执行命令

shell 模块

用法其本和command一样，不过的是其是通过/bin/sh进行执行，所以shell模块可以执行任何命令，就像在本机执行一样，“It is almost exactly like the command module but runs the command through a shell (/bin/sh) on the remote node.”

注解：shell模块调用的/bin/sh指令执行

Ansible 命令都是并发执行的，我们可以针对目标主机执行任何命令。

示例：

```
# ansible docker -m shell -a hostname -o
```

```
# ansible docker -m shell -a 'uname -r' -f 5 -o
```

使用异步执行功能，-P 0 的情况下会直接返回 job_id，然后针对主机根据 job_id 查询执行结果：

```
# ansible docker -B 120 -P 0 -m shell -a 'sleep 10;hostname' -f 5 -o
```

//根据 job_id 查询执行结果,当-P参数大于0时，Ansible会根据job_id去轮询查询执行结果

```
# ansible 192.168.150.12 -m async_status -a 'jid=127378635709.12539'
```


Ansible 组件

▣ Ansible Ad-Hoc 命令

执行命令

command模块

该模块通过-a跟上要执行的命令可以直接执行，不过命令里如果有带有如下字符部分则执行不成功 “so variables like \$HOME and operations like "<", ">", "|", and "&" will not work (use the shell module if you need these features).”

command模块包含如下选项：

- ∅ 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，先切换到该指定的目录
- ∅ 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

注解：command模块不是调用的shell的指令，所以没有bash的环境变量，也不能使用shell的一些操作方式，其他和shell没有区别

版权所有，侵权必究

Ansible 组件

▣ Ansible Ad-Hoc 命令

执行命令

raw模块

用法和shell 模块一样，其也可以执行任意命令，就像在本机执行一样，“Executes a low-down and dirty SSH command, not going through the module subsystem. There is no change handler support for this module. This module does not require python on the remote system”

版权所有，侵权必究

Ansible 组件

❑ Ansible Ad-Hoc 命令

包和服务管理

yum模块

使用yum包管理器来管理软件包，其选项有：

- Ø config_file: yum的配置文件
- Ø disable_gpg_check: 关闭gpg_check
- Ø disablerepo: 不启用某个源
- Ø enablerepo: 启用某个源
- Ø name: 要进行操作的软件包的名字，也可以传递一个url或者一个本地的rpm包的路径
- Ø state: 状态 (present, absent, latest)

示例：

```
# ansible docker -m yum -a 'name=httpd state=latest' -f 5 -o
```

验证：

```
# ansible docker -m shell -a 'rpm -q httpd' -o
```

版权所有，侵权必究

Ansible 组件

❑ Ansible Ad-Hoc 命令

包和服务管理

service模块

用于管理服务

该模块包含如下选项：

- Ø arguments: 给命令行提供一些选项
- Ø enabled: 是否开机启动 yes|no
- Ø name: 必选项，服务名称
- Ø pattern: 定义一个模式，如果通过status指令来查看服务的状态时，没有响应，就会通过ps指令在进程中根据该模式进行查找，如果匹配到，则认为该服务依然在运行
- Ø runlevel: 运行级别
- Ø sleep: 如果执行了restarted，在则stop和start之间沉睡几秒钟
- Ø state: 对当前服务执行启动，停止、重启、重新加载等操作
(started, stopped, restarted, reloaded)

示例：

```
# ansible docker -m service -a 'name=httpd state=started' -o
```

版权所有，侵权必究

Ansible 组件

❑ Ansible Ad-Hoc 命令

用户管理

user 模块

首先可以使用python的crypt命令来生成一个密码，因为 ansible user 的 password 参数需要接受加密后的值

```
>>> import crypt
>>> crypt.crypt('123')
'$6$Q0gb/5mYvx9j.doW$WGkpZH9RzUQiJrh700d73vNNmZUWX7/AiZg2WcaxmnSQ/
aYFTBORmYLHEpI xqkEXoeTgZdnJiLEwsFC8wRyta1'
>>> exit()
# ansible docker -m user -a 'name=tom
password="$6$Q0gb/5mYvx9j.doW$WGkpZH9RzUQiJrh700d73vNNmZUWX7/Ai
Zg2WcaxmnSQ/aYFTBORmYLHEpI xqkEXoeTgZdnJiLEwsFC8wRyta1"' -o
```

验证登录：

```
# ssh 192.168.95.152 -l tom
```

版权所有，侵权必究

Ansible 组件

▣ Ansible Ad-Hoc 命令 组管理

group模块

```
[root@test ansible]# ansible-doc -s group  
less 436
```

Copyright (C) 1984–2009 Mark Nudelman

less comes with NO WARRANTY, to the extent permitted by law.

For information about the terms of redistribution,
see the file named README in the less distribution.

Homepage: <http://www.greenwoodsoftware.com/less>

- name: Add or remove groups

action: group

gid # Optional 'GID' to set for the group.

name= # Name of the group to manage.

state # Whether the group should be present or not on the remote
system # If 'yes', indicates that the group created is a system

版权所有，侵权必究

Ansible 组件

□ Ansible Ad-Hoc 命令

文件操作

file模块：主要用于远程主机上的文件操作，file模块包含如下选项：

Ø 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

Ø group：定义文件/目录的属组

Ø mode：定义文件/目录的权限

Ø owner：定义文件/目录的属主

Ø path：必选项，定义文件/目录的路径

Ø recurse：递归的设置文件的属性，只对目录有效

Ø src：要被链接的源文件的路径，只应用于state=link的情况

Ø dest：被链接到的路径，只应用于state=link的情况

Ø state：

directory：如果目录不存在，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

Ansible 组件

▣ Ansible Ad-Hoc 命令

文件操作

file模块：主要用于远程主机上的文件操作

示例1：创建一个名为 testfile1 的文件，如果 testfile1 文件已经存在，则会更新文件的时间戳

```
# ansible docker -m file -a 'path=/tmp/testfile1 state=touch' -o
```

示例2：创建一个名为 `/tmp/testdir` 的目录，如果

`/testdir/testdir` 目录已经存在，则不进行任何操作。

```
# ansible docker -m file -a 'path=/tmp/testdir state=directory'
```

示例3：testfile1 文件创建软链接文件，软链接名为 linkfile1

```
# ansible docker -m file -a 'path=/tmp/linkfile1 state=link  
src=/tmp/testfile1'
```

示例4：删除远程机器上的指定文件或目录

```
# ansible docker -m file -a 'path=/tmp/linkfile1 state=absent'
```

```
# ansible docker -m file -a 'path=/tmp/testfile1 state=absent'
```

```
# ansible docker -m file -a 'path=/tmp/testdir state=absent'
```

版权所有，侵权必究

Ansible 组件

❑ Ansible Ad-Hoc 命令

文件操作

copy模块：复制文件使用copy模块，文件的变化是通过MD5值来判断

copy模块包含如下选项：

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

Ø content：用于替代"src"，可以直接设定指定文件的值

Ø 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

Ø directory_mode：递归的设定目录的权限，默认为系统默认权限

Ø 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

Ø others：所有的file模块里的选项都可以在这里使用

Ø src：要复制到远程主机的文件在本地的地址，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。

Ø validate : The validation command to run before copying into place. The path to the file to validate is passed in via '%s' which must be present as in the visudo example below.

