

4. Ansible

1. Ansible 简介

1.1 什么是Ansible

Ansible 是一款开源的自动化工具，广泛应用于配置管理、应用部署、任务自动化以及多节点管理等领域。它由 Michael DeHaan 于 2012 年创建，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。

官网文档地址：<https://docs.ansible.com/ansible/latest/>

1.2 为什么选择Ansible

1、无代理架构

Ansible 的无代理架构意味着不需要在被管理的节点上安装任何专门的代理软件。它通过 SSH与被控节点通信，实现任务的执行和数据的传输。

优点：

- **简化部署**：无需在每个被控节点上安装和维护代理程序，减少运维工作量。
- **提高安全性**：减少了潜在的攻击面，因为不需要开放额外的端口或运行额外的服务。
- **节省资源**：避免了代理软件占用系统资源的问题。

2、基于 SSH

Ansible 通过 SSH 协议与被控节点进行通信，这使得它能够在大多数类 Unix 系统（如 Linux、BSD、macOS）上无缝运行。同时，Ansible 也支持 Windows 主机，通过 PowerShell 或 WinRM 进行管理。

优势：

- **广泛兼容性**：支持多种操作系统，无需额外配置。
- **安全性高**：利用现有的 SSH 安全机制，确保数据传输的机密性和完整性。

3、声明式任务

Ansible 使用声明式语言（YAML）定义任务，描述目标状态而非具体的执行步骤。这种方式使得配置更加直观、易读，同时减少了错误和不一致性。

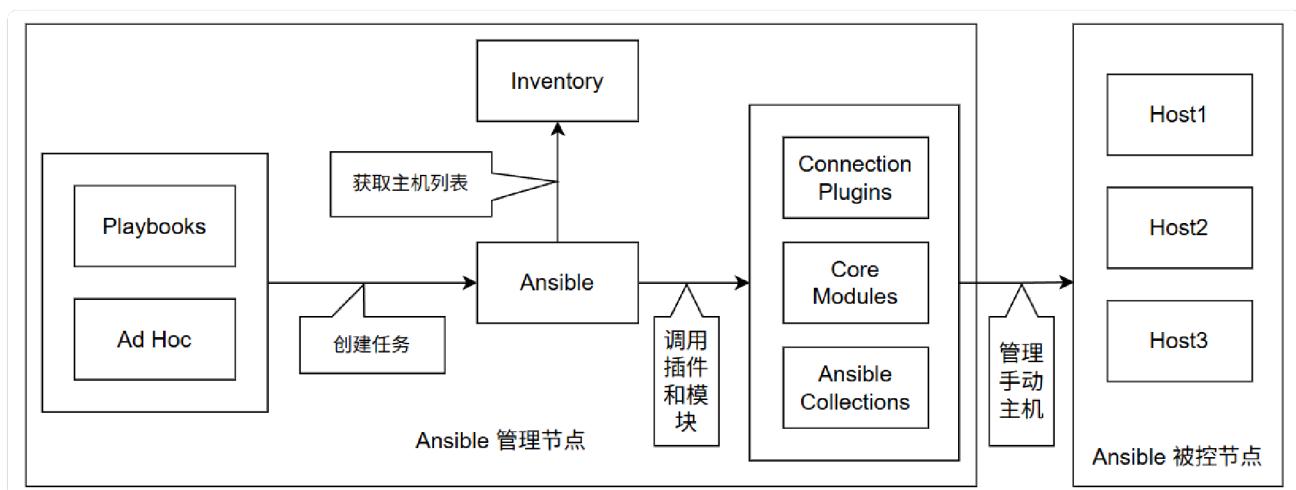
特点：

- **可读性强**：YAML 格式简洁明了，易于理解和编写。
- **可重用性高**：通过 Playbook 和角色（Roles）的复用，实现复杂任务的模块化管理。
- **幂等性**：确保多次运行相同的任务不会导致不一致的状态。

1.3 Ansible能做什么

- 批量部署软件（如安装 nginx、mysql）
- 配置文件下发与模板替换
- 重启服务、管理用户、开防火墙等
- 自动化部署 Web 应用、容器、云服务（AWS、Azure）
- 编排跨主机任务（如部署 Hadoop、K8s、Ceph）

1.4 Ansible的架构



Ansible 管理节点负责下发任务，包含以下组件：

- Ansible：Ansible-core，Ansible 的主程序，包含 Ansible 相关二进程程序
- Inventory：主机清单，用来定义被管理服务器的清单
- Ad Hoc：Ansible 的命令行，用于执行简单的任务或做调试使用

- Playbooks: 剧本，用来定义复杂的自动化任务
- Core Modules: Ansible 核心模块，包含 Ansible 最常用的模块 (Ansible 2.9 以前会包含大量模块，但在 2.10 后 Ansible 只会包含少量的核心模块)
- Ansible Collections: 核心模块只能满足基础功能，通过 Ansible Collections 可以安装新的模块来扩展 Ansible 的功能
- Connection Plugins: 连接插件，连接插件定义 Ansible 连接到远程主机的方式

Ansible 管理主机的流程：

- ① Ansible 通过 Inventory 来定义主机清单
- ② 通过 Playbooks 或 Ad Hoc 来定义任务 (任务通过模块执行) 和要执行任务的主机
- ③ 通过连接插件，将模块推送到被控端
- ④ 在被控端执行模块的任务，执行完成后删除模块

Ansible 的大部分模块都具有幂等性，多次执行只会修改需要改的内容。但部分模块除外，如 `raw`、`com
mand` 和 `shell` 模块。

Ansible 的管理节点只能是 Linux 系统。

Ansible 可以管理的设备和连接方式：

设备类型	默认连接方式	说明
Linux 系统	SSH	默认最常见方式，免密或密钥登录
Windows 系统	WinRM	需配置 WinRM 服务，支持 HTTP/HTTPS
网络设备	SSH 或 API	使用特定模块/Collection，部分设备需 NETCONF
云平台	API (HTTPS)	利用 SDK/REST API，需提供 Access Key、Token 等
Kubernetes / OpenShift	API Server	连接 kube-apiserver，使用 kubeconfig 或 Token
Docker / Podman	本地或远程 API	通过 Unix socket 或远程 TCP 端口
存储/虚拟化设备	API 或 SSH	如 VMware 使用 vCenter API，Dell EMC 用 REST API
HTTP 服务	URI 模块	<code>ansible.builtin.uri</code> 可调用 REST 接口
自定义服务	命令/脚本/API	可用 <code>command</code> ， <code>shell</code> ， <code>script</code> 模块调用

Ansible 会使用被控节点有的连接方式来管理被控节点，如通过 SSH 管理 Linux，因此 Ansible 是无代理的，不需要在被控节点安装 Agent。

2. Ansible 安装

2.1 机器准备

主机	IP	系统	说明
ansible	192.168.72.63	Redhat 9.5	安装ansible
node1	192.168.72.64	Redhat 9.5	
node2	192.168.72.65	Redhat 9.5	

2.2 安装方法

Ansible 有三种安装方式，[源码安装](#)、[发行版安装](#)和 [Python 安装](#)。

使用[发行版安装](#)或 [Python 安装](#)两种方式时，Ansible 的安装包有两个，区别如下：

- `ansible-core`：一种极简语言和运行时包，包含一组内置模块和插件。
- `ansible`：一个更大的“包含电池”的软件包，它添加了社区精选的 Ansible 集合选择，用于自动化各种设备。

在用[源码](#)或者 [Python](#) 安装 Ansible 时，默认不会安装 `sshpass` 软件包，该软件包用来给 Ansible 提供密码验证被控端，因此如果在执行 Ansible 的命令时需要输入 ssh 的密码，则需要该软件包，该软件包通过 `dnf install -y sshpass`。

```
[root@ansible ansible]# ansible servera -m ping  
servera | FAILED! => {  
"msg": "to use the 'ssh' connection type with passwords or pkcs11_provider, you  
must install the sshpass program"  
}
```

2.2.1 源码安装

只需要在 `ansible-controller` 主节点上安装即可。

```
[root@ansible ~]# dnf install python3.12 python3.12-pip sshpass  
[root@ansible ~]# tar xf ansible-2.16.3.tar.gz  
[root@ansible ~]# cd ansible-2.16.3/  
[root@ansible ansible-2.16.3]# python3 -m pip install -r ./requirements.txt  
[root@ansible ansible-2.16.3]# python3 setup.py install
```

源码安装只能安装 [Ansible-core](#)

2.2.2 发行版安装

```
[root@ansible ~]# dnf install -y https://mirrors.aliyun.com/epel/epel-release-latest-9.noarch.rpm

# 安装最简洁的 Ansible
[root@ansible ~]# dnf install ansible-core -y

# 安装包含常用模块的 Ansible
[root@ansible ~]# dnf install ansible -y
```

2.2.3 Python 安装

```
# 安装 Python3 和 pip
[root@ansible ~]# dnf install python3.12 python3.12-pip sshpass

# 安装 Ansible-core
[root@ansible ~]# python3.12 -m pip install ansible-core==2.16.3

# 安装 Ansible
[root@ansible ~]# python3.12 -m pip install ansible
```

2.2.4 设置Ansible参数自动补全

```
[root@ansible ~]# python3 -m pip install argcomplete
[root@ansible ~]# activate-global-python-argcomplete --user
```

重新登录命令行加载一下环境变量就可以看到自动补全了。

2.2.5 快速配置并使用 Ansible

在主控节点上配置。

```
[root@ansible ~]# mkdir ansible
[root@ansible ~]# cd ansible
[root@ansible ansible]# sed -i 's;/inventory=.*;/inventory\ =\ .\./inventory/' ansible.cfg

[root@ansible ansible]# cat <<EOF > ansible.cfg
[defaults]
inventory      = ./inventory
ask_pass       = false
remote_user    = root
log_path       = /var/log/ansible.log
host_key_checking = False
[privilege_escalation]
become = True
become_method = sudo
become_user = root
become_ask_pass = False
[ssh_connection]
```

```
ssh_args = -C -o ControlMaster=auto -o ControlPersist=60s -o StrictHostKeyChecking=no
EOF

[root@ansible ansible]# cat inventory <<EOF
servera
node[1,2]
EOF

[root@ansible ansible]# tail -n1 /etc/hosts
192.168.72.64 node1
192.168.72.65 node2

# 通过 ping 模块测试网络连通性
[root@ansible ansible]# ansible all -k -m ping
SSH password:
servera | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/libexec/platform-python"
    },
    "changed": false,
    "ping": "pong"
}
```

2.3 免密配置

控制节点是运行 Ansible 的主机，负责发送任务并收集结果。被控节点是被 Ansible 管理的主机，无需安装任何额外软件，仅需确保 SSH 服务正常运行，并具备必要的访问权限。

```
[root@ansible-controller ~]# ssh-keygen -t rsa (三次回车，不输入其他信息)
```

复制本机公钥到其它被控节点

```
# 执行命令后，输入正确密码即可
ssh-copy-id root@192.168.72.64
ssh-copy-id root@192.168.72.65
```

配置好后测试连接：

```
# 如果免密做成功则无需密码即可登录
ssh root@192.168.72.64
ssh root@192.168.72.65
```

注意：ssh-copy-id 命令格式有两种：

- ① ssh-copy-id 远端用户@远端IP 或 仅IP
- ② ssh-copy-id -i /root/.ssh/id_rsa.pub 远端用户@远端IP 或 仅IP

- ③ 如果在生成密钥时指定了密钥的名称，此处需要通过 `ssh-copy-id -i` 指定的名称 远程用户@远端
IP 或仅IP

如果能够无密码登录，则配置成功。

3. Ansible 配置文件

Ansible 查找名为 `ansible.cfg` 的文件来作为自己的配置文件：

```
[root@ansible ansible]# ansible --version
ansible [core 2.16.3]
  config file = /root/ansible/ansible.cfg
  configured module search path =
  ['/root/.ansible/plugins/modules', '/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/local/lib/python3.12/site-
  packages/ansible_core-2.16.3-py3.12.egg/ansible
  ansible collection location =
  /root/.ansible/collections:/usr/share/ansible/collections
  executable location = /usr/local/bin/ansible
  python version = 3.12.8 (main, Dec 12 2024, 16:30:29) [GCC 8.5.0 20210514 (Red Hat
  9.5.0-22) (/usr/bin/python3)
  jinja version = 3.1.6
  libyaml = True
```

3.1 文件位置和优先级

Ansible 的配置文件在四个位置可以出现：

- `ANSIBLE_CONFIG` 的环境变量
- 当前工作目录下的 `ansible.cfg`
- 当前用户家目录下的 `.ansible.cfg` 隐藏文件
- 全局默认的配置文件 `/etc/ansible/ansible.cfg`

优先级由高到低：环境变量 → 当前工作目录 → 家目录 → `/etc`

所以执行 Ansible 指令时，先从环境变量中找配置文件，如果环境变量没有，就到当前工作目录下找配置文件，如果当前工作目录依然没有，就到当前用户的家目录下找配置文件，如果当前用户家目录下还没有，就去找 `/etc/ansible/ansible.cfg`。

推荐的 `ansible.cfg` 管理方式的优先级从高到低如下：

- ① 当前目录的 `./ansible.cfg`
- ② 环境变量 `ANSIBLE_CONFIG` 指定的路径
- ③ 用户目录下的 `~/.ansible.cfg`
- ④ 系统路径 `/etc/ansible/ansible.cfg`

3.2 生成一个默认的配置文件

```
[root@ansible ansible]# ansible-config init --disabled > ansible.cfg
```

3.3 文件内容

```
[root@ansible ansible]# grep -E '^[\[\' ansible.cfg
[defaults]
[privilege_escalation]
[persistent_connection]
[connection]
[colors]
[selinux]
[diff]
[galaxy]
[inventory]
[netconf_connection]
[paramiko_connection]
[jinja2]
[tags]
```

Ansible 配置段说明

配置段	用途	说明
<code>[defaults]</code>	默认设置	主要配置 Ansible 的基础行为，比如 <code>inventory</code> ， <code>remote_user</code> ， <code>timeout</code> ， <code>roles_path</code> 等
<code>[privilege_escalation]</code>	提权控制	控制 <code>become</code> ， <code>become_user</code> ， <code>become_method</code> 等提权行为（如 <code>sudo</code> ）
<code>[persistent_connection]</code>	持久连接参数	控制连接插件如 SSH 的持久化（可减少连接开销）
<code>[connection]</code>	通用连接配置	包括远程连接的超时、重试、缓存控制等
<code>[colors]</code>	控制台颜色输出	定义输出中不同类型信息的颜色
<code>[selinux]</code>	SELinux 设置	是否自动管理 <code>semanage</code> 、标签恢复等（对启用 SELinux 的主机有用）
<code>[diff]</code>	显示差异设置	控制是否开启 diff 模式，能在修改配置时显示差异

配置段	用途	说明
[galaxy]	Ansible Galaxy 设置	设置 Galaxy 源、缓存策略、角色下载等
[inventory]	动态库存参数	针对 Inventory 插件的默认设置，比如缓存、路径等
[netconf_connection]	Netconf 连接参数	针对支持 NETCONF 的网络设备（如 Cisco、Juniper）连接设置
[paramiko_connection]	使用 paramiko 时的设置	使用 paramiko (Python SSH 库) 替代 OpenSSH 时的行为控制
[jinja2]	模板渲染设置	控制模板变量、未定义变量行为、Jinja 环境等
[tags]	tag 行为	控制运行 playbook 时关于 tag 的匹配方式

3.4 常用配置选项

[defaults] 常用配置：

配置项	说明	示例
inventory	指定主机清单路径	inventory = ./hosts
remote_user	默认远程登录用户	remote_user = ansible
ask_pass	是否在执行时询问 SSH 密码	ask_pass = false
ask_become_pass	是否询问 sudo 密码	ask_become_pass = true
private_key_file	指定私钥路径	private_key_file = ~/.ssh/id_rsa
host_key_checking	是否启用 SSH 主机密钥检查	host_key_checking = false
timeout	连接超时时间 (秒)	timeout = 30
forks	并发执行主机数	forks = 10
retry_files_enabled	是否生成 .retry 文件	retry_files_enabled = false
log_path	日志文件路径	log_path = /var/log/ansible.log
roles_path	指定角色目录	roles_path = ./roles
gathering	控制 facts 的收集方式 (implicit、explicit、smart)	gathering = smart
fact_caching	是否启用 facts 缓存	fact_caching = jsonfile
fact_caching_connection	facts 缓存路径	fact_caching_connection = ./fact_cache

[privilege_escalation] 提权配置：

配置项	说明	示例
<code>become</code>	是否启用提权	<code>become = true</code>
<code>become_method</code>	使用提权方式 (<code>sudo/su/pbrun/doas</code> 等)	<code>become_method = sudo</code>
<code>become_user</code>	提权后的目标用户	<code>become_user = root</code>
<code>become_ask_pass</code>	是否询问提权密码	<code>become_ask_pass = false</code>

[ssh_connection] (或 [connection]) 连接优化配置:

配置项	说明	示例
<code>pipelining</code>	启用 SSH pipelining, 加速执行	<code>pipelining = true</code>
<code>control_path</code>	SSH 控制连接的 socket 路径	<code>control_path = %(directory)s/%h-%r</code>
<code>ssh_args</code>	传递给 SSH 的参数	<code>ssh_args = -o ControlMaster=auto -o ControlPersist=60s</code>
<code>retries</code>	SSH 失败重试次数	<code>retries = 3</code>

[jinja2] 模板渲染相关:

配置项	说明	示例
<code>undefined</code>	控制未定义变量的处理方式 (如 <code>strict</code>)	<code>undefined = strict</code>
<code>trim_blocks</code>	去除 Jinja2 模板中的空行	<code>trim_blocks = true</code>
<code>lstrip_blocks</code>	去除 Jinja2 左侧空格	<code>lstrip_blocks = true</code>

[diff] 差异显示:

配置项	说明	示例
<code>always</code>	是否始终显示变更 diff	<code>always = true</code>
<code>context</code>	显示上下文行数	<code>context = 5</code>

[galaxy] Ansible Galaxy 设置:

配置项	说明	示例
<code>server_list</code>	指定 Galaxy 服务器	<code>server_list = ansible_galaxy</code>
<code>ignore_certs</code>	忽略 Galaxy HTTPS 证书验证	<code>ignore_certs = false</code>

[inventory] 主机清单设置 (动态库存) :

配置项	说明	示例
enable_plugins	启用的库存插件	enable_plugins = host_list, yaml, ini
cache	启用主机缓存	cache = true
cache_plugin	使用的缓存插件	cache_plugin = jsonfile
cache_timeout	缓存有效时间	cache_timeout = 600

其他配置段（可选）：

段名	用途
[paramiko_connection]	如果使用 paramiko 作为 SSH 连接后端，配置连接行为
[netconf_connection]	管理 NETCONF 网络设备的连接参数
[colors]	控制终端输出颜色样式
[selinux]	配置如何处理 SELinux 标签修复
[tags]	控制 tag 匹配行为，比如 skip_tags 默认值等

3.5 配置案例

```
[defaults]
inventory      = ./inventory      # 主机清单的位置
fork           = 20                # 并发执行的主机数
ask_pass        = False             # 执行 Ansible 时是否询问密码
remote_user     = root              # 表示使用 root 用户来访问被控节点
log_path        = /var/log/ansible.log    # 指定 Ansible 的日志文件位置
host_key_checking = False            # 是否进行 SSH 主机 Key 检查
ansible_python_interpreter = /usr/bin/python3.12      # 指定被控端上 Python 的解释器
[privilege_escalation]
become = True      # 是否提权
become_method = sudo    # 提权的方式
become_user = root      # 提权到哪个用户
become_ask_pass = False     # 执行 Ansible 时是否询问提权密码
```

4. Ansible 主机清单

官方文档：https://docs.ansible.com/ansible/latest/inventory_guide/intro_inventory.html

