28 Ansible基础配置

Ansible基础配置

一、Ansible概述

1.1 Ansible简介

Ansible是一款轻量级的**服务器集中管理（配置管理）**软件，其主要功能是实现IT运维工作的自动化、降低人为操作失误、提高业务自动化率、提升运维工作效率，常用于软件部署自动化、配置自动化、管理自动化、系统化系统任务、持续集成、零宕机平滑升级等。

Ansible在设计上非常注重**简单**的理念。默认采用**SSH**的方式通讯，部署简单，只需要在跳板机或**控制端**部署Ansible环境即可，**被控端无须安装客户端**。

Ansible既可以使用各种**模块**对**被控端**实施**临时性**的**批量管理**（执行命令、安装软件、指定特定任务等），对于一些较为复杂的需要**重复执行**的任务，也可以通过基于YAML格式的**playbook**来管理这些复杂的任务。

相比较于其他自动化运维工具，Ansible的优势如下。

- 轻量级，**无须在客户端安装Agent**，更新时只需要在控制端上进行一次更新即可。
- 批量任务执行可以写成playbook，而且不用分发到远程就可以执行。
- 软件由**Python**编写，维护简单，二次开发更方便。配置文件格式也以INI和YAML为主，上手较快。
- 支持**非root用户**管理操作，支持sudo。
- 支持云计算、大数据平台（如AWS、OpenStack、CloudStack等）。
- Ansible社区非常活跃，Ansible本身提供的模块也非常丰富，第三方资源众多。详细的模块分类可参考官方模块列表：https://docs.ansible.com/ansible/2.9/modules/modules_by_category.html。

2015年，**Redhat公司宣布收购Ansible**，Ansible是免费的，但是Redhat提供的一部分Ansible辅助工具是收费的，比如是**Ansible的图形界面化平台Ansible tower**。

1.2 Ansible工作原理

Ansible**没有客户端**，因此底层**通信依赖于系统软件**。

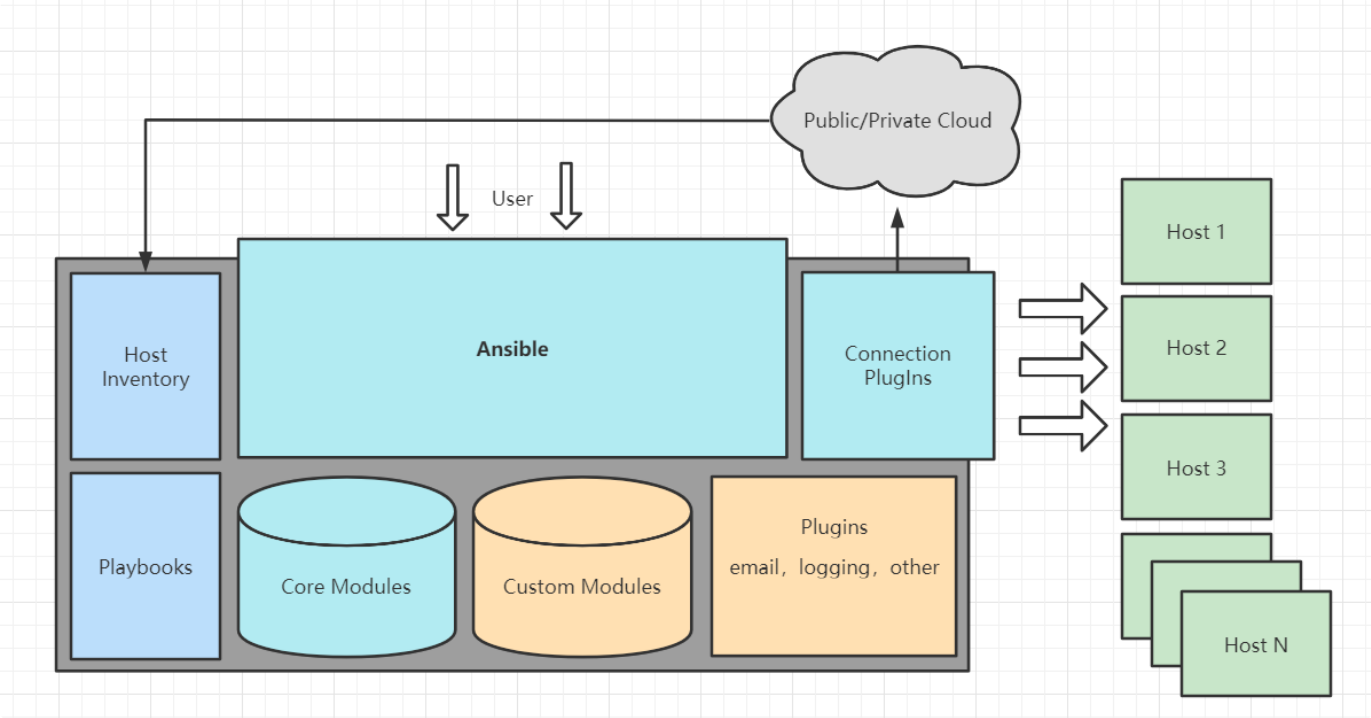
- **控制端**：**必须是Linux系统**。
- 受控端：Linux、Windows等常见操作系统。