Ansible 组件

□ Ansible Ad-Hoc 命令

文件操作

copy模块：复制文件使用copy模块，文件的变化是通过MD5值来判断

copy模块示例

```
# ansible docker -m copy -a 'src=hosts.py dest=/root/hosts.py  
owner=root group=root mode=644 backup=yes' -o
```

//验证文件下发

```
# ansible docker -m shell -a 'md5sum /root/hosts.py' -f 5 -o
```

版权所有，侵权必究

Ansible 组件

❑ Ansible Ad-Hoc 命令

计划任务

cron模块：用于管理计划任务

包含如下选项：

Ø backup：对远程主机上的原任务计划内容修改之前做备份

Ø cron_file：如果指定该选项，则用该文件替换远程主机上的cron.d目录

下的用户的任务计划

Ø day：日（1-31，*，*/2，.....）

Ø hour：小时（0-23，*，*/2，.....）

Ø minute：分钟（0-59，*，*/2，.....）

Ø month：月（1-12，*，*/2，.....）

Ø weekday：周（0-7，*，.....）

Ø job：要执行的任务，依赖于state=present

Ø name：该任务的描述

Ø special_time：指定什么时候执行，参数：

reboot, yearly, annually, monthly, weekly, daily, hourly

Ø state：确认该任务计划是创建还是删除

Ø user：以哪个用户的身份执行

版权所有，侵权必究

Ansible 组件

❑ Ansible Ad-Hoc 命令

计划任务

cron模块示例

示例1：创建计划任务，任务名称为” crontab test”，任务于每天1点5分执行，任务内容为输出test字符。

```
# ansible docker -m cron -a " name='crontab test' minute=5 hour=1  
job='echo test' "
```

示例2：删除计划任务

```
# ansible docker -m cron -a " name='crontab test' state=absent "
```

版权所有，侵权必究

Ansible 组件

❑ Ansible Ad-Hoc 命令

配置挂载

mount 模块

选项：

- ☐ dump
- ☐ fstype: 必选项，挂载文件的类型
- ☐ name: 必选项，挂载点
- ☐ opts: 传递给 mount 命令的参数
- ☐ src: 必选项，要挂载的文件
- ☐ state: 必选项
- ☐ present: 只处理 fstab 中的配置
- ☐ absent: 删除挂载点
- ☐ mounted: 自动创建挂载点并挂载之
- ☐ unmounted: 卸载

版权所有，侵权必究

Ansible 组件

□ Ansible Ad-Hoc 命令

配置挂载

mount 模块示例:

#创建设备

```
# ansible docker -a 'dd if=/dev/zero of=/disk.img bs=4k  
count=1024'
```

#与/dev/loop1 关联

```
# ansible docker -a 'losetup /dev/loop1 /disk.img'
```

#格式化

```
# ansible docker -m filesystem -a 'fstype=ext3 force=yes opts=-F  
dev=/dev/loop1'
```

#挂载

```
# ansible docker -m mount -a 'name=/mnt src=/dev/loop1 fstype=ext3  
state=mounted opts=rw'
```

#创建文件

```
# ansible docker -m file -a 'path=/mnt/test.txt state=touch'
```

版权所有，侵权必究

Ansible 组件

□ Ansible Ad-Hoc 命令

配置挂载

mount 模块示例:

#卸载

```
# ansible docker -m file -a 'path=/mnt/test.txt state=absent'  
count=1024'
```

```
# ansible docker -m mount -a 'name=/mnt src=/dev/loop1 fstype=ext3  
state=unmounted'
```

#取消 关联

```
# ansible docker -a 'losetup -d /dev/loop1'
```

```
# ansible docker -m file -a 'path=/disk.img state=absent'
```

版权所有，侵权必究

Ansible 组件

□ Ansible facts

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。

示例：

```
# ansible 192.168.150.12 -m setup
# ansible 192.168.150.12 -m setup -a
  'filter=ansible_all_ipv4_address'
```

版权所有，侵权必究

playbook 详解

□ playbook 核心元素

Tasks: 任务，由模板定义的操作列表

Variables: 变量

Templates: 模板，即使用模板语法的文件

Handlers: 处理器，当某条件满足时，触发执行的操作

Roles: 角色

版权所有，侵权必究

playbook 详解

□ playbook 基本语法

playbook 是 Ansible 进行配置管理的组件，虽然 Ansible 的日常 Ad-Hoc 命令功能很强大，能完成一些基本配置管理工作，但是 Ad-Hoc 命令无法支撑复杂环境的配置管理工作。在我们实际使用 Ansible 的工作中，大部分时间都是在编写 playbook。

Ansible 的 playbook 文件格式为 YAML 语法

YAML 是 YAML Ain't Markup Language 的首字母缩写

YAML 的语法很简单，结构通过空格来展示，项目使用“-”来代表，键值对使用“：”分割。

规则一：缩进。

YAML 使用一个固定的缩进风格表示数据层级结构关系。不要使用 tab，建议使用两个空格。

规则二：冒号。

Python 的字典是简单的键值对，其他语言的用户也应该知道这个数据类型叫哈希表或者关联数组。字典的 key 在 YAML 中的表现形式是一个以冒号结尾的字符串：

规则三：短横杠。

想要表示列表项，使用一个短横杠加一个空格。多个项使用同样的缩进级别作为同一列表的一部分。

版权所有，侵权必究

playbook 详解

□ playbook 基本语法

示例:

```
# cat nginx.yml
```

```
---
```

```
- hosts: all
```

```
  tasks:
```

```
    - name: Install Nginx Package
```

```
      yum: name=nginx state=present
```

```
    - name: Copy Nginx.conf
```

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

```
      owner=root group=root mode=0644 validate='nginx -t -c %s'
```

```
      notify:
```

```
        - ReStart Nginx Service
```

```
  handlers:
```

```
    - name: ReStart Nginx Service
```

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

版权所有，侵权必究

playbook 详解

□ playbook 基本语法

示例:

hosts文件和模板文件

```
[nginx]
192.168.150.1[1:3]
[nginx:vars]
ansible_python_interpreter=/usr/bin/python2.7
```

复制一份nginx配置文件，做如下修改

```
#worker_processes auto;
worker_processes {{ ansible_processor_cores }};
```

hosts 文件里定义了一个主机组为Nginx，组内包含3台设备，然后指定了Nginx 组的一个变量ansible_python_interpreter，特意指定了一个Python 版本去运行。关于 nginx.conf.j2 文件就是一个默认的nginx.conf 文件，它只是针对Nginx 的 worker_processes 参数通过facts 信息中的CPU 核心数目生成，其他的配置都是默认的。