主机清单用来定义哪些主机在 Ansible 的管理范围。主机清单支持以下写法：

- IP 地址

- 主机名
- 范围 (`node[1:3]`)
- 主机组 (`server`)
- 组的子组 (`子组名:children`)

4.1 主机清单

```
192.168.72.64
```

```
[server]
node[1:2]

[db]
192.168.72.6[4:5]
```

```
[web:children]
server
db
```

直接定义主机名或地址，当前 `192.168.72.64` 不属于任何组。

```
192.168.72.64
```

定义主机组，`server` 主机组包含 `node1` 到 `node2`

```
[server]
node[1:2]
```

组和组之间可以嵌套

```
[web:children]
server
db
```

4.2 选择主机和主机组

4.2.1 匹配所有主机

```
ansible all --list-hosts
hosts (5):
  192.168.72.64
  server
  db
  192.168.72.64
  192.168.72.65
```

4.2.2 匹配指定的主机或主机组

```
ansible server --list-hosts
hosts (1):
server

ansible web --list-hosts
hosts (4):
server
db
192.168.72.64
192.168.72.65
```

4.2.3 匹配多个主机和主机组

```
ansible server --list-hosts
hosts (1):
server

ansible server,db --list-hosts
hosts (4):
server
db
192.168.72.64
192.168.72.65
```

4.2.4 匹配没有组的主机

```
ansible ungrouped --list-hosts
hosts (1):
192.168.72.64
```

4.2.5 使用通配符选择主机

```
ansible node* --list-hosts
hosts (2):
node1
node2

ansible *72.* --list-hosts
hosts (3):
192.168.72.63
192.168.72.64
192.168.72.65
```

4.2.6 通配符逻辑组合

- & : 取交集
- ! : 去反
- | : 取并集

```
# 匹配所有以node开头的主机，排除node2
ansible 'node*,!node2' --list-hosts
```

```
hosts (1):
  node1

# 匹配组server和组web两个组内相同的主机
ansible 'server,&web' --list-hosts
hosts (2):
  node1
  node2

# 在上一个例子在排除node2
ansible 'server,&web,!node2' --list-hosts
hosts (1):
  node1
```

4.2.7 使用正则匹配

```
# '~'后接正则表达式，下边例子表示匹配以no开头的主机
ansible '~^no' --list-hosts
hosts (2):
  node1
  node2
```

4.2.8 通过limit匹配主机

```
ansible all --limit node1,192.168.72.63 --list-hosts
hosts (2):
  node1
  192.168.72.63

# 使用文件匹配
cat <<EOF > ip.txt
node1
node2
192.168.72.64
EOF

ansible all --limit @/root/ansible/ip.txt --list-hosts
hosts (3):
  node1
  node2
  192.168.72.64
```

4.3 特殊的主机 localhost

最后说一个特殊的主机 `localhost`，这是一个特殊的主机，它不需要在主机清单里声明就可以使用。

`localhost` 表示本地主机，未在主机清单中声明时，Ansible 会使用 `local` 方式连接，相当于在本地执行命令，以下是它和 SSH 连接的区别：

项目	local	ssh
执行位置	控制节点本地执行	远程通过 ssh 执行

项目	local	ssh
是否走网络	否	是
权限控制	当前用户	可指定远程用户
常见用途	控制节点配置、本地调试	正式环境、管理多台主机

可以通过在主机清单内明确指定 `localhost` 来设置连接方式为 `ssh`。

`local` 和 `ssh` 连接有个最大的区别就是 `local` 会忽略 `remote_user` 选项，可以看以下例子：

```
# 配置文件
[redhat@ansible ansible]$ cat ansible.cfg
[defaults]
inventory      = ./inventory
fork          = 20
ask_pass       = False
remote_user    = remote-manager
host_key_checking = False
ansible_python_interpreter = /usr/bin/python3.12
[privilegeEscalation]
become = True
become_method = sudo
become_user = root
become_ask_pass = False

# 当前用户
[redhat@ansible ansible]$ id
uid=1000(redhat) gid=1000(redhat) groups=1000(redhat)
context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023

# 查看主机清单，当前 localhost 已被注释
[redhat@ansible ansible]$ cat inventory
#localhost ansible_ssh_password=redhat

# 测试
[redhat@ansible ansible]$ ansible localhost -b -m command -a 'id'
localhost | CHANGED | rc=0 >
uid=0(root) gid=0(root) groups=0(root)
context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
# 可以看到可以执行成功

# 删除主机清单的注释
[redhat@ansible ansible]$ cat inventory
localhost ansible_ssh_password=redhat

# 再次测试
[redhat@ansible ansible]$ ansible localhost -b -m command -a 'id'
localhost | UNREACHABLE! => {
    "changed": false,
    "msg": "Failed to connect to the host via ssh: remote-manager@localhost: Permission denied (publickey,gssapi-keyex,gssapi-with-mic,password,keyboard-interactive).",
    "unreachable": true
}
```

```
# 可以看到报错了，因为 redhat 不能直接 ssh remote-manager 用户
```

可以看到 `local` 方式会忽略 `remote_user` 选项，而 `ssh` 不会忽略。

4.4 Ansible 常用命令

Ansible 有多个命令，不同命令有不同的功能，以下列出一些常用的命令和常用参数。

```
# 设置 ansible 的配置文件  
ansible-config  
  
# 查看 ansible 相关模块和模块文档  
ansible-doc  
  
# 查看 ansible 的主机清单  
ansible-inventory  
  
# 执行临时任务  
ansible  
  
# 执行 ansible 的 playbook  
ansible-playbook  
  
# 设置 ansible 的角色  
ansible-galaxy  
  
# ansible 相关文件加密或解密  
ansible-vault
```

4.4.1 ansible-config 常用参数

```
# 生成一个注释所有配置的 ansible 配置文件  
ansible-config init --disabled -t all > ./ansible.cfg
```

4.4.2 ansible-doc 常用参数

```
# 列出所有支持的模块  
ansible-doc -l  
  
# 查看模块参数  
ansible-doc -s <模块名>  
  
# 查看模块案例  
ansible-doc <模块名>  
# 这个可以通过搜索 EXAMPLE 来快速查看使用方法  
  
# 按类型列出模块  
ansible-doc -t cache -l          # 列出所有缓存模块
```

4.4.3 ansible-inventory 常用参数

```
# 列出所有主机
```

```
ansible-inventory -i inventory --list
-i          # 指定主机清单文件
--list      # 列出所有主机

# 查看某一个主机
ansible-inventory -i inventory --host node1

# 以树状图显示并显示变量
ansible-inventory -i inventory --graph --vars

# 以 toml 格式输出主机清单
ansible-inventory -i inventory --list --toml

# 以 yaml 格式输出主机清单
ansible-inventory -i inventory --list --yaml

# 将内容输出到文件里
ansible-inventory -i inventory --list --output file
```

4.4.4 ansible 常用参数

```
ansible all -u root -k -m command -a 'pwd chdir=/opt removes=/root/anaconda-ks.cfg'
all          # 表示需要执行任务的主机或主机组
-m          # 后接要使用的模块
command     # 使用的模块
-u          # 指定连接用户
-k          # 手动输入密码
-a          # 后接模块的参数
```

4.4.5 ansible-playbook 常用参数

```
ansible-playbook --become --become-method sudo \
    --become-user root -i inventory \
    --ask-pass --ask-become-pass playbook.yaml
--become          # 启动提权
--become-method   # 设置提权方法
--become-user     # 设置提权到什么用户
--ask-pass        # 输入远程用户密码
--ask-become-pass # 设置提权密码

# 检查 playbook 是否有语法错误
ansible-playbook --syntax-check playbook.yaml

# 执行 playbook 但不做更改
ansible-playbook --check playbook.yaml
```

4.4.6 ansible-galaxy 常用选项

```
# 列出所有 role
ansible-galaxy role list

# 初始化一个名为 role 的 role1
ansible-galaxy role init role1

# 下载集合
ansible-galaxy collection download
```

4.4.7 ansible-vault 常用选项

```
# 创建加密文件  
ansible-vault create test1  
  
# 对已有文件进行加密  
ansible-vault encrypt test  
  
# 对字符串 username=root 进行加密  
ansible-vault encrypt_string "username=root"  
  
# 编辑加密文件  
ansible-vault edit test  
  
# 查看加密文件  
ansible-vault view test  
  
# 修改文件密码  
ansible-vault rekey test  
  
# 解密文件  
ansible-vault decrypt test2
```

5. Ansible 模块和集合

官方文档: <https://docs.ansible.com/ansible/latest/collections/ansible/builtin/index.html>

5.1 Ansible 模块

Ansible 通过模块执行任务，例如 `ansible.builtin.hostname` 用于设置主机名。

`ansible.builtin` 是模块的命名空间，不同的模块属于不同的命名空间，一般情况下模块名称不会冲突，但是随着模块数量的增加，不排除模块名冲突的可能，所以将不同的模块放在不同的命名空间里用于区分，如果控制节点安装的模块没有模块名冲突的情况，调用模块时可以不写命名空间，如：`hostname`。

Ansible 是 Python 写的，所以 Ansible 的大部分模块都需要被控主机有 Python 才能使用，少部分模块除外，如：`ansible.builtin.raw`。

5.1.1 状态颜色

Ansible 执行命令后返回结果会有不同的颜色来表示不同的结果状态。这些颜色说明如下：

- 绿色：命令以用户期望的执行了，但是状态没有发生改变。
- 黄色：命令以用户期望的执行了，并且状态发生了改变。
- 紫色：警告信息，说明ansible提示你有更合适的用法。
- 红色：命令错误，执行失败。
- 蓝色：详细的执行过程。

5.1.2 目录规划

对于模块的使用，我们可以使用命令行模式，也可以使用 playbook 模式，官方推荐使用 playbook 模式。在下面的模块使用演示中两种方式都会涉及。

下面我们为 playbook 模式进行目录规划。

```
playbook      # 外层目录
├ module_demo # 模块父目录
│   └ 具体的模块名, 如command_module、system_module
│       ├── inventory      # 清单目录
│       │   ├── group_vars  # 组变量目录
│       │   │   └ server.yml  # 某个组变量文件
│       │   ├── hosts_vars   # 主机变量目录
│       │   │   └ node1.yml   # 某个主机变量文件
│       │   └ hosts          # 清单文件
│       ├── ansible.cfg    # 核心配置文件
│       └ site.yml         # playbook 配置文件
```

1) `ansible.cfg` (核心配置文件)，定义清单文件路径，`ansible` 优先会找当前目录下的 `ansible.cfg` 文件，如果没有再找默认会找 `/etc/ansible/ansible.cfg` 文件。文件内容如下：

```
[defaults]
inventory=inventory/hosts
```

2) `hosts` (清单文件)，定义 `ansible` 批量管理的主机或主机组。文件内容如下：

```
[server]
node[1:2]
```

5.1.3 文件与目录操作模块

模块	功能描述
<code>ansible.builtin.file</code>	管理文件和目录属性（权限、所有权、符号链接等）。
<code>ansible.builtin.copy</code>	将本地文件复制到远程主机。
<code>ansible.builtin.fetch</code>	从远程主机下载文件到主控机。
<code>ansible.builtin.template</code>	使用 <code>Jinja2</code> 模板动态生成文件，并部署到远程主机。
<code>ansible.builtin.unarchive</code>	解压远程主机或从主控机传输的压缩文件
<code>ansible.builtin.stat</code>	获取文件状态

模块	功能描述
ansible.builtin.lineinfile	修改文件中的一行内容
ansible.builtin.blockinfile	向文件中插入多个行

5.1.3.1 file

官方文档: https://docs.ansible.com/ansible/latest/collections/ansible/builtin/file_module.html#ansible-collections-ansible-builtin-file-module

file 模块用于配置管理，远程创建目录、创建文件、创建软硬链接文件等。

file 模块常用参数说明：

- **path (必须)**：指定远程主机目录或文件目录
- **recurse (可选)**：递归授权
- **state (可选)**：指定模块创建类型，可以是以下值之一：
 - **directory**: 在远端创建目录。
 - **touch**: 在远端创建文件。
 - **link**: 在远端创建链接文件。
 - **absent**: 确保文件或目录不存在，如果文件或目录存在，则删除文件或目录。
- **mode (可选)**：设置文件或目录权限。
- **owner (可选)**：设置文件或目录所属者信息。
- **group (可选)**：设置文件或目录所属组信息。

```
# 远程创建目录
[root@ansible-controller ~]# ansible server -m file -a "path=/tmp/hehe
state=director"
# 远程创建文件
[root@ansible-controller ~]# ansible server -m file -a "path=/tmp/he.txt state=touch
mode=0640 owner=root group=root"
# 远程做软链接
[root@ansible-controller ~]# ansible server -m file -a "src=/tmp/he.txt
path=/tmp/he.txt_link state=link"
# 递归创建或更改目录权限
[root@ansible-controller ~]# ansible server -m file -a "path=/tmp/hehe
state=directory owner=root group=root mode=600 recurse=yes"
```

示例演示：

1) 项目结构

```
file_module
└── inventory
    ├── group_vars
    │   └── server.yml
    └── hosts
└── ansible.cfg
└── site.yml
```

2) ansible.cfg

```
[defaults]
inventory=inventory/hosts
```

3) inventory/hosts

```
[server]
node[1:2]
```

4) inventory/group_vars/server.yml

```
ansible_user: root
ansible_password: 123456
```

5) site.yml:

```
---
- name: copy module
  hosts: server
  gather_facts: no
  become: yes

  tasks:
    - name: create a directory if it does not exist
      file:
        path: /opt/test
        state: directory
```

6) 运行剧本

```
[root@ansible-controller ~]# ansible-playbook site.yml
```

7) 验证结果

```
[root@node1 ~]# ls /opt
test
```

5.1.3.2 copy

官方文档: https://docs.ansible.com/ansible/latest/collections/ansible/builtin/copy_m

copy 模块是将文件从控制节点复制到远程主机。这对于在远程主机上部署配置文件、脚本或其他文件非常有用。**注意：所有被管理端需要安装 libselinux-python**

copy 模块常用选项说明如下：

- **src(必需)**: 指定控制节点上要复制的源文件或目录的路径。可以是绝对路径或相对路径。
注意：Ansible 在控制节点上寻找此路径。
- **dest(必需)**: 指定目标文件或目录在远程主机上的路径。可以是绝对路径或相对路径。
注意：Ansible 在远程主机上使用此路径。
- **backup**: 设置为 yes 或 no，表示是否在复制文件之前备份目标文件。默认为 no，表示覆盖目标文件。
- **content**: 直接批量在被管理端文件中添加内容。
- **force**: 设置为 yes 或 no，表示是否强制复制文件，即使目标文件已经存在且与源文件不同。默认为 yes。
- **owner**: 指定复制后文件的所有者。
- **group**: 指定复制后文件的所属用户组。
- **mode**: 指定复制后文件的权限模式。可以使用数字或符号表示法（如：0644 或 u=rw,g=r,o=r）。
- **remote_src**: 设置为 yes 或 no，表示 src 是否在远程主机上。默认为 no，即 src 在控制节点上。
- **validate**: 在复制更新文件到最终目标之前运行的验证命令。验证命令将在临时文件上执行，并通过 %s 占位符接收临时文件路径。

```
# 安装shellcheck
[root@ansible-controller ~]# ansible server -m shell -a 'apt install shellcheck -y'
# 检查目标主机是否有文件
[root@ansible-controller ~]# ansible server -m shell -a 'ls ~/'
# 复制文件test.sh并先验证通过
[root@ansible-controller ~]# ansible server -m copy -a "src=test.sh dest=test.sh
backup=yes owner=root mode=0777 validate='shellcheck %s'"
```

示例结构：

```
copy_module
├── files
│   └── test.txt
├── inventory
│   ├── group_vars
│   │   └── server.yml
│   └── hosts
└── ansible.cfg
└── site.yml
```

1) files/test.txt

```
test
```

2) ansible.cfg

```
[defaults]
inventory=inventory/hosts
```

3) inventory/hosts

```
[server]
node[1:2]
```

4) inventory/group_vars/server.yml

```
ansible_user: root
ansible_password: 123456
```

5) site.yml:

```
---
- name: copy module
  hosts: server
  gather_facts: no
  become: yes

  tasks:
    - name: copy files from local to remote
      copy:
        src: files/test.txt
        dest: /etc/test/test.txt
```

6) 运行剧本

```
[root@ansible-controller ~]# ansible-playbook site.yml
```

7) 验证结果

```
[root@node1 ~]# ls /opt/test
test.txt
```

5.1.3.3 fetch

官方文档: https://docs.ansible.com/ansible/latest/collections/ansible/builtin/fetch_module.html#ansible-collections-ansible-builtin-fetch-module

fetch 模块是用于从远程服务复制文件到本地。

fetch 模块常用参数说明：

- **src (必须)**：远程文件路径。
- **dest (必须)**：本地存放文件路径。
- **flat (可选)**：选择 `false` 或 `true`。如果为 `true`，表示只复制文件不复制路径，且相同文件会覆盖；如果为 `false`，表示复制文件完整路径。默认为 `false`。

`site.yml`

```
---
- name: net tools module
  hosts: server
  gather_facts: no
  become: yes
  tasks:
    - name: Create a directory if it does not exist
      file:
        path: /etc/test
        state: directory
    - name: copy files from local to remote
      copy:
        src: file/test.txt
        dest: /etc/test/test.txt
    - name: test template
      template:
        src: templates/test.j2
        dest: /etc/test/test.cfg

    - name: fetch a file from remote to local
      fetch:
        src: /etc/test/test.txt
        dest: ./tmp/
```

5.1.3.4 template

官方文档：https://docs.ansible.com/ansible/latest/collections/ansible/builtin/template_module.html#ansible-collections-ansible-builtin-template-module

`template` 模块用于定义项目需要的文件。模板采用的是 `Jinja2` 语法 (<https://jinja.palletsprojects.com/en/2.10.x/>)。

`template` 模块常用参数说明：

- **src (必须)**：模板文件源路径
- **dest (必须)**：存放模板文件目标路径
- **owner**：指定复制后文件的所有者。

- group: 指定复制后文件的所属用户组。
- mode: 指定复制后文件的权限模式。可以使用数字或符号表示法（如: 0644 或 u=rw, g=r, o=r）。

示例演示：

1) 示例结构：

```
copy_module
├── inventory
│   ├── group_vars
│   │   └── server.yml
│   ├── host_vars
│   │   ├── node1.yml
│   │   └── node2.yml
│   └── hosts
├── templates
│   └── test.j2
└── ansible.cfg
└── site.yml
```

2) inventory/hosts_vars/node1.yml

```
http_port: 80
host_prefix: test1-
ip_prefix: 192.168.1.
```

3) inventory/hosts_vars/node2.yml

```
http_port: 81
host_prefix: test2-
ip_prefix: 192.168.2.
```

4) templates/test.j2

```
[default]
http_port = {{ http_port }}

[demo]
{% for id in range(201, 210) %}
{{ host_prefix }}{{ "%02d" | format(id-200) }} = {{ ip_prefix }}{{ id }}
{% endfor %}
```

5) ansible.cfg

```
[defaults]
inventory=inventory/hosts
```

6) inventory/hosts

```
[server]
node[1:2]
```

7) inventory/group_vars/server.yml

```
ansible_user: root
ansible_password: 123456
```

8) site.yml:

```
---
- name: module demo
  hosts: server
  gather_facts: no
  become: yes

  tasks:
    - name: test template
      src: templates/test.j2
      dest: /etc/test/test.cfg
```

9) 运行剧本

```
[root@ansible-controller ~]# ansible-playbook site.yml
```

10) 验证结果

```
[root@ansible-controller ~]# ansible server -m shell -a "cat /etc/test/test.cfg"
```

5.1.3.5 unarchive

unarchive 模块是用于解压一个文档。默认情况下，它会在解包之前将源文件从本地系统复制到目标系统。

unarchive 模块常用参数说明：

- **src (必须)**：要解压的存档文件路径。
- **dest (必须)**：解压缩存档后的远程绝对路径，该路径必须存在。
- **remote_src (可选)**：如果remote_src=no (默认)，则表示要解压的本地存档文件路径。如果remote_src=yes，则表示要解压的远程存档文件的路径。