Ansible底层通信依赖于**SSH协议**。

- Linux系统下基于**OpenSSH**通信
- Windows系统下基于**PowerShell**

Ansible是基于**模块**工作的，软件本身没有批量部署能力。

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主要由6部分组成。

- **Playbooks**：任务剧本，编排定义ansible任务集的配置文件，由Ansible顺序依次执行，通常是YAML文件。
- **Inventory**：Ansible管理主机的清单。
- **Modules**：Ansible执行命令的功能模块，多数为内置的核心模块，也可自定义。
- Plugins：模块功能的补充，如连接类型插件、循环插件、变量插件、过滤插件等，该功能不常用。
- API：供第三方程序调用的应用程序编程接口。
- **Ansible：组合Inventory、API、modules、plugins的主程序**。

1.3 Ansible部署

控制节点：域名解析、Ansible

被控节点：SSH、IP、yum源配置、防火墙

Ansible的安装部署非常简单，**只需要安装到控制节点上**，由Ansible管理的主机不需要安装Ansible。

控制节点应为 Linux 或 UNIX 系统，不支持将Windows用作控制节点，但Windows系统可以是受控主机。 Ansible仅依赖于**Python**和**SSH**，**Linux系统默认均已安装**。

Ansible被RedHat官方收购后，其安装源被收录在**EPEL**中，如已安装EPEL可直接YUM或APT安装，通过pip和easy_install的Python第三方包管理器也可以便捷安装Ansible。

下面在CentOS7.9下以yum形式安装Anisle并进行验证。

直接采用**yum安装**，默认安装的是环境为**Python2.7**。

Shell | 复制代码

```shell
# 列出所有文件
rpm -ql ansible
# 列出配置文件
rpm -qc ansible
# 查看帮助
ansible --help
# 列出所有模块
ansible-doc -l
# 查看某模块帮助
ansible-doc -s yum
```