版权所有，侵权必究

playbook 详解

□ playbook 基本语法

使用 `-syntax-check` 参数 playbook 语法检测

```
# ansible-playbook nginx.yaml --syntax-check
```

使用 `-list-task` 参数显示 playbook 文件中所有的 task 名称

```
# ansible-playbook nginx.yaml --list-task
```

使用 `-list-hosts` 参数显示 playbook 文件中针对的目标主机

```
# ansible-playbook nginx.yaml --list-hosts
```

使用 `--limit` 参数，限定执行范围，即使playbook文件中设置了all

```
# ansible-playbook nginx.yml --limit 192.168.150.13
```

确认信息正确后，运行

```
# ansible-playbook nginx.yml
```

运行playbook时指定task，使用`--start-at-task='任务name'`

```
# ansible-playbook nginx.yml --start-at-task='Copy Nginx.conf'
```

版权所有，侵权必究

playbook 详解

□ playbook 基本语法

playbook语法基本特性

- 1) 需要以 “---” (3 个减号) 开始, 且需顶行首写。
- 2) 次行开始正常写 Playbook 的内容, 但笔者建议写明该 Playbook 的功能。
- 3) 使用 # 号注释代码。
- 4) 缩进必须是统一的, 不能将空格和 Tab 混用。
- 5) 缩进的级别必须是一致的, 同样的缩进代表同样的级别, 程序判别配置的级别是通过缩进结合换行来实现的。
- 6) YAML 文件内容和 Linux 系统大小写判断方式保持一致, 是区别大小写的, k/v 的值均需大小写敏感。
- 7) k/v 的值可同行写也可换行写。同行使用 “:” 分隔, 换行写需要以分隔。
- 8) 一个完整的代码块功能需最少元素, 需包括 name: task。
- 9) 一个 name 只能包括一个 task。

版权所有, 侵权必究

playbook 详解

□ playbook的构成

playbook 是由一个或多个“play”组成的列表。play 的主要功能在于将事先归并为一组的主机装扮成事先通过 ansible 中的 task 定义好的角色。从根本上来讲所谓 task 无非是调用 ansible 的一个 module。将多个 play 组织在一个 playbook 中即可以让它们联同起来按事先编排的机制同唱一台大戏。其主要有以下四部分构成：

playbooks 组成：

- ☐ Target section: 定义将要执行 playbook 的远程主机组
- ☐ Variable section: 定义 playbook 运行时需要使用的变量
- ☐ Task section: 定义将要在远程主机上执行的任务列表
- ☐ Handler section: 定义 task 执行完成以后需要调用的任务

而其对应的目录层为五个，如下：

一般所需的目录层有：（视情况可变化）

- ☐ vars 变量层
- ☐ tasks 任务层
- ☐ handlers 触发条件
- ☐ files 文件
- ☐ template 模板

版权所有，侵权必究

playbook 详解

□ playbook的构成

Hosts和Users

在playbook中的每一个play都可以选择在哪些服务器和以什么用户完成，hosts一行可以是一个主机组、主机、多个主机，中间以冒号分隔，可使用通配模式。其中remote_user表示执行的用户账号。

hosts 用于指定要执行指定任务的主机其可以是一个或多个由冒号分隔主机组。

user 执行该任务组的用户

remote_user 则用于指定远程主机上的执行任务的用户，与user相同。

不过remote_user也可用于各task中。也可以通过指定其通过sudo的方式在远程主机上执行的任务其可用于play全局或某任务。

此外甚至可以在sudo时使用sudo_user指定sudo时切换的用户。

sudo 如果设置为yes，执行该任务组的用户在执行任务时，获取root权限。

sudo_user 如果设置user为tom，sudo为yes，sudo_user为jerry，则tom用户会获取jerry用户的权限。

connection 通过什么方式连接到远程主机，默认为ssh。

gather_facts 除非明确说明不需要在远程主机上执行setup模块，否则默认会自动执行。如果确实不需要setup模块所传递过来的变量，可以启用该选项。

版权所有，侵权必究

playbook 详解

□ playbook的构成

Hosts和Users

示例:

```
- hosts: abc
  remote_user: root
```

#指定主机组，可以是一个或多个组。
#指定远程主机执行的用户名

指定远程主机sudo切换用

```
- hosts: abc
  remote_user: root
  become: yes
  # 意为切换用户运行
  become_user: mysql
```

#2.6版本以后的参数，之前是sudo，意
#指定sudo用户为mysql

版权所有，侵权必究

playbook 详解

□ playbook的构成

Tasks list 和action

1: Play的主体部分是task列表，task列表中的各任务按次序逐个在hosts中指定的主机上执行，即在所有主机上完成第一个任务后再开始第二个任务。

在运行playbook时（从上到下执行），如果一个host执行task失败，整个tasks都会回滚，请修正playbook 中的错误，然后重新执行即可。

Task的目的是使用指定的参数执行模块，而在模块参数中可以使用变量，模块执行时幂等的，这意味着多次执行是安全的，因为其结果一致。

2: 每一个task必须有一个名称name，这样在运行playbook时，从其输出的任务执行信息中可以很好的辨别出是属于哪一个task的。如果没有定义name，‘action’的值将会用作输出信息中标记特定的task。

3: 定义一个task，常见的格式:” module: options” 例如: yum:
name=httpd

4: ansible的自带模块中，command模块和shell模块无需使用key=value格式

版权所有，侵权必究

playbook 详解

□ playbook的构成

Tasks list 和action

示例:

tasks:

- name: make sure apache is running
service: name=httpd state=running

在众多模块中只有command和shell模块仅需要给定一个列表而无需使用

“key=value” 格式例如

tasks:

- name: disable selinux
command: /sbin/setenforce 0 如果命令或脚本的退出码不为零可以使用如下方式替代

tasks:

- name: run this command and ignore the result
shell: /usr/bin/somecommand || /bin/true

或者使用ignore_errors来忽略错误信息

tasks:

- name: run this command and ignore the result
shell: /usr/bin/somecommand
ignore_errors: True

版权所有，侵权必究

playbook 详解

□ playbook的构成

handlers

- handlers也是一些task的列表，和一般的task并没有什么区别。
- 是由通知者进行的notify，如果没有被notify，则Handlers不会执行，假如被notify了，则Handlers被执行
- 不管有多少个通知者进行了notify，等到play中的所有task执行完成后，handlers也只会被执行一次
- 注意：在 notify 中定义内容一定要和tasks中定义的 - name 内容一样，这样才能达到触发的效果，否则会不生效。

版权所有，侵权必究

playbook 详解

□ playbook的构成

handlers

示例:

- name: template configuration file
template: src=template.j2 dest=/etc/foo.conf
notify:
 - restart memcached
 - restart apache

handler是task列表这些task与前述的task并没有本质上的不同。

handlers:

- name: restart memcached
service: name=memcached state=restarted
- name: restart apache
service: name=apache state=restarted

