### ansible操作手册

**前言：**

1.蓝色字体表示命令行命令，正式执行时不要复制前面的#号，#号只是提示应该使用root权限操作

2.绿色字体表示注释，有时注释太多就不用绿色表示了

3.注意：本文档的所有操作请先在测环境进行实践，请不要直接在真实的服务器中操作！

**版权声明**：

本文档以开源的形式发布，所有条款如下：

（1）无担保：作者不保证文档内容的准确无误，亦不承担由于使用此文档所导致的任何后果

（2）自由使用：任何人可以出于任何目的而自由地 阅读/链接/打印/转载/引用/分发/再创作 此文档，无需任何附加条件

若您 阅读/链接/打印/转载/引用/分发/再创作 本文档，则说明接受以上2个条款。

作者：李茂福

邮箱：sysyear@163.com

更新日期：2024-05-16

**★第0章、安装ansible**

ansible是开源的自动化运维工具，基于python开发，可实现批量系统配置、程序部署及其他功能。

ansible默认使用ssh协议远程登录到目标主机，将任务脚本（python脚本）复制过去并执行以实现相应功能

**★ansible及ansible-core对应关系**

ansible在2.9.x及之前版本只有一个软件包，名为ansible-2.x.x之类的，包含了所有的ansible命令行工具及模块

在2.10.x及之后版本，分为了2个软件包，将基础命令行工具及常用的核心模块放到名为ansible-base的包里（后更名为ansible-core），其他附加的模块放到名为ansible的包里，且ansible附加包的版本更新较快，版本号也比ansible-core的大，对应版本如下表：

|  |  |  |
| --- | --- | --- |
| ansible版本 | 依赖的ansible-core版本 | 需要的python版本 |
| 2.8.x | 无 | >=3.5 或 >=2.7 |
| 2.9.x | 无 | >=3.5 或 >=2.7 |
| 2.10.7 | ansible-base-2.10.17 | >=3.5 或 >=2.7 |
| 3.4.0 | ansible-base-2.10.17 | >=3.5 或 >=2.7 |
| 4.10.0 | ansible-core-2.11.12 | >=3.5 或 >=2.7 |
| 5.10.0 | ansible-core-2.12.10 | >=3.8 |
| 6.7.0 | ansible\_core-2.13.12 | >=3.8 |
| 7.7.0 | ansible\_core-2.14.10 | >=3.9 |
| 8.4.0 | ansible\_core-2.15.4 | >=3.9 |
| 9.5.1 | ansible\_core-2.16.6 | >=3.10 |
|  |  |  |

**★使用yum安装ansible**

使用yum的方法安装ansible，有的yum仓库可能没有我们需要的ansible版本

# yum install epel-release #centos7的epel仓库里有ansible

# yum install ansible #安装ansible，可指定目标版本号

**★centos8使用yum安装ansible2.9**

centos8的epel里面ansible版本较高，需要python3.9及以上的py环境，若想继续使用ansible2.9，则可使用centos-release-ansible-29的yum仓库

# yum install centos-release-ansible-29.noarch #安装ansible-29的yum仓库

#centos-release-ansible-29.noarch软件包在CentOS-Linux-Extras.repo里面有

[centos-ansible-29]

name=CentOS Configmanagement SIG - ansible-29

mirrorlist=http://mirrorlist.centos.org/?release=$releasever&arch=$basearch&repo=configmanagement-ansible-29

gpgcheck=1

enabled=1

gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-SIG-ConfigManagement

# yum install ansible-*2.9.27-1*.el8.noarch #安装ansible，若缺少依赖则可使用安装光盘去安装依赖

#（先将安装光盘配置为本源yum源）

**★使用pip安装ansible（推荐方法）**

**首先创建一个python虚拟环境，然后在虚拟环境里用pip安装任意版本的ansible**

# yum install python3.9 -y #建议安装python>=3.9版本，才支持更高版本的ansible

#创建虚拟环境并激活

# python\_ver=python3.9

# ansible\_ver=6.7.0

# venv\_dir=~/${python\_ver}-ansible-${ansible\_ver}

# mkdir -p ${venv\_dir}

# ${python\_ver} -m venv ${venv\_dir}

# source ${venv\_dir}/bin/activate #激活python虚拟环境

# pip3 install -U pip #先更新pip软件

# pip3 install "ansible==${ansible\_ver}" #安装ansible时指定版本号

#如果一开始就不知道具体有哪些版本的ansible，可以指定一个大版本号范围，如下：

# pip3 install 'ansible>=6,<7'

# ansible --version #查看ansible版本

ansible [core 2.13.12]

config file = None

configured module search path = ['/root/.ansible/plugins/modules', '/usr/share/ansible/plugins/modules']

ansible python module location = /root/python3.9-ansible-6.7.0/lib64/python3.9/site-packages/ansible

ansible collection location = /root/.ansible/collections:/usr/share/ansible/collections

executable location = /root/python3.9-ansible-6.7.0/bin/ansible

python version = 3.9.16 (main, Dec 23 2022, 09:55:36) [GCC 8.5.0 20210514 (Red Hat 8.5.0-17)]

jinja version = 3.1.4

libyaml = True

如果退出当前系统的登录，之后重新登录的话，需要再次激活此python虚拟环境才有相应版本的ansible命令：

# venv\_dir=~/python3.9-ansible-6.7.0

# source ${venv\_dir}/bin/activate #激活python虚拟环境

# deactivate #退出当前虚拟环境

**★pip使用指定的pypi源（可选）**

# mkdir ~/.pip

# cat > ~/.pip/pip.conf <<EOF

[global]

index-url=http://10.99.1.220:7788/simple

trusted-host=10.99.1.220

[install]

trusted-host=10.99.1.220

EOF

**★第1章、ansible基础配置**

/etc/ansible/ansible.cfg 为全局配置文件，ansible默认会读取环境变量ANSIBLE\_CONFIG指定的配置文件，没有此环境变量再读取当前目录下的ansible.cfg配置，当前目录下没有则再去读取~/.ansible.cfg，也没有才会读取全局配置文件。

/etc/ansible/hosts 为全局的主机清单文件（库存主机清单文件，称为inventory），不同于/etc/hosts，库存主机清单指的是ansible的操作对象，可写域名也可写ip

**★ansible自动化运维相关概念：**

|  |  |
| --- | --- |
| 执行机 | 执行ansible命令的节点，装有ansible命令及配置有需要执行的脚本等文件 |
| 受管节点 | 也叫受管主机，被管理节点，是ansible自动化要操作的目标主机，ansible通过ssh登录此节点并执行相应脚本 |
| inventory | 库存主机清单，ansible自动化要操作的对象集合，文件里的主机可写域名也可写ip |
| playbook | playbook剧本，是ansible要操作的一系列任务的集合，以yaml的格式书写，从上自下执行 |

**★ansible命令行参数**

**-m**  #执行模块的名字，默认使用command模块

**-a**  #模块的参数

**-k**  #使用密码认证，默认使用秘钥认证

**-K**  #提示输入sudo密码，sudo表示提权操作

**-B**  #后台运行超时时间

**-f**  #并行任务数，默认是5

**-i**  #指定库存主机文件，默认为/etc/ansible/hosts

**-l**  #进一步限制所选主机/组模式 --limit=10.99.1.10 表示只对这个主机执行

**-M**  #要执行的模块的路径，默认为/usr/share/ansible/

**-T**  #指定SSH超时时间，默认10S

**-u**  #远程用户，默认是当前用户

# cat > ./inventory <<EOF

10.99.1.10

EOF

# ansible -i ./inventory *10.99.1.10* -m command -a 'hostname' -k # -m后面指定使用command模块，用于登录目标主机（也叫受管主机，被管理节点）后执行命令，-a为传给command模块的参数，即要执行的命令内容为hostname， -k表示使用密码登录目标主机，若不使用-k则默认使用rsa密钥登录目标主机

#如果在使用-k密码登录目标主机时，出现如下报错，说明系统没有安装sshpass软件包

10.99.1.10 | FAILED! => {

"msg": "to use the 'ssh' connection type with passwords, you must install the sshpass program"

# yum install sshpass -y #此软件包在centos-release-ansible-29.noarch的仓库里

**★不检查主机hostKey密钥**

# ansible testgrp2 -m command -a 'hostname' -k

#登录testgrp2所有主机执行命令'hostname'，若从未登录过这些主机，则会报错：

SSH password: #输入密码

10.99.1.42 | FAILED | rc=-1 >>

Using a SSH password instead of a key is not possible because Host Key checking is enabled and sshpass does not support this. Please add this host's fingerprint to your known\_hosts file to manage this host.

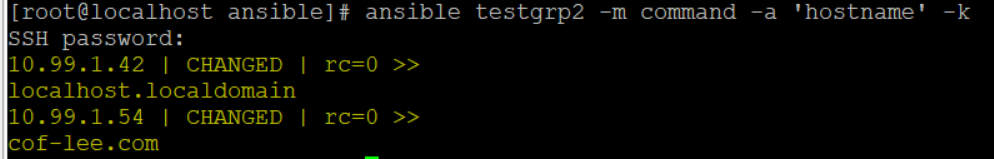
#解决方法为让ansible在远程登录主机时不检查host key，修改配置文件：

# vi /etc/ansible/ansible.cfg #在[defaults]下面添加一行配置

[defaults]

host\_key\_checking = false

#再次重试，输入密码即可



**★iventory主机清单文件的内容**

如果在ansible配置文件里没有特别指明使用哪个inventory文件，则默认读取/etc/ansible/hosts这个主机清单文件，inventory主机清单文件格式为ini配置风格，内容一行为一台主机，单台主机要求放在主机组之前，

主机可以写域名，也可以写ip，同一主机可单独出现，也可放在多个组中（多个组中可以有重复的主机）

主机组使用 [*组名*] 开头（主机组名用方括号括起来，下面跟一堆属于此组的主机）组名不可命名为all和ungrouped

# cat /etc/ansible/hosts #示例内容如下

mariadb.cof-lee.com #主机名

10.99.1.248 # ip地址

10.99.1.42

[testgrp1] #组名，下面有2台主机（属于这个组的主机）

10.99.1.51

10.99.1.52

[testgrp2]

10.99.1.54

10.99.1.42 #同一主机可单独出现，也可放在多个组中

[groupall:children] #支持嵌套，在一个大组下包含上面的小组，下面写的是组名

testgrp1

testgrp2

[pod\_rack01] #组名不可有-减号，可有\_下划线

10.99.1.[60:99] #10.99.1.60到10.99.1.99，支持使用正则规则将连续的若干个ip写在一行

#可以在组后面或者主机名称后添加以下内容指定ssh的账号密码等参数（单台主机的所有信息写同一行，空格隔开）

ansible\_connection=ssh ansible\_ssh\_port=22 ansible\_ssh\_user=lee ansible\_ssh\_pass=passwdxx ansible\_become\_pass=passwdxx

如果每台主机都要单独指定ssh远程登录相关信息，则每台主机后都要写上相关信息

如果所有主机的ssh登录信息相同，则可写在全局的vars变量里，如：

# cat inventory2

host1

host3

[group2]

server2

[group2:vars] #表示group2组的主机默认的ssh相关信息（all表示所有主机）每配置各写一行

ansible\_ssh\_port=4322

[all:vars] #表示所有主机默认的ssh相关信息（all表示所有主机）每配置各写一行

ansible\_ssh\_port=22

ansible\_ssh\_user=cof

ansible\_ssh\_pass=xxx

ansible\_become\_method=sudo

ansible\_become\_password=xxxx

# ansible all --list-hosts #列出inventory主机清单文件里的所有主机（不重复），默认读取/etc/ansible/hosts文件，可用-i指定inventory清单文件

# ansible-inventory -i /etc/ansible/hosts --graph #以树型结构显示主机及主机组

@all:

|--@groupall:

| |--@testgrp1:

| | |--10.99.1.51

| | |--10.99.1.52

| |--@testgrp2:

| | |--10.99.1.42

| | |--10.99.1.54

|--@pod\_rack01:

| |--10.99.1.60

| |--10.99.1.61

| |--10.99.1.62

............

★ansible要操作的目标主机（受管主机）必须在inventory清单文件里有定义，否则会提示以下报错：

[WARNING]: Could not match supplied host pattern, ignoring: serverxx

[WARNING]: No hosts matched, nothing to do

★命令书写顺序建议写为：ansible命令 [-i inventory文件] 要操作的主机/组 模块名及其他参数

★如果在ansible.cfg里指定了inventory文件，则执行ansible命令时可不用再指定inventory文件，除非需要用到的inventory文件不是ansible.cfg里指定的

# ansible 主机组名 --list-hosts #列出指定主机组里的主机

# ansible 主机组名 --list #同上，列出指定主机组里的主机

# ansible testgrp1 --list-hosts

hosts (2):

10.99.1.51

10.99.1.52

# ansible ungrouped --list #列出不在任何主机组里的主机

# ansible "10.99.1.\*2" --list #支持\*匹配

# ansible testgrp1,testgrp2 --list #支持显示2个组的主机（并集）可以不用引号

# ansible "testgrp1,&testgrp2" --list #支持显示2个组都有的主机（交集），要用引号引起来

# ansible 'testgrp1,!mariadb' --list #显示testgrp1主机（要去除mariadb，前面加!号表示排除此主机），有!号时只可用单引号引起来，主机名还支持其他正则匹配，这里不一一列出

**★动态主机清单文件**

If your Ansible inventory fluctuates over time, with hosts spinning up and shutting down in response to business demands, the static inventory solutions described in How to build your inventory will not serve your needs. You may need to track hosts from multiple sources: cloud providers, LDAP, Cobbler, and/or enterprise CMDB systems.

静态主机清单inventory文件里是我们手动写好了的主机信息，如主机名，主机组及远程登录相关参数

如果受管主机信息过多，后续有开通新的服务器或撤销/变更服务器信息，维护这个静态主机清单就比较麻烦

而ansible支持动态的方式去获取主机信息，比如指定的inventory文件为一个python脚本，它返回一个json信息，里面就是通过访问某api获取到的主机/主机组信息（以JSON的格式返回），可以从云服务提供商那里获取相应账号下的虚拟机，或者从LDAP/Cbbler/CMDB/其他信息系统获取主机信息。

动态主机清单文件示例：

（动态主机清单脚本开头必须指定python解释器，且文件有x权限才能被ansible正确识别 #!/usr/bin/python3）

（动态主机清单脚本文件名称不一定要有.py后缀，名称随便取，可以无后缀）

# cat > dynamic\_inventory.py <<EOF

#!/usr/bin/python3  
  
import sys  
import json  
  
# 定义一个名为result的字典，主机/主机组所有信息都在这个字典里，本动态主机清单脚本最终返回这个字典内容  
# ★这里只是个示例，真实的脚本是要去调相应的api模块访问具体的api获取需要的主机信息并做其他更进一步处理  
result = {}  
result['group1'] = {}  
result['group1']['hosts'] = ['mariadb'] # group1组下的主机，可有多个，用逗号隔开  
result['group1']['vars'] = {}  
result['group2'] = {}  
result['group2']['hosts'] = ['10.99.1.238'] # group2组下的主机，可有多个，用逗号隔开  
result['group2']['vars'] = {}  
result['group2']['vars']['ansible\_ssh\_port'] = [22] # group2主机的默认参数设置  
result['ungrouped'] = {}  
result['ungrouped']['hosts'] = ['10.99.1.237', 'host002']  
result['ungrouped']['vars'] = {}  
  
if len(sys.argv) == 2 and sys.argv[1] == '--list':  
 print(json.dumps(result))  
elif len(sys.argv) == 3 and sys.argv[1] == '--host':  
 print(json.dumps({}))  
else:  
 print("Requires an argument, please use --list or --host <host>")

EOF

# chmod +x dynamic\_inventory.py #需要添加可执行权限

# ansible -i dynamic\_inventory.py all --list #列出动态主机清单返回的主机信息

hosts (4):

10.99.1.237

host002

mariadb

10.99.1.238

# ansible-inventory -i dynamic\_inventory.py --graph

@all:

|--@group1:

| |--mariadb

|--@group2:

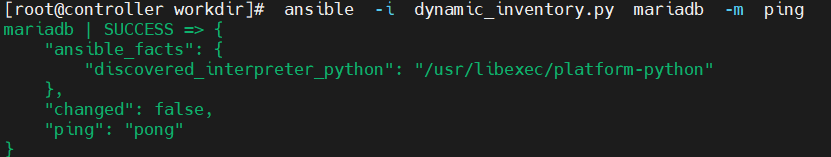
| |--10.99.1.238

|--@ungrouped:

| |--10.99.1.237

| |--host002

# ansible -i dynamic\_inventory.py mariadb -m ping



**★指定inventory主机清单文件**

# vi /etc/ansible/ansible.cfg #在[defaults]下面添加一行配置

[defaults]

inventory = /etc/ansible/hosts #指定受管主机清单文件

#或者在使用命令时直接用参数指定

# ansible -i /root/inventory all --list # -i指定inventory清单文件

# ansible -i ./inventory *mariadb* -m command -a 'ip addr' #登录mariadb这台主机，执行'ip addr'命令，默认使用当前执行此ansible命令的用户的密钥去登录目标主机

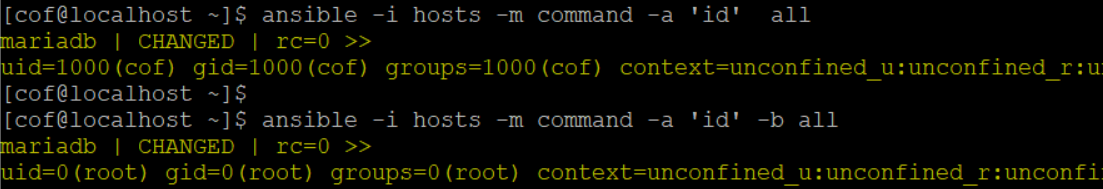
建议是一次批量任务要用到的所有配置文件及相应的脚本，都放某个目录下，inventory指定用此目录下的inventory文件，配置信息写到ansible.cfg配置文件里。

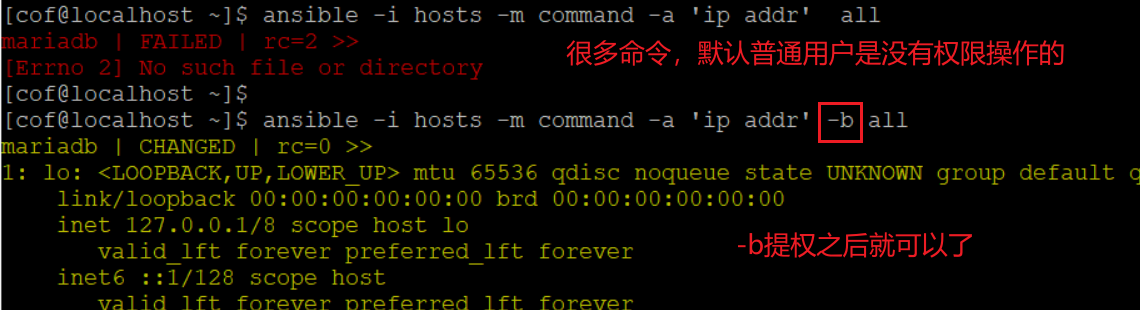
**★sudo提权**

[cof@ ~]$ ansible -i hosts all -m command -a 'id' #查看用户id

[cof@ ~]$ ansible -i hosts all -m command -a 'id' -b

#使用-b参数提权，使用sudo，前提是在目标主机上此用户（cof）已配置有sudo权限（且sudo免密）





如果不想每次都手动添加-b参数进行提权，可在配置文件里指定

# vi /etc/ansible/ansible.cfg #在[privilege\_escalation]下面添加4行

[privilege\_escalation]

become=True

become\_method=sudo

become\_user=root

become\_ask\_pass=False

★如果是普通用户，则默认没有权限修改/etc/ansible/ansible.cfg和/etc/ansible/hosts文件，建议在普通用户的家目录下创建自己的ansible配置文件；比如 ~/.ansible.cfg及里面指定inventory库存主机清单文件