剧本方式：

```
---
- name: net-tools module
  hosts: server
  become: yes

  tasks:
    - name: test get_url
      get_url:
        url: https://www.python.org/ftp/python/3.8.0/Python-3.8.0.tgz
        dest: /home/
```

```

checksum: md5:66dff5228a211e61d6d7ef4a86f5758

- name: test unarchive remote file
  unarchive:
    src: /home/Python-3.8.0.tgz      # 解压文件
    dest: /home/                      # 解压后的目录
    remote_src: yes                  # 解压远程

- name: test unarchive local file
  unarchive:
    src: Python-3.7.5.tgz          # 本地
    dest: /home/                   # 远端目录

```

5.1.4 软件包管理模块

模块	功能描述
ansible.builtin.hostname	设置主机名
ansible.builtin.user	创建、修改和删除系统用户
ansible.builtin.group	创建、修改和删除用户组
ansible.builtin.service	启动、停止、重启和启用服务
ansible.builtin.systemd	管理 systemd 服务的启动、状态等
ansible.builtin.yum	在基于 RedHat 的系统上安装、更新和删除软件包。
ansible.builtin.apt	在基于 Debian 的系统上安装、更新和删除软件包。
ansible.builtin.dnf	用于支持 DNF 包管理的系统（如 CentOS 8）
ansible.builtin.package	跨平台的软件包管理模块，会自动选择合适的管理工具。

5.1.4.1 dnf

官方文档: https://docs.ansible.com/ansible/latest/collections/ansible/builtin/dnf_module.html#ansible-collections-ansible-builtin-dnf-module

dnf 模块用于在 Redhat 系列系统中批量安装软件，相当于到远程主机上执行 `dnf -y install` 命令。

dnf 模块常用参数说明：

- **name (必须)**：指定要安装的软件包名称，如 `httpd`、`nginx`、`tomcat`。
- **state (可选)**：指定软件包的状态，可以是以下值之一：
 - `installed/present`: 确保软件包已安装，如果软件包未安装，则进行安装。
 - `removed/absent`: 确保软件包未安装，如果软件包已安装，则进行卸载。

- **latest**: 确保软件包已安装到最新版本，如果软件包已安装但不是最新版本，则进行更新。

```
[root@ansible-controller ~]# ansible server -m dnf -a "name=nginx state=present"
```

```
[root@ansible-controller ~]# ansible server -m dnf -a "name=nginx state=absent"
```

剧本方式：

```
---
- name: package module
  hosts: server
  gather_facts: no
  become: yes

  tasks:
    - name: test dnf module
      dnf:
        name: git
        state: present
    - name: delete git
      dnf:
        name: git
        state: absent
    # 安装多个软件方式一
    - name: ensure a list of packages installed
      dnf:
        name: "{{ packages }}"
      vars:
        packages:
          - httpd
          - httpd-tools
    # 安装多个软件方式二
    - name: install a list of packages
      dnf:
        name:
          - nginx
          - postgresql
          - postgresql-server
        state: present
    # 只下载安装包但不安装
    - name: download the nginx package but do not install it
      dnf:
        name:
          - nginx
        state: latest
        download_only: true
```

5.1.4.2 apt

官方文档：https://docs.ansible.com/ansible/latest/collections/ansible/builtin/apt_module.html#ansible-collections-ansible-builtin-apt-module

apt/yum模块是基于 Debian / Redhat 的系统上安装、升级或移除软件包。

apt 模块常用参数：

- **name (必需)**：要操作的软件包的名称。
- **state (可选)**：指定软件包的状态，可以是以下值之一：
 - **present**: 确保软件包已安装，如果软件包未安装，则进行安装。
 - **absent**: 确保软件包未安装，如果软件包已安装，则进行卸载。
 - **latest**: 确保软件包已安装到最新版本，如果软件包已安装但不是最新版本，则进行更新。
- **update_cache (可选)**：设置为 **yes** 或 **no**，表示是否更新软件包缓存。如果设置为 **yes**，在安装软件包之前会更新本地软件包信息。默认为 **no**。
- **force (可选)**：设置为 **yes** 或 **no**，表示是否强制重新安装软件包，即使软件包已经是最新版本。默认为 **no**。
- **allow_unauthenticated (可选)**：设置为 **yes** 或 **no**，表示是否允许安装未经验证的软件包。默认为 **no**，即只允许安装经过验证的软件包。
- **autoremove (可选)**：设置为 **yes** 或 **no**，表示是否自动删除不再需要的依赖软件包。默认为 **no**。
- **cache_valid_time (可选)**：用于指定软件包缓存的有效时间，单位为秒。默认为 **0**，表示不使用缓存。
- **deb (可选)**：用于安装本地 **deb** 格式的软件包。指定 **deb** 软件包的路径。
- **default_release (可选)**：用于指定默认的发行版 (**distribution**)。在多个发行版同时存在的情况下，指定要使用的发行版。

```
[root@ansible-controller ~]# ansible server -m apt -a "name=nginx state=present"
# 安装软件包
[root@ansible-controller ~]# ansible server -m apt -a "name=nginx state=absent" # 移
除软件包
```

剧本方式：

```
---
- name: package module
  hosts: server
  gather_facts: yes
  become: yes

  tasks:
    - name: test dnf module
      dnf:
        name: git
        state: present
      when: ansible_facts['distribution'] == "Redhat"
    - name: delete git
```

```
apt:  
  name: git  
  state: absent  
when: ansible_facts['distribution'] == "Ubuntu"
```

5.1.4.3 package

官方文档: https://docs.ansible.com/ansible/latest/collections/ansible/builtin/package_module.html#ansible-collections-ansible-builtin-package-module

package 模块用于不关心操作系统底层，让其自动识别操作系统来选择合适的安装工具。

package 模块常用参数说明：

- **name (必须)**：包名称，也可以是指定版本。
- **state (必须)**：安装状态，可选值如下：
 - **present**: 确保软件包已安装，如果软件包未安装，则进行安装。
 - **absent**: 确保软件包未安装，如果软件包已安装，则进行卸载。
 - **latest**: 确保软件包已安装到最新版本，如果软件包已安装但不是最新版本，则进行更新。
- **use (可选)**：用于确定使用哪种包管理器，默认值为 `auto`，表示可使用 `yum`、`dnf`、`apt` 等工具。

剧本方式：

```
---  
- name: package module  
  hosts: server  
  gather_facts: no  
  become: yes  
  
  tasks:  
    - name: test package module  
      package:  
        name: git  
        state: present  
    - name: delete git  
      package:  
        name: git  
        state: absent
```

缺点：不同操作系统中包名可能不同，用 package 模块的话就不太合适。

5.1.4.4 pip

pip 模块是用于管理 python 库依赖的工具。

```
---

- name: install python 3
  hosts: server

  tasks:
    - name: install python on redhat
      become: yes
      dnf:
        name:
          - python3
          - python3-devel
          - python3-pip
          - python-setuptools
        state: present
      when: ansible_facts['distribution'] == "Redhat"

    - name: install repositories
      become: yes
      apt_repository:
        repo: "{{ item }}"
        update_cache: true
      loop:
        - ppa:deadsnakes/ppa
      when: ansible_facts['distribution'] == "Ubuntu"

    - name: install Python system packages
      become: yes
      apt:
        name:
          - python3.13.2
          - python3.13.2-devel
          - python-setuptools
          - python3-pip
        state: present
        force_apt_get: yes
      when: ansible_facts['distribution'] == "Ubuntu"

    - name: pip install flask
      pip:
        name: flask
        state: present
        virtualenv: /home/vir/ # 安装到虚拟环境中
        virtualenv_command: /usr/bin/python3 -m venv      # 在 /home/vir 目录下生成虚拟环境
```

验证测试：

```
[root@ansible-node1 ~]# cd /home
[root@ansible-node1 home]# source vir/bin/active
[root@ansible-node1 home]# python
```

5.1.4.5 user

官方文档: https://docs.ansible.com/ansible/latest/collections/ansible/builtin/user_module.html#ansible-collections-ansible-builtin-user-module

`user` 模块是用于创建、修改或删除用户。

`user` 模块常用选项说明:

- `name` (必需) : 要创建或管理的用户的名称。
- `state` (可选) : 指定用户的状态, 可以是以下值之一:
 - `present`: 确保用户存在, 如果用户不存在, 则创建该用户。
 - `absent`: 确保用户不存在, 如果用户存在, 则删除该用户。
- `uid` (可选) : 指定用户的用户ID (UID)。
- `group` (可选) : 指定用户所属的主用户组。可以是组名或组ID。
- `groups` (可选) : 指定用户所属的其他附加用户组。可以是组名或组 ID 的列表。
- `home` (可选) : 指定用户的 Home 目录路径。
- `shell` (可选) : 指定用户的默认 Shell。
- `create_home` (可选) : 设置为 `yes` 或 `no`, 表示是否创建家目录。默认为 `yes` 不创建。
- `remove` (可选) : 设置为 `yes` 或 `no`, 表示是否删除家目录, 默认为 `no` 不删除。
- `password` (可选) : 指定用户的密码, 密码加密形式。

```
[root@ansible-controller ~]# ansible server -m user -a "name=testuser state=present"
[root@ansible-controller ~]# ansible server -m user -a "name=testuser state=absent"

# 创建普通用户并设置登录密码
[root@ansible-controller ~]# echo "123456" | openssl | passwd -1 -stdin # 给指定的密码
内容加密, 注意需要加密后用户才能登录

[root@ansible-controller ~]# ansible server -m user -a "name=haha password=刚加密后字
符串"

# 安装模块
[root@ansible-controller ~]# pip install passlib
# 加密密码
[root@ansible-controller ~]# ansible all -i localhost, -m debug -a "msg={{ '123456' |
password_hash('sha512', 'mysecretsat') }}"
```

剧本方式：

```
---
- name: test package
  hosts: server
  gather_facts: no
  become: yes

  tasks:
    - name: create user
      user:
        name: demo
        password: "{{ '123456' | password_hash('sha512') }}"
    - name: delete user
      user:
        name: demo
        state: absent
        remove: yes # 删除家目录, 默认
```

注意：创建用户时如果希望指定密码，需要提供 `password` 参数，参数的值必须是经过 `hash` 加密后的字符串。

5.1.4.6 group

官方文档：https://docs.ansible.com/ansible/latest/collections/ansible/builtin/group_module.html#ansible-collections-ansible-builtin-group-module

`group` 模块是用于创建、修改或删除用户组。

`group` 模块常用的选项说明：

- `name` (必需)：要创建或管理的用户组的名称。
- `state` (可选)：指定用户组的状态，可以是以下值之一：
 - `present`: 确保用户组存在，如果用户组不存在，则创建该用户组。
 - `absent`: 确保用户组不存在，如果用户组存在，则删除该用户组。
- `gid` (可选)：指定用户组的组 ID (GID)。
- `system` (可选)：设置为 `yes` 或 `no`，表示是否创建系统用户组。默认为 `no`。

```
[root@ansible-controller ~]# ansible server -m group -a "name=testgroup
state=present"
```

```
[root@ansible-controller ~]# ansible server -m group -a "name=testgroup
state=absent"
```

5.1.4.7 service

官方文档: https://docs.ansible.com/ansible/latest/collections/ansible/builtin/service_module.html#ansible-collections-ansible-builtin-service-module

service 模块用于启动、停止、重新启动或重载系统服务。常用选项说明如下：

- **name (必需)**：要操作的服务的名称。
- **state (可选)**：指定服务的状态，可以是以下值之一：
 - **started**: 确保服务处于启动状态，如果服务未启动，则启动该服务。
 - **stopped**: 确保服务处于停止状态，如果服务正在运行，则停止该服务。
 - **restarted**: 总是会重新启动服务。
 - **reloaded**: 总是重新加载服务。
- **enabled**: 确保服务在系统启动时自动启动。
- **arguments (可选)**：用于指定启动或停止服务时的额外参数。这可以是一个字符串或列表，用于传递给服务脚本。
- **sleep (可选)**：用于在执行操作之前添加延迟，以确保服务配置已生效。单位为秒。

templates/index.html.j2

```
<h1>hello ansible</h1>
<h2>{{ ansible_facts['distribution'] }}</h2>
```

site.yml

```
---
- name: system module
  hosts: server
  gather_facts: no
  become: yes

  tasks:
    - name: install nginx on ubuntu
      apt:
        name: nginx
        state: present
        force_apt_get: yes
      when: ansible_facts['distribution'] == 'Ubuntu'

    - name: change index.html on ubuntu
      template:
        src: templates/index.html.j2
        dest: /var/www/html/index.html
      when: ansible_facts['distribution'] == 'Ubuntu'
```

```
- name: install nginx on redhat
  dnf:
    name:
      - nginx
    state: present
  when: ansible_facts['distribution'] == 'Redhat'

- name: change index.html on redhat
  template:
    src: templates/index.html.j2
    dest: /usr/share/nginx/html/index.html
  when: ansible_facts['distribution'] == 'Redhat'

- name: Start service nginx
  service:
    name: nginx
    state: started
```

运行playbook

```
[root@ansible-controller ~]# ansible-playbook site.xml
```

5.1.4.8 systemd

官方文档: https://docs.ansible.com/ansible/latest/collections/ansible/builtin/systemd_module.html#ansible-collections-ansible-builtin-systemd-module

systemd 模块用于启动、停止、重新启动或重载系统服务。常用选项说明如下：

- **name (必需)**：要操作的服务的名称。
- **state (可选)**：指定服务的状态，可以是以下值之一：
 - **started**: 确保服务处于启动状态，如果服务未启动，则启动该服务。
 - **stopped**: 确保服务处于停止状态，如果服务正在运行，则停止该服务。
 - **restarted**: 总是会重新启动服务。
 - **reload**: 总是重新加载服务。
- **enabled**: 确保服务在系统启动时自动启动，yes 开机启动，no 关闭开机启动，默认值 no。
- **daemon_reload**: 重载systemd整个的配置文件。

```
[root@ansible-controller ~]# ansible server -m command -a "dnf -y install httpd"
[root@ansible-controller ~]# ansible server -m command -a "systemctl status httpd"

[root@ansible-controller ~]# ansible server -m systemd -a "name=httpd state=started
enabled=yes"
[root@ansible-controller ~]# ansible server -m systemd -a "name=httpd state=stopped
enabled=no"
```

5.1.5 命令执行模块

模块名	功能描述
ansible.builtin.ping	检测远程主机的连通性
ansible.builtin.command	运行普通命令（不支持管道、重定向）
ansible.builtin.shell	运行 shell 命令（支持管道、重定向等）
ansible.builtin.raw	原始命令执行，不走模块系统
ansible.builtin.script	将本地脚本上传并执行

5.1.5.1 ping

官方文档: https://docs.ansible.com/ansible/latest/collections/ansible/builtin/ping_module.html#ansible-collections-ansible-builtin-ping-module

这个模块的作用是检测远程主机的连通性。

```
[root@ansible-controller ~]# ansible server -m ping
```

5.1.5.2 command/shell

官方文档: https://docs.ansible.com/ansible/latest/collections/ansible/builtin/command_module.html#ansible-collections-ansible-builtin-command-module

这两个模块都是在目标主机上执行命令或命令字符串。

- `shell` 模块用于在远程主机上执行复杂的命令，包括使用管道、重定向和其他 `shell` 功能。这意味着你可以像在命令行中一样运行命令，包括使用通配符、环境变量和其他 `shell` 特性。
- `command` 模块用于在远程主机上执行单个命令，但不会通过 `shell` 运行它。这意味着不会使用 `shell` 的特性，如通配符扩展或重定向。它更适合用于简单的命令，而不涉及 `shell` 的特殊功能。

- 由于 command 模块不会在 shell 中执行命令，因此它比 shell 模块更安全，因为不会受到 shell 注入等攻击。

```
# command 模块
[root@ansible-controller ~]# ansible server -m command -a "ls ~/.ssh"

[root@ansible-controller ~]# vim test.sh
#!/bin/bash
echo "Hello World"
[root@ansible-controller ~]# scp test.sh root@192.168.72.64:/home/admin
[root@ansible-controller ~]# ansible server -m command -a "chmod +x
/home/admin/test.sh"

# shell 模块
[root@ansible-controller ~]# ansible server -m shell -a "/home/admin/test.sh"

[root@ansible-controller ~]# ansible server -m command -a "echo test > ~/test.txt"
[root@ansible-controller ~]# ansible server -m command -a "cat ~/test.txt"
[root@ansible-controller ~]# ansible server -m shell -a "echo test > ~/test.txt"
[root@ansible-controller ~]# ansible server -m shell -a "cat ~/test.txt"
```

command 模块不支持特殊字符，所以不会有内容显示，而 shell 模块是支持复杂命令的，因此会显示我们预期的内容。

剧本方式：

1) 项目结构

```
command_module
├── inventory
│   ├── group_vars
│   │   └── server.yml
│   └── hosts
└── ansible.cfg
└── site.yml
```

2) ansible.cfg

```
[defaults]
inventory=inventory/hosts
```

3) inventory/hosts

```
[server]
node[1:2]
```

4) inventory/group_vars/server.yml

```
ansible_user: root
ansible_password: 123456
```

5) site.yml:

```
---
- name: command module
  hosts: server
  become: yes
  tasks:
    - name: test command
      command: cat /etc/hosts
```

6) 运行剧本

```
[root@ansible-controller ~]# ansible-playbook site.yml
```

执行上面的剧本，会发现没有任何输出！那要如何做才会有输出呢？我们需要通过变量去接收，然后再输出这个变量的值就可以了。

7) 修改site.yml

```
---
- name: command module
  hosts: server
  become: yes

  tasks:
    - name: test command
      command: cat /etc/hosts
      register: hosts_value      # 将命令执行的结果赋值给这个变量

    - debug:
        msg: "{{ hosts_value }}"
```

8) 运行剧本

```
[root@ansible-controller ~]# ansible-playbook site.yml
```

此时就可以看到运行结果了。

5.1.5.3 script

官方文档: https://docs.ansible.com/ansible/latest/collections/ansible/builtin/script_module.html#ansible-collections-ansible-builtin-script-module

`script` 模块用于编写脚本和执行脚本。即在本地运行模块，等同于在远程执行，不需要将脚本文件进行推送到目标主机上再执行。

```
# 管理端
[root@ansible-controller ~]# vim /root/dnf_wget.sh
[root@ansible-controller ~]# cat /root/dnf_wget.sh
#!/bin/bash
dnf -y install wget

[root@ansible-controller ~]# chmod +x /root/dnf_wget.sh
[root@ansible-controller ~]# ansible server -m script -a "/root/dnf_wget.sh"

[root@ansible-controller ~]# ansible server -m shell -a "wget -V"
```

5.1.6 归档与压缩

模块名	功能描述
<code>ansible.builtin.archive</code>	创建归档文件 (如 <code>tar.gz</code>)
<code>ansible.builtin.unarchive</code>	解压归档文件

5.1.7 权限与认证

模块名	功能描述
<code>ansible.builtin.seboolean</code>	管理 SELinux 布尔值
<code>ansible.builtin.selinux</code>	设置 SELinux 模式
<code>ansible.builtin.authorized_key</code>	添加 SSH 公钥到用户 <code>~/.ssh/authorized_keys</code>

5.1.7.1 selinux

官方文档: https://docs.ansible.com/ansible/latest/collections/ansible/builtin/selinux_module.html#ansible-collections-ansible-builtin-selinux-module

`selinux` 模块用于控制 `selinux` 开启或关闭。

`selinux` 模块常用参数说明:

- `name:`
- `selinux`
 - `policy: targeted`
 - `state: disabled`

```
# 先远程查看被管理端selinux是开着的
[root@ansible-controller ~]# ansible server -m command -a 'getenforce'

# 远程关闭被管理端selinux
[root@ansible-controller ~]# ansible server -m selinux -a "state=disabled"
# 远程查看
[root@ansible-controller ~]# ansible server -m command -a 'getenforce'
```