Bash | 复制代码

```
[root@root ~]# yum install ansible -y
yum install ansible -y --nogpgcheck
# 仅显示依赖部分
Dependencies Resolved

================================================================================
================================================================================
 Package
    Arch                                                Version
                                    Repository
          Size
================================================================================
Installing:
 ansible
    noarch                                              2.9.25-1.el7
                                    epel
           17 M
Installing for dependencies:
 PyYAML
    x86_64                                              3.10-11.el7
                                    base
          153 k
 libyaml
    x86_64                                              0.1.4-11.el7_0
                                    base
           55 k
 python-babel
    noarch                                              0.9.6-8.el7
                                    base
          1.4 M
 python-backports
    x86_64                                              1.0-8.el7
                                    base
          5.8 k
 python-backports-ssl_match_hostname
    noarch                                              3.5.0.1-1.el7
                                    base
           13 k
 python-cffi
    x86_64                                              1.6.0-5.el7
```

```
                                                    base
18         218 k
 python-enum34
    noarch                                          1.0.4-1.el7
                                                    base
19          52 k
 python-idna
    noarch                                          2.4-1.el7
                                                    base
20          94 k
 python-ipaddress
    noarch                                          1.0.16-2.el7
                                                    base
21          34 k
 python-jinja2
    noarch                                          2.7.2-4.el7
                                                    base
22         519 k
 python-markupsafe
    x86_64                                          0.11-10.el7
                                                    base
23          25 k
 python-paramiko
    noarch                                          2.1.1-9.el7
                                                    base
24         269 k
 python-ply
    noarch                                          3.4-11.el7
                                                    base
25         123 k
 python-pycparser
    noarch                                          2.14-1.el7
                                                    base
26         104 k
 python-setuptools
    noarch                                          0.9.8-7.el7
                                                    base
27         397 k
 python2-cryptography
    x86_64                                          1.7.2-2.el7
                                                    base
28         502 k
 python2-httplib2
    noarch                                          0.18.1-3.el7
                                                    epel
29         125 k
 python2-jmespath
    noarch                                          0.9.4-2.el7
```

```
                                           epel
30          41 k
 python2-pyasn1
    noarch                                                 0.1.9-7.el7
                                           base
31         100 k
 sshpass
    x86_64                                                 1.06-2.el7
                                           extras
32          21 k

33
34  Transaction Summary
    ================================================================================
    ================================================================================
    ================================================================================
    ==============
35  Install  1 Package (+20 Dependent packages)
36
37  Total download size: 21 M
38  Installed size: 122 M
```

验证，查看已安装Ansible版本。

Bash | 复制代码

```
1  [root@root ~]# ansible --version
2  ansible 2.9.25
3    config file = /etc/ansible/ansible.cfg
4    configured module search path = [u'/root/.ansible/plugins/modules', u'/usr/share/ansible/plugins/modules']
5    ansible python module location = /usr/lib/python2.7/site-packages/ansible
6    executable location = /usr/bin/ansible
7    python version = 2.7.5 (default, Nov 16 2020, 22:23:17) [GCC 4.8.5 20150623 (Red Hat 4.8.5-44)]
```

1.4 Ansible的目录结构

Ansible是**开源工具**，整个开发过程或二次开发均遵循GPL协议，所以所有源码均可见。

Ansible的相关目录如下：

- 配置文件目录 `/etc/ansible/` ，主要功能为：Inventory主机信息配置、Ansible工具功能配置等。Ansible所有配置均存放在该目录下，运维日常的所有配置类操作也均基于此目录进行。
- 执行文件目录 `/usr/bin/` ，主要功能为：Ansible系列命令默认存放目录。Ansible所有的可执行文件均存放在该目录下。 `ll /usr/bin | grep ansible`
- Python库目录 `/usr/lib/pythonXXX/site-packages/` ，该目录是系统当前默认的Python路

径，因为Ansible是基于Python编写的，所以Ansible的所有lib库文件和模块文件也均存放于该目录下。

Ansible的默认配置文件存放在 `/etc/ansible/` 目录下，**均默认遵循INI文件格式**。

- **`hosts`** 文件为Inventory文件，用于定义Ansible的主机列表配置。
- **`ansible.cfg`** 文件为Anisble配置文件。
  `ansible.cfg` 配置文件可以存在于多个地方，Ansible读取配置文件的优先级为 当前命令执行目录的`ansible.cfg` → 用户家目录下的`ansible.cfg` → `/etc/ansible.cfg` 。
  `ansible.cfg` 配置的**所有内容均可在命令行通过参数的形式传递**或定义在**Playbooks**中。
  配置文件 `ansible.cfg` 约有350行语句，大多数为注释行默认配置项。

```bash
[root@root ~]# ll /etc/ansible
total 28
-rw-r--r-- 1 root root 19985 Aug 22 04:07 ansible.cfg
-rw-r--r-- 1 root root  1016 Aug 22 04:07 hosts
drwxr-xr-x 2 root root  4096 Aug 22 04:07 roles
```