★如果配置了 become\_ask\_pass=False ，则需要给普通用户配置sudo免密

**★ansible用户提权4种方式**

1、直接用root账号远程，使用密钥和密码都行

2、用普通用户远程，再sudo免密

3、用普通用户远程，再sudo不免密，需要指定ssh\_sudo\_pass=

4、用普通用户远程，再su切换为root，需要指定ssh\_su\_pass=

（在ansible 2.0版本之后，使用ansible\_become\_pass替换了之前的ansible\_sudo\_pass和ansible\_su\_pass）

5、自由组合设置

①ansible.cfg配置文件里只设置需要提权，不写具体的方法：

[privilege\_escalation]

become=True

#默认就是ansible\_become\_user=root ansible\_become\_method=sudo become\_ask\_pass=false

②具体提权方法在inventory里设置，不同主机设置不同的become方法（如果ansible.cfg里写了具体的提权方法，也没关系，inventory里的方法优先级高）

# cat inventory

server1 ansible\_become\_user=root ansible\_become\_method=su ansible\_become\_pass=rootpasswd

server2 ansible\_become\_user=root ansible\_become\_method=sudo ansible\_become\_pass=userpassxx

server3 ansible\_become\_user=root ansible\_become\_method=sudo #在运行命令时加 --ask-become-pass；或者配置了sudo免密就不用加--ask-become-pass

**★例：**

使用普通用户ssh远程登录，再su切换到root用户（非sudo）

userxx# vi ~/.ansible.cfg

[defaults]

host\_key\_checking=false

inventory=~/playbooks/inventory

[privilege\_escalation]

become=True

become\_method=su

become\_user=root

become\_ask\_pass=false

userxx# vi ~/playbooks/inventory

serverc ansible\_become\_pass=xxxx

**★指定受管主机的python解释器**

有时ansible在目标主机上找不到python解释器的路径，可能是由环境变量导致的，或者受管主机上真的没有把python解释器位置放到env环境变量里，这时需要我们手动在执行时指定受管主机的python解释器路径：

# ansible 10.99.1.49 -m ping

fatal: [10.99.1.49]: FAILED! => {"changed": false, "msg": "Python Module not found: firewalld and its python module are required for this module, version 0.2.11 or newer required (0.3.9 or newer for offline operations)"}

# ansible 10.99.1.49 -m ping -e 'ansible\_python\_interpreter=/usr/libexec/platform-python' #指定python路径

# vi inventory #或者把python解释器路径信息写到inventory里

10.99.1.49 ansible\_python\_interpreter=/usr/libexec/platform-python

# ansible all -m setup -a 'filter=discovered\_interpreter\_python' #查看所有主机的python解释器

10.99.1.81 | SUCCESS => {

"ansible\_facts": {

"discovered\_interpreter\_python": "/usr/bin/python"

},

"changed": false

}

10.99.1.82 | SUCCESS => {

"ansible\_facts": {

"discovered\_interpreter\_python": "/usr/bin/python3.9"

},

"changed": false

}

10.99.1.83 | SUCCESS => {

"ansible\_facts": {

"discovered\_interpreter\_python": "/usr/libexec/platform-python"

},

"changed": false

有时会提示：

[WARNING]: Platform linux on host 10.99.1.xx is using the discovered Python interpreter at /usr/bin/python3.9, but future installation of another

Python interpreter could change the meaning of that path. See https://docs.ansible.com/ansible-

core/2.11/reference\_appendices/interpreter\_discovery.html for more information.

配置文件添加以下一行参数可关闭：

interpreter\_python=auto\_legacy\_silent

**★ansible配置汇总**

**★ansible加载配置文件优先级如下：**

1、ANSIBLE\_CONFIG环境变量指定的配置文件，需要export ANSIBLE\_CONFIG=/pathTo/ansible.cfg

2、当前目录下的ansible.cfg文件

3、当前用户的家目录下的.ansible.cfg文件（注意：文件名前有个.），即 ~/.ansible.cfg

4、全局配置文件/etc/ansible/ansible.cfg

# vi ansible.cfg #添加以下配置，ansible配置行尾不可有注释，可换行注释，等号左右可有空格

|  |  |
| --- | --- |
| [defaults] |  |
| inventory = ./inventory | 指定库存主机清单文件 |
| host\_key\_checking = False | 不检查目标主机（受管节点）的host key是否在执行机的受信任主机列表里（执行机的~/.ssh/known\_hosts文件里） |
| log\_path=./ansible.log | 指定日志保存文件 |
| roles\_path=~/.ansible/roles/ | 指定role角色存储位置 |
| collections\_path=~/.collections | 指定collection集合存储位置，会在此目录下生成ansible\_collections子目录，安装的集合最终存放在~/.collections/ansible\_collections/ |
| remote\_user=cof | 指定远程登录目标主机的用户 |
| forks = 16 | 任务并行执行，playbook里的每个任务在执行时，同时下发给forks指定的数量个主机，默认5个 |
| gathering=explicit | gathering=explicit表示默认不收集facts信息，如果想收集facts则必须在playbook里指定 gather\_facts: **Ture**  gathering=smart表示默认收集facts，但facts已有的情况下不会再收集，即使用缓存的facts  gathering=implicit表示默认收集facts，要禁止收集则必须在playbook里指定 gather\_facts: **False** |
| deprecation\_warnings=False | 关闭deprecation\_warnings提示信息 |
| interpreter\_python=auto\_legacy\_silent | 关闭interpreter\_discovery提示信息 |
| executable=/bin/sh | 指定shell模块使用的命令解释器，默认是 /bin/sh |
| nocolor=0 | #输出内容带上彩色，0表示开启(有彩色)，1表示关闭 |
| timeout = 10 | SSH超时时间，默认10秒 |
| [privilege\_escalation] |  |
| become=True | 自动提权 |
| become\_method=sudo | 提权方式为sudo |
| become\_user=root |  |
| become\_ask\_pass=False | 未配置sudo免密时需要指定sudo密码，在inventory文件里对应主机后面写ansible\_become\_pass=*passxx* |
| [ssh\_connection] |  |
| ssh\_args = -C -o ControlMaster=auto -o ControlPersist=300s | 在首次连接后300s内无需再次认证登录，默认60s |
| pipelining = true | ansible在执行任务时会生成一个python脚本，再创建额外的ssh会话将脚本复制到目标主机去执行。可配置pipelining优化，在pipelining模式下ansible生成的python脚本不会额外创建ssh会话去复制到目标主机，而是通过管道传递给ssh会话（登录目标主机的会话），这样可节省资源和时间 |
| use\_tty = false | 不创建额外的ssh会话 |
| control\_path\_dir = ~/.ansible/cp | 默认是~/.ansible/cp |
| control\_path = %(directory)s/%%h-%%r | 默认是${control\_path\_dir}/nnn，这里指定为${control\_path\_dir}/%h-%r |

**★第2章、ansible模块及使用**

**官方文档：** https://docs.ansible.com/ansible/latest/collections/index.html

https://docs.ansible.com/ansible/latest/collections/index\_module.html

**★ansible常用模块**

# ansible-doc -l #列出所有模块

# ansible-doc -t module -l #同上，列出所有模块（不指定-t时默认就是指模块）

-t {become,cache,callback,cliconf,connection,httpapi,inventory,lookup,netconf,shell,vars,**module**,strategy,role,keyword}

# ansible-doc -l | wc -l #查看模块数量

3387

#ansible-2.9默认有3387个模块，2.8版本才2834个模块，到了8.4.0版本模块数量已经增长到了8265个

#ansible模块的本质就是python脚本，由对应的"模块名称.py"脚本去解析模块的参数并执行相应功能

ansible2.8及以前版本只有role的概念（role角色，详见后面第8章“role角色”）

从ansible-2.9开始，引入了collection的概念（模块及role的集合），因为模块越来越多，不同厂商及开发者开发的role及模块可能会同名，造成冲突，于是将role及模块进行分组（类似java/python的package包的概念，以包的管理模式来结构化管理各个role及模块），将相同类型的role及模块放到同一个collection集合里。但2.9版本的ansible仍然将自带的所有模块（3387个）杂合在一起，未分组到collection里，只是引入collection这个概念。

从ansible 2.10开始，安装包里自带的模块被分组到不同的collection里了，分组后的模块名要写FQCN名称（完全限定集合名称），类似 xx.yy.modulexx之类的名称，可能为了兼容某些旧的脚本，仍留了60多个模块未分组（使用原来的一级模块名就可引用此模块）

★从2.10开始，使用ansible-doc列出模块时，仍使用短名称的模块（未分组的）只有60多个了

ansible-2.10.7# ansible-doc --list | awk '{print$1}' | grep -v "\." | wc -l

68

2.10版本，用ansible-doc列出的仍使用短名称的模块如下：

add\_host apt apt\_key apt\_repository assemble assert async\_status blockinfile

command copy cron debconf debug dnf dpkg\_selections expect fail fetch file find

gather\_facts get\_url getent git group group\_by hostname

import\_playbook import\_role import\_tasks include include\_role include\_tasks include\_vars

iptables known\_hosts lineinfile meta package package\_facts pause ping pip

raw reboot replace rpm\_key script service service\_facts set\_fact set\_stats setup

shell slurp stat subversion systemd sysvinit tempfile template

unarchive uri user wait\_for wait\_for\_connection yum yum\_repository

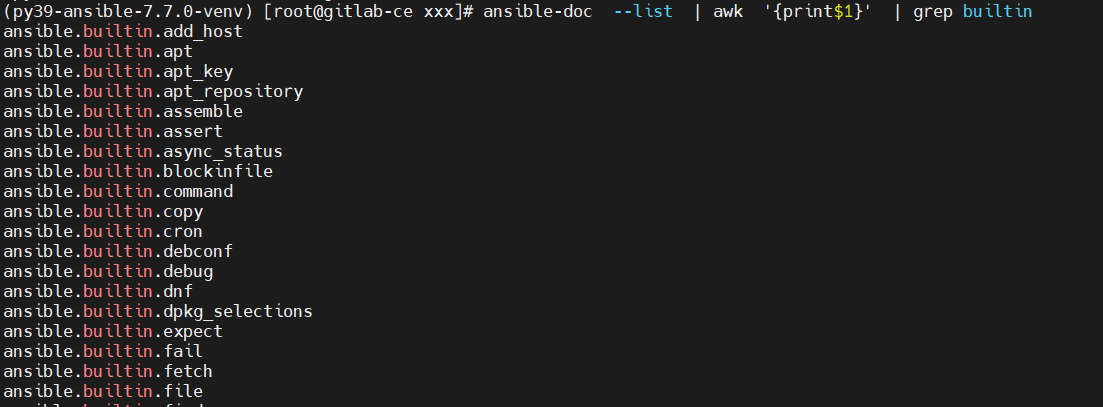
**说明：**

这60多个模块只是在ansible-doc里无FQCN完全限定名称，实际上已分组到ansible.builtin了，即2.10后的所有版本在使用这些模块时都可写为 ansible.builtin.模块名

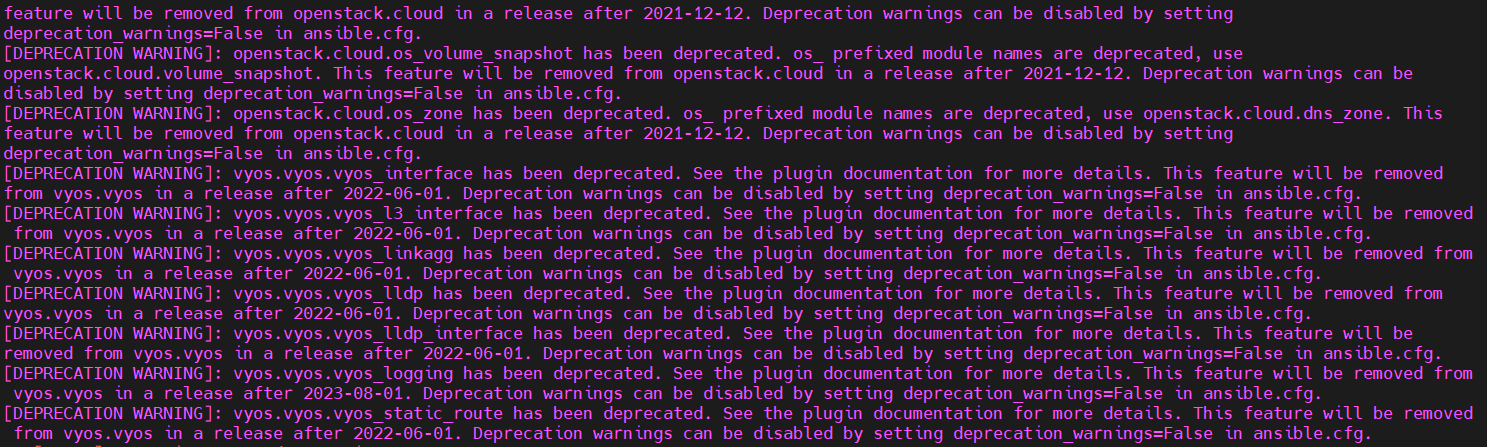
其他分组的模块名在新版本ansible>=2.10里建议只写FQCN名称

★到了7.x版本，之前的60多个未分组的模块，在使用ansible-doc查看时也终于显示为FQCN名称了，都带上了ansible.builtin这个前缀（2.x到6.x只是查看时不显示ansible.builtin前缀，使用时已经可以写FQCN名称了）

ansible-7.7.0# ansible-doc --list | awk '{print$1}' | grep builtin



**★如果执行ansible命令一直输出提示信息[DEPRECATION WARNING]: xxxxxx**



可在ansible.cfg里添加以下配置，关闭deprecation\_warnings提示信息

# vi ansible.cfg #在[defaults]下添加一行

[defaults]

deprecation\_warnings=False

**★ansible模块分组：**

|  |  |
| --- | --- |
| ansible<=2.9的模块名 | ansible>=2.10的模块名 |
| firewalld | ansible.posix.firewalld |
| selinux | ansible.posix.selinux |
| sysctl | ansible.posix.sysctl |
| parted | community.general.parted |
| lvg | community.general.lvg |
| filesystem | community.general.filesystem  dellemc.unity.filesystem #匹配到2个FQCN |
| group | ansible.builtin.group  awx.awx.group #匹配到多个FQCN  microsoft.ad.group |
| script | ansible.builtin.script |
| shell | ansible.builtin.shell |

上表说明，不同的ansible版本，对模块分组的情况不尽相同，ansible 2.10之后的几个大版本之间对模块的分组都有细微的差异，一定要确认自己当前环境使用的ansible版本及模块分组情况，根据实际情况书写模块名称。

★如果使用了高版本的ansible（>=2.10），而之前写的playbook剧本都是写的短模块名称，怎么办啊？

大家不要惊慌，几乎没有影响，旧的playbook剧本仍然可正常执行，因为写了短模块名称，ansible会匹配第一个找到的模块，目前大多数模块还没有短名称命名上的冲突，且新旧版本的同一个模块，其参数写法及功能没变化，不需要修改代码，如果模块名称有冲突，只需要改冲突的模块名称即可。

不过，随着ansible不断地开发升级，新的版本加入的各个厂商的模块越来越多，短模块名称冲突的可能性也随之提升，建议使用新版本ansible（>=2.10）时，书写的playbook剧本要使用FQCN模块名称。

# ansible-doc 模块名 #查看指定模块的帮助信息

# ansible-doc parted #查看parted模块的帮助信息，示例内容如下：

- name: Create a new ext4 primary partition #任务名称

community.general.parted: #模块名称（2.10及之后版本的写法）旧版本就写 parted:

device: /dev/sdb #此模块的参数及参数值

number: 1 #此模块的参数及参数值

state: present #此模块的参数及参数值

fs\_type: ext4 #此模块的参数及参数值

★先写一个inventory主机清单文件，里面写上要操作的目标主机，本例中写主机名，在/etc/hosts里写静态解析

# cat > inventory <<EOF

mariadb ansible\_ssh\_user=root ansible\_ssh\_pass=passwdxx #根据实际情况修改相关信息

EOF

# cat >> /etc/hosts <<EOF

10.99.1.xx mariadb #根据实际情况修改主机ip

EOF

★命令书写顺序建议写为：ansible命令 [-i inventory文件] 要操作的主机/组 模块名及其他参数

★如果在ansible.cfg里指定了inventory文件，则执行ansible命令时可不用再指定inventory文件，除非需要用到的inventory文件不是ansible.cfg里指定的

ansible-2.9# ansible -i ./inventory mariadb -m ping #使用ping模块，实际是使用ssh远程登录目标主机，以检测mariadb主机是否可连接（这个ansible的ping模块不是icmp协议，是要能ssh远程登录才回复pong）

mariadb | SUCCESS => {

"ansible\_facts": {

"discovered\_interpreter\_python": "/usr/bin/python"

},

"changed": false,

"ping": "pong"

}

ansible-2.9# ansible -i ./inventory all -m shell -a 'ip addr | grep inet'

#使用shell模块，默认的command模块不支持带特殊字符的命令，用shell可以，特殊字符有 | < > $ ; 等

ansible-2.9# ansible -i ./inventory all -m script -a '/home/cof/my.sh'

#使用script模块，可以把执行机本地的my.sh文件内容发送到目标主机去执行

ansible-2.9# ansible -i ./inventory all -m copy -a 'src=/home/cof/my.sh dest=/tmp/'

#使用copy模块，可以把执行机本地的文件scp复制到目标主机的dest指定的目录下

ansible-2.9# ansible -i ./inventory all -m file -a 'path=/tmp/test.h state=touch \

owner=cof group=root mode=0760'

#使用file模块，可以在目标主机上创建文件，path指定目标主机上的文件路径及名称

state=可指定为absent, directory, file, hard, link, touch；如果不写state=参数，则默认为修改已存在文件权限，如：

ansible-2.9# ansible -i ./inventory all -m file -a 'path=/tmp/test.h mode=0750' #修改权限为0750

ansible-2.9# ansible -i ./inventory all -m get\_url -a "url=http://10.99.1.248/Centos7-ext/docker.rpm \

dest=/tmp/ mode=0550 force=yes"