5.1.8 网络与远程连接

模块名	功能描述
ansible.builtin.uri	发 HTTP 请求
ansible.builtin.get_url	下载文件
ansible.builtin.wait_for	等待端口/文件状态变更

5.1.8.1 get_url

官方文档: https://docs.ansible.com/ansible/latest/collections/ansible/builtin/get_url_module.html#ansible-collections-ansible-builtin-get-url-module

get_url 模块用于文件下载。常用参数说明:

- url (必须) : 下载地址
- dest (必须) : 下载到本地的绝对路径
- mode (可选) : 权限
- checksum (可选) : 对资源做校验, 可取值如下:
 - sha256:
 - md5:

```
[root@ansible-controller ~]# ansible server -m get_url -a
'url=http://rpms.famillecollect.com/enterprise/remi-release-6.rpm dest=/tmp
mode=0666'
```

剧本方式:

```

---
- name: net-tools module
  hosts: server
  become: yes

  tasks:
    - name: test get_url
      get_url:
        url: https://www.python.org/ftp/python/3.8.0/Python-3.8.0.tgz
        dest: /home/
        checksum: md5:66dfffb5228a211e61d6d7ef4a86f5758

```

5.1.8.2 uri 模块

uri 模块用于从指定网络地址中获取文件

uri 模块常用参数说明：

- **url (必须)**：通过 http 或 https 指定要获取的文件域名路径
- **src (可选)**：提交到远程服务器的文件的路径。
- **dest (可选)**：存放下载文件的目录路径。
- **method (可选)**：请求方式，默认为 GET。

site.yml

```

---
- name: net tools module
  hosts: server
  gather_facts: no
  become: yes
  tasks:
    - name: Check that you can connect(GET) to a page and it returns a status 200
      uri:
        url: http://www.example.com
        return_content: yes # 返回网站的内容
        register: result
    - debug:
        var: result

```

5.1.9 控制逻辑模块

模块名	功能描述
ansible.builtin.debug	输出调试信息
ansible.builtin.pause	暂停执行
ansible.builtin.assert	条件断言
ansible.builtin.set_fact	设置自定义变量

5.1.9.1 set_fact

官方文档: https://docs.ansible.com/ansible/latest/collections/ansible/builtin/set_fact_module.html#ansible-collections-ansible-builtin-set-fact-module

`set_fact` 模块用于操作允许设置与当前主机关联的变量。这些变量将在通过设置它们的主机运行可翻译的剧本期间可用于后续的剧本。

`set_fact` 模块参数说明:

- `key_value` (必须) : 以 `key=value` 对进行设置。
- `cacheable` (可选) : 设为 `true` 或 `false`, 表示是否进行缓存, 默认为 `false`。

`site.yml`

```
---
- name: test facts
  hosts: server
  gather_facts: no

  tasks:
    - name: gather facts
      setup:
        gather_subset: min
    - name: test set facts
      set_fact:
        test_set_fact: 'set from set_fact'
    - debug:
        var: hostvars[ansible_host]
```

运行playbook

```
[root@ansible-controller ~]# ansible-playbook site.yml
```

5.1.9.2 debug

官方文档: https://docs.ansible.com/ansible/latest/collections/ansible/builtin/debug_module.html#ansible-collections-ansible-builtin-debug-module

`debug` 模块输出调试信息的模块。常用选项说明如下:

- `var` (必需) : 指定要调试的变量名, 可以是已定义的变量名或者是一个表达式, 与 `msg` 选项互斥。
- `msg` (可选) : 自定义输出消息, 默认值为"Hello World!"。

- **verbosity (可选)**：指定调试信息的输出级别，当设置为 3 时，debug 任务只会在使用 -vvv 参数或更高级别时才执行。
- -v 参数可以用多次来增加输出的详细程序。每多一个 -v，输出的详细程序就会增加一组级。

```
# 通过debug模块输出ansible_host变量
[root@ansible-controller ~]# ansible server -m debug -a "var=ansible_host"

# 输出变量a，通过 -e 来提供额外参数
[root@ansible-controller ~]# ansible server -m debug -a "var=a" -e "a=aaa b=bbb
c=ccc"

# 输出详细信息
[root@ansible-controller ~]# ansible server -m debug -a "var=c verbosity=3" -e
"a=aaa b=bbb c=ccc" -vvv
```

5.1.10 其他模块

5.1.10.1 gather_facts

官方文档：https://docs.ansible.com/ansible/latest/collections/ansible/builtin/file_module.html#ansible-collections-ansible-builtin-file-module

`gather_facts` 模块用于远程主机的信息，如IP地址，操作系统的发行版本等。

`gather_facts` 模块常用参数说明：

- **parallel (必须)**：设置为 yes 或 no，表示是否获取远程主机信息。设置为 no 表示不获取，可提升性能。默认为 yes。

```
[root@ansible-controller ~]# ansible server -m gather_facts

# 如果希望将信息写入文件，则
[root@ansible-controller ~]# ansible server -m gather_facts --tree ./facts
```

示例演示：

1) 项目结构

```
system_module
├── inventory
│   ├── group_vars
│   │   └── server.yml
│   └── hosts
└── ansible.cfg
└── site.yml
```

2) ansible.cfg

```
[defaults]
inventory=inventory/hosts
```

3) inventory/hosts

```
[server]
node[1:2]
```

4) inventory/group_vars/server.yml

```
ansible_user: root
ansible_password: 123456
```

5) site.yml:

```
---
- name: copy module
  hosts: server
  gather_facts: yes      # 默认就有，可不写

  tasks:
    - name: test ping
      ansible.builtin.ping:          # 可以是完整路径，也可以是简单的，推荐完整路径，这样可以
        方便链接到模块文档
      ping:

    - name: print facts
      debug:
        msg: "{{ ansible_date_time }}"
```

6) 运行剧本

```
[root@ansible-controller ~]# ansible-playbook site.yml
```

5.1.10.2 cron

官方文档: https://docs.ansible.com/ansible/latest/collections/ansible/builtin/cron_module.html#ansible-collections-ansible-builtin-cron-module

cron 模块用于远程添加定时任务。相当于在远程主机上批量执行 crontab -e 命令。

```
# 远程添加定时任务，未设置注释信息
[root@ansible-controller ~]# ansible server -m cron -a "minute=00 hour=01 day=*
month=* weekday=* job='/bin/sh /root/a.sh' state=present"

# 远程添加定时任务，并设置注释信息，防止定时任务重复
[root@ansible-controller ~]# ansible server -m cron -a "minute=00 hour=01 day=*
month=* weekday=* name='注释信息' job='/bin/sh /root/a.sh' state=present"

# 远程注释定时任务
[root@ansible-controller ~]# ansible server -m cron -a "minute=00 hour=01 day=*
month=* weekday=* name='cron1' job='/bin/sh /root/a.sh' state=present disabled=yes"

# 远程删除定时任务
[root@ansible-controller ~]# ansible server -m cron -a "minute=00 hour=01 day=*
month=* weekday=* name='cron1' job='/bin/sh /root/a.sh' state=absent"
```

5.1.10.3 mount

官方文档：https://docs.ansible.com/ansible/latest/collections/ansible/builtin/mount_module.html#ansible-collections-ansible-builtin-mount-module

`mount` 模块主要用于远程挂载。

`mount` 模块常用参数说明：

- `src` (必须)：指定被挂载的原目录。
- `path` (必须)：指定挂载到的目标目录。
- `fstype` (可选)：挂载的文件类型。
- `state` (可选)：挂载或卸载的状态，常用值有：
 - `present`: 开机挂载，不会直接挂载设备，仅将配置写入`/etc/fstab`，不会马上挂载。
 - `mounted`: 马上挂载设备，并将配置写入 `/etc/fstab` 文件。
 - `unmounted`: 马上卸载设备，但不会清除 `/etc/fstab` 写入的配置。
 - `absent`: 马上卸载设备，并且会清除`/etc/fstab`写入的配置。

```
# 管理端 (192.168.72.63) 和被控端都要安装
[root@ansible-controller ~]# dnf -y install nfs-utils
[root@ansible-controller ~]# vim /etc/exports
/data *(rw,no_root_squash)
[root@ansible-controller ~]# systemctl start nfs

# 立刻挂载并写入/etc/fstab中
[root@ansible-controller ~]# ansible server -m mount -a "src=192.168.73.63:/data
path=/opt fstype=nfs opts=defaults,noatime state=mounted"

# 立刻卸载并清除/etc/fstab中信息
[root@ansible-controller ~]# ansible server -m mount -a "src=192.168.72.63:/data
path=/opt fstype=nfs opts=defaults,noatime state=absent"
```

5.1.10.4 setup

官方文档: https://docs.ansible.com/ansible/latest/collections/ansible/builtin/setup_module.html#ansible-collections-ansible-builtin-setup-module

setup 模块用于主机信息获取。

setup 模块常用参数说明:

- **filter (可选)** : 过滤出指定变量的信息, 常用可取值如下:
 - `ansible_all_ipv4_addresses`: 仅显示ipv4的信息。
 - `ansible_devices`: 仅显示磁盘设备信息。
 - `ansible_distribution`: 显示是什么系统, 例:centos,suse等。
 - `ansible_distribution_major_version`: 显示是系统主版本。
 - `ansible_distribution_version`: 仅显示系统版本。
 - `ansible_machine`: 显示系统类型, 例:32位, 还是64位。
 - `ansible_eth0`: 仅显示eth0的信息
 - `ansible_hostname`: 仅显示主机名
 - `ansible_fqdn`: 仅显示主机名。
 - `ansible_kernel`: 仅显示内核版本。
 - `ansible_lvm`: 显示lvm相关信息。
 - `ansible_memtotal_mb`: 显示系统总内存。
 - `ansible_memfree_mb`: 显示可用系统内存。

- `ansible_memory_mb`: 详细显示内存情况。
- `ansible_swaptotal_mb`: 显示总的swap内存。
- `ansible_swapfree_mb`: 显示swap内存的可用内存。
- `ansible_mounts`: 显示系统磁盘挂载情况。
- `ansible_processor`: 显示cpu个数(具体显示每个cpu的型号)
- `ansible_processor_vcpus`: 显示cpu个数(只显示总的个数)。

```
# 获取主机所有信息
[root@ansible-controller ~]# ansible server -m setup

# 获取主机的ipv4相关信息
[root@ansible-controller ~]# ansible server -m setup -a
"filter=ansible_default_ipv4"

# 获取主机名信息
[root@ansible-controller ~]# ansible server -m setup -a "filter=ansible_fqdn"

# 获取内存信息
[root@ansible-controller ~]# ansible server -m setup -a "filter=ansible_memory_mb"
```

5.1.10.5 git

官方文档: https://docs.ansible.com/ansible/latest/collections/ansible/builtin/git_module.html#ansible-collections-ansible-builtin-git-module

git 模块是在远程主机上执行 Git 相关操作，例如从 Git 仓库克隆代码、拉取更新、推送更新等。

git 模块常用选项说明：

- `repo` (必需) : Git 仓库的 URL, 可以是 HTTPS 或 SSH 协议的仓库地址。
- `dest` (必需) : 目标目录, 用于存放从 Git 仓库克隆的代码。
- `version` (可选) : 指定要克隆的 Git 分支或标签, 默认为 HEAD。可以指定分支名、标签名或提交哈希值。
- `update` (可选) : 是否拉取仓库的更新, 默认为 yes。如果设置为 yes, 在执行任务时会自动拉取远程仓库的最新代码; 如果设置为 no, 则只在仓库不存在时进行克隆, 不会拉取更新。

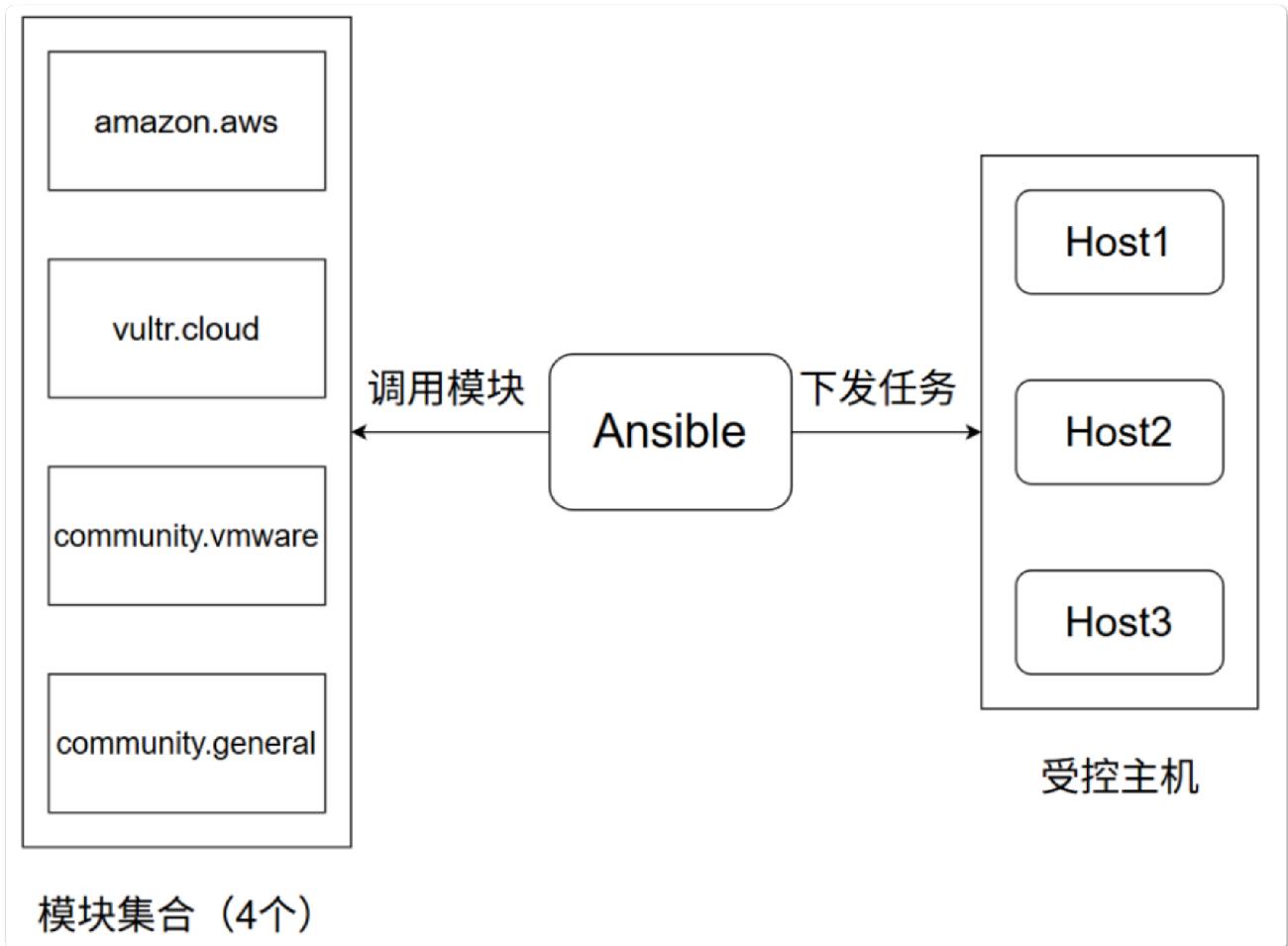
```
# 安装 git
[root@ansible-controller ~]# ansible server -m apt -a "name=git state=present"

[root@ansible-controller ~]# ansible server -m git -a
"repo=https://github.com/ansible/ansible-examples.git dest=/home/admin/ansible-ex"

[root@ansible-controller ~]# ansible server -m command -a "ls /home/admin/ansible-ex"
```

5.2 Ansible 集合

Ansible-core 自带一组模块，以 `ansible.builtin` 开头，这些模块是远远不够的，需要其他模块来扩展 Ansible 的功能，为了方便这些模块的管理，所以将这些扩展模块分类，一类模块就组成一个集合。



下载集合：

```
ansible-galaxy collection download community.general
```

6. Ansible Playbook

官方文档: https://docs.ansible.com/ansible/latest/getting_started/get_started_playbook.html

6.1 Ansible Playbook

Ansible Playbook 使用 yaml 格式，有严格的对其要求，一个 playbook 由多个 play 组成。

6.1.1 Playbook 例子

```
---
- name: init      # 定义 Playbook 的名字
  hosts: all      # 定义被控主机范围
  tasks:          # 定义要执行的任务，下边一个 - 对应一个任务（一个 - 表示一个列表）
    - name: set hostname      # 第一个任务的名称
      ansible.builtin.hostname:      # 第一个任务使用的模块
        name: "{{ inventory_hostname }}.{{ host_search_name }}"
    - name: dnf install package  # 第二个任务名称
      ansible.builtin.dnf:        # 第二个任务使用的模块
        name:
          - bash-completion
          - vim
          - gcc
          - make
          - git
          - wget
          - tar
          - bzip2
          - unzip
          - python3
          - sysstat
        state: present
    - name: set hosts
      ansible.builtin.template:
        src: ./templates/hosts.j2
        dest: /etc/hosts
        owner: root
        group: root
        mode: '0644'
```

`---`：表示 yaml 文件的文档分隔符，多个 yaml 文档可以放在一个文件里，用 `---` 来做分隔

`- name`：用于定义 Playbook 和 Play 的名称，类似于注释

`hosts`：定义 Playbook 作用的主机范围

`tasks`：定义 Playbook 的多个自动化任务，下边包含多个任务

YAML 对缩进有严格要求，比方说 `- name` 和 `hosts` 是一个等级（在一个列表），所以缩进是对齐的，`tasks` 下的 `- name` 和 `ansible.builtin.hostname` 是它的子项（是 `tasks` 下的列表），所以会多缩进两个空格（缩进以第一个字母算，不考虑 `-`）。

```
- name: set hosts
  ansible.builtin.template:
    src: ./templates/hosts.j2
    dest: /etc/hosts
    owner: root
    group: root
    mode: '0644'
```