版权所有，侵权必究

playbook 详解

□ playbook 变量与引用

1. 通过 Inventory 文件定义主机以及主机组变量

示例:

```
[root@ansible ~]# cat hosts
```

```
192.168.150.11 key=11
```

```
192.168.150.12 key=12
```

```
192.168.150.13 key=13
```

```
[nginx]
```

```
192.168.150.1[1:3]
```

```
[nginx:vars]
```

```
ansible_python_interpreter=/usr/bin/python2.7
```

编写一个 playbook 文件来验证变量的引用是否正确

```
# cat variable1.yml
```

```
- hosts: all
```

```
gather_facts: False
```

```
tasks:
```

```
- name: display Host Variable from hostfile
```

```
debug: msg="The {{ inventory_hostname }} Variable is {{ key }}"
```

版权所有，侵权必究

playbook 详解

□ playbook 变量与引用

1. 通过 Inventory 文件定义主机以及主机组变量

示例:

```
# ansible-playbook -i hosts variable1.yml --syntax-check  
# ansible-playbook -i hosts variable1.yml
```

修改hosts文件如下:

```
#192.168.150.11 key=11  
#192.168.150.12 key=12  
#192.168.150.13 key=13  
[nginx]  
192.168.150.1[1:3]  
[nginx:vars]  
ansible_python_interpreter=/usr/bin/python2.7  
key=nginx
```

再次验证, 所有主机都处于 nginx 组内, 所以该组定义的变量针对组内所有主机都生效。

```
# ansible-playbook -i hosts variable1.yml
```

版权所有, 侵权必究

playbook 详解

□ playbook 变量与引用

2. 通过 `/etc/ansible/` 下的文件定义主机以及主机组变量

默认使用 `yum` 安装 `Ansible` 的配置文件是在 `/etc/ansible/` 目录下，我们还可以使用在该目录下新建 `host_vars` 和 `group_vars` 目录来针对主机和主机组定义变量。

示例：

目录结构如下：

```
[root@ansiblecontrol ansible]# tree ./
```

```
./
├── ansible.cfg
├── group_vars
│   └── nginx
├── hosts
├── host_vars
│   ├── 192.168.150.11
│   ├── 192.168.150.12
│   └── 192.168.150.13
└── roles
```

版权所有，侵权必究

playbook 详解

□ playbook 变量与引用

2. 通过 /etc/ansible/ 下的文件定义主机以及主机组变量

示例:

文件内容:

```
[root@ansible ansible]# head host_vars/*
```

```
==> host_vars/192.168.150.11 <==
```

```
key: 192.168.150.11
```

```
==> host_vars/192.168.150.12 <==
```

```
key: 192.168.150.12
```

```
==> host_vars/192.168.150.13 <==
```

```
key: 192.168.150.13
```

```
[root@ansible ansible]# cat group_vars/nginx
```

```
key: NGINX
```

测试host_vars

```
# ansible-playbook -i hosts /root/variable1.yml
```

版权所有，侵权必究

playbook 详解

□ playbook 变量与引用

2. 通过 /etc/ansible/ 下的文件定义主机以及主机组变量

示例:

删除host_vars下主机变量文件, 验证组变量文件

```
[root@ansible ansible]# rm -f host_vars/*
```

```
[root@ansible ansible]# ansible-playbook -i hosts  
/root/variable1.yml
```

版权所有，侵权必究

playbook 详解

□ playbook 变量与引用

3. 通过 `ansible-playbook` 命令行传入
前面两种是最常用的变量定义方式，通过 `ansible-playbook` 命令行传参的方式定义变量，但是默认传进去的变量都是全局变量

```
# ansible-playbook -i hosts /root/variable1.yml -e 'key=HELL0'
```

目前 `Ansible-playbook` 还支持指定文件的方式传入变量，变量文件的内容支持 `YAML` 和 `JSON` 两种格式

```
[root@ansible ansible]# cat var.yml
```

```
key: ansible
```

```
[root@ansible ansible]# cat var.json
```

```
{"key": "json"}
```

验证

```
# ansible-playbook -i hosts /root/variable1.yml -e '@var.json'
```

```
# ansible-playbook -i hosts /root/variable1.yml -e '@var.yml'
```

版权所有，侵权必究

playbook 详解

□ playbook 变量与引用

4. 在 playbook 文件内使用 vars

可以在 playbook 文件内通过 vars 字段定义变量

示例:

```
# cat /root/variable2.yml
```

```
---
```

```
- hosts: all
```

```
  gather_facts: False
```

```
  vars:
```

```
    key: Ansible
```

```
  tasks:
```

```
    - name: display Host Variable from hostfile
```

```
      debug: msg="The {{ inventory_hostname }} Variable is
```

```
{{ key }}"
```

验证:

```
# ansible-playbook -i hosts /root/variable2.yml
```

版权所有，侵权必究

playbook 详解

□ playbook 变量与引用

5. 在 playbook 文件内使用 vars_files

可以在 playbook 文件内通过 vars_files 字段引用变量， 首先把所有的变量定义到某个文件内， 然后在 playbook 文件内使用 vars_files 参数引用这个变量文件

示例：

```
# cat variable3.yml
```

```
---
```

```
- hosts: all
  gather_facts: False
  vars_files:
    - var.yml
  tasks:
    - name: display Host Variable from hostfile
      debug: msg="The {{ inventory_hostname }} Vaule is
{{ key }}"
```

验证：

```
# ansible-playbook -i hosts variable3.yml
```

版权所有，侵权必究

playbook 详解

□ playbook 变量与引用

6 . 使用 register 内的变量

Ansible playbook 内 task 之间还可以互相传递数据, Ansible task 之间传递数据使用 register 方式。

示例:

```
# cat variable4.yml
- hosts: all
  gather_facts: False
  tasks:
    - name: register variable
      shell: hostname
      register: info
    - name: display register
      debug: msg="The variable is {{ info }}"
```

把第 1 个 task 执行 hostname 的结果 register 给 info 这个变量, 然后在第2 个 task 把这个结果使用 debug 模块打印出来

```
# ansible-playbook variable.yml -1 192.168.150.12
```

版权所有，侵权必究

playbook 详解

□ playbook 变量与引用

6 . 使用 register 内的变量

示例:

下面想标准输出 stdout 的信息时, 只需要指定 stdout 这个 key 即可

```
# cat variable5.yml
```

```
---
```

```
- hosts: all
```

```
  gather_facts: False
```

```
  tasks:
```

```
    - name: register variable
```

```
      shell: hostname
```

```
      register: info
```

```
    - name: display register
```

```
      debug: msg="The variable is {{ info['stdout'] }}"
```

```
# ansible-playbook -i hosts variable5.yml -l 192.168.150.12
```

版权所有，侵权必究

playbook 详解

□ playbook 变量与引用

7. 使用 vars_prompt 传入

Ansible 还支持在运行 playbook 的时候通过交互式的方式给定义好的参数传入变量值，只需在 playbook 中定义 vars_prompt 的变量名和交互式提示内容即可。