#使用get\_url模块，可以在目标主机上面下载指定的网站文件force=yes表示下载的文件内容和已存在的文件不一样时，替换原有文件，内容一样则不下载；不指定force=参数则表示存在原文件就不下载，不存在才下载

ansible-2.9# ansible -i ./inventory all -m cron -a "name='mycron job' minute='\*/5' job='ls /tmp'"

#使用cron模块，可以在目标主机上面创建任务计划crontab

ansible-2.9# ansible -i ./inventory all -m service -a "name=sshd state=restarted enabled=yes"

#使用service模块，可以在目标主机上面操作系统服务，7/8默认用systemd的服务

state=可选值有 started/stopped/restarted/reloaded

ansible-2.9# ansible -i ./inventory all -m user -a "name=fee state=present \

uid=1498 group=root home=/home/fee"

#使用user模块，可以在目标主机上面创建用户

ansible-2.9# ansible -i ./inventory all -m debug -a 'msg="xxxx eerwsfsd" '

#使用debug模块，可以打印字符串，msg为常量，var为变量，var=某变量

ansible-2.9# ansible -i ./inventory all -m yum -a "name=*bridge-utils* state=latest disable\_gpg\_check=yes"

#使用yum模块，可以在目标主机上面安装指定的软件，state=installed安装软件，state=latest表示安装最新版本，state=absent或者state=removed表示删除已安装的软件

ansible-2.9# ansible -i ./inventory all -m yum -a "name=bridge-utils state=latest \

disable\_gpg\_check=yes download\_only=true download\_dir=/tmp"

#仅下载软件包到指定目录下

ansible-2.9# ansible -i ./inventory all -m yum\_repository -a 'name="myrepo" \

description="my local repo" baseurl="file:///mnt/" gpgcheck=yes \

gpgkey="file://mnt/RPM-GPG-KEY-CentOS-7" enabled=yes'

#使用ym\_repository模块，可在目标主机上新增一个repo配置文件，文件名为myrepo.repo，内容如下：

# cat /etc/yum.repos.d/myrepo.repo

[myrepo] #同name参数值，yum\_repository的name为仓库ID

baseurl = file:///mnt/

enabled = 1

gpgcheck = 1

gpgkey = file://mnt/RPM-GPG-KEY-CentOS-7

name = my local repo #同description参数值，yum\_repository的description为仓库名称

ansible-2.9# ansible -i ./inventory all -m sysctl -a "name=net.ipv4.ip\_forward value=1 reload=yes"

#使用sysctl模块，可以在目标主机上面修改sysctl参数

★2.10及之后版本在模块名不冲突时也可写模块的短名称，新编写的ansible剧本或命令，建议按2.10之后的标准来写，新版本ansible模块的写法如下：

ansible>=2.10# ansible -i ./inventory mariadb -m ansible.builtin.ping

ansible>=2.10# ansible -i ./inventory mariadb -m ansible.builtin.shell -a 'ip addr | grep inet'

ansible>=2.10# ansible -i ./inventory all -m ansible.posix.sysctl -a \

"name=net.ipv4.ip\_forward value=1 reload=yes"

**★查看模块的完全限定名称：**

如果不确定当前ansible版本对某个模块的分组情况，可以使用ansible-doc命令查询

# ansible-doc --list | awk '{print$1}' | grep 短模块名



**★常用模块的新旧版本FQCN名称对照**

|  |  |  |
| --- | --- | --- |
| ansible<=2.9的模块名 | ansible>=2.10的模块名 |  |
| async\_status | ansible.builtin.async\_status |  |
| command | ansible.builtin.command |  |
| copy | ansible.builtin.copy |  |
| cron | ansible.builtin.cron |  |
| debug | ansible.builtin.debug |  |
| dnf | ansible.builtin.dnf |  |
| fail | ansible.builtin.fail |  |
| fetch | ansible.builtin.fetch |  |
| file | ansible.builtin.file |  |
| get\_url | ansible.builtin.get\_url |  |
| group | ansible.builtin.group |  |
| import\_playbook | ansible.builtin.import\_playbook |  |
| import\_tasks | ansible.builtin.import\_tasks |  |
| include | ansible.builtin.include | ansible-core>=2.16将弃用此模块 |
| include\_role | ansible.builtin.include\_role |  |
| include\_vars | ansible.builtin.include\_vars |  |
| lineinfile | ansible.builtin.lineinfile |  |
| ping | ansible.builtin.ping |  |
| script | ansible.builtin.script |  |
| service | ansible.builtin.service |  |
| setup | ansible.builtin.setup |  |
| shell | ansible.builtin.shell |  |
| stat | ansible.builtin.stat |  |
| template | ansible.builtin.template |  |
| uri | ansible.builtin.uri |  |
| user | ansible.builtin.user |  |
| wait\_for | ansible.builtin.wait\_for |  |
| wait\_for\_connection | ansible.builtin.wait\_for\_connection |  |
| yum | ansible.builtin.yum |  |
| yum\_repository | ansible.builtin.yum\_repository |  |
| firewalld | ansible.posix.firewalld |  |
| mount | ansible.posix.mount |  |
| filesystem | community.general.filesystem |  |
| lvg | community.general.lvg |  |
| lvol | community.general.lvol |  |
| parted | community.general.parted |  |

**★ansible-playbook**

将要执行的所有任务脚本写在一个配置文件里，自动按书写的任务脚本去执行完所有任务，这就是ansible-playbook的功能，任务清单文件一般使用yaml格式进行书写

# vi mytask.yml #适配ansible<=2.9版本，内容如下，yaml文件格式，以---开头，以...结束，...可省略

---

- name: playbook name xx #playbook名称可省，默认显示为hosts指定的主机名/组名

become: true #可选

become\_user: root #可选，提权相关操作除非有针对性，否则默认写在ansible.cfg配置文件里

gather\_facts: no #不获取facts信息，按实际情况来决定是否要开启

hosts: mariadb

tasks:

- name: task name yy #任务名称可省略，默认显示为执行的模块名

user: #使用user模块

name: tom #要创建的用户名

uid: 1433

state: present #创建用户

- name: #任务名称可省略，默认显示为执行的模块名

lineinfile: #使用lineinfile模块，可将一行内容追加到原有文件后面

path: /etc/hosts #要操作的文件名

line: '10.99.1.1 gateway.xxx.com' #要插入的内容

state: present

# vi mytask.yml #适配ansible>=2.10版本，内容如下，yaml文件格式，以---开头，以...结束，...可省略

---

- name: playbook name xx #playbook名称可省，默认显示为hosts指定的主机名/组名

become: true #可选

become\_user: root #可选，提权相关操作除非有针对性，否则默认写在ansible.cfg配置文件里

gather\_facts: no #不获取facts信息，按实际情况来决定是否要开启

hosts: mariadb

tasks:

- name: task name yy #任务名称可省略，默认显示为执行的模块名

ansible.builtin.user: #使用完全限定名称的模块

name: tom #要创建的用户名

uid: 1433

state: present #创建用户

- name: #任务名称可省略，默认显示为执行的模块名

ansible.builtin.lineinfile: #使用完全限定名称的模块

path: /etc/hosts #要操作的文件名

line: '10.99.1.1 gateway.xxx.com' #要插入的内容

state: present

#新旧版本的playbook剧本，就模块名称写法不一样，其他的都一样

★命令书写顺序建议写为：ansible命令 [-i inventory文件] playbook剧本文件 其他参数

★如果在ansible.cfg里指定了inventory文件，则执行ansible命令时可不用再指定inventory文件，除非需要用到的inventory文件不是ansible.cfg里指定的

# ansible-playbook -i ./inventory mytask.yml --syntax-check #检查mytask.yml文件语法是否正确

# ansible-playbook -i ./inventory mytask.yml -C # -C表示空运行（--check），不真正执行命令，只是测试；-i指定库存主机清单文件

# ansible-playbook -i ./inventory mytask.yml #执行playbook，可重复执行playbook，如果在目标主机已完成操作，则操作的命令不会重复执行，结果显示已ok，changed=0

#但是有些用shell模块执行的命令，可能会重复执行导致报错，需要手动修改任务脚本再进行第二次执行



★在执行playbook时也可额外指定其他参数

# ansible-playbook play-xx.yml --start-at-task="task name xx" #只执行指定的task

# ansible-playbook play-xx.yml --check #-C，空运行，只模拟测试，不实际执行任务

# ansible-playbook play-xx.yml --step #分步骤执行，每步骤会提示是否执行，需要输入y确认

# ansible-playbook play-xx.yml --forks=10 #指定执行任务的并行数量，同时给指定的数量台主机执行playbook里的任务（每个任务同时下发到指定数量台主机）

**★playbook里引入其他playbook**

# cat > play-A.yml <<EOF

---

- name: play A

hosts: mariadb

tasks:

- name: task A

ansible.builtin.debug:

msg: "debug msg of task A"

EOF

# cat > play-import-testx.yml <<EOF

---

- name: play test ww # playbook 1

hosts: mariadb

gather\_facts: no

tasks:

- name: task ww test

ansible.builtin.debug: # ansible<=2.9就只写debug

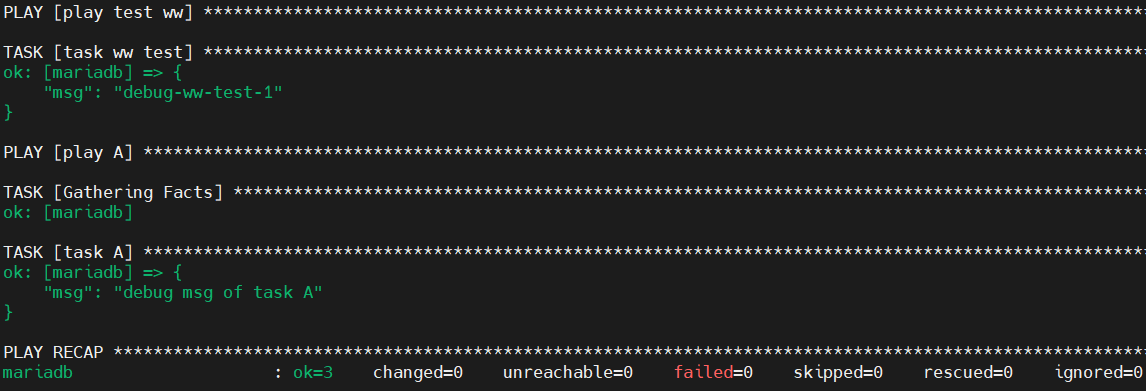
msg: "debug-ww-test-1"

- name: import tag playbook # playbook 2

ansible.builtin.import\_playbook: play-A.yml #引入其他playbook文件，从这里开始执行此playbook的任务，ansible<=2.9就只写import\_playbook

EOF

# ansible-playbook -i ./inventory play-import-testx.yml



**★文件操作相关模块**

**①file**

# cat > play-file.yml <<EOF

---

- name: play xx

hosts: mariadb

gather\_facts: no

tasks:

- name: task file xx

ansible.builtin.file: #在目标主机上创建/删除/属性设置某文件 # ansible<=2.9就只写file

path: /tmp/test-file1

owner: root

group: root

mode: 0640

state: touch #touch表示创建文件

EOF

# ansible-playbook -i inventory play-file.yml

**★创建目录及设置selinux文件类型**

ansible.builtin.file: # ansible<=2.9就只写file

path: /dirxx

state: directory #创建目录

group: cof

mode: '0775'

setype: httpd\_sys\_content\_t #设置selinux文件类型

**★创建链接**

ansible.builtin.file: # ansible<=2.9就只写file

src: /dirxx #源文件/目录

dest: /tmp/dirxx #链接名称

state: link #创建类型：软链接

**②copy**

**★新建文件并写入一行内容**

ansible.builtin.copy: # ansible<=2.9就只写copy

content: "hello\n" # copy模块默认不换行，需要手动添加换行符

dest: /dirxx/h.txt #如果文件已存在，则覆盖内容

setype: httpd\_sys\_content\_t

**★将执行机本地文件复制到受管主机上**

ansible.builtin.copy:

src: test.sh #**执行机**本地的文件

dest: /tmp/test.sh #复制到目标主机上

**★将受管主机本地文件复制到受管主机本地另一目录上**

ansible.builtin.copy:

src: /etc/hostname #**受管主机**本地的文件

dest: /tmp/test.sh #复制到目标主机上

remote\_src: yes #此参数说明src源文件/目录为**受管主机**的，而不是执行机的。若不带此参数则src表示执行机的源文件

**③synchronize**

使用synchronize模块时要求**执行机及目标主机上要安装有rsync命令**

# cat > play-synchronize.yml <<EOF

---

- name: play xx

hosts: mariadb

gather\_facts: no

tasks:

- name: task file xx

ansible.posix.synchronize: #使用rsync同步文件，ansible<=2.9就只写synchronize

src: /root/workdir/hosts #将执行机上的此文件，rsync同步到目标主机

dest: /tmp/ #目标主机的目录

EOF

# ansible-playbook -i hosts play-synchronize.yml

**④fetch**

# cat > play-fetch.yml <<EOF

---

- name:

hosts: mariadb

gather\_facts: no

tasks:

- name: task file xx

ansible.builtin.fetch: #将文件从目标主机上复制到执行机，以文件树的格式保存，拷贝完整路径

# ansible<=2.9就只写fetch

src: /root/anaconda-ks.cfg #src只能为一个文件

dest: /fetch #执行机上的一个主目录，文件树的根，此目录会自动创建

flat: no #允许覆盖

EOF

# ansible-playbook -i inventory play-fetch.yml

# tree /fetch

/fetch

└── mariadb #同ansible剧本中定义的主机名，此目录下才是拉取的文件

└── root

└── anaconda-ks.cfg

**⑤lineinfile**

# cat > play-lineinfile.yml <<EOF

---

- name: play xx

hosts: mariadb

gather\_facts: no

tasks:

- name: task file xx

ansible.builtin.lineinfile: #在目标主机上操作某文件（行操作）ansible<=2.9就只写lineinfile

path: /etc/hosts

line: "10.99.1.89 ansible.com" **#追加一行**

EOF

#其他操作

ansible.builtin.lineinfile: #在目标主机上操作某文件（行操作）

path: /etc/selinux/config

regexp: '^SELINUX=' #正则匹配

line: "SELINUX=permissive" **#将匹配到的行替换成这行**

ansible.builtin.lineinfile: #在目标主机上操作某文件（行操作）

path: /root/test

regexp: '^whhh' #匹配

state: absent **#将匹配到的行删除**

ansible.builtin.lineinfile: #在目标主机上操作某文件（行操作）

path: /root/test

insertafter: '^hwww' **#在匹配的行之后插入一行**

line: "insertxxxxafter"

ansible.builtin.lineinfile: #在目标主机上操作某文件（行操作）

path: /root/test

insertbefore: '^hwww'  **#在匹配的行之前插入一行**

line: "insertxxxxbefore"

**★★其他常用模块playbook写法**

ansible的模块多达几千个，本小节不可能一一列出，需要使用时可查看帮助文档

# ansible-doc 模块名 #查看指定模块的帮助信息

**★assert判断**

# cat > test\_assert.yml <<EOF

---

- name: play test assert

hosts: mariadb

vars:

var1: "abcd"

var2: "abcd"

tasks:

- name: test assert 1

ansible.builtin.assert:

that:

- var1 == var2

fail\_msg: "not equal"

success\_msg: "equal"

- name: test assert 2

ansible.builtin.assert:

that:

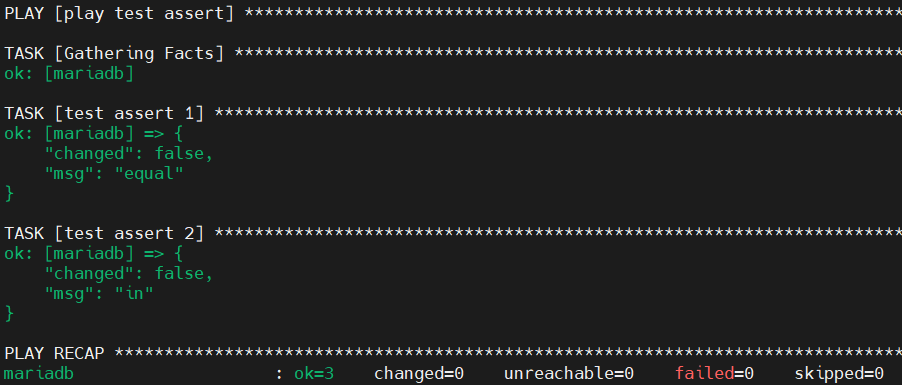
- var1 in var2

fail\_msg: "not in"

success\_msg: "in"

EOF

# ansible-playbook test\_assert.yml



**★挂载光盘**

ansible.posix.mount: #模块名称，在写入playbook tasks里时需要注意整段缩进

# ansible<=2.9就只写mount

path: /mnt #挂载点

src: /dev/sr0

fstype: iso9660

opts: ro,noauto #noauto表示不自动随开机挂载，手动mount -a也不会挂载

state: present #只写入/etc/fstab，未立即挂载；值为mounted则立即挂载

**★parted创建分区**

community.general.parted: #模块名称，在写入playbook tasks里时需要注意整段缩进

# ansible<=2.9就只写parted

device: /dev/sdb

number: 1

part\_start: 0% #只可用0%表示磁盘开始位置，不可用0

part\_end: 3GiB #不写part\_start则默认磁盘最前面

state: present #parted创建分区必须指明state: present，否则不会创建分区

community.general.parted:

device: /dev/sdb

number: 2

part\_start: 3GiB

part\_end: 4GiB #不写part\_end则默认到磁盘最后

state: present

**★lvm操作**

community.general.lvg: #创建vg并加入物理卷 # ansible<=2.9就只写lvg

vg: vg\_xx #vg可以是已存在的，也可以是未创建的

pvs: /dev/sdb1

pesize: 32 #PE大小，单位MB

state: present #可省略，默认就是present

community.general.lvg:

vg: vg\_xx

pvs: /dev/sdb2,/dev/sdb3

pesize: 32

community.general.lvol: #在已创建的vg里创建lv # ansible<=2.9就只写lvol

vg: vg\_xx

lv: lv\_xx

size: 5000 # lv大小，单位MB；可写为5g（但1g为1024MB）注意大小是否够用

state: present #可省略，默认就是present

**★创建文件系统**

community.general.filesystem: # ansible<=2.9就只写filesystem

fstype: ext4

**dev**: /dev/vg\_xx/lv\_xx

**★创建目录**

ansible.builtin.file: # ansible<=2.9就只写file

**path**: /lvpoint

state: directory

owner: cof

group: cof

mode: '0775'

setype: httpd\_sys\_content\_t

**★挂载文件系统**

ansible.posix.mount: # ansible<=2.9就只写mount

path: /lvpoint

**src**: /dev/vg\_xx/lv\_xx

fstype: ext4

state: present #只写入/etc/fstab，未立即挂载

ansible.posix.mount:

path: /lvpoint

src: /dev/vg\_xx/lv\_xx

fstype: ext4

state: mounted #写入/etc/fstab，并立即挂载

**★firewalld防火墙**

ansible.posix.firewalld: # ansible<=2.9就只写firewalld

service: http

permanent: yes #永久配置

state: enabled #允许放通

immediate: yes #立即生效

ansible.posix.firewalld:

zone: public # public的zone参数可省略

port: 8200-8210/tcp

permanent: yes

state: enabled

immediate: yes

ansible.posix.firewalld:

rich\_rule: rule family="ipv4" source address="10.1.2.0/24" service name="ssh" accept

permanent: yes #取消规则可以不用这个参数

state: disabled #取消规则，取消原来这条规则的设置

immediate: yes

**★shell**

ansible.builtin.shell: # ansible<=2.9就只写shell

cmd: bash test.sh

args:

chdir: /tmp

ansible.builtin.shell: chdir=/tmp bash test.sh #也可一行写完

**★cron**

ansible.builtin.cron: # ansible<=2.9就只写cron

name: "ping gw" #crontab文件里仅用作提示，必须用引号括起来

minute: "\*/2" #必须用引号括起来

user: cof

job: /usr/bin/ping -c 3 10.99.1.1 #crontab里的命令尽量使用绝对路径

执行后到目标主机上查看效果：

# crontab -l -u cof

#Ansible: ping gw

\*/2 \* \* \* \* ping -c 3 10.99.1.1

**★uri**

ansible.builtin.uri: #使用uri模块， # ansible<=2.9就只写uri

url: http://10.99.1.248/Centos7-ext/docker.rpm #要测试的url

status\_code: 200 #期望返回的结果状态

**★get\_url**

ansible.builtin.get\_url: # ansible<=2.9就只写get\_url

url: http://10.99.1.248/Centos7-ext/docker.rpm

dest: /tmp/docker-xxx.rpm #可指定下载后的文件名

mode: 0550

force: yes #如果本地已有此文件，可覆盖

**★user**

ansible.builtin.user: # ansible<=2.9就只写user

name: testuser1

uid: 1999

group: testgrp #主组为testgrp，而不是testuser1了，不创建testuser1的主组

home: /home/testuser1

shell: /bin/bash

password: "{{ testuser1passwd | password\_hash('sha512') }}"

ansible.builtin.user:

name: testuser1

uid: 1999

groups: testgrp #加到附加组testgrp，主组同用户名，创建testuser1的主组

**★第3章、ansible变量及facts信息采集**

**★ansible变量**

ansible变量有3种（变量命名不可和python内置函数相同，命名规范符合python编程语言的规范）

|  |  |
| --- | --- |
| 全局范围 | 从命令行设置的或者ansible配置文件里指定的变量，优先级最高 |
| playbook范围 | 在playbook和相关结构中设置的变量，优先级次高 |
| 主机范围 | 在主机组和主机上设置的变量（主机清单中）优先级最低，会被全局或playbook里的同名变量覆盖，主机后面的变量优先级比组后面的优先级高 |