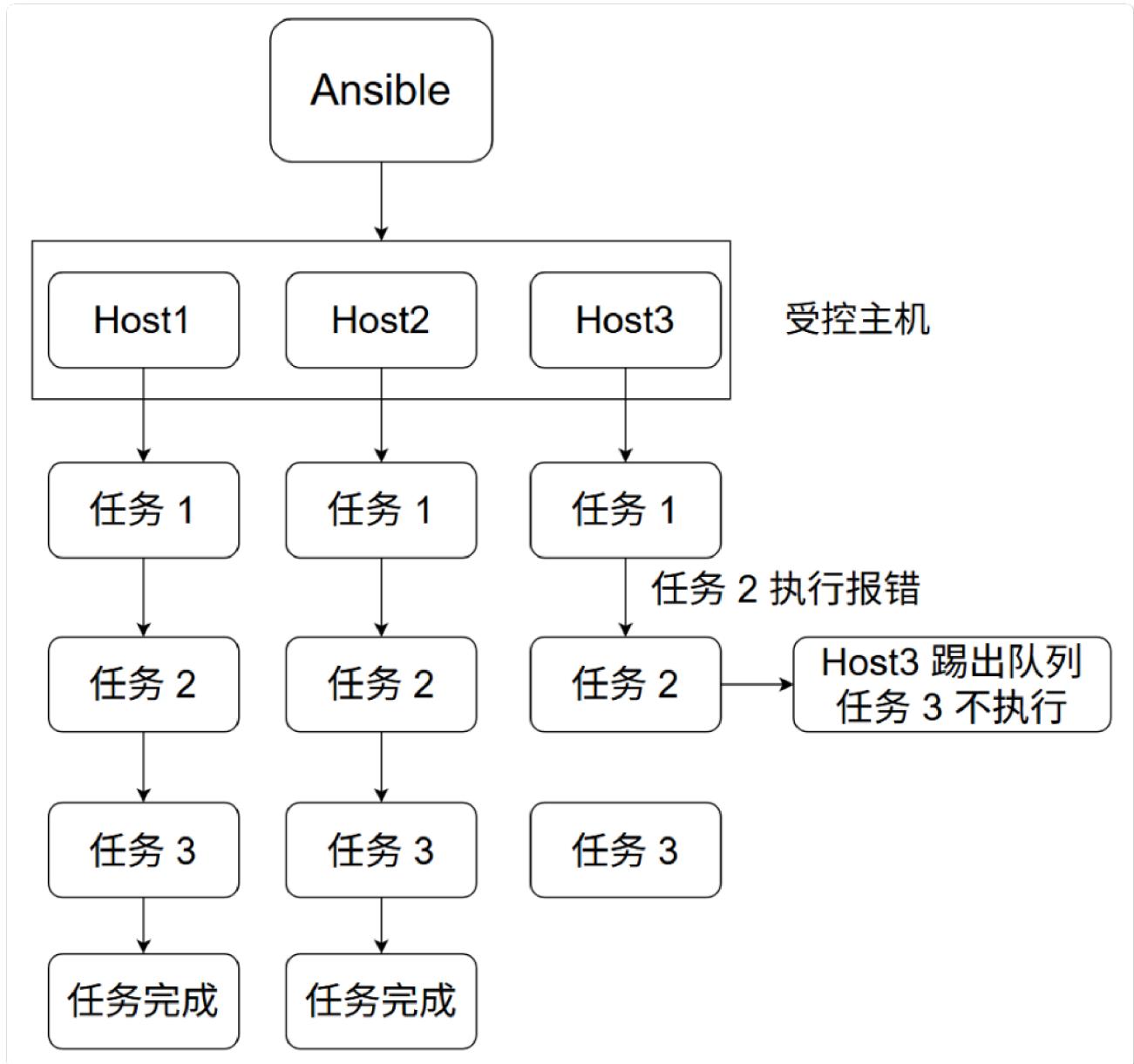
这个表示一个 play，play 的名字为 `set hosts`，使用的模块为 `ansible.builtin.template`，模块使用的参数有 `src`、`dest`、`owner`、`group`、`mode`。

6.1.2 命令常用选项

- ① 可以通过 `ansible-playbook` 命令来执行 Playbook，通过 `-v` 可以查看更详细的信息。
 - `-v`：提供任务和主机状态的详细信息，但不包括变量值和详细调试信息。
 - `-vv`：提供更详细的任务执行信息，包括变量值。
 - `-vvv`：显示包括变量展开和任务详细信息在内的更多调试信息。
 - `-vvvv`：显示所有调试信息，包括 HTTP 请求和响应的详细内容。
- ② 可以通过 `--syntax-check` 选项来对 Playbook 的格式进行检查
- ③ 可以通过 `--check` 来执行 Playbook，但并不做实际的改变

6.1.3 队列错误处理机制

Ansible Playbook 的机制是如果某个主机的某个 Play 执行错误，那么这个主机就会被移除执行队列，该主机相关的后续所有 Play 都不会执行。



有时候我们可能希望任务报错后节点继续执行后边的任务，这个时候可以使用 `ignore_errors: yes` 来实现。

```

- name: dnf install package
  ansible.builtin.dnf:
    name:      sysstat
    state:    present
    ignore_errors: yes
  
```

6.2 Ansible Handlers

Ansible Handlers 表示一组特殊的任务，它只会在特定的情况下才会被执行，例如配置 sssd 服务，可以通过 Ansible Handlers 来实现只有 sssd 服务的配置文件发生改变后才执行 sssd 服务的重启。

Ansible Handlers 有两个常用选项，`notify`、`handlers`和`listen`。

`handlers`: 定义一些特殊任务，这些特殊任务在某种条件下才会被执行，执行条件和 `notify` 有关

`notify`: 在某个 play 下追加 `notify` 后，表示这个 play 执行成功后并且做出修改后触发 `handler` 的某个特殊任务

6.2.1 Ansible Handlers 例子

```
---
- name: set sssd
  hosts: all
  gather_facts: false
  vars:
    sssd_packages:
      - sssd
      - sssd-tools
      - oddjob
      - oddjob-mkhomedir
      - libsssd_sudo
  tasks:
    - name: install packages
      ansible.builtin.yum:
        name: "{{ sssd_packages }}"
        state: present
    - name: set sssd.conf
      ansible.builtin.template:
        src: templates/sssd.conf.j2
        dest: /etc/sssd/sssd.conf
        mode: '0600'
        owner: root
        group: root
      notify: set_pam_and_service
    - name: set oddjob service
      ansible.builtin.systemd:
        name: oddjobd
        state: started
        enabled: true
  handlers:
    - name: set authselect
      ansible.builtin.command: "authselect select sssd with-sudo with-mkhomedir --force"
      listen: set_pam_and_service
    - name: set sssd service
      ansible.builtin.systemd:
        name: sssd
        state: restarted
      listen: set_pam_and_service
```

在这个 Playbook 中，在名为 `set sssd.conf` 的 Play 下定义了一个 `notify`，`notify` 的名字为 `set_pam_and_service`，然后在 `handlers` 中定义了两个 Play，分别是 `set authselect` 和 `set sssd service`，这两个 Play 都一个 `listen` 选项，`listen` 的内容均为 `set_pam_and_service`，这个实现的结果就是当 `set sssd.conf` 执行成功且做出改变之后就会触发 `set authselect` 和 `set sssd service` 这两个任务，当 `set sssd.conf` 没有做出改变时，`set authselect` 和 `set sssd service` 这两个任务会直接被跳过。

上述例子里是一个 Play 触发两个 handlers 的 Play，如果 handlers 里被触发的 Play 只有一个，那么可以不用 listen，可以直接将 handlers 里 Play 的 name 设置为和 notify 内容相同。（如果 handlers 里多个 Play 的名字为 set_pam_and_service，那么只有第一个 Play 会被执行）

```
---
- name: set sssd
  hosts: all
  gather_facts: false
  tasks:
    - name: set sssd.conf
      ansible.builtin.template:
        src: templates/sssd.conf.j2
        dest: /etc/sssd/sssd.conf
        mode: '0600'
        owner: root
        group: root
      notify: set_pam_and_service
  handlers:
    - name: set_pam_and_service
      ansible.builtin.systemd:
        name: sssd
        state: restarted
```

6.2.2 强制执行 handlers

Ansible 的机制是如果某个主机的某个 Play 执行错误，那么这个主机就会被移除执行队列，该主机相关的后续所有 Play 都不会执行，所以为了保证 handlers 能够正常执行，可以添加 force_handlers: true 选项，这样无论是否有错误任务产生 handlers 都会正常执行。

```
---
- name: set sssd
  hosts: all
  gather_facts: false
  force_handlers: true
  tasks:
    ...

```

6.3 Ansible 变量的定义和引用

- Ansible 的变量区分大小写
- Ansible 的变量不要重名，特别是不要和内置变量冲突
- Ansible 的变量名称可以使用数字、字母和下划线组成，但是只能以字母开头

6.3.1 连接变量

连接变量是 Ansible 的内置变量，这个变量控制 Ansible 对被控主机的连接方式。

可以在主机清单、Playbook 和 `ansible -e` 处设置变量。

- `ansible_host`: 指定清单中主机的真实 IP 地址
- `ansible_port`: 指定清单中主机的端口
- `ansible_user`: 指定清单中主机的连接用户
- `ansible_become`: 是否进行特权升级
- `ansible_become_user`: 通过特权升级到哪个用户
- `ansible_become_password`: 提升特权时，如果需要密码的话，可以通过该变量指定
- `ansible_sudo_exec`: 如果 `sudo` 命令不在默认路径，需要指定 `sudo` 命令路径
- `ansible_connection`: `ssh` 连接的类型: `local`, `ssh`, `paramiko`, 默认是 `ssh`
- `ansible_ssh_password`: `ssh` 连接时的密码
- `ansible_ssh_private_key_file`: 秘钥文件路径，如果不使用 `ssh-agent` 管理秘钥文件时可以使用此选项
- `ansible_ssh_executable`: 如果 `ssh` 指令不在默认路径当中，可以使用该变量来定义其路径
- `ansible_ssh_extra_args`: 额外的 `ssh` 参数。
- `ansible_python_interpreter`: 指定受控主机的 `python` 的位置

以下是在主机清单中设置变量示例：

```
[all:vars]
ansible_ssh_user = root
ansible_ssh_password = redhat
ansible_become = true
ansible_become_user = redhat
```

6.3.2 定义变量

6.3.2.1 主机清单定义

```
node1 MYSQL_VERSION=9.3.0 MYSQL_MASTER=true

[web]
node1
node2

[web:vars]
HTTPD_VERSION=2.4

[all:vars]
SYSTEM_TYPE=RHEL
```

- 主机变量写在主机后边用空格分隔，如 `MYSQL_VERSION` 和 `MYSQL_MASTER`
- 组变量通过组名接 `:vars` 定义，如 `[web:vars]`
- 所有主机的变量可以通过 `[all:vars]` 来定义

也可用 `yaml` 格式定义主机清单

```
ungrouped:
  hosts:
    node1:
      MYSQL_VERSION: 9.3.0
      MYSQL_MASTER: true

  web:
    hosts:
      node1:
      node2:
    vars:
      HTTPD_VERSION: 2.4

  all:
    vars:
      SYSTEM_TYPE: RHEL
```

6.3.2.2 Ansible Playbook 定义并使用变量

通过 `vars` 关键字设置

```
---
- name: set sssd
  hosts: all
  gather_facts: false
  vars:
    sssd_packages:
      - sssd
      - sssd-tools
      - oddjob
      - oddjob-mkhomedir
      - libsss_sudo
  tasks:
    - name: install packages
      ansible.builtin.yum:
        name: "{{ sssd_packages }}"
```

`sssd_packages`就是定义的变量

通过 set_fact 模块设置

```
- name: set fact
hosts: localhost
gather_facts: false
tasks:
- name: set fact
  ansible.builtin.set_fact:
    var1: one
    var2:
      - two
      - three
- name: print vars
  ansible.builtin.debug:
    msg: "{{ var1 }} and {{ var2 }} "
```

var1 和 var2 都是通过 `set_fact` 模块设置的变量，通过 `set_fact` 设置的变量可以给后续的模块使用。

通过文件设置变量

自定义变量文件

```
cat vars_file.yml
---
var3: three
var4: four

cat test.yml
- name: set fact
  hosts: localhost
  gather_facts: false
  vars_files:
  - ./vars_file.yml
  tasks:
  - name: print vars
    ansible.builtin.debug:
      msg: "{{ var3 }} and {{ var4 }} "
```

将变量写入 `vars_file.yml` 文件，在 `playbook` 中通过 `vars_files` 加载变量文件

设置主机变量

```
cat host_vars/localhost/vars.yml
---
HTTPD_VERSION: 2.4
cat test.yml
- name: set fact
  hosts: localhost
  gather_facts: false
  tasks:
  - name: print vars
    ansible.builtin.debug:
      msg: "{{ HTTPD_VERSION }}"
```

在 Ansible 配置文件所在目录创建一个 `host_vars` 目录，在 `host_vars` 目录下创建和清单中对应名称或地址的目录，如 `./host_vars/localhost/`（表示设置 `localhost` 主机的变量），在 `./host_vars/localhost/` 下创建任意名称的文件（文件取什么名字都可以，最好以 `.yml` 或 `.yaml` 结尾，方便观看），将变量写入文件，如 `./host_vars/localhost/vars.yml`，Ansible 会自动取读取同名目录下的文件，Playbook 可以直接引用变量。

设置主机组变量

```
cat inventory
[webserver]
node1
node2
cat group_vars/webserver/vars.yml
---
SYSTEM_TYPE: RHEL
cat test.yml
- name: set fact
  hosts: webserver
  gather_facts: false
  tasks:
    - name: print vars
      ansible.builtin.debug:
        msg: "{{ SYSTEM_TYPE }}"
```

和 `host_vars` 类似，只不过主机换成了组。

注册变量

```
cat test.yml
- name: set fact
  hosts: webserver
  gather_facts: false
  tasks:
    - name: register
      ansible.builtin.command: id
      register: register_var
    - name: print vars
      ansible.builtin.debug:
        msg: "{{ register_var }}"
```

在模块执行成功后可以通过 `register` 来将输出注册为变量，注册的变量输出如下

```
ok: [server1] => {
  "msg": {
    "ansible_facts": {
      "discovered_interpreter_python": "/usr/libexec/platform-python"
    },
    "changed": true,
    "cmd": [
      "id"
    ],
    "delta": "0:00:00.004153",
```

```
        "end": "2024-08-04 23:09:14.460466",
        "failed": false,
        "msg": "",
        "rc": 0,
        "start": "2024-08-04 23:09:14.456313",
        "stderr": "",
        "stderr_lines": [],
        "stdout": "uid=0(root) gid=0(root) groups=0(root)
context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023",
        "stdout_lines": [
            "uid=0(root) gid=0(root) groups=0(root)
context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023"
        ]
    }
}
```

这里有价值的几个输出为 `failed`、`rc`、`stderr`、`stderr_lines`、`stdout` 和 `stdout_lines`，这几个输出可用于判断执行是否成功和后续模块对这次执行结果内容的引用。

6.3.2.3 通过 Ansible Ad-Hoc 设置变量

```
ansible localhost -e SYSTEM_TYPE=RHEL -m debug -a 'msg="{{ SYSTEM_TYPE }}"'
```

使用 `-e` 选项设置变量，优先级最高。

6.3.3 Ansible Playbook 引用变量

6.3.3.1 通过 `{{}}` 引用

```
---
- name: set sssd
  hosts: all
  gather_facts: false
  vars:
    sssd_packages:
      - sssd
      - sssd-tools
      - oddjob
      - oddjob-mkhomedir
      - libsss_sudo
  tasks:
    - name: install packages
      ansible.builtin.yum:
        name: "{{ sssd_packages }}"
"{{ sssd_packages }}`就是变量引用的方式，如果是多个变量写法如下`"{{ var1 }} {{ var2 }}"
```

playbook 在使用变量的时候需要注意双引号的使用，如果某个段落的开头就是调用变量，即以 `{{` 开头，那么这个段落就需要使用双引号引起起来，但如果开始是字符串，那么可以不使用双引号。

```
---
- name: vars example
  hosts: all
  vars:
    var1: one
    var2: two
  tasks:
    - name: debug1
      ansible.builtin.debug:
        msg: vars is {{ var1 }} and {{ var2 }}
    - name: debug2
      ansible.builtin.debug:
        msg: "{{ var1 }} and {{ var2 }}"
```

template 模块也可以使用变量，使用方法和上边一样。

6.3.3.2 通过 [] 引用

[] 引用和 {{}} 的区别就是 [] 内的值会被当成变量处理，举个例子：

```
---
- name: test
  hosts: localhost
  vars:
    netcard: ens18
  tasks:
    - name: loop list
      debug:
        msg: "{{ ansible_facts[netcard].ipv4.address }}"
```

上边的 "{{ ansible_facts[netcard].ipv4.address }}" 使用了 []，netcard 也会被当成变量处理，实际要打印的变量为 ansible_facts.ens18.ipv4.address。

有的地方在引用变量是会使用 ['']，这个因为带有 ''，所以里边的值不会当成变量处理，比如：ansible_facts['ens18'].ipv4.address 就是 ansible_facts.ens18.ipv4.address。

注意使用 [] 时，左边是没有 . 的，举个例子：ansible_facts[netcard].ipv4['address']。

6.4 Ansible 事实变量和魔法变量

6.4.1 事实变量

顾名思义事实变量就是根据事实定义的变量。比方说被控主机的配置信息，如 cpu、内存、硬盘、IP 和系统版本等信息。

6.4.1.1 默认的事实变量

`ansible_facts` 是 Ansible 的事实变量。

```
ansible localhost -m ansible.builtin.setup
```

通过 `ansible.builtin.setup` 可以查看受控主机的事实变量。

```
ansible localhost -m ansible.builtin.setup -a 'filter=ansible_distribution'
ansible localhost -m ansible.builtin.setup -a
'filter=ansible_distribution_file_variety'
ansible localhost -m ansible.builtin.setup -a
'filter=ansible_distribution_major_version'
ansible localhost -m ansible.builtin.setup -a 'filter=ansible_distribution_version'
```

通过 `-a 'filter=ansible_distribution'` 来进行过滤，所有 `ansible.builtin.setup` 模块看到的变量都可以引用。

`ansible.builtin.setup` 模块的 `filter` 参数只能做一层过滤，想要获取更细的过滤需要使用 `ansible.builtin.debug` 模块。

举个例子：

```
[root@awx-1 ansible]# ansible localhost -m ansible.builtin.setup -a
'filter=ansible_python'
localhost | SUCCESS => {
    "ansible_facts": {
        "ansible_python": {
            "executable": "/usr/bin/python3.11",
            "has_sslcontext": true,
            "type": "cpython",
            "version": {
                "major": 3,
                "micro": 5,
                "minor": 11,
                "releaselevel": "final",
                "serial": 0
            },
            "version_info": [
                3,
                5,
                11,
                "final",
                0
            ]
        }
    }
}
```

```
        "11",
        "5,
        "final",
        0
    ]
}
},
"changed": false
}
[root@awx-1 ansible]# ansible localhost -m ansible.builtin.setup -a
'filter=ansible_python.version'
localhost | SUCCESS => {
    "ansible_facts": {},
    "changed": false
}
[root@awx-1 ansible]# ansible localhost -m ansible.builtin.setup -a
'filter=ansible_python.version'
localhost | SUCCESS => {
    "ansible_facts": {},
    "changed": false
}
[root@awx-1 ansible]# ansible localhost -m ansible.builtin.debug -a
'var=ansible_python.version'
localhost | SUCCESS => {
    "ansible_python.version": {
        "major": 3,
        "micro": 5,
        "minor": 11,
        "releaselevel": "final",
        "serial": 0
    }
}
```

6.4.1.2 自定义事实变量

```
cat /etc/ansible/facts.d/install факт
[install]
install_date=2024-8-4

cat /etc/ansible/facts.d/uptime факт
{
    "uptime": {
        "time": "10d",
        "health": "true"
    }
}
```

在受控主机创建 `/etc/ansible/facts.d/`，在 `/etc/ansible/facts.d/` 目录下创建变量文件（必须以 `.fact` 结尾），如 `/etc/ansible/facts.d/install.fact`，文件内容可以是 `INI` 格式，也可以是 `JSON` 格式。

```
ansible localhost -m setup -a 'filter=ansible_local'
localhost | SUCCESS => {
    "ansible_facts": {
```

```
"ansible_local": {
    "install_vars": {
        "install": {
            "install_date": "2024-8-4"
        }
    },
    "uptime_vars": {
        "uptime": {
            "health": "true",
            "time": "10d"
        }
    }
},
"changed": false
}
```

通过 `-a 'filter=ansible_local'` 可以查找自定义事实变量，`ansible_local` 表示自定义事实变量，`install_vars` 和自定义事实变量文件名对应，`install` 对应文件里的 `[install]`，`"install_date": "2024-8-4"` 对应文件里的 `install_date=2024-8-4`

```
ansible localhost -m ansible.builtin.debug -a 'msg="{{ ansible_local.uptime_vars.uptime.time }}"'
ansible localhost -m ansible.builtin.debug -a
'var=ansible_local.uptime_vars.uptime.time'
```

打印变量的时候使用 `ansible.builtin.debug` 模块打印比较方便。

6.4.2 魔法变量

官方文档：https://docs.ansible.com/ansible/latest/reference_appendices/special_variables.html#magic-variables

Ansible 魔法变量是反映当前的一些状态的变量，如配置文件位置、Ansible 版本、Python 版本、当前主机在主机清单中的名字和所属主机组，这些变量的值都是随着配置文件或主机清单等来确定的，可以用 `ansible all -m debug -a 'var=hostvars'` 来查看所有魔法变量。

```
#查看所有魔法变量
ansible all -m debug -a 'var=hostvars'

#查看主机清单中设置的主机名
ansible all -m debug -a 'var=inventory_hostname'

#查看主机所属组
ansible all -m debug -a 'var=group_names'
```