示例：

```
# cat variable6.yml
```

```
---
```

- hosts: all
- gather_facts: False
- vars_prompt:
 - name: "one"
 - prompt: "please input one value"
 - private: no
 - name: "two"
 - prompt: "please input two value"
 - default: 'good'
 - private: yes

版权所有，侵权必究

playbook 详解

□ playbook 变量与引用

7. 使用 vars_prompt 传入

示例:

```
# cat variable6.yml
```

```
tasks:
```

- name: display one value
debug: msg="one value is {{ one }}"
- name: display two value
debug: msg="two value is {{ two }}"

测试

```
# ansible-playbook -i hosts variable6.yml -l 192.168.150.12
```

版权所有，侵权必究

playbook 详解

□ playbook 标准 Loops循环

7. 使用 vars_prompt 传入

示例：分别打印one two两个值

```
# cat loop1.yml
```

```
---
```

- hosts: all
- gather_facts: False
- tasks:
 - name: debug loops
 - debug: msg="name -----> {{ item }}"
 - with_items:
 - one
 - two

运行

```
# ansible-playbook -i hosts loop1.yml -l 192.168.150.12
```

版权所有，侵权必究

playbook 详解

□ playbook when 条件判断

很多任务只有在特定条件下才能执行，这就是 when 语句发挥作用的地方。比如我们只需要在数据库服务器上安装 MySQL 软件，根据系统种类来确定是使用 apt 模块还是 yum 模块进行软件包管理

```
# cat when.yml
- hosts: all
  gather_facts: true
  tasks:
    - name: say redhat hello task
      shell: echo "RedHat" `date` by `hostname` >> /tmp/hello.log
      when: ansible_os_family == "RedHat"
    - name: say other linux hello task
      shell: echo "Not RedHat" `date` by `hostname` >> /tmp/hello.log
      when: ansible_os_family != "RedHat"
```

测试：

```
# ansible-playbook when.yml
```

版权所有，侵权必究

playbook 实战

□ 实战1：playbook一键化部署apache服务

1、配置apache.yml文件，实现安装和启动httpd

```
# cat apache.yml
```

```
---
```

- hosts: webservers
remote_user: root
gather_facts: False
vars:
 web: httpd
tasks:
 - name: Install Httpd
 yum: name={{ web }} state=latest

 - name: start httpd
 service: name={{ web }} state=started

2、语法检测

```
# ansible-playbook apache.yml --syntax-check
```

版权所有，侵权必究

playbook 实战

□ 实战1：playbook一键化部署apache服务

3、查看部署任务

```
# ansible-playbook apache.yml --list-tasks
```

4、推送并验证

```
# ansible-playbook apache.yml
```

5、修改playbook文件，增加配置测试页

```
# cat apache.yml
```

```
---
```

```
- hosts: webservers
  remote_user: root
  gather_facts: False
  vars:
    web: httpd
  tasks:
    - name: Install Httpd
      yum: name={{ web }} state=latest
    - name: create index.html
      copy: content="This is test Page."
            dest=/var/www/html/index.html
    - name: start httpd
      service: name={{ web }} state=started
```

playbook 实战

□ 实战1：playbook一键化部署apache服务

6、测试语法

```
# ansible-playbook apache.yml --syntax-check
```

7、推送并验证 # ansible-playbook apache.yml

8、修改playbook，推送httpd.conf配置文件，开机自启

```
# cat apache.yml
```

```
---
```

```
- hosts: webservers
  remote_user: root
  gather_facts: False
  vars:
    web: httpd
  tasks:
    - name: Install Httpd
      yum: name={{ web }} state=latest

    - name: create index.html
      copy: content="This is test Page."
            dest=/var/www/html/index.html
```

版权所有，侵权必究

playbook 实战

□ 实战1：playbook一键化部署apache服务

8、修改playbook，推送httpd.conf配置文件，开机自启

```
# cat apache.yml
```

- name: configure httpd
copy: src=httpd.conf dest=/etc/httpd/conf/httpd.conf
notify:
 - restart httpd
- name: start httpd
service: name={{ web }} state=started enabled=true

handlers:

- name: restart httpd
service: name={{ web }} state=restart

9、准备一个配置文件模板，然后测试playbook语法

```
# scp web1:/etc/httpd/conf/httpd.conf .
```

```
# ansible-playbook apache.yml --syntax-check
```

10、推送测试并验证 # ansible-playbook apache.yml

版权所有，侵权必究

playbook 实战

□ 实战2: playbook源码安装服务

```
# mkdir nginx
cd nginx/
# tree ./
./
├── nginx.conf
└── nginx.yml
0 directories, 2 files
# cat nginx.yml
- hosts: localhost
  vars:
    - nginx_version: 1.12.2
    - nginx_user: www
  tasks:
    - name: add nginx run user
      user: name={{ nginx_user }}
    - name: nginx dependence
      yum: name={{ item }} state=latest
      with_items:
        - openssl-devel
        - pcre-devel
```

版权所有，侵权必究

playbook 实战

□ 实战2: playbook源码安装服务

```
# cat nginx.yml
- name: install nginx
  hosts: all
  become: yes
  tasks:
    - name: download nginx-{{ nginx_version }}.tar.gz
      get_url:
        url: http://nginx.org/download/nginx-{{ nginx_version }}.tar.gz
        dest: "{{ lookup('env', 'HOME') }}/nginx-{{ nginx_version }}.tar.gz"
    - name: install nginx
      shell: cd {{ lookup('env', 'HOME') }};tar -xf nginx-{{ nginx_version }}.tar.gz;cd nginx-{{ nginx_version }};./configure --user={{ nginx_user }} --group={{ nginx_user }} --prefix=/usr/local/nginx --with-http_stub_status_module --with-http_ssl_module --with-pcre --with-http_realip_module;make -j`grep processor /proc/cpuinfo|wc -l`&& make install
    - name: copy conf file nginx.conf
      template: src=nginx.conf
```


playbook 实战

□ 实战2: playbook源码安装服务

```
# cat nginx.yml
dest=/usr/local/nginx/conf/nginx.conf
- name: create vhosts dir
  file: path=/data/www state=directory
- name: start nginx services
  shell: /usr/local/nginx/sbin/nginx
```

版权所有，侵权必究

playbook进阶

□ 巧用Includes

Includes使用场景

有时，我们发现大量的 Playbook 内容需要重复编写，各 Tasks 之间功能需相互调用才能完成各自功能，Playbook 庞大到维护困难，这时我们需要使用 Includes。

Includes用法

基本的 include 语句用法

与其他语言的 include 语句一样，直接 include 即可。

示例：

```
# cat tasks/check_httpd.yml
```

```
---
```

```
# possibly saved as tasks/check_httpd.yml
```

```
- name: check package httpd
```

```
  shell: rpm -q httpd
```

main.yml 文件中调用 include 的方法

```
# cat main.yml
```

```
---
```

```
- hosts: webservers
```

```
  gather_facts: False
```

```
  tasks:
```

```
    - include: tasks/check_httpd.yml
```