**python3内置函数：**

abs() dict() help() min() setattr()

all() dir() hex() next() slice()

any() divmod() id() object() sorted()

ascii() enumerate() input() oct() staticmethod()

compile() globals() map() reversed() \_\_import\_\_()

complex() hasattr() max() round()

delattr() hash() memoryview() set()

bin() eval() int() open() str()

bool() exec() isinstance() ord() sum()

bytearray() ﬁlter() issubclass() pow() super()

bytes() ﬂoat() iter() print() tuple()

callable() format() len() property() type()

chr() frozenset() list() range() vars()

classmethod() getattr() locals() repr() zip()

**★变量的使用：**

|  |
| --- |
| ①在playbook脚本中单独一行使用变量时，必须用 "{{ }}" 括起来，在{{}}花括号外面必须有双引号或单引号，表示它是一个变量，要转换为相应的字符串（输出的字符串不含这个"{{ }}"） |
| ②在playbook脚本中如果变量前面有其他字符串，则只需要使用 {{ }} 括起来，花括号外面不需要引号，如果用了引号，则输出字符串时 这引号也属于字符串的一部分一同输出 |
| ③在playbook脚本中如果变量前面没有其他字符串，但在变量后面有其他字符串，则变量花括号{{ }}前面必须有引号，}}花括号后面没有直接跟引号，对应的最后一个引号是在后面的字符串的末尾添加 |

**比如以下4种写法为正确的写法：**

ansible.builtin.debug: #模块名，ansible<=2.9就只写debug

msg: #模块参数

- "{{ varxx }}" #①变量单独为一行

- this is {{varxx}} #②变量前面有其他字符串，则不用引号了（变量前不能有:冒号，否则需要引号）

- "{{ varxx }} is this" #③变量后面有其他字符串，则引号包含所有变量及字符串

- "{{ varxx }} {{ varyy }}" #多个变量都在一对引号内

# cat > test\_vars.yml <<EOF

---

- name: playxx

hosts: mariadb

vars: #定义变量

varxx: varxxxxxxx

varyy: varyyyyyyy

tasks:

- name: taskxx

ansible.builtin.debug: # ansible<=2.9就只写debug

msg:

- "{{ varxx }}"

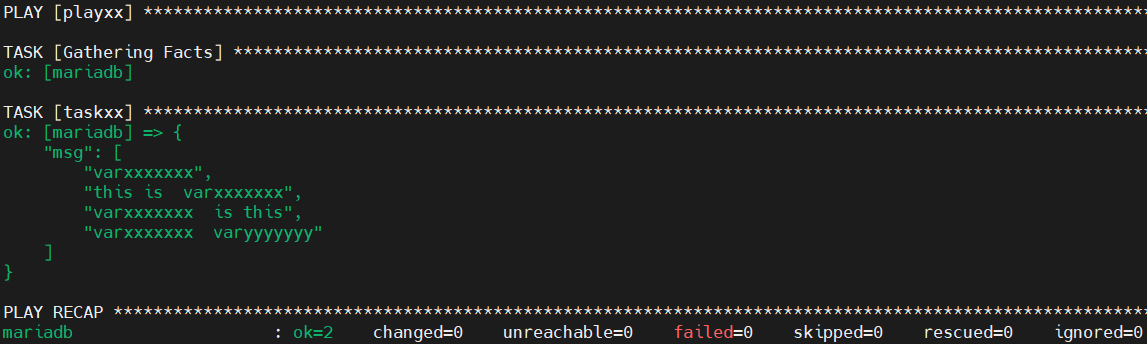
- this is {{varxx}} #变量前面有其他字符串，则不用引号了

- "{{ varxx }} is this" #变量后面有其他字符串，则引号包含所有变量及字符串

- "{{ varxx }} {{ varyy }}" #多个变量都在一对引号内

EOF

# ansible-playbook -i inventory test\_vars.yml #结果如下：



# cat > mytask3.yml <<EOF

---

- name: playbook name xxx

hosts: mariadb

vars: #在playbook清单文件中定义变量

username: xyz

userid: 1490

tasks:

- name: create user "{{username}}"

ansible.builtin.user: #使用user模块， # ansible<=2.9就只写user

name: "{{username}}"

uid: "{{userid}}"

EOF

★变量也可单独在一个文件中定义，在playbook清单中引用此变量文件：

# cat > mytask3-vars.yml <<EOF

username: xyz

userid: 1490

EOF

# cat > mytask3.yml <<EOF

---

- name: playbook name xxx

hosts: mariadb

vars\_files: #在playbook清单文件中引入变量文件

- mytask3-vars.yml

tasks:

- name: new user {{username}}

ansible.builtin.user: #使用user模块 # ansible<=2.9就只写user

name: "{{username}}" #使用变量（变量文件里的username变量）

uid: "{{userid}}" #使用变量（变量文件里的userid变量）

EOF

★playbook默认会去读取当前目录下的host\_vars和group\_vars子目录下的变量文件，主机找host\_vars子目录下同主机名的变量文件，主机组找group\_vars子目录下同主机组名的变量文件（没有.yml后缀）

★全局变量（优先级最高）在执行ansible-playbook命令时使用-e参数带上的变量，如：

# ansible-playbook -i inventory mytask3.yml -e "username=helo userid=1399"

★命令行手动引入变量文件

# ansible-playbook -i inventory mytask3.yml -e "@mytask3-vars.yml"

**★数组变量**

创建一个变量文件：

# cat > uservars.yml <<EOF

users:

user1:

username: user1

userid: 1421

userhome: /home/user1

user2:

username: user2

userid: 1422

userhome: /home/user2

EOF

# cat > mytask4.yml <<EOF

---

- name: playbook name xxx

hosts: mariadb

vars\_files: #在playbook清单文件中引入变量文件

- uservars.yml

tasks:

- name: new user {{users.user1.username}}

ansible.builtin.user: #使用user模块 # ansible<=2.9就只写user

name: "{{users.user1.username}}"

uid: "{{users.user1.userid}}"

home: "{{users.user1.userhome}}"

- name: new user {{users.user2.username}}

ansible.builtin.user: #使用user模块 # ansible<=2.9就只写user

name: "{{users.user2.username}}"

uid: "{{users.user2.userid}}"

home: "{{users.user2.userhome}}"

EOF

# ansible-playbook mytask4.yml

**★ansible魔法变量**

magic variable是ansible的内置变量，可直接使用，常用的魔法变量如下：

{{ hostvars }} #列出受管主机的信息（ansible\_facts的所有信息）

{{ group\_names }} #当前受管主机所属的组（inventory里的组）

{{ groups }} #列出主机清单文件中所有的组（含all这个组（all组包含所有主机））

{{ inventory\_hostname }} #列出主机清单文件中所配置的当前主机的主机名

# ansible -i inventory mariadb -m debug -a 'msg="{{ group\_names }}"'

mariadb | SUCCESS => {

"msg": [

"ungrouped"

]

}

# ansible -i inventory mariadb -m debug -a 'msg="{{ inventory\_hostname }}"'

mariadb | SUCCESS => {

"msg": "mariadb"

}

**★ansible\_facts主机信息采集**

每次执行ansible-playbook时会调用setup模块自动采集目标主机数据（包括主机名称，网络信息，磁盘信息等）

TASK [Gathering Facts] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [mariadb]

可先查看有哪些主机信息，再去使用其中需要用的信息

# ansible -i inventory mariadb -m ansible.builtin.setup #查看所有获取的信息，ansible<=2.9就只写setup

#查看指定的信息

# ansible -i inventory mariadb -m ansible.builtin.setup -a 'filter="ansible\_all\_ipv4\_addresses"'

# ansible -i inventory mariadb -m ansible.builtin.setup -a 'filter="ansible\_default\_ipv4"' #有默认网关的ip

# ansible -i inventory mariadb -m ansible.builtin.setup -a 'filter="ansible\_devices"'

# ansible -i inventory mariadb -m ansible.builtin.setup -a 'filter="ansible\_ens33"'

# ansible -i inventory mariadb -m ansible.builtin.setup -a 'filter="ansible\_hostname"' #主机名称

在debug模块输出具体主机信息时，分层级去获取，第一级变量名为"ansible\_facts"，其他所有信息都处于这级之下

第二级名称要去掉ansible\_字符串前缀（有的需要，有的不需要去除，暂未找到原因，需要多测试来掌握规律，ansible\_local不用去除前缀）



**★常用的ansible\_facts信息结点：**

"ansible\_facts"

"ansible\_architecture": "x86\_64"

"ansible\_distribution": "CentOS",

"ansible\_distribution\_major\_version": "7",

"ansible\_distribution\_release": "Core",

"ansible\_distribution\_version": "7.9",

"ansible\_nodename": "localhost.localdomain", #同/etc/hostname里的主机名称

"ansible\_hostname": "localhost", # hostname短名称

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

"ansible\_fqdn": "localhost.localdomain", #完全限定域名

"ansible\_local": #自定义的

"mariadb": { #主机名

"test": {

"var1": "value1",

"var2": "value2"

}

}

},

"ansible\_machine": "x86\_64",

"ansible\_machine\_id": "c8a0573cabe54a69ab8ab381a605569c",

"ansible\_memfree\_mb": 3242,

"ansible\_memtotal\_mb": 3770

"ansible\_swapfree\_mb": 0,

"ansible\_swaptotal\_mb": 0,

"ansible\_processor\_cores": 1,

"ansible\_processor\_count": 2,

"ansible\_processor\_threads\_per\_core": 1,

"ansible\_processor\_vcpus": 2,

"ansible\_product\_name": "VMware Virtual Platform",

"ansible\_product\_serial": "VMware-56 4d 53 ae 74 6b 8c ba-f4 c7 1f 71 c9 11 bd 0f",

"ansible\_selinux": {

"config\_mode": "enforcing",

"mode": "enforcing",

"policyvers": 31,

"status": "enabled",

"type": "targeted"

}

"ansible\_ens33"

"ipv4": {

"address": "10.99.1.43",

"broadcast": "10.99.1.255",

"netmask": "255.255.255.0",

"network": "10.99.1.0"

},

"mtu": 1500

"ansible\_default\_ipv4": { #有默认路由的接口ip

"address": "192.168.2.104",

"gateway": "192.168.2.1",

"ansible\_devices": {

"sda": {

"holders": [],

"host": "SCSI storage controller: Broadcom / LSI 53c1030 PCI-X Fusion-MPT Dual Ultra320 SCSI (rev 01)",

},

"partitions": {

"sda1": {

"size": "1.00 GB",

"uuid": "a18c7371-2024-440d-a706-6e9e7f4b71b2"

},

"sda2": {

"size": "18.00 GB",

"uuid": "bb79f057-e68f-4528-a944-76c490b0782c"

}

},

"sectorsize": "512",

"size": "20.00 GB",

},

"sr0": {

"host": "IDE interface: Intel Corporation 82371AB/EB/MB PIIX4 IDE (rev 01)",

"model": "VMware IDE CDR10",

"size": "1024.00 MB",

}

},

"ansible\_lvm": {

"lvs": {

"*lv\_test*": {

"size\_g": "2.94",

"vg": "vg\_test"

}

},

"pvs": {

"/dev/sdb1": {

"free\_g": "4.62",

"size\_g": "4.62",

"vg": "vg\_test"

}

},

"vgs": {

"*vg\_test*": {

"free\_g": "1.68",

"num\_lvs": "1",

"num\_pvs": "1",

"size\_g": "4.62"

}

}

},

# cat > get-facts.yml <<EOF

---

- name: get facts xx

hosts: mariadb

tasks:

- name: print macaddress

ansible.builtin.debug: # ansible<=2.9就只写debug

msg:

- the ip address is {{ansible\_facts.ens33.ipv4.address}}

- the macaddress is {{ansible\_facts.ens33.macaddress}}

#上面第二级网口名称ens33，在查看时为ansible\_ens33，在使用时要去掉ansible\_ ，使用以下写法则不用去前缀

- the ip address is {{ansible\_ens33.ipv4.address}}

- the macaddress is {{ansible\_ens33.macaddress}}

EOF

# ansible-playbook get-facts.yml

TASK [print macaddress] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [mariadb] => {

"msg": [

"the ip address is 10.99.1.48",

"the macaddress is 00:0c:29:9e:24:8f",

"the ip address is 10.99.1.48",

"the macaddress is 00:0c:29:9e:24:8f"

]

★默认所有playbook执行时会自动采集主机信息，比较耗时，如果不需要采集信息，可在playbook文件里关闭

# vi facts.yml

---

- name: get facts xx

hosts: mariadb

gather\_facts: no #关闭主机信息采集

tasks:

- name: print macaddress

...........

在playbook里关闭后，也可在任务里开启，gather\_facts就是调用的setup模块

# vi facts.yml

---

- name: get facts xx

hosts: mariadb

gather\_facts: no #关闭主机信息采集

tasks:

- name: setup facts

ansible.builtin.setup: #这里是一个任务，ansible<=2.9就只写setup

#（setup任务的内容就上面这一行，下面不是它的参数了，而是其他任务）

- name: print macaddress

............

**★在playbook里刷新facts信息**

#有时刚刚创建一个vg逻辑卷组，在它之后的任务如果要判断此vg是否在facts变量里有定义，默认是会失败了，因为一开始获取的facts信息里没有此vg，可以在创建vg的任务之后再创建一个新的任务去刷新facts信息

- name: refresh facts

ansible.builtin.setup: # ansible<=2.9就只写setup

filter: ansible\_lvm #新任务只获取ansible\_lvm这个facts信息

#接下来就可判断是否已创建了vg

**★主机本地fact文件**

额外的facts信息文件，需要在目标主机上创建/etc/ansible/facts.d目录，在此目录下创建一个fact文件，名称为 主机名.fact ，内容随便写，有层级即可（支持ini、json格式）

maridb# mkdir -p /etc/ansible/facts.d/

maridb# cat > /etc/ansible/facts.d/mariadb.fact <<EOF

[test]

var1=value1

var2=value2

EOF

然后在ansible执行机查看：

# ansible -i inventory mariadb -m ansible.builtin.setup -a 'filter="ansible\_local"'

mariadb | SUCCESS => {

"ansible\_facts": {

"ansible\_local": {

"mariadb": { #目标主机名

"test": {

"var1": "value1",

"var2": "value2"

}

}

},

**★playbook引入其他vars/tasks文件**

**★include引入task**

旧版本可使用include模块引入其他任务清单文件，也可使用include\_tasks及import\_takss模块，

ansible-core>=2.16不再支持include，建议只使用include\_tasks及import\_tasks模块

# cat > taskxx.yml <<EOF

- name: new user taskxx

ansible.builtin.user: # ansible<=2.9就只写user

name: "{{username}}"

uid: "{{userid}}"

state: present

EOF

# cat > playxx.yml <<EOF

---

- name: my play xx

hosts: mariadb

tasks:

- name: include task from file

ansible.builtin.include\_tasks: taskxx.yml # ansible<=2.9就只写include\_tasks

vars:

username: userxx

userid: 1423

EOF

# ansible-playbook -i inventory playxx.yml

# cat > task6.yml <<EOF

- name: service {{servicestate}} {{servicename}} #任务名称不能以变量开头，前面一定要有普通字符

ansible.builtin.service: # ansible<=2.9就只写service

name: "{{servicename}}"

state: "{{servicestate}}"

EOF

# cat > play6.yml <<EOF

---

- name: my play xx

hosts: mariadb

tasks:

- name: include task from file

ansible.builtin.include: task6.yml # ansible<=2.9就只写include

vars:

servicename: sshd

servicestate: restarted

register: output

- name: debug include task

ansible.builtin.debug: # ansible<=2.9就只写debug

var: output

EOF

# ansible-playbook -i inventory play6.yml

**★第4章、ansible-vault机密管理**

有些配置清单文件不想让人看到里面的账号密码内容，可使用ansible-vault工具对配置文件进行加密解密操作

# ansible-vault create myvault-play.yml #创建加密文件，要求设置加密密码，操作方法同vi

New Vault password:

Confirm New Vault password:

#上面输入2次密码后，就进入vi界面，按下字母i进行编辑，最后保存并退出

---

- name: test vault

hosts: mariadb

tasks:

- name: test vault task

ansible.builtin.debug: # ansible<=2.9就只写debug

msg: host in "{{group\_names}}"