上边列出几个查看的例子，其他的以此类推。

魔法变量和事实变量的区别

我自己的理解，事实变量记录着当前受控主机的相关信息，比如CPU、内存、硬盘，当前主机的 python 版本，这些变量都是从受控主机采集的信息，所以它们的值会随着受控主机的改变而改变，Ansible 是没有办法在 server2 主机上使用 server1 的事实变量的。

但是魔法变量不一样，如受控主机在主机清单的名字、主机组名字、Ansible 的配置文件、控制节点的 python 路径等，这些都是受 Ansible 配置文件或主机清单来配置的，因为在 Ansible 配置文件和主机清单确定之后，魔法变量的值也都确定了，所以是可以在 server2 上使用 server1 的魔法变量的，例如：

```
ansible server1,server2 -m debug -a 'var=hostvars.server1.inventory_hostname'
```

这个命令会在 server1 和 server2 都输出 server1。

再举一个例子：

```
[root@awx-1 ansible]# cat hosts
servera ansible_ssh_host=127.0.0.1 ansible_ssh_user=root ansible_ssh_password=redhat
[root@awx-1 ansible]# ansible servera -i hosts -m ansible.builtin.debug -a
'var=hostvars.servera.inventory_hostname'
servera | SUCCESS => {
    "hostvars.servera.inventory_hostname": "servera"
}

[root@awx-1 ansible]# ansible servera -i hosts -m ansible.builtin.debug -a
'var=ansible_hostname'
servera | SUCCESS => {
    "ansible_hostname": "awx-1"
}
```

主机清单里有 servera 主机，通过 servera 主机查看魔法变量就能看到 servera，这个变量就是通过主机清单定义的，跟被控主机的配置没有任何关系。

6.5 Ansible 变量的优先级、判断和循环

6.5.1 Ansible 变量优先级

优先级是从高到低

- ① 命令行设置的变量优先级最高
- ② role 中的 vars 变量
- ③ playbook 中定义的变量其次
- ④ host_vars 目录和 group_vars 目录的变量 (host_vars 优先于 group_vars)

⑤ inventory 变量

⑥ role 中的 default 变量

- 变量名相同，高优先级会覆盖低优先级变量
- 变量名相同，主机变量会覆盖主机组变量
- 变量名相同，子组覆盖嵌套组的变量

6.5.2 变量的条件判断

6.5.2.1 判断的类型

针对变量有以下判断方法：

条件判断类型	示例
等于（字符串）	ansible_machine == "x86_64"
等于（数字）	ansible_distribution_major_version == 8
小于	ansible_memfree_mb < 1024
大于	ansible_memfree_mb > 1024
小于等于	ansible_memfree_mb ≤ 1024
大于等于	ansible_memfree_mb ≥ 1024
不等于	ansible_memfree_mb ≠ 1024
变量存在	custom_var is defined
变量不存在	custom_var is not defined
布尔变量为 True、True、Yes 或 1	ansible_selinux_python_present
布尔变量为 False、False、No 或 0	not ansible_selinux_python_present
第一个变量存在，且在第二个变量的列表里	ansible_distribution in supported_distros
变量是否为空	custom_var == ""
变量是否不为空	custom_var ≠ "
变量是路径时的文件类型	custom_var is file (directory, link, mount, exists)

通过 `ansible -e` 或 `ansible-playbook -e` 设置的变量为字符串。

布尔变量需要用 `var=true` 和 `var=false` 设置。

数字需要通过 `json` 方式设置：`'{ "var": 100 }'`。

判断可以组合使用，比如：

- `custom_var` is defined and `custom_var == "RedHat"`
- `ansible_distribution == "Rocky" or (custom_var is defined and custom_var == "Linux")`
- 多个判断可以用列表

```
when:
  - my_var is defined
  - my_var == 'abc'
```

6.5.2.2 例子

```
---
- name: test
  hosts: localhost
  tasks:
    - name: test
      debug:
        msg:
          - "{{ ansible_facts.hostname }}"
      when: ansible_distribution == "Rocky" or (custom_var is defined and custom_var == "Linux")
```

6.5.3 列表变量和字典变量

变量有列表和字典两种。

6.5.3.1 列表

列表是一个有序的元素集合，元素可以是数字、字符串、字典、甚至是另一个列表。可以通过下标访问列表的元素。

以下是列表变量定义的方式：

```
list_var1:
  - var1
  - var2
  - var3

list_var2: [ "var4", "var5", "var6" ]
```

6.5.3.2 字典

字典是一个无序的键值对集合，每个键都有对应的值。字典可以嵌套，键和值的类型可以是任何数据类型。

以下是字典变量定义的方式：

```
dict_var1:  
  var7: 7  
  var8: "eight"  
  
dict_var2: { var9: "nine", var10: 10 }
```

6.5.3.3 列表和字典的混合

```
list_var:  
  - var1: 1  
  - var2: "two"  
dict_var:  
  var3:  
    - var3_1  
    - var3_2  
  var4: "four"
```

6.5.4 遍历变量

遍历变量通过 `loop` 实现，以下是一个在 Playbook 遍历变量的例子：

```
---  
- name: test  
  hosts: localhost  
  vars:  
    user_list:  
      - name: "root"  
        password: "redhat"  
        host: "192.168.1.1"  
      - name: "admin"  
        password: "redhat"  
        host: "192.168.1.2"  
  tasks:  
    - name: loop list  
      debug:  
        msg: "{{ item }}"  
        loop: "{{ user_list }}"  
    - name: loop list.name  
      debug:  
        msg: "{{ item.name }}"  
        loop: "{{ user_list }}"
```

早版本的 Ansible 可能会有 `with_items` 或 `with_dict`，不过已经被淘汰了（可能还支持），新版本统一用 `loop`。

6.6 Ansible 通过 ansible-doc 查询模块选项和使用案例

6.6.1 查看Ansible版本

```
[root@awx-1 ansible]# ansible --version
ansible [core 2.16.3]
  config file = /root/ansible/ansible.cfg
  configured module search path =
  ['/root/.ansible/plugins/modules', '/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python3.12/site-packages/ansible
  ansible collection location =
  /root/.ansible/collections:/usr/share/ansible/collections
  executable location = /usr/bin/ansible
  python version = 3.12.8 (main, Dec 12 2024, 16:30:29) [GCC 8.5.0 20210514 (Red Hat
  8.5.0-22)] (/usr/bin/python3.12)
  jinja version = 3.1.2
  libyaml = True
```

6.6.2 通过 ansible-doc 查询模块

通过 `ansible-doc -l` 可以列出当前所有可用模块：

```
[root@awx-1 ansible]# ansible-doc -l | grep ansible.builtin | head
ansible.builtin.add_host
    Add a host (and alternatively a group) to the ansible-playbo ...
ansible.builtin.apt
    ...
ansible.builtin.apt_key
    Ad ...
ansible.builtin.apt_repository
    Add and r ...
ansible.builtin.assemble
    Assemble configuratio ...
ansible.builtin.assert
    Asserts give ...
ansible.builtin.async_status
    Obtain statu ...
ansible.builtin.blockinfile
    Insert/update/remove a text block surr ...
ansible.builtin.command
    Execu ...
ansible.builtin.copy
    Copy fil ...
```

通过 `ansible-doc ansible.builtin.yum` 可以查询 `ansible.builtin.yum` 使用说明：

```
[root@awx-1 ansible]# ansible-doc ansible.builtin.yum
```

查询的信息有以下重要段落（比方说通过 `/EXAMPLE` 来搜索例子）：

- **OPTIONS**: 可用选项 (选项) 针对 **OPTIONS**, 有两个重要的属性:

- `type` : 定义了选项的数据结构 (`list/str/bool` 等)

比如 `ansible.builtin.yum` 模块的 `name` 选项的 `type` 为 `list`, 所以写法如下:

```
- name: Download the nginx package but do not install it
  ansible.builtin.yum:
    name:
      - nginx
      - mysql
    state: latest
```

- `list` : 列表
- `str` : 字符串
- `bool` : 布尔值 (`yes/no`、`true/false`)
- `default` : 定义选项的默认值

- **EXAMPLES**: 使用例子**EXAMPLE** 里列出了常用方法, 不会用模块的时候可以参考 **EXAMPLE**。

通过 `ansible-doc -s ansible.builtin.yum` 可以列出模块所有选项:

```
[root@awx-1 ansible]# ansible-doc -s ansible.builtin.yum
```

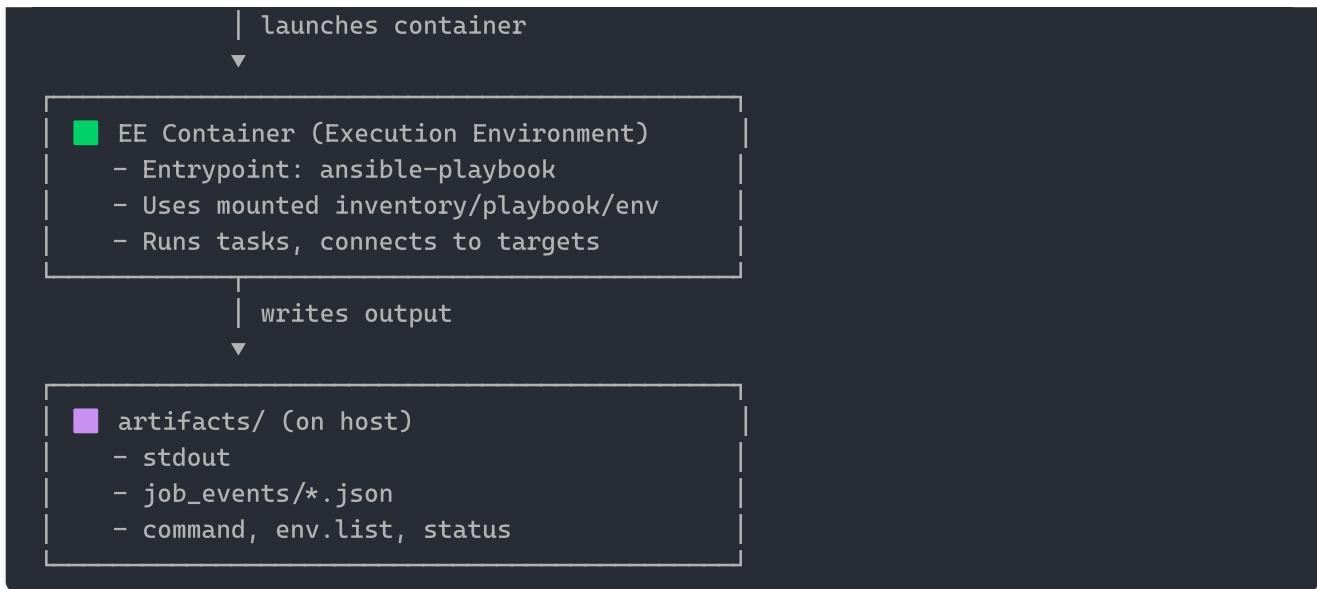
6.7 Ansible-navigator 介绍

简单讲, `ansible-navigator` 就是通过容器来运行 **Ansible** 任务, `ansible-navigator` 调用 `ansible-runner` 接口来启动容器并执行任务。

```
■ ansible-navigator
  - Reads .ansible-navigator.yml
  - Parses CLI arguments
  - Constructs execution context
```

calls ansible-runner API
▼

```
■ ansible-runner (Python API)
  - Builds podman/docker command
  - Prepares artifacts directory structure
  - Generates env.list, cmd, settings
```



`ansible-navigator` 可以通过容器完成 `Ansible` 所有的命令。

6.7.1 查看帮助文档

通过 `ansible-navigator --help` 可以查看帮助文档。

通过 `ansible-navigator <subcommand> --help` 可以查看子选项帮助文档（比方说 `ansible-navigator run --help`）。

```
[root@ansible-controller ansible-navigator]# ansible-navigator --help
Usage: ansible-navigator [options]

Options (global):
-h      --help                                Show this help message and exit
--version                               Show the application version and
exit
--rad   --ansible-runner-artifact-dir          The directory path to store
artifacts generated by ansible-runner
--rac   --ansible-runner-rotate-artifacts-count Keep ansible-runner artifact
directories, for last n runs, if set to 0 artifact directories won't be deleted
--rt    --ansible-runner-timeout                The timeout value after which
ansible-runner will forcefully stop the execution
--rwje  --ansible-runner-write-job-events      Write ansible-runner job_events in
the artifact directory (true|false)
--cdcp  --collection-doc-cache-path           The path to collection doc cache
(default: /root/.cache/ansible-navigator/collection_doc_cache.db)
--ce    --container-engine                     Specify the container engine
(auto=podman then docker) (auto|podman|docker) (default: auto)
--co    --container-options                   Extra parameters passed to the
container engine command
--dc    --display-color                      Enable the use of color for mode
interactive and stdout (true|false) (default: true)
--ecmd  --editor-command                    Specify the editor command (default:
vi +{line_number} {filename})
--econ  --editor-console                   Specify if the editor is console
based (true|false) (default: true)
--ee    --execution-environment            Enable or disable the use of an
execution environment (true|false) (default: true)
```

```

--eei  --execution-environment-image          Specify the name of the execution
environment image (default: ghcr.io/ansible/community-ansible-dev-tools:latest)
--eev  --execution-environment-volume-mounts  Specify volume to be bind mounted
within an execution environment (--eov /home/user/test:/home/user/test:Z)
--la   --log-append                         Specify if log messages should be
appended to an existing log file, otherwise a new log file will be created per
session (true|false)                           (default: true)
--lf   --log-file                           Specify the full path for the
ansible-navigator log file (default: /root/ansible-navigator/ansible-navigator.log)
--ll   --log-level                          Specify the ansible-navigator log
level (debug|info|warning|error|critical) (default: warning)
-m    --mode                               Specify the user-interface mode
(stdout|interactive) (default: interactive)
--osc4 --osc4                             Enable or disable terminal color
changing support with OSC 4 (true|false) (default: true)
--penv --pass-environment-variable        Specify an existing environment
variable to be passed through to and set within the execution environment (--penv
MY_VAR)
--pa   --pull-arguments                   Specify any additional parameters
that should be added to the pull command when pulling an execution environment from
a container                                         registry. e.g. --pa='--tls-
verify=false'
--pp   --pull-policy                      Specify the image pull policy
always:Always pull the image, missing:Pull if not locally available, never:Never
pull the image, tag:if the
                                         image tag is 'latest', always pull
the image, otherwise pull if not locally available (always|missing|never|tag)
(default: tag)
--senv --set-environment-variable       Specify an environment variable and
a value to be set within the execution environment (--senv MY_VAR=42)
--tz   --time-zone                        Specify the IANA time zone to use or
'local' to use the system time zone (default: utc)

```

Subcommands:

{subcommand} --help	
builder	Build [execution environment]
(https://ansible.readthedocs.io/en/latest/getting_started_ee/index.html) (container image)	
collections	Explore available collections
config	Explore the current ansible
configuration	
doc	Review documentation for a module or
plugin	
exec	Run a command within an execution
environment	
images	Explore execution environment images
inventory	Explore an inventory
lint	Lint a file or directory for common
errors and issues	
replay	Explore a previous run using a
playbook artifact	
run	Run a playbook
settings	Review the current ansible-navigator
settings	
welcome	Start at the welcome page

6.7.2 ansible-navigator 子命令

子命令	功能描述	常见用途
<code>run</code>	运行一个 playbook	执行自动化任务
<code>exec</code>	在执行环境中运行命令	进入容器环境进行调试、执行 shell
<code>images</code>	查看本地已有的执行环境镜像	管理 EE 镜像
<code>builder</code>	构建 EE 容器镜像 (使用 ansible-builder)	自定义执行环境
<code>collections</code>	查看当前可用的 Ansible Collections	了解已安装的模块包
<code>config</code>	查看当前 ansible 配置 (ansible.cfg)	快速查明配置来源
<code>doc</code>	查看模块或插件文档	查阅用法，如：doc -t module ping
<code>inventory</code>	浏览并测试 inventory 文件	查看组/主机结构或变量
<code>lint</code>	对 Playbook、role 进行语法检查	检查最佳实践、错误
<code>replay</code>	回顾已执行过的 playbook (artifact)	用于调试和回顾历史执行
<code>settings</code>	显示当前的 ansible-navigator 设置	查看配置项及来源
<code>welcome</code>	打开欢迎界面 (TUI 模式)	从 TUI 界面开始导航

6.7.3 ansible-navigator 通用选项

基础选项

参数	含义	默认值
<code>-h</code> , <code>--help</code>	显示帮助信息	-
<code>--version</code>	显示版本信息	-
<code>-m</code> , <code>--mode</code>	UI 模式： <code>stdout</code> 或 <code>interactive</code>	<code>interactive</code>
<code>--tz</code>	设置时区 (如 <code>utc</code> 、 <code>local</code>)	<code>utc</code>

执行环境相关 (Execution Environment)

参数	含义	默认值
<code>--ee</code> , <code>--execution-environment</code>	是否启用 EE 容器	<code>true</code>
<code>--eei</code> , <code>--execution-environment-image</code>	指定 EE 镜像名	<code>ghcr.io/ansible/community-ansible-dev-tools:latest</code>
<code>--ce</code> , <code>--container-engine</code>	容器引擎： <code>auto</code> / <code>podman</code> / <code>docker</code>	<code>auto</code>

参数	含义	默认值
--co , --container-options	添加额外 <code>docker/podman run</code> 的参数	-
--eenv , --execution-environment-volume-mounts	执行环境挂载卷 (多次使用)	-
--penv , --pass-environment-variable	传入已有环境变量	-
--senv , --set-environment-variable	在 EE 中设置新变量	-
--pa , --pull-arguments	拉取镜像时的额外参数 (<code>docker/podman pull</code> 的参数, 如 <code>--tls-verify=false</code>)	-
--pp , --pull-policy	镜像拉取策略: <code>always</code> / <code>missing</code> / <code>never</code> / <code>tag</code>	<code>tag</code>

这里解释一下 `--pull-policy` :

- `always` : 总是拉取镜像
- `missing` : 本地没有时拉取镜像
- `never` : 从不拉取镜像
- `tag` : 当镜像标签为 `latest` 时总是拉取镜像, 为其他标签时, 只在本地没有时拉取镜像

日志与调试

参数	含义	默认值
--ll , --log-level	日志级别 (debug/info/warning/...)	<code>warning</code>
--lf , --log-file	日志文件路径 (默认保存到当前目录下的 <code>ansible-navigator.log</code> 文件中)	-
--la , --log-append	是否追加写入日志文件	<code>true</code>

ansible-runner 相关

参数	含义	默认值
--rad , --ansible-runner-artifact-dir	设置 artifact 输出目录	在 /tmp 下创建临时目录
--rac , --ansible-runner-rotate-artifacts-count	保留最近多少次执行记录	不清理 (默认)
--rt , --ansible-runner-timeout	runner 执行超时时间 (秒)	-
--rwje , --ansible-runner-write-job-events	是否写入 job events 文件	false

UI 和颜色控制

参数	含义	默认值
--dc , --display-color	是否启用颜色输出	true
--osc4 , --osc4	是否支持 OSC4 改变终端色彩	true

编辑器行为 (用于 doc、lint 等交互)

参数	含义	默认值
--ecmd , --editor-command	打开文件时使用的编辑器命令	vi +{line_number} {filename}
--econ , --editor-console	编辑器是否基于控制台	true

集合文档缓存

参数	含义	默认值
--cdcp , --collection-doc-cache-path	集合文档缓存路径	/root/.cache/ansible-navigator/collection_doc_cache.db