二、定义主机与组

Anisble在运行时必须指定主机的作用域。

Ansible默认通过读取**默认Inventory文件** **`/etc/ansible/hosts`** 指定Ansible作用的主机列表。

当然，也可以在运行ansible系列命令时用 **`-i`** **参数指定临时主机列表文件**。

2.1 定义主机与组规则

2.1.1 简易规则

Inventory文件简易定义规则如下。

- 支持多种格式，常见的格式为**INI**和**YAML**。
- 每行为一个单独的配置。
- `#` 为注释符号。
- 定义主机可以使用**主机名**或**IP地址**。
- 主机可以分组，格式为 `[组名]` ，组名必须为单独一行，其后的行均认为是该组成员，直到遇到新的分组。
- 同一台主机**可以属于多个组**。
- 如果SSH采用的不是默认的22端口，那么可以在主机后面指定SSH端口。
- 如果多个主机名有相同的模式，可使用范围语法，格式为 `[起始:终止]` 。

案例：INI格式的Inventory文件

```Bash
#IP格式主机
192.168.1.101
#主机组
[webservers]
#主机名格式主机
alpha.example.org
beta.example.org
#可指定SSH端口
192.168.1.100:2222
192.168.1.110
#主机名可使用范围语法
web[01:10].example.org
web[a:f].example.org
```

思考：192.168.1.101属于哪个组？webservers组有哪些主机？

2.1.2 定义变量

在日常工作中，通常会遇到**非标准化的需求配置**，例如如考虑到安全性问题，业务人员通常将企业内部的Web服务80端口修改为其他端口号。该需求可通过为Inventory配置定义变量来实现。

变量根据**变量的性质**可以分为**自定义变量**和**系统定义的连接变量**，常见的连接变量如下表。

参数	作用
ansible_ssh_host	受控主机名
ansible_ssh_port	端口号
ansible_ssh_user	默认账号
ansible_ssh_pass	默认密码
ansible_shell_type	Shell终端类型

变量根据**作用范围**可分为**主机变量**和**组变量**。

- 主机变量：主机变量作用范围为**指定主机**，直接定义在主机后。

```Bash
[atlanta]
host1 http_port=80 maxRequestsPerChild=808
host2 http_port=303 maxRequestsPerChild=909
```

- 组变量：组变量的作用是**覆盖组中的所有成员**，下面定义一个新块，块名由 `[组名+:vars]` 组成。

```
[atlanta]
host1
host2
[atlanta:vars]
ntp_server=ntp.atlanta.example.com
proxy=proxy.atlanta.example.com
```

Bash | 复制代码

2.1.3 默认组

Ansible有两个默认组： `all` 和 `ungrouped` 。

- **`all` 包含所有主机。**
- **`ungrouped` 包含所有未分组的主机。**

> ⚠️ **注意！命名分组时应避免使用 `all` 和 `ungrouped` 。**

2.2 案例：定义主机和组

2.2.1 节点规划

控制节点：192.168.149.4

被控节点：192.168.149.3和192.168.149.5

2.2.2 免密登录（可选）

```
[root@zabbix ~]# ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
/root/.ssh/id_rsa already exists.
Overwrite (y/n)? n
[root@zabbix ~]# ssh-copy-id -i /root/.ssh/id_rsa.pub root@192.168.149.3
The authenticity of host '192.168.149.3 (192.168.149.3)' can't be establis
hed.
ECDSA key fingerprint is 2f:60:fc:7a:fb:a9:56:ee:08:38:29:66:b4:8f:16:a9.
Are you sure you want to continue connecting (yes/no)? y
Please type 'yes' or 'no': yes
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to f
ilter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are
prompted now it is to install the new keys
root@192.168.149.3's password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'root@192.168.149.3'"
and check to make sure that only the key(s) you wanted were added.
[root@zabbix ~]# ssh-copy-id -i /root/.ssh/id_rsa.pub root@192.168.149.5
The authenticity of host '192.168.149.5 (192.168.149.5)' can't be establis
hed.
ECDSA key fingerprint is 2f:60:fc:7a:fb:a9:56:ee:08:38:29:66:b4:8f:16:a9.
Are you sure you want to continue connecting (yes/no)? yes
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to f
ilter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are
prompted now it is to install the new keys
root@192.168.149.5's password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'root@192.168.149.5'"
and check to make sure that only the key(s) you wanted were added.
```