# ansible-vault view myvault-play.yml #查看内容，要求输入密码

# ansible-vault edit myvault-play.yml #编辑内容，要求输入密码

#加密现有配置清单文件，加密后原文件被加密，不会备份原明文文件

# ansible-vault encrypt facts.yml

#可用--output指定加密后的文件名，原文件还在

# ansible-vault encrypt facts.yml --output=facts-vault.yml

#解密

# ansible-vault decrypt facts-vault.yml

# ansible-vault decrypt facts-vault.yml --output=newfacts.yml

**#在执行playbook时手动输入vault文件的密码**

# ansible-playbook -i hosts myvault-play.yml --vault-id @prompt

# ansible-playbook -i hosts myvault-play.yml --ask-vault-pass

**也可将明文密码写入一个文件里，在执行playbook时指定解密密码所处的文件**

# echo "passwdxx" > myvault-play.yml.pass

# chmod 600 myvault-play.yml.pass

# ansible-playbook -i hosts myvault-play.yml --vault-id myvault-play.yml.pass

#创建加密文件时指定用vault-pass里的明文密码去加密

# ansible-vault create myvault-play2.yml --vault-password-file=myvault-play.yml.pass

★示例：批量创建用户（用户信息文件含密码为加密的）

# ansible-vault create userinfo.secret

newusers:

- name: user10

passwd: passwdxx

- name: user11

passwd: passwdxx1

# cat > newuser-play.yml <<EOF

---

- name: create new user

hosts: mariadb

vars\_files:

- userinfo.secret

tasks:

- name: create new user task

ansible.builtin.user: # ansible<=2.9就只写user

name: "{{ item.name }}"

password: "{{ item.passwd | password\_hash('sha512') }}"

with\_items: "{{ newusers }}"

EOF

# ansible-playbook -i hosts newuser-play.yml --vault-id @prompt

# ansible-playbook -i hosts newuser-play.yml --ask-vault-pass

# ansible-vault rekey vault.yml #更换加密密码（更新密钥）

Vault password: #输入原来的密码

New Vault password: #输入新密码

Confirm New Vault password: #输入新密码

Rekey successful

**★第5章ansible-playbook流程控制**

recursive adj. 递归的; 循环的;一层一层嵌套的

iteration n. 迭代，就是遍历，从0到N，一个周期

ergodic n. 遍历性； adj. 遍历性的

loop v. 使成环，环形运动； n.环，环形；无数个周期，死循环

circulate v. 循环

**★loop遍历赋值**

**①基本loop**

loop会遍历它下面的list或dict，并给一个全局变量item赋值

# cat > play-test-loop1.yml <<EOF

---

- name: add users

hosts: mariadb

tasks:

- name: add user

ansible.builtin.user: # ansible<=2.9就只写user

name: "{{ item }}"

state: present

groups: "wheel"

loop: #使用loop，给上面的item迭代赋值，一次loop赋一行

- usertest1

- usertest2

EOF

# ansible-playbook -i inventory play-test-loop1.yml

**②字典loop**

# cat > play-test-loop2.yml <<EOF

---

- name: add users

hosts: mariadb

tasks:

- name: add user

ansible.builtin.user: # ansible<=2.9就只写user

name: "{{ item.name }}"

state: present

uid: "{{ item.uid }}"

loop: #使用loop，给上面的item迭代赋值，一次loop赋一组

- name: usertest11

uid: 1451

- name: usertest12

uid: 1452

EOF

# ansible-playbook -i hosts play-test-loop2.yml

早期版本的ansible定义了一些用于循环的关键字，都以with\_开头

with\_items 同loop，但with\_items会将列表扁平化处理，将一行的list拆开赋值

with\_nested 嵌套循环

with\_list 同loop，不做扁平化处理

**③loop和with\_items区别**

loop一次读取一整个键值对或一整行list，但with\_items会拆开分别算一次循环赋值

cat > play-test-loop3.yml <<EOF

---

- name: debug add users

hosts: mariadb

gather\_facts: no

tasks:

- name: debug add user task

ansible.builtin.debug: # ansible<=2.9就只写debug

msg: "{{ item }}"

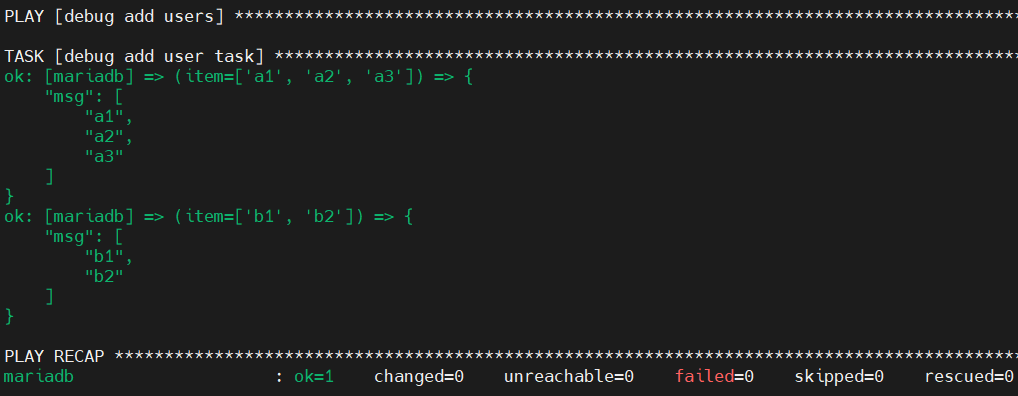
loop:

- [a1,a2,a3]

- [b1,b2]

EOF

# ansible-playbook play-test-loop3.yml #loop执行结果：



cat > play-test-with\_items1.yml <<EOF

---

- name: debug add users

hosts: mariadb

gather\_facts: no

tasks:

- name: debug add user task

ansible.builtin.debug: # ansible<=2.9就只写debug

msg: "{{ item }}"

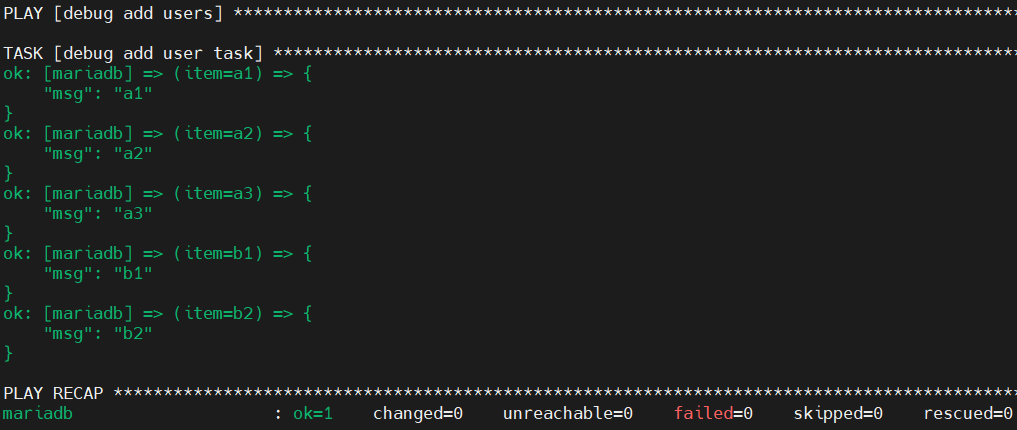
with\_items: #with\_items会拆开list/dick分别算一次循环赋值

- [a1,a2,a3]

- [b1,b2]

EOF

# ansible-playbook play-test-with\_items1.yml #with\_items执行结果：



**④with\_nested嵌套**

with\_nested表示组合式地取值（第一个list取第一个值后，取第2个list分别取各个值进行组合，再取第一个list的第二个值，取第2个list分别取各个值进行组合……）

cat > play-test-with\_nested1.yml <<EOF

- name: debug add users

hosts: mariadb

gather\_facts: no

tasks:

- name: debug add user task

ansible.builtin.debug: # ansible<=2.9就只写debug

msg: "{{ item }}"

with\_nested:

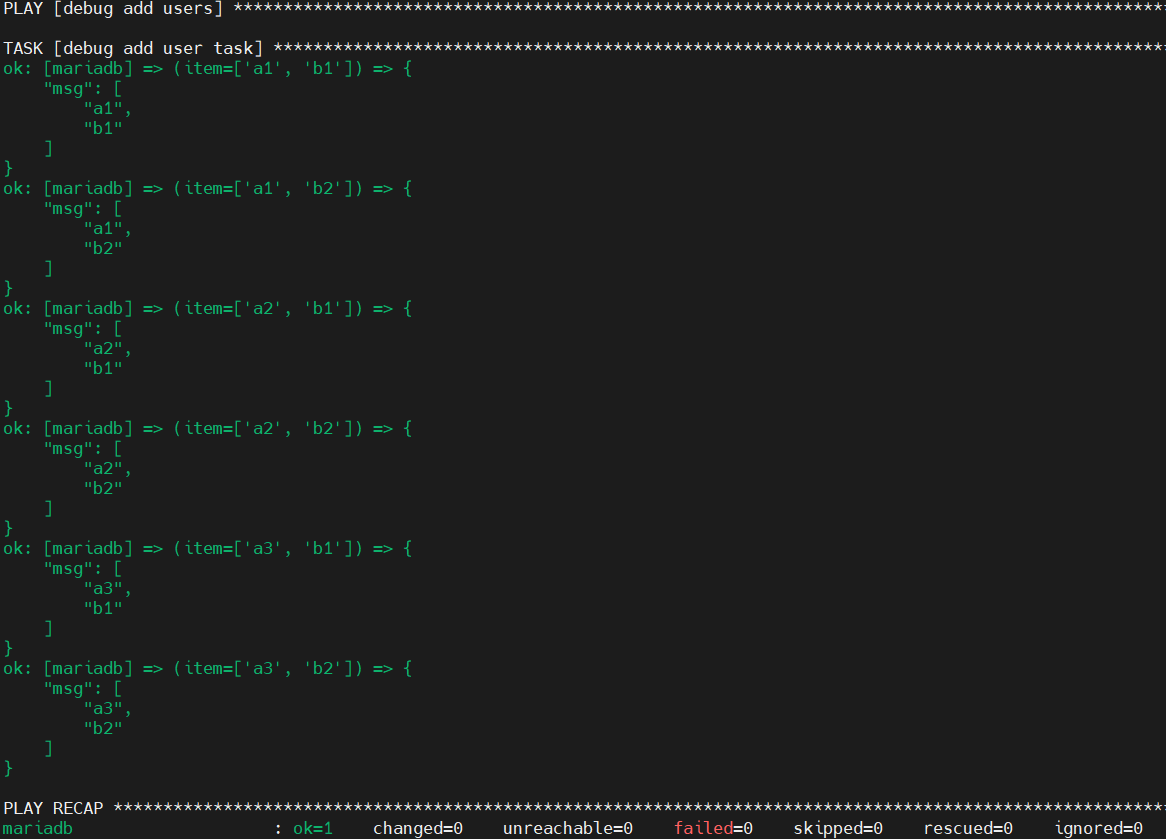
- [a1,a2,a3]

- [b1,b2]

EOF

# ansible-playbook play-test-with\_nested1.yml

# with\_nested执行结果：（a1与b1组合赋值给item，a1与b2组合，a2再......）



**⑤loop读取外部文件变量示例**

# cat > myvars.yml <<EOF

user1: [a1,b1,c1,d1]

user2: [a2,b2,c2,d2]

EOF

# cat > play-test-loop4.yml <<EOF

---

- name: debug add users

hosts: mariadb

gather\_facts: no

tasks:

- name: include vars

ansible.builtin.include\_vars: myvars.yml

- name: debug add user task

ansible.builtin.debug: # ansible<=2.9就只写debug

msg: "{{ item }}"

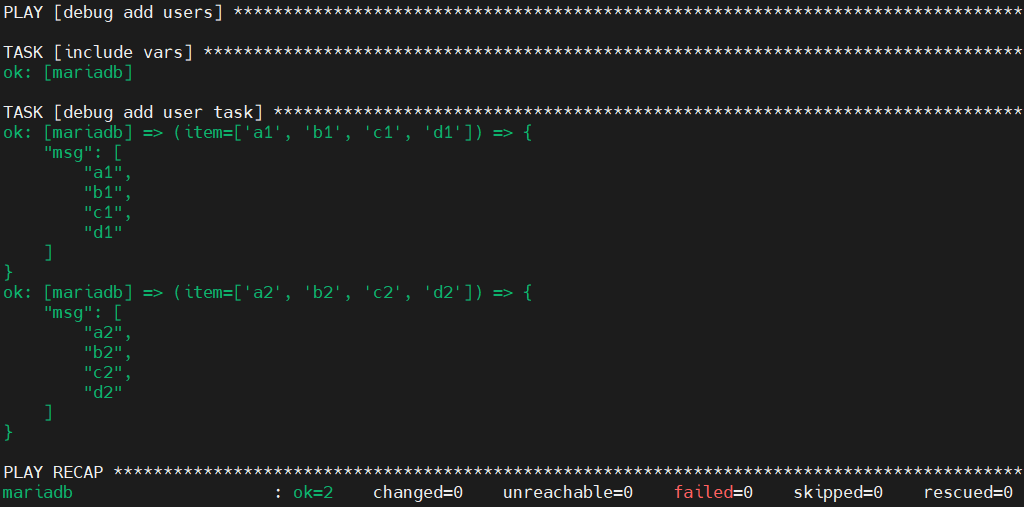
loop:

- "{{ user1 }}"

- "{{ user2 }}"

EOF

# ansible-playbook -i inventory play-test-loop4.yml



**★when条件判断**

when判断，满足某些条件就执行某些playbook任务

**①when条件判断true**

# cat > play-test-when1.yml <<EOF

---

- name: test when

hosts: mariadb

vars:

run\_task: true # run\_task的值直接为bool值的true

tasks:

- name: test when run my task

ansible.builtin.debug: # ansible<=2.9就只写debug

msg: running task now

when: run\_task #当when后面的条件为true时，才执行此task

EOF

# ansible-playbook -i inventory play-test-when1.yml

**②判断变量是否已定义**

# cat > play-test-when2.yml <<EOF

---

- name: test when

hosts: mariadb

vars:

run\_task: valuexx #run\_task的值为普通字符串

tasks:

- name: test when run my task

ansible.builtin.debug: # ansible<=2.9就只写debug

msg: running task now

when: run\_task is defined #当run\_task变量被定义时，才执行此task

EOF

# ansible-playbook play-test-when2.yml

**★when常用的判断语句**

is defined

is not defined

> >= < <=

== !=

not #对bool值取反

and or

*变量* in *列表中*

**★when后面的变量可以直接使用facts收集的变量，如ansible\_xxxx**

"ansible\_facts"下一级开始

# cat > play-test-when3.yml <<EOF

---

- name: test when

hosts: mariadb

vars:

surpportarch:

- x86\_64

- amd64

- x64

tasks:

- name: test when run my task

ansible.builtin.debug: # ansible<=2.9就只写debug

msg: running task now

when: ansible\_architecture in surpportarch

EOF

# ansible-playbook -i inventory play-test-when3.yml

**★register返回任务运行结果及判断**

# cat > play-test-register1.yml <<EOF

---

- name: test register

hosts: mariadb

tasks:

- name: test run task1

ansible.builtin.shell: "cat /etc/resolv.conf" # ansible<=2.9就只写shell

**register**: ret\_msg #返回运行状态结果的字典

ignore\_errors: true

- name: debug ret\_msg succeeded

ansible.builtin.debug: # ansible<=2.9就只写debug

msg: task1 succeeded

when: ret\_msg is succeeded

- name: debug ret\_msg failed

ansible.builtin.debug:

msg: task1 failed

when: ret\_msg is failed

- name: debug ret\_msg skipped

ansible.builtin.debug:

msg: task1 skipped

when: ret\_msg is skipped

- name: debug ret\_msg changed

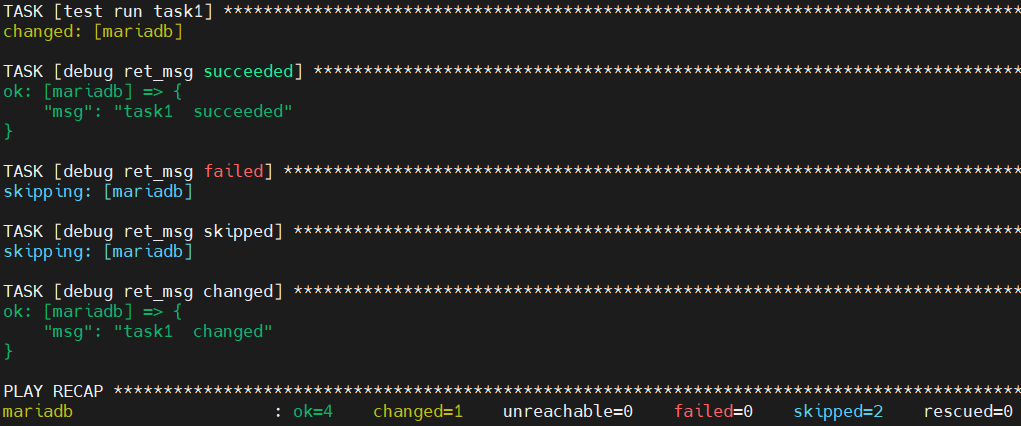
ansible.builtin.debug:

msg: task1 changed

when: ret\_msg is changed

EOF

# ansible-playbook -i inventory play-test-register1.yml



**★changed\_when**

# cat > play-test-changed\_when1.yml <<EOF

---

- name: play test

hosts: mariadb

tasks:

- name: task 1

ansible.builtin.shell: 'echo haha' # ansible<=2.9就只写shell

register: result #注册的是返回结果字典

changed\_when: '"haha" in result.stdout' #满足条件时就将本任务置为changed状态

notify:

- debug\_start

handlers:

- name: debug\_start

ansible.builtin.debug: # ansible<=2.9就只写debug

msg: "task1 changed"

EOF

# ansible-playbook play-test-changed\_when1.yml

**★failed\_when**

# cat > play-test2.yml <<EOF

---

- name: play test2

hosts: mariadb

tasks:

- name: task 1

ansible.builtin.shell: 'setenforce 0' # ansible<=2.9就只写shell

register: result

failed\_when: '"SELinux is disabled" in result.stdout' #满足条件时认为是changed状态

notify:

- debug\_start

handlers:

- name: debug\_start

ansible.builtin.debug: # ansible<=2.9就只写debug

msg: "debug changed"

EOF

# ansible-playbook -i hosts play-test2.yml

#当我们要设置setenforce 0关闭selinux时，如果系统本来已关闭了selinux，则执行此命令后会报错：

[root@localhost ~]# setenforce 0

setenforce: SELinux is disabled

但此报错本身表示selinux已关闭，符合我们的预期，算是成功配置了，但结果是error，此时可以使用failed\_when将此结果置为changed成功状态

**★ignore\_errors**

# cat > ignore\_errors.yml <<EOF

---

- name: play ig

hosts: test66

tasks:

- name: txx

ansible.builtin.shell: init 6 # ansible<=2.9就只写shell

async: 1

poll: 0

ignore\_errors: true #必须配合async和poll，忽略本任务的失败

- name: tdeb

ansible.builtin.debug: # ansible<=2.9就只写debug

msg: after init task

EOF

# ansible-playbook ignore\_errors.yml

#本来当txx任务重启系统时会导致失去连接，而判断任务失败，playbook中一旦某任务失败就会中断执行，不会再去执行之后的其他任务。配置ignore\_errors: true后会忽略当前任务的失败而继续执行之后的任务。