针对布尔值选项，有的布尔值后需要加 True/False，比方说 --log-append false。

有的是默认为 false，想要设置为 true 时，直接添加选项而不用加 true，比方说 --ansible-runner-write-job-events。

6.7.4 Ansible-navigator 常用选项

参数	含义	示例
-m , --mode	指定运行模式: stdout / interactive	--mode stdout
--ee	是否启用执行环境 (容器)	--ee true
--eei	执行环境镜像名称	--eei quay.io/ansible/awx-ee:24.6.1

参数	含义	示例
--pa	拉取 EE 镜像时附加参数	--pa '--tls-verify=false'
--pp	EE 镜像拉取策略	--pp always / missing / never / tag
--ce	指定容器引擎	--ce docker / --ce podman
--co	额外容器参数	--co "--net=host"
--eev	执行环境内挂载路径 (可多次使用)	--eev /data:/data:Z
--penv	传入宿主机环境变量	--penv http_proxy
--senv	设置 EE 内的新环境变量	--senv ANSIBLE_STDOUT_CALLBACK=debug
--rad	runner artifact 输出路径	--rad ./artifacts
--ll	日志级别	--ll debug
--lf	日志文件路径	--lf /tmp/navigator.log
--rad	设置 artifact 输出目录	在 /tmp 下创建临时目录
--rac	保留最近多少次执行记录	不清理 (默认)
--rt	runner 执行超时时间 (秒)	-
--rwje	是否写入 job events 文件	false

下边写一个例子：

```
[root@ansible-controller ansible-navigator]# ansible-navigator \
--eei quay.io/ansible/awx-ee:24.6.1 --pa='--tls-verify=false' --pp missing \
--ll debug --la false --lf ./log/ansible-navigator.log \
-m stdout \
run test.yml \
--pas ./artifacts/{playbook_name}/{playbook_name}-{time_stamp}.json
```

有些参数，像 `--pa`、`--co` 或 `--eev` 等，其实就是 `podman` (或 `Docker`) 在启动容器时加的选项，就不做过多介绍了 (后边会去验证)。

剩下的一些参数也不难理解，多测试看看，像 `--rac` 或 `--rad` 后边再说。

6.7.5 Ansible-navigator 常用操作

前边把所有子命令和选项都列出来了，不过我常用的不多，就下边这些：

- 通过 `-m stdout` 来设置非交互模式
- 通过 `--eei` 设置使用的镜像
- 通过 `--pp` 设置镜像拉取策略
- 通过 `ansible-navigator.yml` 为 `ansible-navigator` 设置持久化参数

- 设置 `ansible-navigator` 保存执行过程中的事件和执行结果，以便回放或故障排查
- 常用子选项：
 - `run`：跑 Playbook
 - `exec`：跑单条 Ansible 命令用
 - `settings`：查看 `ansible-navigator` 的设置，也可以用它导出一个默认的设置文件（就是上边那个 `ansible-navigator.yml`）
- 通过 `ansible-builder` 来构建 Ansible 执行环境容器镜像（`ansible-navigator builder` 也能构建，不过感觉没 `ansible-builder` 方便）

6.8 案例实战

案例1：远程批量安装 httpd，若修改配置文件则重新推送后，但不会重启服务，因此新的配置不会生效

```
# httpd_install.yaml
# 第一步：定义hosts，对应主机清单文件中的配置
# 第二步：定义任务
# 第三步：定义具体操作
# remote_user: 指定远程主机上使用的用户
# gather_facts: 执行playbook时，默认会收集目标主机的信息，禁用掉能提高效率，默认为 no
- hosts: server
  remote_user: root
  gather_facts: no
  tasks:
    - name: install httpd
      dnf: name=httpd,httpd-tools state=installed
    - name: configure httpd
      copy: src=/root/httpd.conf dest=/etc/httpd/conf/httpd.conf
    - name: start httpd server
      systemd: name=httpd state=started enabled=yes
```

然后在控制端安装 httpd，从而获取到配置文件。

```
[root@ansible-controller ~]# dnf -y install httpd
[root@ansible-controller ~]# cp /etc/httpd/conf/httpd.conf /root/
```

然后再执行 playbook：

```
[root@ansible-controller ~]# ansible-playbook install_httpd.yml
```

此时可以通过浏览器来访问 httpd 服务。但如果我们将控制端的配置文件端口修改为 81，然后再推送，但服务并没有生效。

```
[root@ansible-controller ~]# vim /root/httpd.conf
....
Listen=81    # 将80改为81

# 推送
[root@ansible-controller ~]# ansible-playbook install_httpd.yml

# 检查被控端的配置
[root@ansible-controller ~]# ansible server -m shell -a "cat
/etc/httpd/conf/httpd.conf | grep 81"
Listen 81
```

此时，新的端口是不能在浏览器中访问的。

```
curl 127.0.0.1:81
```

案例2：远程批量安装 httpd，如果修改了配置文件，则重新推送并触发重启服务，让新的配置生效

```
- hosts: server
  remote_user: root
  gather_facts: no
  tasks:
    - name: install httpd
      dnf: name=httpd,httpd-tools state=installed
    - name: configure httpd
      copy: src=/root/httpd.conf dest=/etc/httpd/conf/httpd.conf
      notify: Restart httpd server
    - name: start httpd server
      systemd: name=httpd state=started enabled=yes
  handlers:
    - name: Restart httpd server
      systemd: name=httpd state=restarted
```

检查语法是否有误：

```
[root@ansible-controller ~]# ansible-playbook --syntax-check httpd_install.yml
```

此时再修改配置文件的端口后重新推送。

```
[root@ansible-controller ~]# vim /root/httpd.conf
....
Listen=82    # 将81改为82

# 推送
[root@ansible-controller ~]# ansible-playbook install_httpd.yml

# 检查被控端的配置
[root@ansible-controller ~]# ansible server -m shell -a "cat
/etc/httpd/conf/httpd.conf | grep 82"
Listen 82
```

这时再打开浏览器，就可以使用 82 端口来访问了。

```
curl 127.0.0.1:82
```

案例3：远程卸载 httpd，并且删除相应的用户和配置文件。

```
# httpd_removed.yml
- hosts: server
  remote_user: root
  gather_facts: no
  tasks:
    - name: remove httpd
      dnf: name=httpd,httpd-tools state=absent
    - name: remove apache user
      user: name=apache state=absent
    - name: remove configure file
      file: name=/etc/httpd state=absent
```

执行这个 playbook 配置。

```
[root@ansible-controller ~]# ansible-playbook httpd_removed.yml
```

案例4：在管理端安装 nfs 服务，在被管理端批量挂载 nfs 的共享目录。

```
[root@ansible-controller ~]# dnf -y install nfs-utils
[root@ansible-controller ~]# vim /etc/exports
/data *(rw,no_root_squash)
[root@ansible-controller ~]# mkdir /data
[root@ansible-controller ~]# echo 111 > /data/a.txt
```

然后编写剧本配置文件：

```
# web_mount.yml
- hosts: server
  remote_user: root
  gather_facts: no
  tasks:
    - name: Mount nfs server
      mount: src=192.168.72.63:/data path=/data fstype=nfs opts=defaults
state=mounted
```

执行剧本：

```
[root@ansible-controller ~]# ansible-playbook web_mount.yml
```

案例5：远程批量安装 rsync 服务，并设置管理端修改配置文件变动后执行时触发重启服务

准备配置文件：

```
[root@ansible-controller ~]# cat conf/rsync.conf
uid = www
gid = www
```

```
port = 873
fake super = yes
use chroot = no
max connections = 200
timeout = 600
ignore errors
read only = false
list = false
auth users = rsync backup
secrets file = /etc/rsyncd.password
log file = /var/log/rsyncd.log
[data]
path=/data
```

编写playbook配置文件。

```
# rsync_install.yaml
- hosts:
  remote_user: root
  gather_facts: no
  tasks:
    - name: Install Rsync Server
      dnf: name=rsync state=installed
    - name: configure rsync server
      copy: src=./conf/rsync.conf dest=/etc/rsync.conf
      notify: Restart Rsync Server
    - name: create virt user
      copy: content='rsync_backup:1' dest=/etc/rsyncd.password mode=600
    - name: create group
      group: name=www gid=666
    - name: create user
      user: name=www uid=666 group=www create_home=no shell=/sbin/nologin
    - name: create data
      file: path=/data state=directory recurse=yes owner=www group=www mode=755
    - name: start rsync server
      systemd: name=rsyncd state=started enabled=yes
  handlers:
    - name: Restart Rsync Server
      systemd: name=rsyncd state=restarted
```

运行这个配置文件，来实现批量安装配置。

```
[root@ansible-controller ~]# ansible-playbook rsync_install.yaml
```

接下来进行功能测试：

```
# 在主控端
[root@ansible-controller ~]# dnf -y install rsync
[root@ansible-controller ~]# echo 1 > /etc/rsync.pass
[root@ansible-controller ~]# chmod -R 600 /etc/rsync.pass
[root@ansible-controller ~]# echo 111 > a.txt
[root@ansible-controller ~]# rsync -av a.txt rsync_backup@192.168.72.64::data --password-file=/etc/rsync.pass
[root@ansible-controller ~]# rsync -av a.txt rsync_backup@192.168.72.65::data --password-file=/etc/rsync.pass
```

```
# 在被控端
[root@ansible-node1 ~]# systemctl status rsyncd
[root@ansible-node1 ~]# ls /data
a.txt
[root@ansible-node1 ~]# cat /data/a.txt
111
```

案例6：远程批量安装 nfs 服务，并在管理端修改配置文件后执行触发重启服务

首先在主控端创建好共享目录，并配置好共享设置。

```
[root@ansible-controller ~]# mkdir /data ~/conf
[root@ansible-controller ~]# cat conf/exports
/data *(rw,no_root_squash)
```

然后编写playbook配置文件。

```
# nfs_install.yaml
- hosts: server
  remote_user: no
  gather_facts: no
  tasks:
    - name: Install nfs server
      dnf: name=nfs-utils state=installed
    - name: configure nfs server
      copy: src=./conf/exports dest=/etc/exports
      notify: restart nfs server
    - name: create share data directory
      file: path=/data state=directory recurse=yes owner=root group=root mode=755
    - name: start nfs server
      service: name=nfs-server state=started enabled=yes
  handlers:
    - name: restart nfs server
      service: name=nfs-server state=restarted
```

执行这个 playbook 来批量安装。

使用测试：

```
# 挂载
[root@ansible-controller ~]# mount -t nfs 192.168.72.64:/data /tmp
# 验证
[root@ansible-controller ~]# df -h
# 卸载
[root@ansible-controller ~]# umount /tmp
```

案例7：远程批量添加定时任务

首先定义 playbook 配置文件。

```
# cron-add.yaml
- hosts: server
  remote_user: root
  gather_facts: no
  tasks:
    - name: Crontab Scripts
      cron: name='dellog scripts' minute=00 hour=01 job="/bin/sh /server/scripts/delete_log.sh &>/dev/null state=present"
```

然后执行这个配置：

```
[root@ansible-controller ~]# ansible-playbook cron_add.yaml

# 要被控节点查看
[root@ansible-node1 ~]# crontab -l
```

案例8：远程批量源码安装 nginx 服务

首先在主控端创建nginx源码下载目录。

```
[root@ansible-controller ~]# ls
nginx-1.23.3.tar.gz nginx_source_code_deploy.yaml
```

然后再创建 playbook 配置文件。

```
# nginx_source_code_deploy.yaml
- hosts: server
  remote_user: root
  gather_facts: no
  vars:
    src_nginx: /root/nginx-1.23.3.tar.gz
    nginx_unzip_dir: /usr/local/src
    nginx_install_dir: /usr/local/nginx
    nginx_unzip_name: nginx-1.23.3
  tasks:
    #- name: Install complie dependencies
    #  dnf: name={{ item }} state=installed
    #  loop:
    #    - gcc
    #    - gcc-c++
    #    - openssl-devel
    #    - openssl
    #    - zlib
    #    - zlib-devel
    #    - pcre
    #    - pcre-devel
    - name: Install compile dependencies
      dnf: name=gcc,gcc-c++,openssl-devel,openssl,zlib,zlib-devel,pcre,pcre-devel,vim,wget state=installed
    - name: Unarchive nginx package
      unarchive:
        src: "{{ src_nginx }}"
        dest: "{{ nginx_unzip_dir }}"
    - name: config and compile nginx
      shell: |
```

```
useradd -s /sbin/nologin nginx
cd {{ nginx_unzip_dir }}
cd {{ nginx_unzip_name }}
./configure --user=nginx --group=nginx --prefix={{ nginx_install_dir }} --
with-http_stub_status_module --with-http_ssl_module
make && make install
- name: start nginx
  shell: /usr/local/nginx/sbin/nginx
```

接下来运行这个剧本。

```
[root@ansible-controller ~]# ansible-playbook nginx_source_code_deploy.yaml
```

最后就是进行测试：

```
# 在被控端
[root@ansible-node1 ~]# curl 127.0.0.1
```

案例9：远程批量二进制安装tomcat服务

首先在主控端下载好二进制安装文件：

```
[root@ansible-controller ~]# wget https://download.oracle.com/java/21/latest/jdk-
21_linux-x64_bin.tar.gz
[root@ansible-controller ~]# wget https://dlcdn.apache.org/tomcat/tomcat-
11/v11.0.2/bin/apache-tomcat-11.0.2.tar.gz
```

编写剧本文件：

```
# tomcat_deploy.yaml
- hosts: server
  remote_user: root
  gather_facts: no
  vars:
    src_jdk: /root/jdk-21_linux-x64_bin.tar.gz
    jdk_install_dir: /usr/local/
    jdk_unzip_name: jdk21
    src_tomcat: /root/apache-tomcat-11.0.2.tar.gz
    tomcat_install_dir: /usr/local/
    tomcat_unzip_name: tomcat
  tasks:
    - name: Unarchive jdk package
      unarchive:
        src: "{{ src_jdk }}"
        dest: "{{ jdk_install_dir }}"
    - name: config jdk
      shell: |
        echo "export JAVA_HOME=/usr/local/{{ jdk_install_dir }}" >> ~/.bashrc
        echo "export PATH=$JAVA_HOME/bin:$PATH" >> ~/.bashrc
        source ~/.bashrc
    - name: Unarchive tomcat package
      unarchive:
        src: "{{ src_tomcat }}"
        dest: "{{ tomcat_install_dir }}"
    - name: config tomcat
```

```
shell: |
  echo "export TOMCAT_HOME/usr/local/{{ tomcat_unzip_name }}" >> ~/.bashrc
  echo "export PATH=$PATH:$TOMCAT_HOME/bin" >> ~/.bashrc
  source ~/.bashrc
- name: start tomcat
  shell: nohup "{{tomcat_install_dir}}/{{tomcat_unzip_name}}/bin/startup.sh &"
```

执行这个剧本文件：

```
[root@ansible-controller ~]# ansible-playbook tomcat_deploy.yaml
```

最后测试：

```
[root@ansible-node1 ~]# curl 127.0.0.1
```

案例10：远程批量二进制安装mysql8.4.3服务

1) 配置文件

```
ll /ansible
my.cnf
mysqld.service
mysql.yaml
```

2) my.cnf 文件内容

```
root@bole:/ansible# cat my.cnf
[mysql]
#设置mysql客户端默认字符集
default-character-set=utf8mb4
[mysqld]
skip-name-resolve
user=mysql
ngram_token_size=2
server-id=1
default_password_lifetime=0
port=3306
#设置安装目录
basedir=/app/mysql
#数据存放目录
datadir=/app/mysql/data
log-error=/app/mysql/logs/err.log
#允许最大连接数
max_connections=1000
#服务端默认使用的字符集
character-set-server=utf8mb4
#创捷新表时默认的储存引擎
default-storage-engine=INNODB
#忘记密码时使用
#skip-grant-tables
#不区分大小写
lower_case_table_names=1
#认证方式
default_authentication_plugin=mysql_native_password
max_allowed_packet=500M
```

```
sql_mode=NO_ENGINE_SUBSTITUTION,STRICT_TRANS_TABLES
wait_timeout=28800
interactive_timeout=28800
max_connect_errors=100
max_user_connections=0
#日志文件大小
max_binlog_size=100M
```

3) mysqld.service 文件内容

```
[Unit]
Description=MySQL Server
After=network.target
After=syslog.target

[Service]
Type=notify
User=mysql
Group=mysql
TimeoutSec=0
ExecStart=/app/mysql/bin/mysqld --defaults-file=/app/mysql/conf/my.cnf $MYSQLD_OPTS
LimitNOFILE=10000
Restart=on-failure
RestartPreventExitStatus=1
Environment=MYSQLD_PARENT_PID=1
PrivateTmp=false

[Install]
WantedBy=multi-user.target
```

4) mysql.yaml 文件内容

```
- name: Install MySQL
  hosts: server
  gather_facts: no
#创建用户组
  tasks:
    - name: Create MySQL group
      group:
        name: mysql
        system: yes
        gid: 306
    - name: Create MySQL user
      user:
        name: mysql
        shell: /sbin/nologin
        system: yes
        group: mysql
        uid: 306
        home: /data/mysql
        create_home: no
    #将shell的标准输出赋予 id_output
    - name: Check creation of user and group
      shell: id mysql
      register: id_output
#将标准输出打印，加stdout即使捕获register模块的标准输出
```

```
- name: Print output of 'id mysql'
  debug:
    var: id_output.stdout
#创建app目录
- name: Create directory for MySQL installation
  file:
    path: /app
    state: directory
    mode: 0755
#下载mysql的tar包并将其解压
- name: Unzip MySQL package
  block:
    - name: Download MySQL package
      get_url:
        url: https://cdn.mysql.com/Downloads/MySQL-8.4/mysql-8.4.3-linux-glibc2.28-x86_64.tar.xz
      dest: /app

    - name: Unzip MySQL package to temporary directory
      unarchive:
        src: /app/mysql-8.4.3-linux-glibc2.28-x86_64.tar.xz
        dest: /app
        copy: no

    - name: Tar.xz Mysql
      unarchive:
        src: /app/mysql-8.4.3-linux-glibc2.28-x86_64.tar.xz
        dest: /app
        copy: no

    - name: Move MySQL directory to final location
      shell: mv /app/mysql-8.4.3-linux-glibc2.28-x86_64 /app/mysql
      args:
        creates: /app/mysql

    - name: Set ownership and permissions for MySQL directory
      file:
        path: /app/mysql
        state: directory
        owner: mysql
        group: mysql
        mode: 0755
#创建所需的目录
- name: Create MySQL directories
  file:
    path: "{{ item }}"
    state: directory
    owner: mysql
    group: mysql
    mode: 0755
  with_items:
    - /app/mysql
    - /app/mysql/logs
    - /app/mysql/conf
    - /app/mysql/data
#安装依赖
- name: Install dependencies
  apt:
```

```

name:
  - libaio1
  - libaio-dev
  - libncurses5-dev
  update_cache: yes
#拷贝配置文件及service文件
- name: Copy MySQL configuration files
copy:
  src: /ansible/my.cnf
  dest: /app/mysql/conf
  owner: mysql
  group: mysql
  mode: 0644

- name: Copy Mysql.service
copy:
  src: /ansible/mysqld.service
  dest: /usr/lib/systemd/system
  owner: root
  group: root
  mode: 0644
#安装mysql并启动
- name: Initialize MySQL
  shell: /app/mysql/bin/mysqld --defaults-file=/app/mysql/conf/my.cnf --initialize
  environment:
    MYSQL_ROOT_PASSWORD: 123456

- name: Start MySQL service
  systemd:
    name: mysql
    state: started
    enabled: yes

```

5) 执行playbook

```
[root@ansible-controller ~]# ansible-playbook mysql.yaml
```

6) 验证查看

```
# 在被控机上
systemctl status mysqld
```

7) 登录验证

```
# 在被控机上
cat /app/mysql/logs/err.log | grep 'temporary password'
# 登录
mysql -uroot -p ''
# 修改密码
alter user 'root'@'localhost' identified by '123456'
```

7. Ansible Role

官方文档: https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_reuse_roles.html