版权所有，侵权必究

playbook进阶

□ 巧用Includes

Includes用法

在 include 语句中使用参数

这里有两个知识点 一是如何在被 include 的文件中定义参数，二是如何向 include 文件中传入参数。

在被 include 的文件中定义参数。在被 include 的文件

tasks/check_httpd.yml ，使用 {{ pkg_name }} 定义了一个名字为 pkg_name 的参数

```
# cat tasks/check_httpd.yml
```

```
---
```

```
# possibly saved as tasks/check_httpd.yml
```

```
- name: check package httpd
  shell: rpm -q {{ pkg_name }}
```

版权所有，侵权必究

playbook进阶

□ 巧用Includes

Includes用法

在 include 语句中使用参数

1)执行的 Playbook 中传参数，可以加在行尾，使用空格分隔。

```
# cat main.yml
```

```
---
```

- hosts: webservers
gather_facts: False
tasks:
 - include: tasks/check_httpd.yml pkg_name=httpd

2)使用 YAML 字典传参数。

```
# cat main.yml
```

```
---
```

- hosts: webservers
gather_facts: False
tasks:
 - include: tasks/check_httpd.yml
- vars:
 - pkg_name: httpd

版权所有，侵权必究

playbook进阶

□ 巧用Includes

Includes用法

在 include 语句中使用参数

3) 写成一个类 JSON 的形式传参数

```
# cat main.yml
```

```
---
```

- hosts: webservers
- gather_facts: False
- tasks:
 - { include: tasks/check_httpd.yml, pkg_name: httpd }

另外，可以在handler 里面加 include ，也可以在全局（或者叫 Plays ）加 include 。

版权所有，侵权必究

playbook进阶

▣ 巧用Roles

Includes 使 Playbook 更加整洁有效，清晰健壮。在自动化发布项目，总文件数达几百个，如果仅通过简单的 Includes 不停地引用，那最终的结果难以想象。

role 有比 include 更强大灵活的代码重用和分享机制。include 类似于编程语言中的 include, 是重用单个文件的，重用的功能有限。而 role 类似于编程语言中的“Package”，可以重用一组文件，形成完整的功能。

Ansible 十分提倡在 Playbook 中使用 role，并且提供了一个分享 role 的平台 Ansible Galaxy
(<https://galaxy.ansible.com/>) 。

版权所有，侵权必究

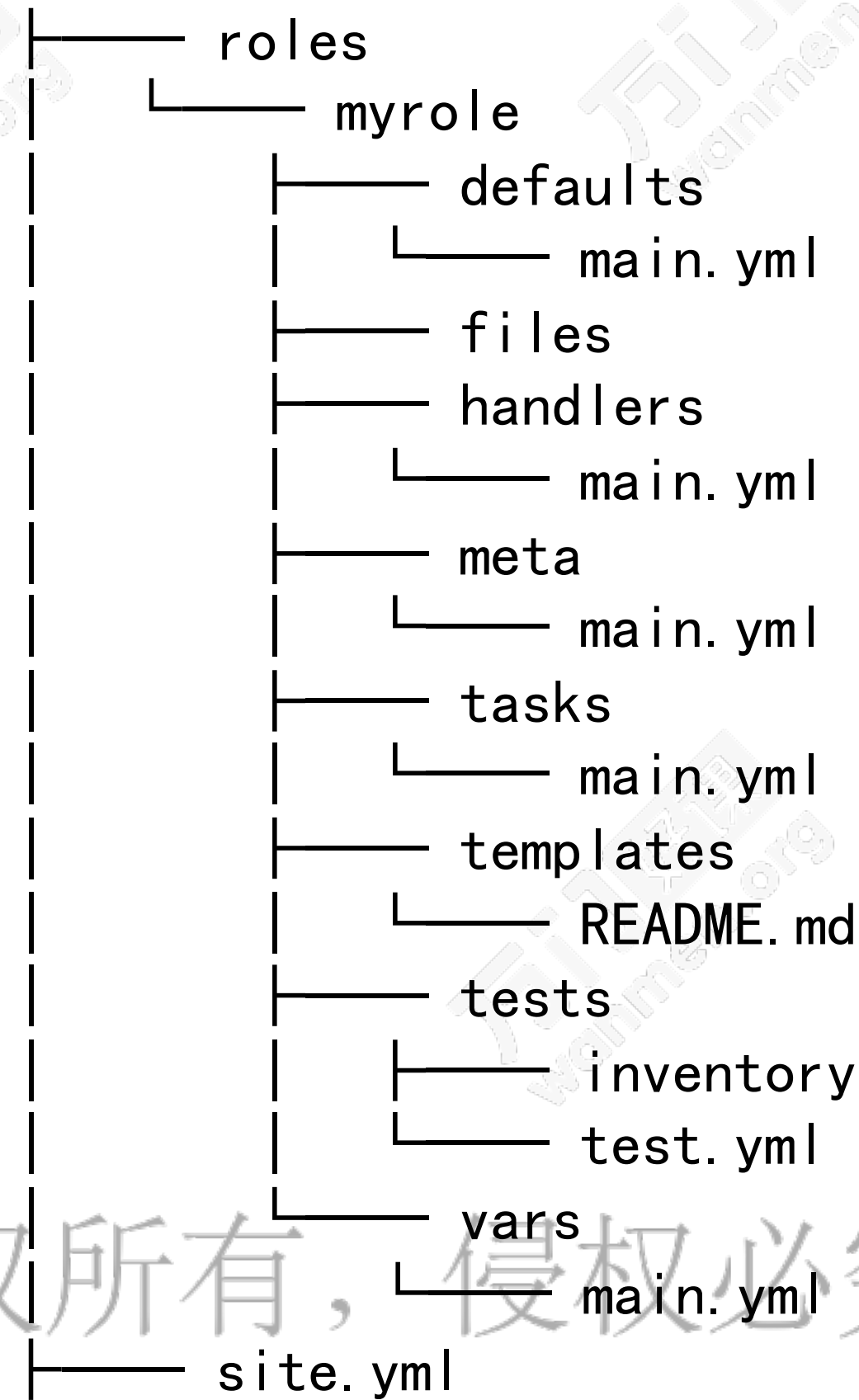
playbook进阶

巧用Roles

定义 role 完整的目录结构

在 Ansible 中，通过遵循特定的目录结构，就可以实现对 role 的定义。

下面的目录结构定义了一个 role：名字为 myrole。Playbook 文件中的 site.yml 时，调用了这个 role。



版权所有，侵权必究

playbook进阶

□ 巧用Roles

定义 role 完整的目录结构

site.yml 中调用 role

- hosts: webservers

 roles:

 - myrole

Ansible 并不要求 role 包含上述所有的目录及文件，可以根据 role 的功能，加入对应的目录和文件即可。下面是每个目录和文件的功能

1. 如果roles 文件 role/x/tasks/main.yml 存在，则文件中列出的任务都将被添加到 Play 中。
2. 如果文件roles/x/handlers/main.yml 存在，则文件中列出 handler 都将被添加到 Play 中。
3. 如果文件role/x/vars/main.yml存在，则文件中列出的变量都将被添加到 Play 中。
4. 如果文件role/x/defaults/main.yml存在，则文件中列出的变量都会被添加到Play 中。
5. 如果文件role/x/meta/main.yml 存在，则文件中列出的所有依赖的 role 都将被添加到 Play 中。
6. 此外，下面的文件不需要绝对或者是相对路径，和放在同一个目录下的文件一样，直接使用即可。
 - copy 或者 script 使用 roles/x/files / 下的文件。
 - template 使用 roles/x/templates 下的文件。
 - include 使用 roles/x/tasks 下的文件。

版权所有，侵权必究

playbook进阶

□ 巧用Roles

带参数的 role

下面定义一个带参数的 role，名字是 myrole，其目录结构如下。

```
main.yml
roles
  myrole
    tasks
      main.yml
```

在 roles/myrole/tasks/main.yml 中，使用 {{ }} 定义的变量就可以了。

```
---
- name: use param
  debug: msg="{{ param }}"
```

使用带参数的 role

在 main.yml 中可以用如下方法使用 myrole 了。

```
---
- hosts: webservers
  roles:
    - { role: myrole, param: 'Call some_role for the 1st time' }
    - { role: myrole, param: 'Call some role for the 2nd time' }
```

版权所有，侵权必究

playbook进阶

□ 巧用Roles

role 和任务的执行顺序

如果一个 Playbook 中同时出现 role 和任务，那么它们的调用顺序是怎样的呢？

pre_tasks > role > tasks > post_tasks

示例：

```
# cat role_1.yml
```

```
---
```

```
- hosts: webservers
```

```
  user: root
```

```
  pre_tasks:
```

```
    - name: one
```

```
      shell: echo "hello"
```

```
  roles:
```

```
    - { role: some_role }
```

```
  tasks:
```

```
    - name: task
```

```
      shell: echo "still busy"
```

```
  post_tasks:
```

```
    - name: post
```

```
      shell: echo "googdbye"
```

版权所有，侵权必究

playbook进阶

□ 巧用Roles

role 和任务的执行顺序

如果一个 Playbook 中同时出现 role 和任务，那么它们的调用顺序是怎样的呢？

pre_tasks > role > tasks > post_tasks

示例：

```
# mkdir roles/some_role/tasks -p
# cat roles/some_role/tasks/main.yml
```

```
- name: some role
  shell: echo "I'm some role."
```

执行测试

```
# ansible-playbook role_1.yml
```

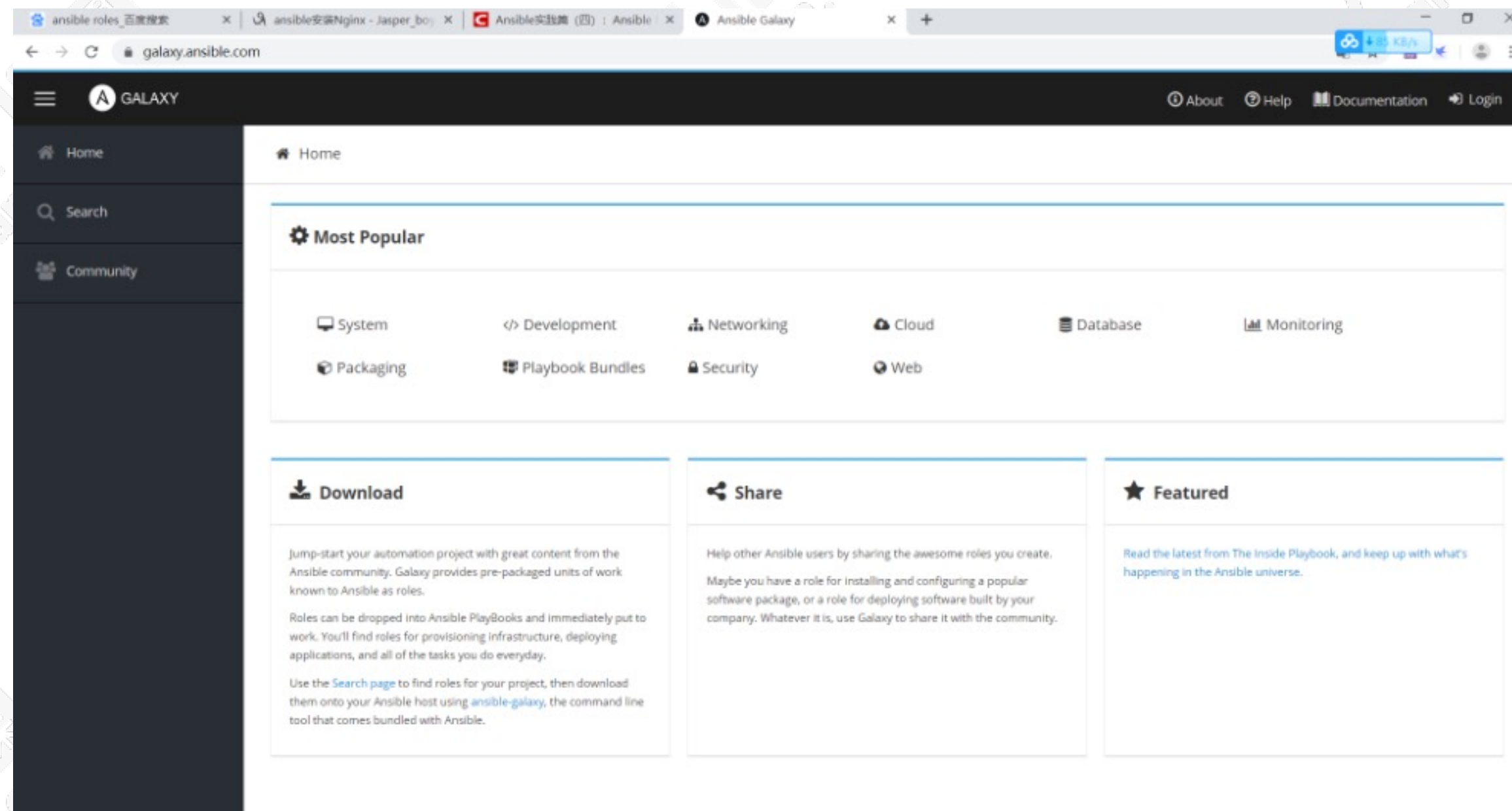
版权所有，侵权必究

playbook进阶

▣ Ansible Galaxy 网站介绍

从 Ansible Galaxy 网站上下载 role

访问 <https://galaxy.ansible.com/>，找到你需要的 role。



版权所有，侵权必究

playbook进阶

□ 使用tags

如果 Playbook 文件比较大，并且在执行的时候只是想执行部分功能，那么这个时候有没有解决方案呢？Playbook 提供了标签（tags）可以实现部分运行。

标签的基本用法

ansible中可以对play、role、include、task打一个tag(标签)，然后：

- ∅ 当命令ansible-playbook有-t参数时，只会执行-t指定的tag
- ∅ 当命令ansible-playbook有--skip-tags参数时，则除了--skip-tags指定的tag外，执行其他所有