**★fail**

fail模块用来创建一个失败的任务，起到中断playbook的作用

# cat > fail.yml <<EOF

---

- name: pxx

hosts: test66

tasks:

- name: txx

ansible.builtin.fail: # ansible<=2.9就只写fail

msg: "sdd is not exist"

when: ansible\_devices.sdd is not defined #条件满足时就认为failed

EOF

# ansible-playbook fail.yml

**★block-rescue-always流程处理**

# cat > play-block-testx.yml <<EOF

---

- name: play test xx

hosts: mariadb

tasks:

- name: task ping test #可省略此行，若省略，则下面要写成 - block:

block: #首先执行这段内的任务（类似编程语言中的try测试模块）

- name: block test

ansible.builtin.uri: # ansible<=2.9就只写uri

url: http://10.99.1.248/Centos7-ext/docker.rpm

status\_code: 200

rescue: #仅当上面block测试任务失败时才执行rescue的任务（类似编程中的except）

- name: test when block error

ansible.builtin.debug: # ansible<=2.9就只写debug

msg: "block error"

always: #无论上面block测试任务失败与否都执行此任务（类似编程中的finally）

- name: always test

ansible.builtin.debug: # ansible<=2.9就只写debug

msg: "always test"

EOF

# ansible-playbook -i hosts play-block-testx.yml

**★notify触发handlers任务**

当普通任务成功完成后，触发notify，handlers里再做判断（异步回调机制），如果有相应的notify成功则进行handlers里的任务。handlers与tasks同级

# cat > play-test-handlers1.yml <<EOF

---

- name: play handlers

hosts: mariadb

force\_handlers: yes #当有notify触发时，强制执行后面对应的handlers，不会被中间失败的任务打断

tasks:

- name: copy conf file

ansible.builtin.copy: # ansible<=2.9就只写copy

src: /etc/ssh/sshd\_config-back #执行机本地的文件

dest: /etc/ssh/sshd\_config #复制到目标主机上

notify: #本任务成功后，触发一个notify信号，名称为restart\_sshd

- restart\_sshd

handlers:

- name: restart\_sshd #名称必须同notify的名称（必须是changed），执行本任务

ansible.builtin.service: # ansible<=2.9就只写service

name: sshd

state: restarted

EOF

# ansible-playbook play-test-handlers1.yml

**★给任务打tags**

# cat > play-tag-testx.yml <<EOF

---

- name: play test xx

hosts: mariadb

gather\_facts: no

tasks:

- name: task deb1 test

ansible.builtin.debug: # ansible<=2.9就只写debug

msg: "debug-test-1 deb1"

tags:

- deb1

- name: task deb2 test

ansible.builtin.debug:

msg: "xxx debug 2 test deb2"

tags:

- deb2

- name: task deb3 test

ansible.builtin.debug:

msg: "xxx debug 3 test deb3"

tags:

- deb3

- never

- name: task deb4 test

ansible.builtin.debug:

msg: "always debug 4 test always"

tags:

- always #执行playbook时无论是否指定调用此tag，都会执行此任务

EOF

# ansible-playbook play-tag-testx.yml #正常执行所有任务，带有tag名为never的任务默认不会被执行

# ansible-playbook play-tag-testx.yml --tags=deb1 #只执行tag值为deb1的任务（tag值为always的任务永远都要执行）

# ansible-playbook play-tag-testx.yml --tags=deb1,deb2 #只执行tag值为deb1及deb2的任务（tag值为always的任务永远都要执行）

# ansible-playbook play-tag-testx.yml --tags=deb3 #只执行tag值为deb3的任务，deb3任务带有名为never的tag，所以默认不会执行，如果想要执行它则需要显示指定它的tag（tag值为always的任务永远都要执行）

# ansible-playbook play-tag-testx.yml --list-tags #查看所有tag（never不会显示）

# ansible-playbook play-tag-testx.yml --skip-tags=deb2 #跳过此tag的任务（never不会执行）

# ansible-playbook play-tag-testx.yml --tags=tagged #只执行所有带tag的任务（never不会执行）

# ansible-playbook play-tag-testx.yml --tags=untagged #只执行不带tag的任务（always也执行）

**★第6章、jinja2模板**

jinja2可直接调用facts变量及playbook变量。用途一般是**将执行机上的配置模板**转为实际主机需要的配置文件并替换目标主机上的旧的配置文件

**jinja2分隔符：**

{{ 表达式/变量 }} {# 表达式，变量用{{}}花括号括起来，花括号外面没有引号 #}

{% if/for/endfor %} {# 流程控制用 {% %}括起来 #}

{# 注释信息 #}

**jinja2特殊变量：**

{{ loop.index }} {# 循环索引，从1开始 #}

{{ loop.index0 }} {# 循环索引，从0开始 #}

{{ loop.length }} {# 迭代对象长度 #}

# cat > template-xx.j2 <<EOF

ip address:

{{ ansible\_facts.all\_ipv4\_addresses }}

hostname:

{{ ansible\_facts.hostname }}

EOF

# cat > play-j2.yml <<EOF

---

- name: play xx

hosts: mariadb

tasks:

- name: task xx

ansible.builtin.template: #是template模块负责调用jinja2，# ansible<=2.9就只写template

src: template-xx.j2

dest: /tmp/test\_config #将模板里的变量转为实际内容并保存到目标主机的/tmp/test\_config

- name:

ansible.builtin.shell: cat /tmp/test\_config # ansible<=2.9就只写shell

register: return\_test\_config

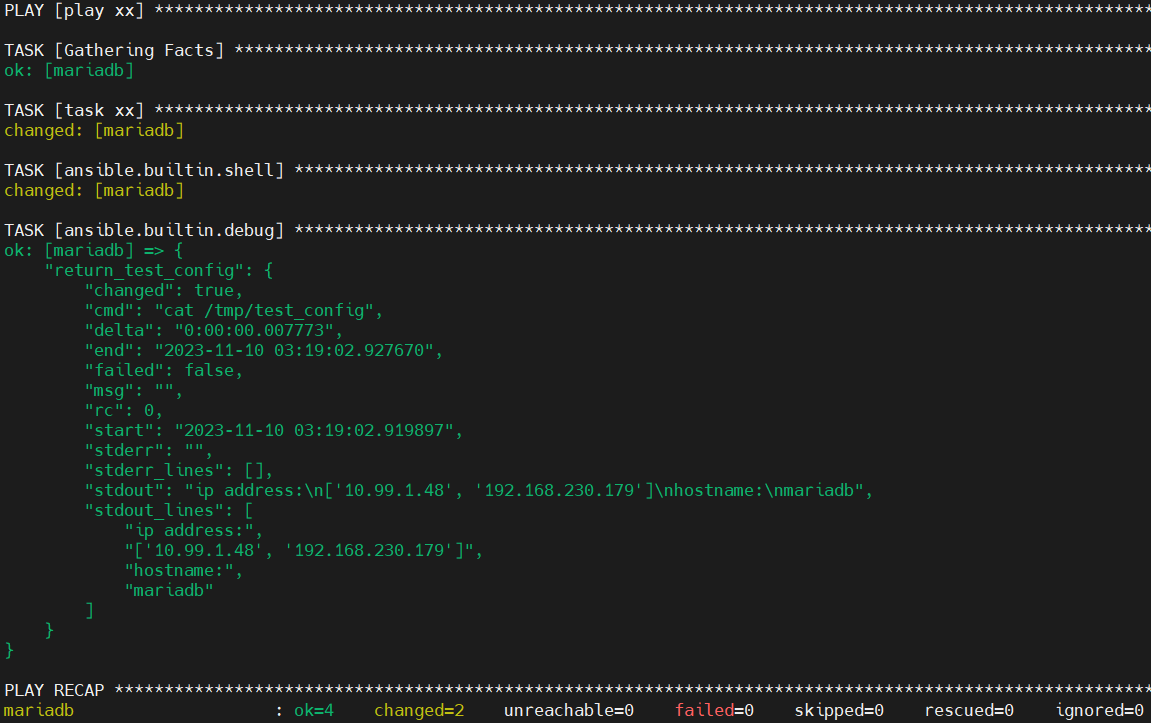
- name:

ansible.builtin.debug: # ansible<=2.9就只写debug

var: return\_test\_config

EOF

# ansible-playbook -i inventory play-j2.yml



**★jinja2的for循环：**

# vi template-for.j2 #正式写时不要在行末写 {#注释#}，会影响结果的换行

|  |  |
| --- | --- |
| {% for user in users %} | 遍历users并赋值给user |
| {{ user }} | 使用user变量 |
| {% endfor %} |  |

# cat > play-j2-for.yml <<EOF

---

- name: play j2 for

hosts: mariadb

vars:

users:

- cof

- lee

- tom

tasks:

- name: task j2 for

ansible.builtin.template: # ansible<=2.9就只写template

src: template-for.j2

dest: /tmp/test\_for\_config #将模板里的变量转为实际内容并保存到目标主机的/tmp/test\_for\_config

EOF

# ansible-playbook -i inventory play-j2-for.yml

mariadb# cat /tmp/test\_for\_config

cof

lee

tom

**★if判断：**

# vi template-for.j2 #正式写时不要在行末写 {#注释#}，会影响结果的换行

|  |  |
| --- | --- |
| {% for user in users if not user=="cof" %} | 将cof排除 |
| user {{ loop.index }}-{{ user }} | 第1个user为正常字符（常量），后面{{}}里的为变量 |
| {% endfor %} |  |

# ansible-playbook -i inventory play-j2-for.yml

mariadb# cat /tmp/test\_for\_config

user 1-lee

user 2-tom

**★变量默认值**

当某变量不存在时，可定义一个默认值

{{ *varxx* | default('NULL',true) }}

#当变量varxx不存在时，赋默认值 NULL，后面的true表示对布尔类型的数据生效；如果不定义默认值，当变量不存在时，任务失败

**示例：**

cat > testdefault.j2 <<EOF

i am {{ username1 | default("NULL",true) }}

u are {{ username2 | default("NONE",true) }}

EOF

cat > testdefault.yml <<EOF

---

- name: play xx

hosts: mariadb

vars:

- username1: userxxxxx1

tasks:

- name: task xx

ansible.builtin.template: # ansible<=2.9就只写template

src: testdefault.j2

dest: /tmp/testdefault.txt

EOF

ansible-playbook testdefault.yml

**结果：**

mariadb# cat /tmp/testdefault.txt

i am userxxxxx

u are NONE

**★playbook及jinja2模板里写变量的区别**

在playbook剧本中以及在jinja2模板文件中，普通变量写法没区别，主要是ansible\_facts获取的变量写法有些不同

在debug模块输出具体主机信息时，分层级去获取，第一级变量名为"ansible\_facts"，其他所有信息都处于这级之下

**★setup模块的filter里的facts变量**

**二级全称**，只可用第二级facts名称去过滤，双引号前后不可有空格，第二级变量名称不可以去除ansible\_前缀

# ansible mariadb -m ansible.builtin.setup -a 'filter="ansible\_default\_ipv4"'

**★playbook里使用的facts变量**

可以从第一级开始，第二级名称要去掉“ansible\_”字符串前缀（有的不需要去除，暂未找到原因，需要多测试，ansible\_local不用去除前缀）

变量为多级结构时，用.分隔或[]分隔，[]方括号里的变量名要用引号括起来

"{{ ansible\_facts.ens33.ipv4.address }}" #从第一级开始写变量，则第二级变量名去除ansible\_前缀

'{{ ansible\_facts["ens33"]["macaddress"] }}' **#里面一层和外面一层的引号不可相同**

也可从二级开始写，这时二级不可去除“ansible\_”字符串前缀，

"{{ ansible\_ens33.ipv4.address }}" #从第二级开始写变量，第二级变量名不可以去除ansible\_前缀

**★when判断语句使用的facts变量**

when判断语句里的facts变量可以从第一级开始，第二级名称要去掉“ansible\_”字符串前缀（有的不需要去除，暂未找到原因，需要多测试，ansible\_local不用去除前缀）

可有多级，用.分隔或[]分隔，[]方括号里的变量名要用引号括起来

也可从二级开头，这时二级不可去除“ansible\_”字符串前缀，

**和其他地方不同的是：不需要用引号及{{ }}花括号包围整个变量**

when: ansible\_facts.bios\_version == "6.00" #when后的变量名不用 "{{ }}" 括起来，二级去除前缀

when: ansible\_default\_ipv4["address"] == "10.99.1.43" #when后的变量名不用 "{{ }}" 括起来，二级不去除前缀

**★hostvars主机信息列表变量写法**

只能从hostvars['主机名']开始，接下来是ansible\_facts一级，到多级，一级不可省略，二级必须去除“ansible\_”字符串前缀（ansible\_local不用去除前缀）

用.分隔或[]分隔，[]方括号里的变量名要用引号括起来

"{{ hostvars['mariadb'].ansible\_facts.lvm }}" #第二级变量名去除ansible\_前缀

"{{ hostvars['mariadb']['ansible\_facts']['lvm'] }}"

**★jinja2里的facts变量**

可以从第一级开始，第二级名称要去掉“ansible\_”字符串前缀（有的不需要去除，暂未找到原因，需要多测试，ansible\_local不用去除前缀）

可有多级，用.分隔或[]分隔，[]方括号里的变量名要用引号括起来

{{ }}花括号外无论何时都不用引号包围，方括号之前不加.点号，方括号之后若不用方括号分隔，则需要在后面加点

也可从二级开头，这时二级不可去除“ansible\_”字符串前缀，

{{ ansible\_facts.default\_ipv4.address }} {{ ansible\_facts.hostname }} #第二级变量名去除ansible\_前缀

{{ ansible\_default\_ipv4['address'] }} {{ ansible\_facts.hostname }} #第二级变量名不可去除ansible\_前缀

**★第7章、ansible任务处理**

**★delegate\_to委派任务**

playbook里的任务下发到目标主机上执行，有时目标由于系统重启或服务重启了，导致此任务没法在目标主机上实现，可以将本应下发到目标主机的任务委派给其他主机去执行

比如有个检测目标主机的某服务是否在线，某端口是否可连接的任务，当此目标主机重启系统时，此任务明显是不可能成功执行的，因为目标主机重启了，执行不了此任务。这时可以将此任务委派给其他还在线的主机，让其他主机去检测目标主机的某服务是否在线。

委派（delegate）可以是tasks下的任务，也可是handlers下的任务

# cat > play-delegate\_to.yml <<EOF

---

- name: play xx

hosts: 10.99.1.248

tasks:

- name: task1 restart httpd

ansible.builtin.service: # ansible<=2.9就只写service

name: httpd

state: restarted

notify:

- detect\_httpd

handlers:

- name: detect\_httpd #如果不委派，此handlers会在目标主机上执行

delegate\_to: "{{item}}" #将此任务委派给item表示的主机

ansible.builtin.uri: # ansible<=2.9就只写uri

url: http://{{ inventory\_hostname }}:808/index.html #这里访问的目标是 触发notify的任务所在的主机，即10.99.1.248

with\_items:

- mariadb

- localhost #表示执行机自己，执行此playbook的ansible执行机

EOF

# ansible-playbook -i inventory play-delegate\_to.yml



每执行一次task1触发notify，都会委派此handlers任务给其中一个item，如果item有多个主机则每个主机都执行一次此handlers任务

delegate\_to: 主机名 #委派给一台主机，此种写法只允许委派给一台，若要委派给多台使用with\_items循环

**★async/wait\_for异步处理任务**

同步为forks，并行；异步表示在重启服务，下载大文件，以及其他需要待的任务

async表示任务的最大超时时间，一旦过了这个时间则判定任务失败

poll用来检查async的值是否满足，值为2表示每2秒检查一次async是否超时了

**①async**

# cat > play-async.yml <<EOF

---

- name: play xx async

hosts: mariadb

tasks:

- name: task sleep xx

ansible.builtin.shell: # ansible<=2.9就只写shell

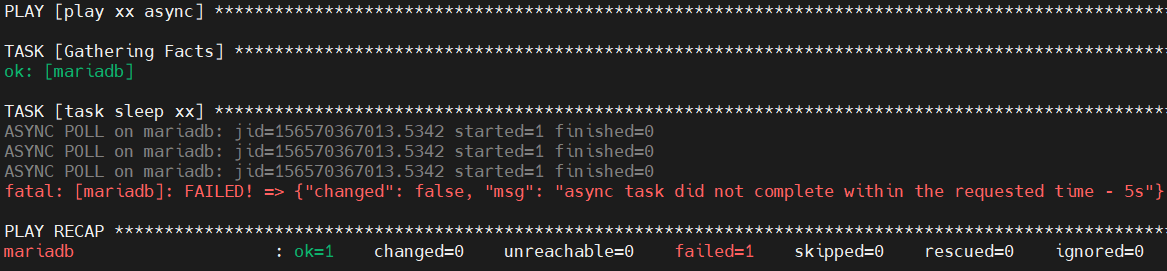
cmd: sleep 10

async: 5 #若5秒内此任务未完成，则判定它失败

poll: 2 #每2秒检查一次async是否超时了

EOF

# ansible-playbook -i inventory play-async.yml #结果失败，符合预期



**②async\_status模块**

# cat > play-async.yml <<EOF

---

- name: play xx async

hosts: mariadb

tasks:

- name: task sleep xx

ansible.builtin.shell: # ansible<=2.9就只写shell

cmd: "sleep 30" #执行长时间的任务

async: 60 #花更多的时间去等待

poll: 0 #表示不等待此任务完成，直接执行后续的其他任务，判定此任务是否完成的事情交给下面的async\_status模块去实现

register: deploy\_xx

- name: waitting for deploy\_xx #这个任务会等待，阻塞后续任务执行

ansible.builtin.async\_status: # ansible<=2.9就只写async\_status，判断任务最好放在最后执行

jid: "{{ deploy\_xx.ansible\_job\_id }}"

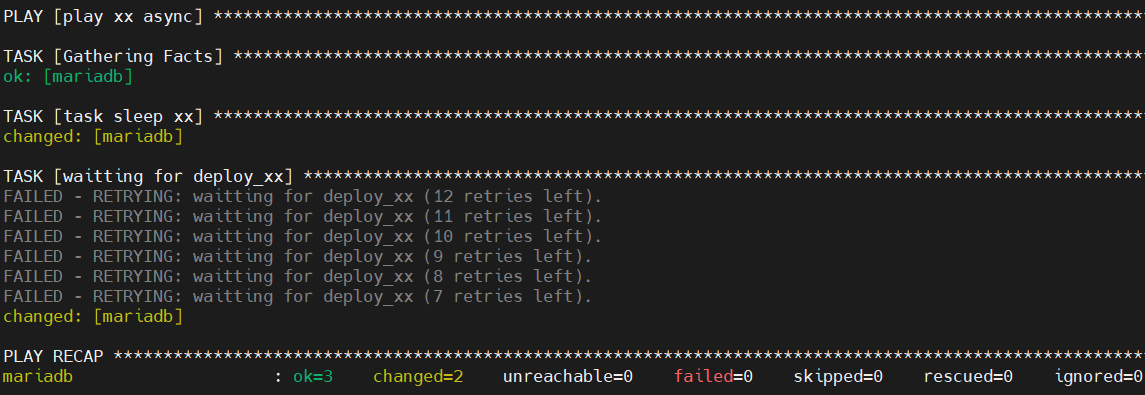
register: deploy\_xx\_result

until: deploy\_xx\_result.finished

retries: 12 #检查12次，默认5秒检查一次

EOF

# ansible-playbook -i inventory play-async.yml



默认每5秒检查一次

**③wait\_for模块**

# cat > play-wait\_for.yml <<EOF

---

- name: play xx

hosts: mariadb

tasks:

- name: task reboot os

ansible.builtin.shell: init 6 #重启受管主机，谨慎操作（不可对正式环境操作）

async: 1

poll: 0

ignore\_errors: true

- name: waitting for os start

ansible.builtin.wait\_for: #判断某tcp端口是否可连接， # ansible<=2.9就只写wait\_for

host: "{{ inventory\_hostname }}"

port: 22

connect\_timeout: 2

timeout: 5

search\_regex: "OpenSSH"

register: check\_ssh\_port

until: check\_ssh\_port is success

retries: 10 #检测10次，默认每5秒一次

delay: 6

delegate\_to: 10.99.1.244 #委托给10.99.1.244这台受管主机去检测

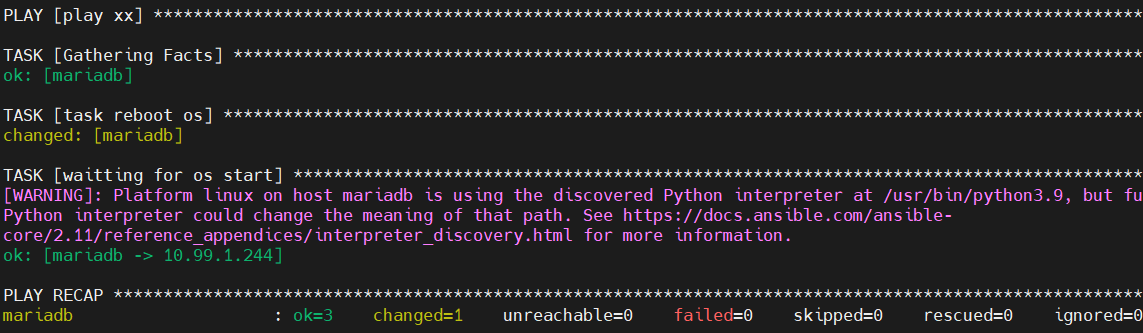
EOF

# ansible-playbook -i inventory play-wait\_for.yml #委托机上的执行者要有sudo权限



上面报错了，原因是当前我使用了user1这个用户执行的ansible剧本，委托给10.99.1.244去检测时，10.99.1.244上的user1用户没有sudo权限，在10.99.1.244上给user1添加sudo权限：

10.99.1.244（root）# echo "user1 ALL=(ALL) NOPASSWD: ALL" > /etc/sudoers.d/user1



**④wait\_for\_connection模块**

# cat > wait\_for\_connection.yml <<EOF

---

- name: pxx

hosts: mariadb

tasks:

- name: restart managed host

ansible.builtin.shell: init 6 #重启受管主机，谨慎操作

async: 1

poll: 0

ignore\_errors: true

- name: wait for managed host to come back

ansible.builtin.wait\_for\_connection: # ansible<=2.9就只写wait\_for\_connection

delay: 2 #延迟检测，默认0秒

connect\_timeout: 5 #连接超时设为5秒，默认5秒

sleep: 5 #每5秒检测一次，默认1秒

timeout: 300 #一共等待的时间，默认600秒

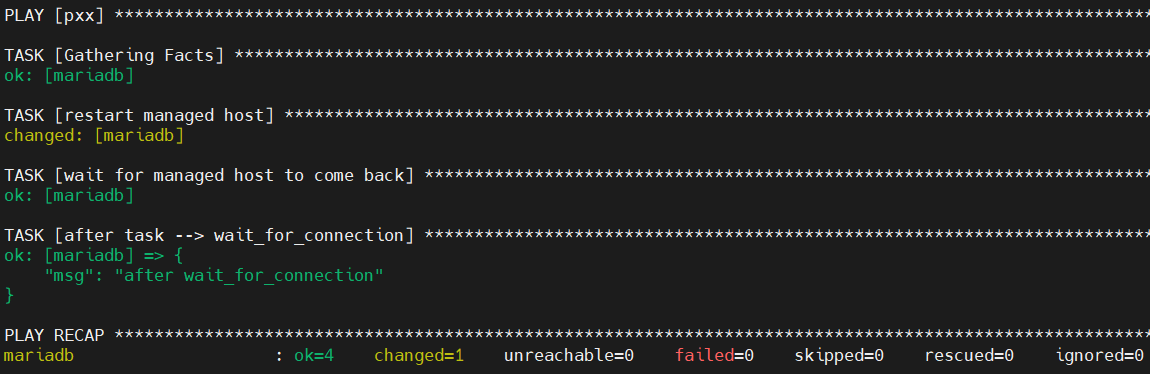
- name: after task --> wait\_for\_connection

ansible.builtin.debug: # ansible<=2.9就只写debug

msg: "after wait\_for\_connection"

EOF

# ansible-playbook wait\_for\_connection.yml



**★serial串行处理playbook**

默认情况下，playbook在执行一个任务时，是同时下发给多台主机执行的，某个任务在所有目标主机上完成后，再继续下一个任务，直到整个playbook完成。

现在想让整个playbook所有任务下发给一台主机完成后，再下发给另一台主机，可使用serial串行的方式

# cat > play-serial.yml <<EOF

---

- name: play xx

hosts: all #本playbook是给所有主机执行的

serial: 1 #但一次只下发给一个主机，一台主机执行完所有任务再下发给另一台主机

tasks:

- name: task 1

ansible.builtin.debug: # ansible<=2.9就只写setup

msg: connect to {{ inventory\_hostname }}

- name: task 2

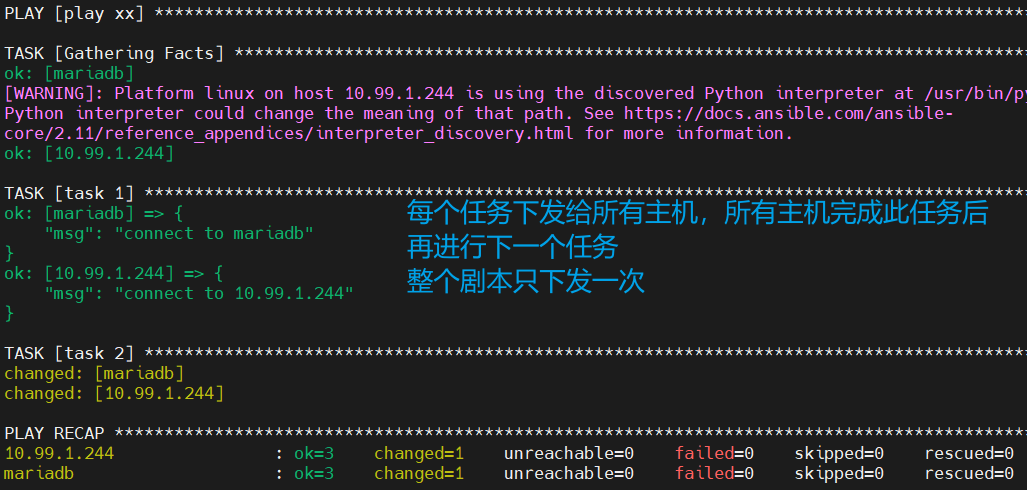
ansible.builtin.shell: sleep 5

EOF

# ansible-playbook -i inventory play-serial.yml #执行效果如下图，



**注释掉剧本里的 serial: 1 后，效果如下：**



如果serial: 指定数量为2，则每次playbook的任务下发2个主机，剩下的主机等下一次playbook

forks并行是基于任务的，serial串行是基于playbook的

就算forks指定为5，只是同一任务同时下发给5台主机，最终等所有主机完成此任务，才进行下一个任务

而如果serial指定为5，是同一playbook剧本只给5台主机执行（至于每次下发剧本时forks是多少它不关心这个）

等完成这5台，才进行下一次playbook的下发

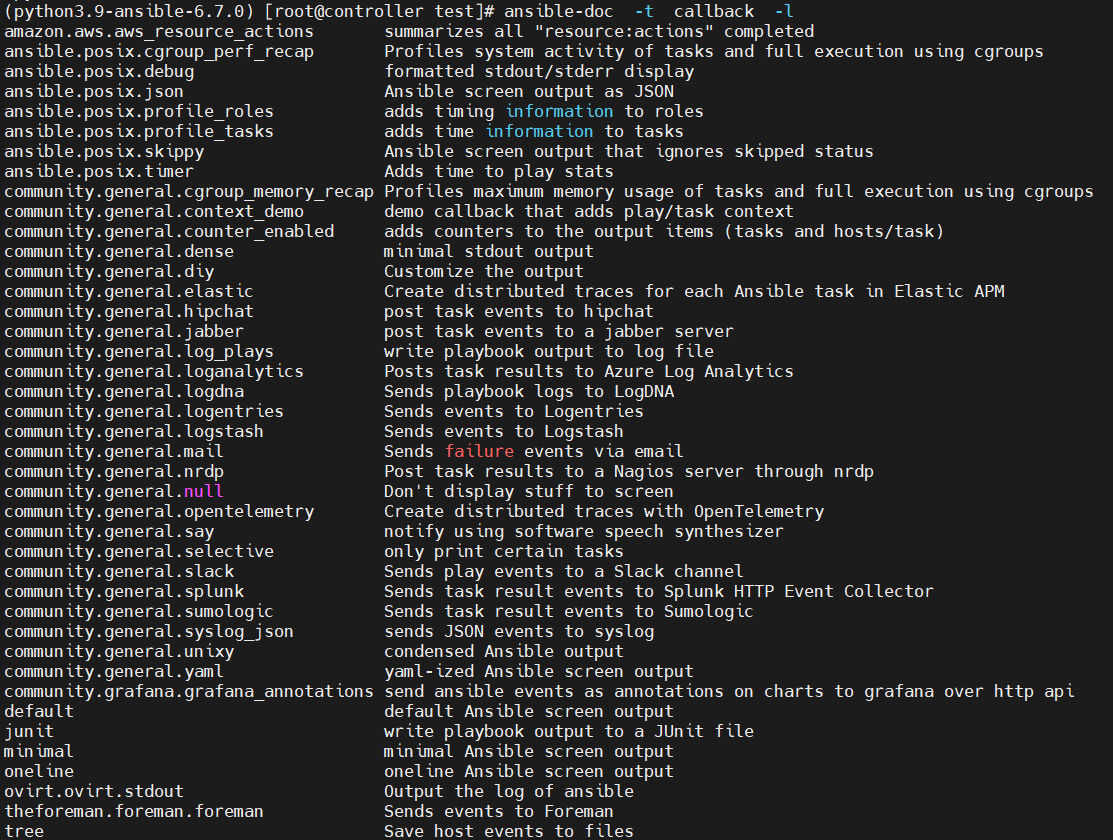
**★第8章、插件及过滤器**

ansible提供了模块、角色等功能给我们使用，还额外提供了许多好用的辅助功能，比如 callback、lookup等插件以及各种过滤器（实现某些功能的函数）

**★callback插件**

# ansible-doc -t callback -l #列出所有callback插件

-t {become,cache,**callback**,cliconf,connection,httpapi,inventory,lookup,netconf,shell,vars,module,strategy,role,keyword}

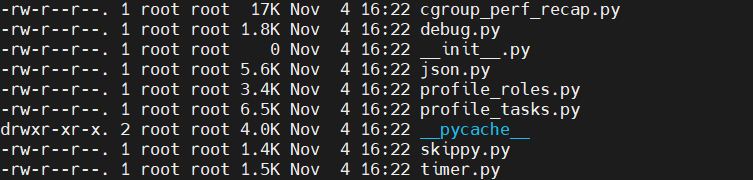


ansible>=4，ansible-core>=2.11才支持此插件，之前的版本也有插件了，但ansible-core-2.11以下设置没生效，原因未知

**插件路径（以ansible.posix集合为例）：**

# venv\_dir=~/python3.9-ansible-4.10.0

# ls -lh ${venv\_dir}/lib/python3.9/site-packages/ansible\_collections/ansible/posix/plugins/callback/



**★callback-timer/profile\_tasks**

# cat > ansible.cfg <<EOF

[defaults]

inventory=./inventory

host\_key\_checking=false

#启用callback插件的timer及profile\_tasks计时功能，多个插件之间用逗号隔开

callbacks\_enabled=ansible.posix.timer,ansible.posix.profile\_tasks

[privilege\_escalation]

become=true

become\_method=sudo

become\_user=root

become\_ask\_pass=false

EOF

# cat > test1.yml <<EOF

---

- name: paly 1

hosts: all

tasks:

- name: task 1

ansible.builtin.ping:

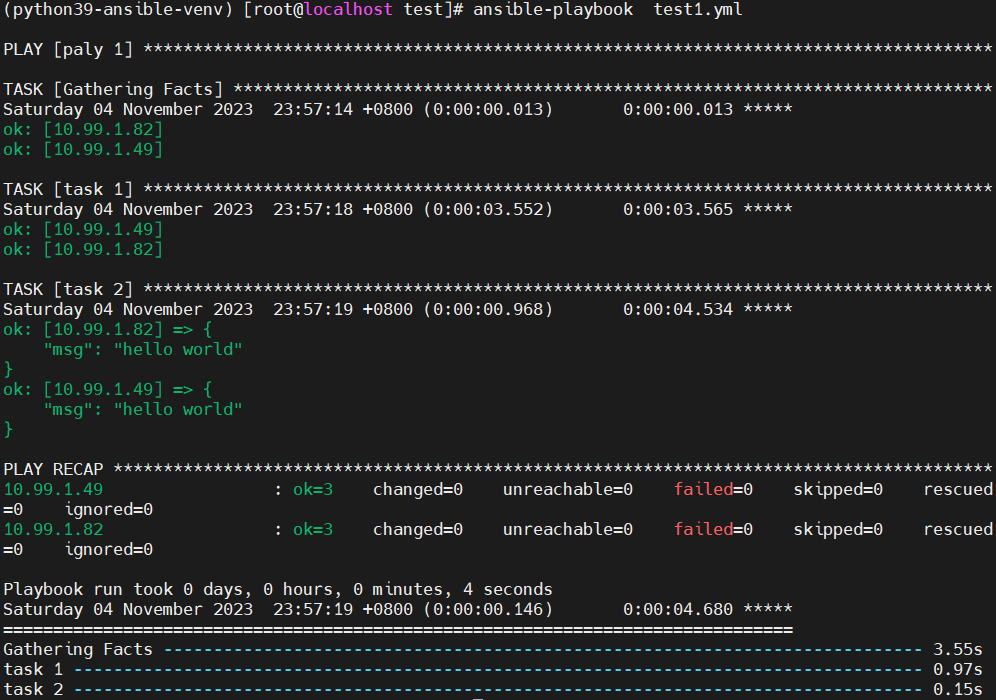
- name: task 2

ansible.builtin.debug:

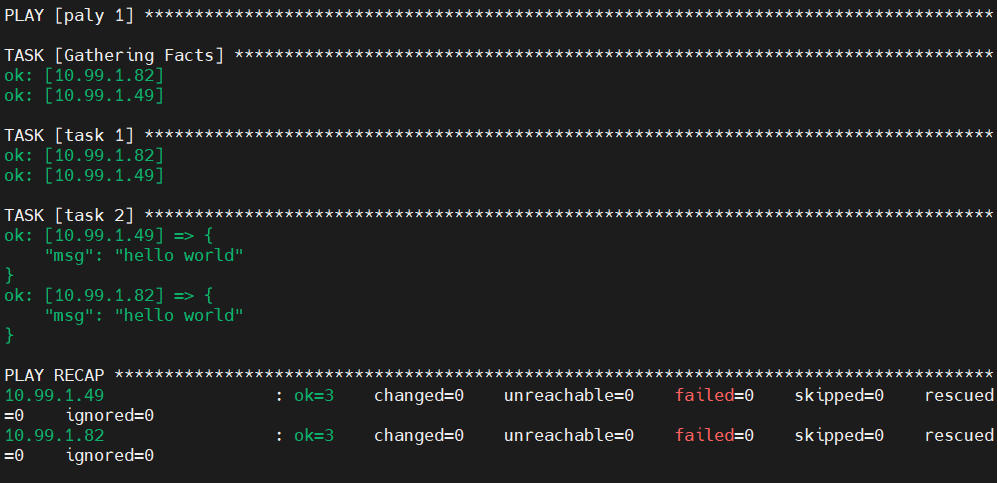
msg: hello world

EOF

# ansible-playbook test1.yml #执行playbook，效果如下：



#ansible <=2.9版本不支持插件，效果如下：

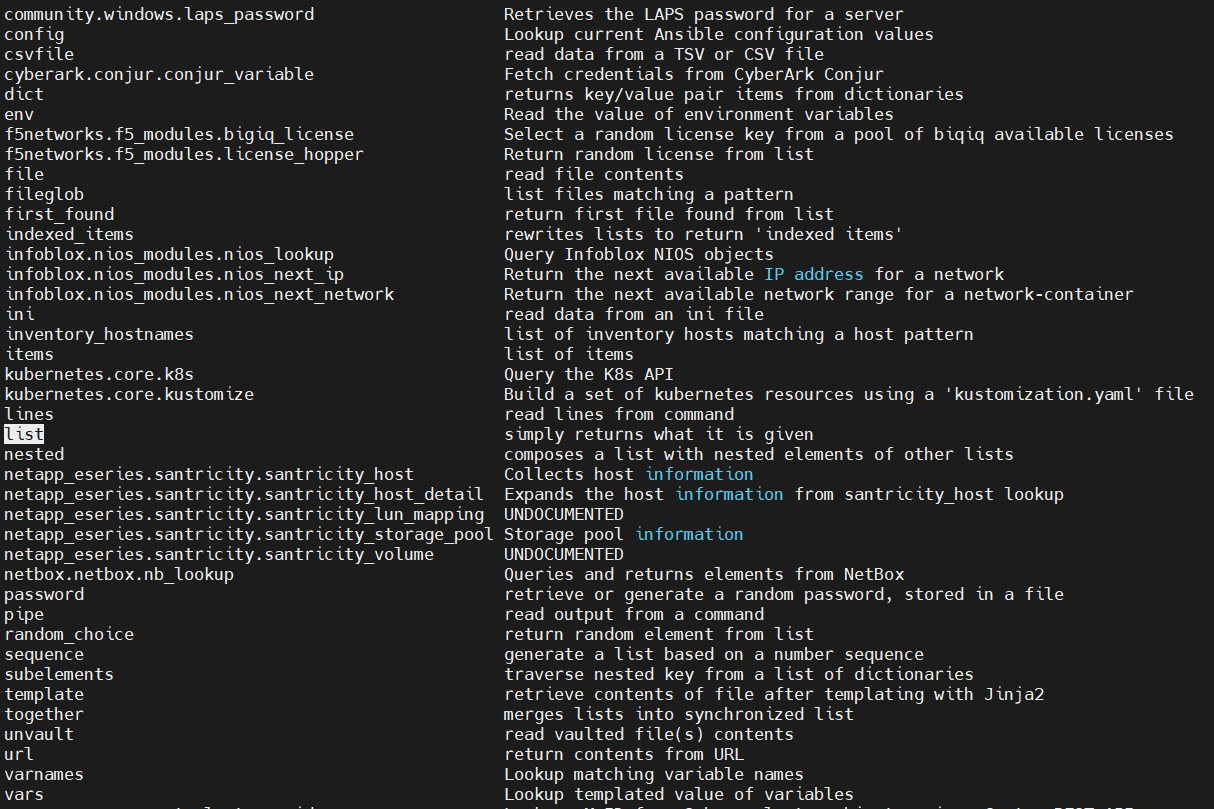


**★lookup插件**

lookup插件的用法如下:

lookup('插件名', 被处理数据或参数)