2.2.3 定义主机与组

主机和组定义好之后，可以使用 `ansible` 或者 `ansible-inventory` 命令进行验证。

案例：定义主机与组，并用ansible命令验证

Bash | 复制代码

```
# 定义主机和组
[root@zabbix ~]# vi /etc/ansible/hosts
192.168.149.3
[test]
192.168.149.5
# 查看所有主机
[root@zabbix ~]# ansible all  --list-hosts
  hosts (2):
    192.168.149.3
    192.168.149.5
# 查看test组的主机
[root@zabbix ~]# ansible test  --list-hosts
  hosts (1):
    192.168.149.5
# 查看所有未分组的主机
[root@zabbix ~]# ansible ungrouped --list-hosts
  hosts (1):
    192.168.149.3
# 也可以用IP或用主机名操作单个主机
[root@zabbix ~]# ansible 192.168.149.3 -m ping
192.168.149.3 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "ping": "pong"
}
```

以 `ansible-inventory` 命令进行验证。

Shell | 复制代码

```
# 以JSON格式显示主机与组结构
[root@zabbix ~]# ansible-inventory --list
{
    "_meta": {
        "hostvars": {}
    },
    "all": {
        "children": [
            "test",
            "ungrouped"
        ]
    },
    "test": {
        "hosts": [
            "192.168.149.5"
        ]
    },
    "ungrouped": {
        "hosts": [
            "192.168.149.3"
        ]
    }
}
# 以目录树的形式显示主机与组结构
[root@zabbix ~]# ansible-inventory --graph
@all:
  |--@test:
  |  |--192.168.149.5
  |--@ungrouped:
  |  |--192.168.149.3
# 以YAML格式显示主机与组结构
[root@zabbix ~]# ansible-inventory -y --list
all:
  children:
    test:
      hosts:
        192.168.149.5: {}
    ungrouped:
      hosts:
        192.168.149.3: {}
```

案例：定义YAML格式的Inventory文件

```Bash
[root@zabbix ~]# vi host.yml
---
all:
  hosts:
    192.168.149.3:
  children:
    test:
      hosts:
        192.168.149.5:
[root@zabbix ~]# ansible all --list-hosts -i host.yml
  hosts (2):
    192.168.149.3
    192.168.149.5
[root@zabbix ~]# ansible test --list-hosts -i host.yml
  hosts (1):
    192.168.149.5
```

案例：设置主机和组变量

主机变量优先于组变量，但**playbook中定义的变量的优先级比这两者更高**。

准备工作：到受控节点删除密钥，取消免密登录，此时使用ping命令测试失败。

```Shell
[root@database ~]# vi .ssh/authorized_keys
```

```Shell
[root@zabbix ~]# ansible all -m ping
192.168.149.3 | UNREACHABLE! => {
    "changed": false,
    "msg": "Failed to connect to the host via ssh: Permission denied (publickey,gssapi-keyex,gssapi-with-mic,password).",
    "unreachable": true
}
192.168.149.5 | UNREACHABLE! => {
    "changed": false,
    "msg": "Failed to connect to the host via ssh: Permission denied (publickey,gssapi-keyex,gssapi-with-mic,password).",
    "unreachable": true
}
```

```bash
# 原始inventory文件
[root@zabbix ~]# vi /etc/ansible/hosts
192.168.149.3
[test]
192.168.149.5
```

通过设置主机变量替代免密登录操作。

```bash
# 取消免密登录后，不能正常工作
[root@zabbix ~]# ansible ungrouped -m ping
192.168.149.3 | UNREACHABLE! => {
    "changed": false,
    "msg": "Failed to connect to the host via ssh: Permission denied (publickey,gssapi-keyex,gssapi-with-mic,password).",
    "unreachable": true
}
# 设置主机变量，提供SSH账号密码
[root@zabbix ~]# vi /etc/ansible/hosts
192.168.149.3 ansible_user=root  ansible_password=000000
[test]
192.168.149.5
# 结果发现设置变量的主机正常工作，未设置的不能正常工作
[root@zabbix ~]# ansible all -m ping
192.168.149.5 | UNREACHABLE! => {
    "changed": false,
    "msg": "Failed to connect to the host via ssh: Permission denied (publickey,gssapi-keyex,gssapi-with-mic,password).",
    "unreachable": true
}
192.168.149.3 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "ping": "pong"
}
```