7.1 概述

角色允许您根据已知的文件结构自动加载相关的变量、文件、任务、处理程序和其他Ansible工件。将内容分组为角色后，您可以轻松地重用它们并与其他用户共享。

角色 roles 是 ansible 自 1.2 版本引入的新特性，用于层次性、结构化的组织 playbook，roles 能够根据层次型结构自动装载变量文件，tasks 任务以及 handlers 触发器等。简单来说，roles 就是通过分别将变量、文件、任务、模板及处理器放置于单独的目录中，并可以便捷的 include 他们的一种机制。其实就是将一个大的 playbook 文件，进行分类拆分，从而达到根据需要复用的目的。

Ansible Playbook Roles 是 Ansible 中用于组织和管理任务的一种结构。它允许将相关的任务、变量和文件组织成可重用的组件，以便在多个 Playbooks 中共享和调用。使用 Roles 可以更好地组织 Playbooks，使其更易于理解、维护和扩展。

7.2 优势

- 可重用性：将任务组织成 Role 后，可以在多个 Playbooks 中重复使用，避免了重复编写相同的任务。
- 可维护性：将不同功能的任务拆分成独立的 Role，可以更好 地组织和管理配置，使 Playbooks 更易于维护。
- 模块化：将不同功能拆分成 Role，可以实现模块化的配置，使得系统更易于扩展和修改。
- 共享和社区支持：可以将自己创建的 Role 共享给其他人使用，同时也可 以从社区获取开源的 Role。

7.3 目录结构

Ansible角色具有定义的目录结构，其中包含七个主要的标准目录。每个角色中必须至少包含其中一个目录。您可以省略角色不使用的任何目录。例如：

```
roles/
  common/          # 这个层次结构代表一个“角色”
    tasks/
      main.yml     # ← 如果需要，任务文件可以包含较小的文件
    handlers/
      main.yml     # ← 处理程序文件
    templates/
      ntp.conf.j2  # ← 与模板资源一起使用的文件
    files/
      bar.txt      # ← 与复制资源一起使用的文件
      foo.sh       # ← 与脚本资源一起使用的脚本文件
    vars/
      main.yml     # ← 与此角色关联的变量
  defaults/
    main.yml       # ← 此角色的默认较低优先级变量
  meta/
    main.yml       # ← 角色依赖
```

当然，我们可以使用 `ansible-galaxy` 命令来帮我们初始化一个 `role` 目录结构。

官方文档：https://docs.ansible.com/ansible/latest/galaxy/user_guide.html

例如：

```
cd roles
ansible-galaxy role init role-name
```

执行这个命令后，就会在 `role-name` 目录下自动帮我们生成角色的相关目录。

7.4 案例实战

案例1：使用角色部署安装httpd

1) 首先在主控端定义角色目录层次结构

```
[root@ansible-controller ~]# ls httpd/
default files handlers meta tasks templates vars
[root@ansible-controller ~]# tree httpd/
httpd/
├── default
├── files
│   └── httpd.conf
├── handlers
│   └── main.yml
├── meta
└── tasks
    ├── config_httpd.yml
    ├── index_httpd.yml
    └── install_httpd.yml
```

```
|   |- main.yml
|   |- service_httpd.yml
|- templates
|   |- index.html
└ vars
    └ main.yml
```

2) host/hosts

```
[server]
192.168.72.64
192.168.72.65
```

3) playbook-all-roles.yaml

```
- hosts: server
  remote_user: root
  gather_facts: no
  roles:
    - role: httpd
    #- role: nginx
```

4) handlers/main.yml

```
- name: restart
  systemd:
    name: httpd
    state: restarted
```

5) tasks/main.yml

```
- include: install_httpd.yml
- include: config_httpd.yml
- include: index_httpd.yml
- include: service_httpd.yml
```

6) tasks/install_httpd.yml

```
- name: install httpd
  dnf:
    name: httpd
    state: installed
```

7) tasks/config_httpd.yml

```
- name: config httpd
  copy:
    src: httpd.conf
    dest: /etc/httpd/conf/httpd.conf
  notify: restart
```

8) tasks/index_httpd.yml

```
- name: index httpd
  template:
    src: index.html
    dest: /var/www/html/index.html
```

9) tasks/service_httpd.yml

```
- name: service httpd
  systemd:
    name: httpd
    state: started
```

10) templates/index.html

```
<html>
  <head>
    <meta charset="UTF-8">
    <title>index.html</title>
  </head>
  <body>
    <h1>{{ index_content }}</h1>
  </body>
</html>
```

11) vars/main.yml

```
index_content: "首页"
```

12) 执行playbook

```
[root@ansible-controller ~]# ansible-playbook -i host/hosts playbook-all-roles.yaml
```

13) 验证查看

```
# 在被控主机上
[root@ansible-node1 ~]# curl 127.0.0.1
```

案例2：使用角色源码部署nginx

1) 首先在主控端定义角色目录层次结构

```
[root@ansible-controller ~]# ls nginx/
default files handlers meta tasks templates vars
[root@ansible-controller ~]# tree nginx/
nginx/
├── default
│   └── files
│       ├── index.html
│       ├── nginx-1.23.3.tar.gz
│       └── nginx.conf
│           └── www.test.com.conf
└── handlers
    └── main.yml
```

```
|- meta
|- tasks
|   |- compile_nginx.yml
|   |- config_nginx.yml
|   |- index_nginx.yml
|   |- install_nginx_compile.yml
|   |- install_nginx_dependency.yml
|   |- main.yml
|   |- nginx_package_transfer_and_unzip.yml
|   |- service_nginx.yml
|- templates
└─ vars
    └─ main.yml
```

2) host/hosts

```
[server]
192.168.72.64
192.168.72.65
[nfs]
192.168.72.66
```

3) playbook-all-roles.yaml

```
- hosts: server
  remote_user: root
  gather_facts: no
  roles:
    #- role: httpd
    - role: nginx
    #- role: tomcat
    #- role: mysql
```

4) files/nginx.conf

```
worker_processes 1;
events {
    worker_connections 1024;
}
http {
    include      mime.types;
    default_type application/octet.stream;
    include      /usr/local/nginx/conf/vhost/*.conf;
    sendfile    on;
    keepalive_timeout 65;
}
```

5) files/[www.test.com.conf](#)

```
server {
    listen 88;
    server_name www.test.com;
    location / {
        root html;
        index index.html index.htm
    }
}
```

6) handlers/main.yml

```
- name: restart
  shell: |
    pkill nginx
    /usr/local/nginx/sbin/nginx
```

7) vars/main.yml

```
src_nginx: "nginx-1.23.3.tar.gz"
nginx_unzip_dir: "/usr/local"
nginx_install_dir: "/usr/local/nginx"
nginx_unzip_name: "nginx-1.23.3"
src_nginx_conf: "nginx.conf"
src_nginx_include_conf: "www.test.com.conf"
dest_nginx_conf: "/usr/local/nginx/conf/nginx.conf"
dest_nginx_include_conf: "/usr/local/nginx/conf/vhost/www.test.com.conf"
src_nginx_index: "index.html"
dest_nginx_index: "/usr/local/nginx/html/"
```

8) tasks/main.yml

```
- include: install_nginx_compile.yml
- include: install_nginx_dependency.yml
- include: nginx_package_transfer_and_unzip.yml
- include: compile_nginx.yml
- include: config_nginx.yml
- include: index_nginx.yml
- include: service_nginx.yml
```

9) tasks/install_nginx_compile.yml

```
- name: install compile
  dnf:
    name:
      - gcc
      - gcc-c++
    state: installed
```

10) tasks/install_nginx_dependency.yml

```

- name: config httpd
dnf:
  name:
    - openssl
    - openssl-devel
    - zlib
    - zlib-devel
    - pcre
    - pcre-devel
  state: installed

```

11) tasks/nginx_package_transfer_and_unzip.yml

```

- name: index transfer
unarchive:
  src: "{{ src_nginx }}"
  dest: "{{ nginx_unzip_dir }}"

```

12) tasks/compile_nginx.yml

```

- name: create nginx user
  shell: |
    user_name='cat /etc/passwd|grep nginx|wc -l'
    [ ${user_name} -eq 0 ] && useradd -s /sbin/nologin nginx || break
- name: config nginx
  shell: |
    cd {{ nginx_unzip_dir }}
    cd {{ nginx_unzip_name }}
    ./configure --user=nginx --group=nginx --prefix={{ nginx_install_dir }} --with-
http_stub_status_module --with-http_ssl_module
    make
    make install

```

13) tasks/config_nginx.yml

```

- name: transfer and config nginx.conf
  copy:
    src: {{ src_nginx_conf }}
    dest: {{ dest_nginx_conf }}
  notify: restart

- name: create nginx include conf
  shell: |
    cd {{ nginx_install_dir }}
    [ ! -d conf/vhost ] && mkdir conf/vhost || break

- name: transfer and config nginx include conf
  copy:
    src: {{ src_nginx_include_conf }}
    dest: {{ dest_nginx_include_conf }}
  notify: restart

```

14) tasks/index_nginx.yml

```
- name: transfer nginx index.html
  copy:
    src: {{ src_nginx_index }}
    dest: {{ dest_nginx_index }}
```

15) tasks/service_nginx.yml

```
- name: start nginx
  shell: /usr/local/nginx/sbin/nginx
```

16) 执行playbook启动nginx

```
[root@ansible-controller ~]# ansible-playbook -i host/hosts playbook-all-roles.yaml
```

17) 验证查看

```
# 在被控主机上
[root@ansible-node1 ~]# curl 127.0.0.1
```

案例3：使用角色二进制部署tomcat

1) 首先在主控端定义角色目录层次结构

```
[root@ansible-controller ~]# ls tomcat/
default files handlers meta tasks templates vars
[root@ansible-controller ~]# tree tomcat/
tomcat/
├── default
├── files
│   ├── apache-tomcat-11.0.2.tar.gz
│   ├── index.html
│   ├── jdk-21
│   └── server.xml
├── handlers
│   └── main.yml
├── meta
└── tasks
    └── config_tomcat.yml
    ├── index_tomcat.yml
    ├── install_jdk.yml
    ├── install_tomcat.yml
    ├── main.yml
    └── service_tomcat.yml
└── templates
└── vars
    └── main.yml
```

2) host/hosts

```
[server]
192.168.72.64
192.168.72.65
```

3) playbook-all-roles.yaml

```
- hosts: server
  remote_user: root
  gather_facts: no
  roles:
    #- role: httpd
    #- role: nginx
    - role: tomcat
    #- role: mysql
```

4) files/index.html

```
tomcat test index
```

5) files/server.xml

```
<?xml version="1.0" encoding="utf-8"?>
<Server port="8005" shutdown="SHUTDOWN">

  <Connector port="8081" protocol="HTTP/1.1"
             connectionTimeout="20000"
             redirectPort="8443"/>
</Server>
```

6) handlers/main.yml

```
- name: restart
  shell: |
    ps -ef | grep tomcat | grep -v grep | awk '{print $2}' | xargs kill -9
    cd "{{ tomcat_install_dir }}"
    cd "{{ tomcat_unzip_name }}"/bin
    nohup ./startup.sh &
```

7) vars/main.yml

```
src_jdk: "jdk-21_linux-x64_bin.tar.gz"
jdk_install_dir: "/usr/local/"
jdk_unzip_name: "jdk-21"
src_tomcat: "apache-tomcat-11.0.2.tar.gz"
tomcat_install_dir: "/usr/local/"
tomcat_unzip_name: "tomcat-11.0.2"
src_tomcat_index: "index.html"
dest_tomcat_index: "/usr/local/tomcat-11.0.2/webapps/ROOT"
src_tomcat_config: "server.xml"
dest_tomcat_config: "/usr/local/tomcat-11.0.2/conf/server.xml"
```

8) tasks/main.yml

```
- include: install_jdk.yml
- include: install_tomcat.yml
- include: config_tomcat.yml
- include: index_tomcat.yml
- include: service_nginx.yml
```

9) tasks/install_jdk.yml

```
- name: unarchive jdk package
  unarchive:
    src: "{{ src_jdk }}"
    dest: "{{ jdk_install_dir }}"
- name: set jdk env
  shell: |
    echo "export JAVA_HOME=/usr/local/{{ jdk_unzip_name }}" >> ~/.bashrc
    echo "export PATH=$JAVA_HOME/bin:$PATH" >> ~/.bashrc
    source ~/.bashrc
```

10) tasks/install_tomcat.yml

```
- name: unarchive tomcat package
  unarchive:
    src: "{{ src_tomcat }}"
    dest: "{{ tomcat_install_dir }}"
```

11) tasks/config_tomcat.yml

```
- name: config tomcat
  copy:
    src: "{{ src_tomcat_config }}"
    dest: "{{ dest_tomcat_config }}"
  notify: restart
```

12) tasks/index_tomcat.yml

```
- name: transfer tomcat index.html
  copy:
    src: {{ src_tomcat_index }}
    dest: {{ dest_tomcat_index }}
```

13) tasks/service_nginx.yml

```
- name: start tomcat
  shell: |
    cd "{{ tomcat_install_dir }}"
    cd "{{ tomcat_unzip_name }}"/bin
    nohup ./start.sh &
```

14) 执行playbook启动nginx

```
[root@ansible-controller ~]# ansible-playbook -i host/hosts playbook-all-roles.yaml
```

15) 验证查看

```
# 在被控主机上  
[root@ansible-node1 ~]# curl 127.0.0.1:8081
```

案例4：使用角色二进制部署mysql8.4.3

1) 首先在主控端定义角色目录层次结构

```
[root@ansible-controller ~]# ls mysql/  
default files handlers meta tasks templates vars  
[root@ansible-controller ~]# tree mysql/  
mysql/  
├── default  
├── files  
│   ├── my.cnf  
│   ├── mysql-8.4.3-linux-glibc2.28-x86_64.tar.xz  
│   └── mysqld.service  
├── handlers  
│   └── main.yml  
├── meta  
└── tasks  
    ├── create_mysql_user_and_dir.yml  
    ├── init_mysql.yml  
    ├── install_msq_dependency.yml  
    ├── unzip_mv_mysql.yml  
    ├── main.yml  
    ├── service_msq.yml  
    ├── transfer_mysql_config.yml  
    ├── transfer_mysqld_service.yml  
    └── transfer_mysql.yml  
└── templates  
└── vars  
    └── main.yml
```

2) host/hosts

```
[server]  
192.168.72.64  
192.168.72.65
```

3) playbook-all-roles.yaml

```
- hosts: server  
  remote_user: root  
  gather_facts: no  
  roles:  
    #- role: httpd  
    #- role: nginx  
    #- role: tomcat  
    - role: mysql
```

4) files/my.cnf

```
[mysql]
#设置mysql客户端默认字符集
default-character-set=utf8mb4
[mysqld]
skip-name-resolve
user=mysql
ngram_token_size=2
server-id=1
default_password_lifetime=0
port=3306
#设置安装目录
basedir=/app/mysql
#数据存放目录
datadir=/app/mysql/data
log-error=/app/mysql/logs/err.log
#允许最大连接数
max_connections=1000
#服务端默认使用的字符集
character-set-server=utf8mb4
#创建新表时默认的储存引擎
default-storage-engine=INNODB
#忘记密码时使用
#skip-grant-tables
#不区分大小写
lower_case_table_names=1
#认证方式
default_authentication_plugin=mysql_native_password
max_allowed_packet=500M
sql_mode=NO_ENGINE_SUBSTITUTION,STRICT_TRANS_TABLES
wait_timeout=28800
interactive_timeout=28800
max_connect_errors=100
max_user_connections=0
#日志文件大小
max_binlog_size=100M
```

5) files/mysqld.service

```
[Unit]
Description=MySQL Server
After=network.target
After=syslog.target

[Service]
Type=notify
User=mysql
Group=mysql
TimeoutSec=0
ExecStart=/app/mysql/bin/mysqld --defaults-file=/app/mysql/conf/my.cnf $MYSQLD_OPTS
LimitNOFILE=10000
Restart=on-failure
RestartPreventExitStatus=1
Environment=MYSQLD_PARENT_PID=1
PrivateTmp=false

[Install]
WantedBy=multi-user.target
```

6) handlers/main.yml

```
- name: restart
  shell: |
    ps -ef | grep tomcat | grep -v grep | awk '{print $2}' | xargs kill -9
    cd "{{ tomcat_install_dir }}"
    cd "{{ tomcat_unzip_name }}"/bin
    nohup ./startup.sh &
```

7) vars/main.yml

```
src_mysql: "mysql-8.4.3-linux-glibc2.28-x86_64.tar.xz"
mysql_install_dir: "/app/mysql"
mysql_data_dir: "/app/mysql/data"
mysql_log_dir: "/app/mysql/log"
mysql_unzip_name: "mysql-8.4.3-linux-glibc2.28-x86_64"
config_mysql: "my.cnf"
service_mysql: "mysqld.service"
```

8) tasks/main.yml

```
- include: install_msq_dependency.yml
- include: transfer_mysql.yml
- include: unzip_mv_mysql.yml
- include: create_mysql_user_and_dir.yml
- include: transfer_mysql_config.yml
- include: init_mysql.yml
- include: transfer_mysqld_service.yml
- include: service_msq.yml
```

9) tasks/install_msq_dependency.yml

```
- name: install mysql dependency
  dnf:
    name:
      - libaio1
      - libaio-dev
      - libncurses5-dev
    state: installed
    update_cache: yes
```

10) tasks/transfer_mysql.yml

```
- name: transfer mysql package
  copy:
    src: "{{ src_mysql }}"
    dest: "/opt/"
```

11) tasks/unzip_mv_mysql.yml

```

- name: create data
  shell: |
    cd /
    [ ! -d app ] && mkdir app || break
- name: Unarchive mysql package
  shell: |
    cd /opt
    tar -zxf {{ src_mysql }}
    mv {{ mysql_unzip_name }} {{ mysql_instll_dir }}

```

12) tasks/reate_mysql_user_and_dir.yml

```

- name: create mysql user
  shell: |
    user_name='cat /etc/passwd|grep mysql|wc -l'
    [ ${user_name} -eq 0 ] && useradd -s /sbin/nologin mysql || break
- name: create mysql log data
  shell: |
    mkdir {{ mysql_data_dir }}
    mkdir {{ mysql_log_dir }}
    chown -R mysql:mysql {{ mysql_install_dir }}
    echo "export PATH=/app/mysql/bin/:$PATH" >> ~/.bashrc
    source ~/.bashrc

```

13) tasks/transfer_mysql_config.yml

```

- name: transfer my.cnf
  copy:
    src: {{ config_mysql }}
    dest: /etc/
  notify: restart

```

14) tasks/init_mysql.yml

```

- name: Initialize MySQL
  shell: /app/mysql/bin/mysqld --defaults-file=/app/mysql/conf/my.cnf --initialize
  shell: mysqld --initialize --user=mysql --basedir={{ mysql_install_dir }} --
  datadir={{ mysql_data_dir }}
  environment:
    MYSQL_ROOT_PASSWORD: 123456

```

15) tasks/transfer_mysqld_service.yml

```

- name: transfer mysqld.service
  copy:
    src: {{ service_mysql }}
    dest: /usr/lib/systemd/system
    owner: mysql
    group: mysql

```

16) tasks/service_msqql.yml

```
- name: start mysql
  systemd:
    name: mysqld
    state: started
    enabled: yes
```

17执行playbook启动nginx

```
[root@ansible-controller ~]# ansible-playbook -i host/hosts playbook-all-roles.yaml
```

18) 验证查看

```
# 在被控主机上
[root@ansible-node1 ~]# ps -ef | grep mysql

[root@ansible-node1 ~]# init_mysql_pass='cat /app/mysql/log/err.log | grep password
| awk '{print $NF}''
[root@ansible-node1 ~]# mysql -uroot -p '${init_mysql_pass}' -e "set
password='123456'"
[root@ansible-node1 ~]# alter user 'root'@'localhost' identified by '123456'
```