版权所有，侵权必究

playbook进阶

□ 使用tags

标签的基本用法示例

tasks:

- name: install httpd
 - yum: name={ { item } } state=installed
 - with items:
 - httpd
 - tags:
 - packages
- name: copy httpd.conf
 - template : src=templates/httpd.conf.j2
 - dest=/etc/httpd/conf/httpd.conf
 - tags:
 - configuration
- name: copy index.html
 - template: src=templates/index.html.j2
 - dest=/var/www/html/index.html
 - tags:
 - configuration

版权所有，侵权必究

playbook进阶

□ 使用tags

标签的基本用法示例

我们在执行的时候，如果不加任何 `tag` 参数，那么会执行所有标签对应的任务。`ansible-playbook example.yml`

如果指定执行安装部分的任务，则可以利用关键字 `tags`或`t`指定需要执行的标签名。`ansible-playbook example.yml --tags "packages"`

如果指定不执行 `tag packages` 对应的任务，则可以利用关键字 `skip-tags`。`ansible-playbook example.yml --skip-tags "configuration"`

版权所有，侵权必究

playbook进阶

▣ Jinja2 实现模板高度自定义

Jinja2 读取变量的方式，通常两个花括号中间加变量名且花括号和变量名间需有空格分隔，如 `{{ variable }}`

Jinja2 For 循环

变量的提取使用 `{{ variable }}`，`{% statement execution %}` 括起来的内容为

Jinja2 命令执行语句，命令语法如下：

```
{% for item in all_items %}  
{{ item }}  
{% endfor %}
```

示例：为远程主机生成服务器列表，该列表从 192.168.150.201 web01.test 开始，到 192.168.150.211 web11.test 结束。

```
{% for id in range(201, 211) %}  
192.168.150.{{ id }} web{{ "%02d" | format(id-200) }}.test  
{% endfor %}
```

版权所有，侵权必究

playbook进阶

□ Jinja2 实现模板高度自定义

Jinja2 If 条件

Jinja 中 If 条件判断的使用格式如下：

```
{% if my_conditional %}  
...  
{% endif %}
```

示例：如果在主机组（hosts里面定义的）设置为主dns，否则设置为从dns

```
{% if node in group_names %}  
{% set zone_type = 'master' %}  
{% set zone_dir = 'data' %}
```

```
{% else %}  
{% set zone_type = 'slave' %}  
{% set zone_dir = 'slaves' %}  
{% endif %}
```

```
{% if node not in group_names %}  
masters { 192.168.1.100; };
```

playbook进阶

□ 实战一 通过role远程部署nginx并配置

1. 首先创建目录结构

```
# tree roles/
```

```
roles/
├── nginx
│   ├── files
│   │   └── index.html
│   ├── handlers
│   │   └── main.yaml
│   ├── tasks
│   │   └── main.yaml
│   ├── templates
│   │   └── nginx.conf.j2
│   └── vars
│       └── main.yaml
└── site.yaml
```

6 directories, 6 files

版权所有，侵权必究

playbook进阶

□ 实战一 通过role远程部署nginx并配置

1. 首先创建目录结构

准备目录结构

```
# mkdir roles/nginx/{files,handles,tasks,templates,vars} -p
# touch roles/site.yaml roles/nginx/{handles,tasks,vars}/main.yaml
# echo 1234 > roles/nginx/files/index.html
# yum install -y nginx && cp /etc/nginx/nginx.conf
  roles/nginx/templates/nginx.conf.j2
```

此处安装nginx是为了得到nginx的配置文件，通过修改此配置文件的配置，并将文件拷贝到服务主机上，实现修改服务主机的配置。

版权所有，侵权必究

playbook进阶

□ 实战一 通过role远程部署nginx并配置

2. 编写任务

template不同于copy模块，复制的文件中可以存在变量

```
# vim roles/nginx/tasks/main.yaml
```

```
---
```

- name: install nginx package
yum: name={{ item }} state=latest
with_items:
 - epel-release
 - nginx
- name: copy index.html
copy: src=index.html dest=/usr/share/nginx/html/index.html
- name: copy nginx.conf template
template: src=nginx.conf.j2 dest=/etc/nginx/nginx.conf
notify: restart nginx
- name: make sure nginx service running
service: name=nginx state=started enabled=yes

playbook进阶

□ 实战一 通过role远程部署nginx并配置

3. 准备配置文件

复制一份配置文件进行如下修改

```
# vim roles/nginx/templates/nginx.conf.j2
```

```
# * Official English Documentation: http://nginx.org/en/docs/
# * Official Russian Documentation: http://nginx.org/ru/docs/

user nginx;
worker_processes {{ ansible_processor_cores }};
error_log /var/log/nginx/error.log;
pid /run/nginx.pid;

# Load dynamic modules. See /usr/share/nginx/README.dynamic.
include /usr/share/nginx/modules/*.conf;

events {
    worker_connections {{ worker_connections }};
    "roles/nginx/templates/nginx.conf.j2" 90L, 2512C
```

调用系统已有变量

调用自定义变量

版权所有，侵权必究

playbook进阶

□ 实战一 通过role远程部署nginx并配置

4. 编写变量

```
# vim nginx/vars/main.yaml
worker_connections: 10240
```

5. 编写处理程序

```
# vim roles/nginx/handlers/main.yaml
__
- name: restart nginx
  service: name=nginx state=restarted
```

6. 编写剧本

```
# vim roles/site.yaml
- hosts: host1
  roles:
    - nginx
```

7. 实施 `ansible-playbook site.yaml`

版权所有，侵权必究

playbook进阶

□ 实战二 批量部署tomcat

1 案例场景

某公司申请了云计算100台虚拟机，需要进行tomcat环境搭建。

2 步骤分析

Tomcat安装过程如下：

- 1、安装jdk
- 2、创建tomcat用户
- 3、安装tomcat
- 4、配置tomcat，重启等

如定义playbook呢？

- 1、定义hosts
- 2、定义roles
- 3、定义vars
- 4、编排playbooks

3 实战部署

以下实战部署参考官方案例，结合企业实际应用修改。

事先将需要安装的jdk和tomcat软件通过http方式提供下载，配置过程略。

总结

- ❑ Ansible 架构及特点
- ❑ Ansible 安装与配置
- ❑ Ansible 组件
- ❑ playbook 详解
- ❑ playbook 进阶

版权所有，侵权必究

作业

□ 使用ansible进行lnmp环境部署

- 1、使用yum进行lnmp的安装，使用一个yaml文件实现；
- 2、使用yum进行lnmp的安装，为提高playbook的可用性，可使用role的方式将playbook进行拆分；

版权所有，侵权必究



谢谢观看

更多好课，请关注万门大学APP