通过设置组变量替代免密登录操作。

Bash | 复制代码

```
# 设置test组变量
[root@zabbix ~]# vi /etc/ansible/hosts
192.168.149.3
[test]
192.168.149.5
[test:vars]
ansible_user=root
ansible_password=000000
# 仅test组主机正常工作
[root@zabbix ~]# ansible all -m ping
192.168.149.3 | UNREACHABLE! => {
    "changed": false,
    "msg": "Failed to connect to the host via ssh: Permission denied (publickey,gssapi-keyex,gssapi-with-mic,password).",
    "unreachable": true
}
192.168.149.5 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "ping": "pong"
}

# 设置all组变量
[root@zabbix ~]# vi /etc/ansible/hosts
192.168.149.3
[test]
192.168.149.5
[all:vars]
ansible_user=root
ansible_password=000000
# 所有主机均正常工作
[root@zabbix ~]# ansible all -m ping
192.168.149.5 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "ping": "pong"
}
192.168.149.3 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
```

```
45        "changed": false,
46        "ping": "pong"
47    }
```

扩展内容

下面是ansible内置的变量清单，可以收藏备用。

主机连接配置

- ansible_connection：与主机的连接类型。可以是ansible的连接插件的名称。SSH协议类型为smart、ssh或paramiko。默认值是smart。

所有连接配置

- ansible_host：要连接到远程主机的名称。
- ansible_port：要连接到远程主机的端口，默认为22。
- ansible_user：连接到远程主机的用户名。
- ansible_password：连接到远程主机的用户名密码。

SSH连接配置

- ansible_ssh_private_key_file：ssh使用的私有文件，适用于有多个秘钥，但不想使用ssh agent情况。
- ansible_ssh_common_args：该设置附加到默认的sftp、scp和ssh命令行上。
- ansible_sftp_extra_args：该设置总是附加到默认的sftp命令行上。
- ansible_scp_extra_args：该设置总是附加到默认的scp命令行上。
- ansible_ssh_extra_args：该设置总是附加到默认的ssh命令行上。
- ansible_ssh_pipelining：确定是否使用SSH pipelining，该参数会覆盖ansible.cfg中的pipelining设置。
- ansible_ssh_executable：此设置会覆盖使用系统ssh的默认行为，会覆盖ansible.cfg中的ssh_executable参数。

特权提升配置

- ansible_become：允许特权升级，等同于ansible_sudo、ansible_su。
- ansible_become_method：设置特权提升的方法，比如sudo。
- ansible_become_user：设置特权提升的用户，等同于ansible_sudo_user，`ansible_su_user
- ansible_become_password：设置特权用户的密码,等同于ansible_sudo_password，ansible_su_password
- ansible_become_exe：设置提权方法所用的可执行文件，等同于ansible_sudo_exe,ansible_su_exe
- ansible_become_flags：设置提权方法所用的参数，等同于ansible_sudo_flags,ansible_su_flags

远程主机环境配置

- ansible_shell_type：目标系统的shell类型.默认情况下,命令的执行使用 'sh' 语法,可设置为 'csh' 或 'fish'。

- ansible_python_interpreter：目标主机的 python 路径.适用于的情况: 系统中有多个 Python, 或者命令路径不是"/usr/bin/python",比如 *BSD, 或者 /usr/bin/python。
- ansible_*_interpreter：这里的"*"可以是 ruby 或 perl 或其他语言的解释器，作用和 ansible_python_interpreter 类似。
- ansible_shell_executable：这将设置ansible控制器将在目标机器上使用的shell，覆盖ansible.cfg中的配置，默认为/bin/sh。

小结

- Ansible：特性、工作原理、配置文件
- 定义主机与组：规则、变量

课程目标

- 知识目标：了解Ansible的特性、工作原理、目录结构。
- 技能目标：能够根据需求灵活定义主机与组。

课外拓展

- 了解更多Ansible的应用场景。

参考资料

- Ansible官方文档：https://docs.ansible.com/
- 《DevOps和自动化运维实践》，余洪春，机械工业出版社
- 《Ansible权威指南》，李松涛，机械工业出版