# ansible-doc -t lookup -l #列出lookup所有插件



**★lookup-list**

当我们需要使用循环时，可以使用"with\_"开头的关键字处理数据，也可以使用"loop"关键字，使用with\_xxx关键字时，等同于loop与lookup('xxx', [listxx])的组合，lookup插件可以作为循环的数据源

比如：

# cat > test\_with\_list.yml <<EOF

---

- name: paly with\_list

hosts: mariadb

tasks:

- name: task with\_list

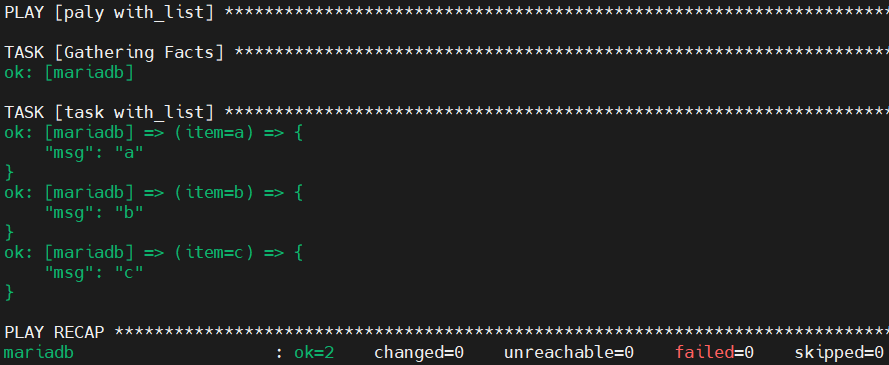
ansible.builtin.debug:

msg: "{{ item }}"

with\_list: ['a', 'b', 'c']

EOF

# ansible-playbook test\_with\_list.yml #执行playbook，效果如下：



# cat > test\_lookup\_list.yml <<EOF

---

- name: paly loop lookup

hosts: mariadb

tasks:

- name: task loop lookup

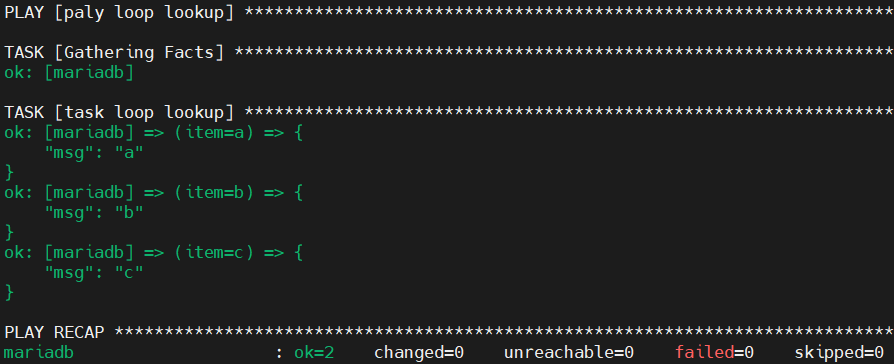
ansible.builtin.debug:

msg: "{{ item }}"

loop: "{{ lookup('list', ['a', 'b', 'c'] ) }}"

EOF

# ansible-playbook test\_lookup\_list.yml #执行playbook，效果如下：



**★lookup-file**

lookup的file插件可以获取到指定文件的文件内容（文件位于ansible执行机中），例如：

# cat > test\_lookup\_file.yml <<EOF

---

- name: paly lookup

hosts: mariadb

tasks:

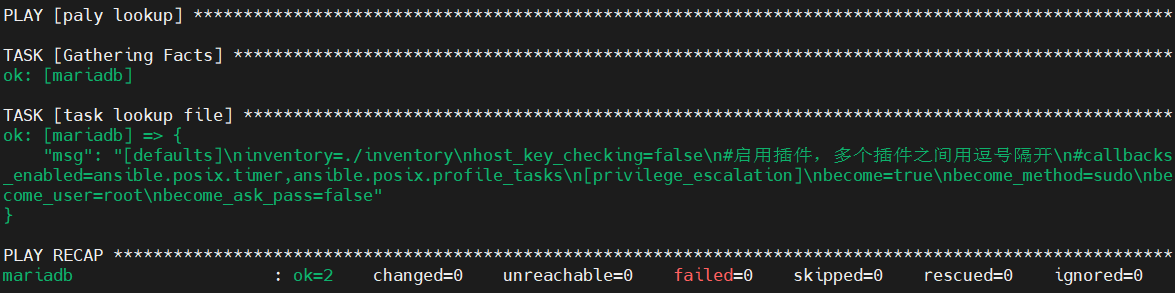
- name: task lookup file

ansible.builtin.debug:

msg: "{{ lookup('file', './ansible.cfg' ) }}"

EOF

# ansible-playbook test\_lookup\_file.yml #执行playbook，效果如下（目标主机为mariadb，但读取文件时是读取执行机本地的文件）：



**也可读取多个文件：**

# cat > test\_lookup\_file.yml <<EOF

---

- name: paly lookup

hosts: mariadb

tasks:

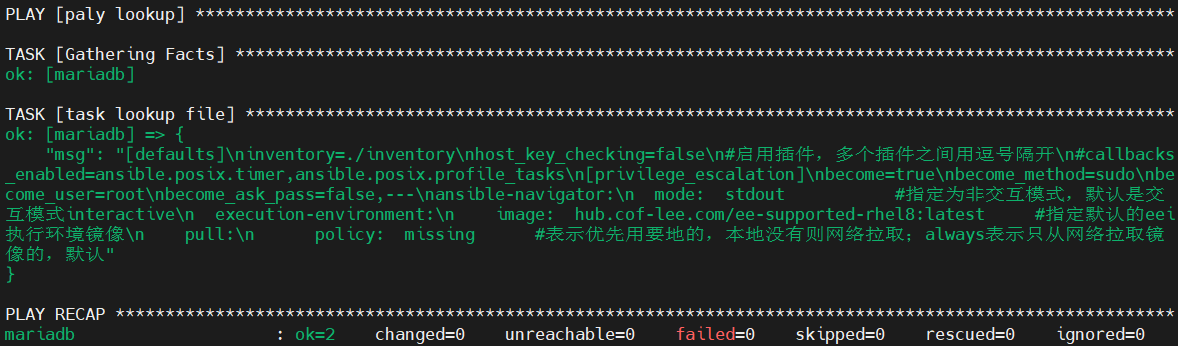
- name: task lookup file

ansible.builtin.debug:

msg: "{{ lookup('file', './ansible.cfg' ,'./ansible-navigator.yml') }}"

EOF

# ansible-playbook test\_lookup\_file.yml #执行playbook，效果如下（多个文件内容变为一行输出，返回的是一个字符串）：



默认情况下读取多个文件时是按一行字符串输出的（所有文件的内容都输出到一行了），想让不同文件内容换行输出，需要指定wantlist=True参数（返回字符串列表）：

# cat > test\_lookup\_file.yml <<EOF

---

- name: paly lookup

hosts: mariadb

tasks:

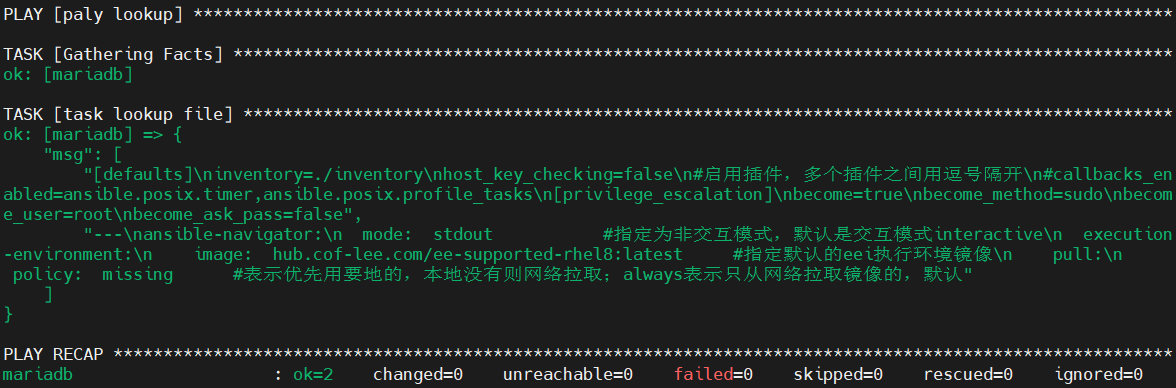
- name: task lookup file

ansible.builtin.debug:

msg: "{{ lookup('file', './ansible.cfg' ,'./ansible-navigator.yml', wantlist=true) }}"

EOF

# ansible-playbook test\_lookup\_file.yml #执行playbook，效果如下



在2.5版本的ansible中，引入了一个新的jinja2函数，这个函数叫做"query"，通过query函数也可以调用lookup插件，只是通过query函数调用lookup插件时，默认就是返回一个列表（等同于指定了wantlist=True参数的lookup）

# cat > test\_query\_file.yml <<EOF

---

- name: paly query

hosts: mariadb

tasks:

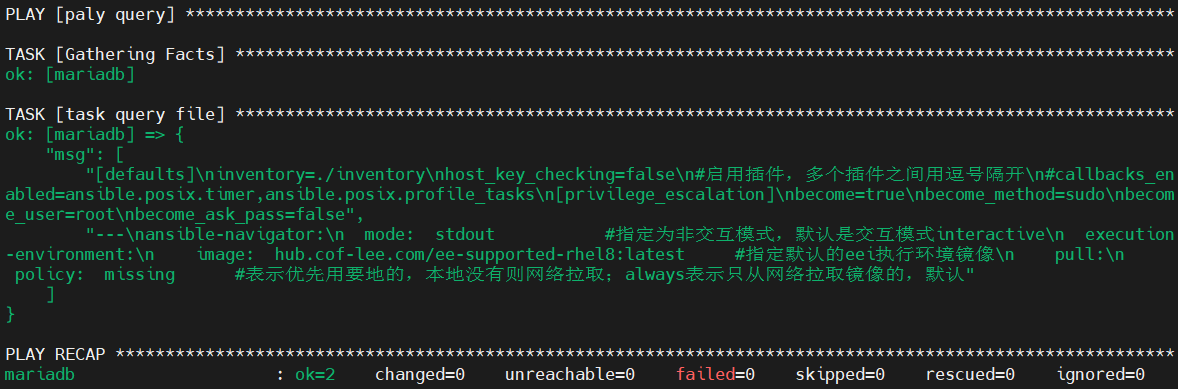
- name: task query file

ansible.builtin.debug:

msg: "{{ query('file', './ansible.cfg' ,'./ansible-navigator.yml') }}"

EOF

# ansible-playbook test\_query\_file.yml #执行playbook，效果如下



而"query"函数又有一个简写的格式"q"，所以，如下写法也是等价的

msg: "{{ query('file', './ansible.cfg' ,'./ansible-navigator.yml') }}"

msg: "{{ q('file', './ansible.cfg' ,'./ansible-navigator.yml') }}"

在2.6版本的ansible中，可以使用errors参数控制lookup插件出错时的处理机制，比如想要在lookup插件执行出错时忽略错误，可以将errors的值设置为ignore，如：

# cat > test\_lookup\_filexx.yml <<EOF

---

- name: paly lookup

hosts: mariadb

tasks:

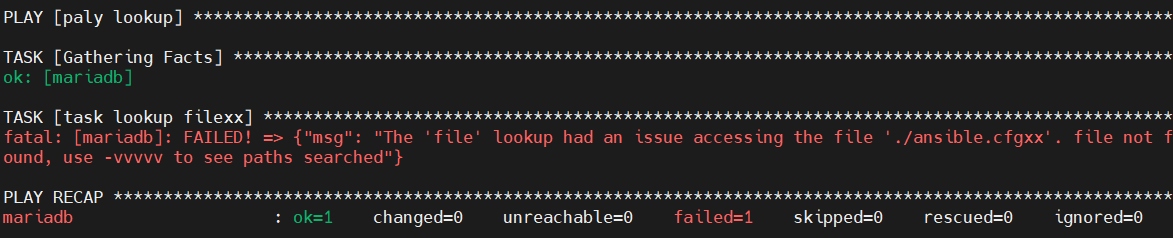
- name: task lookup filexx

ansible.builtin.debug:

msg: "{{ lookup('file', './ansible.cfgxx') }}"

EOF

# ansible-playbook test\_lookup\_filexx.yml #执行playbook，效果如下（有报错）



# cat > test\_lookup\_filexx.yml <<EOF

---

- name: paly lookup

hosts: mariadb

tasks:

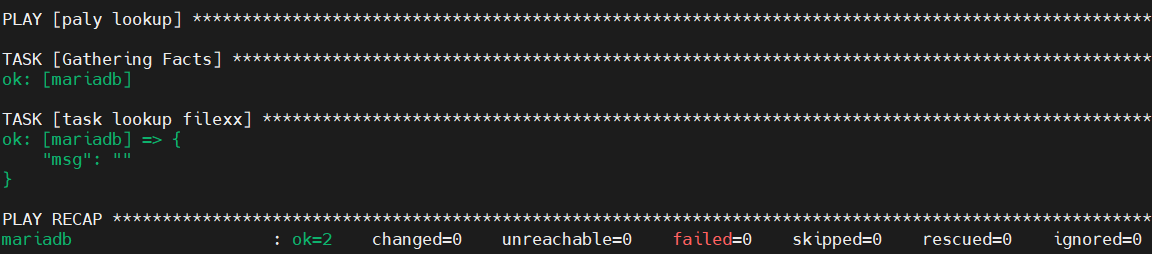
- name: task lookup filexx

ansible.builtin.debug:

msg: "{{ lookup('file', './ansible.cfgxx', errors='ignore') }}"

EOF

# ansible-playbook test\_lookup\_filexx.yml #执行playbook，效果如下（无报错了）



**★.splitlines()按行拆分文件内容**

如果我们的参数写在一个文本文件里，一行有多个参数（多个参数用冒号隔开），有若干行，如何读取这个文件呢？

例：

现有需求为：要创建若干用户，用户的username,password,uid写在一行（用冒号隔开），每个用户的信息单独放一行

# cat > userinfo.txt <<EOF

username1:passwd1:3001

username2:passwd2:3002

username3:passwd3:3003

EOF

# cat > test\_lookup\_fileyy.yml <<EOF

---

- name: paly lookup

hosts: mariadb

tasks:

- name: task lookup fileyy

ansible.builtin.user:

name: "{{ item.split(':').0 }}"

password: "{{ item.split(':').1 | password\_hash('sha512') }}"

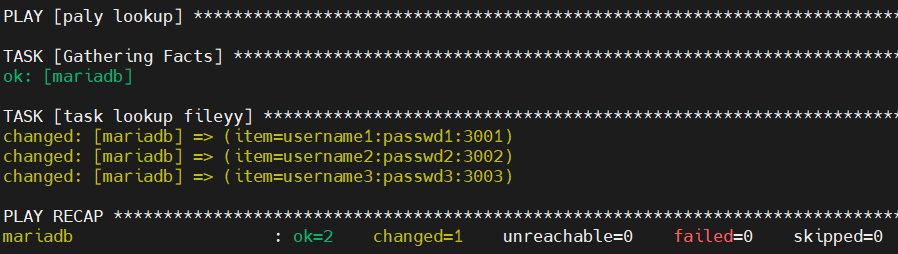
uid: "{{ item.split(':').2 }}"

state: present

loop: "{{ lookup('file', './userinfo.txt').splitlines() }}"

EOF

# ansible-playbook test\_lookup\_fileyy.yml #执行playbook，效果如下



**★lookup-password**

lookup的password插件可用于生成一个随机密码并将明文密码保存到文本文件里（文件存储在执行机本地）

要求ansible>=4.0

**示例1：**

# cat > test\_lookup\_password.yml <<EOF

---

- name: paly lookup

hosts: mariadb

vars:

username: cof

tasks:

- name: task lookup password

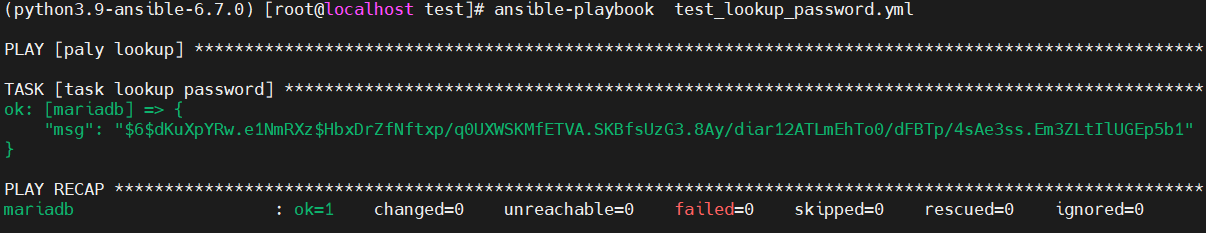
ansible.builtin.debug:

msg: "{{ lookup('ansible.builtin.password', 'password-{{ username }}', chars='digits', length=6, encrypt='sha512\_crypt') }}"

EOF

# lookup()第1个参数为插件名，第2个参数为明文密码保存文件名，之后参数为生成随机密码的控制参数

# ansible-playbook test\_lookup\_password.yml #执行playbook，效果如下（目标主机为mariadb，但密码保存文件是在执行机本地的文件）返回值是加密了的password\_hash



执行机：# cat password-cof #保存到文件里的密码是明文的，且附带salt盐值（16字符）

790125 salt=dKuXpYRw.e1NmRXz

**解析：**

chars='digits', length=6 #表示生成随机密码由'digits'组成，即仅数字，密码总长度为6。随机密码还可由字母及其他特殊符号组成，chars='多个字符类型间用逗号隔开'

digits 数字

ascii\_letters 英文字母，含大小写

ascii\_lowercase 小写英文字母

ascii\_uppercase 大写英文字母

punctuation 符号

**示例2：**

生成包含数字，大小写字母及符号组成的随机密码，总长度12

# cat > test\_lookup\_password2.yml <<EOF

---

- name: paly lookup

hosts: mariadb

vars:

username: cof

tasks:

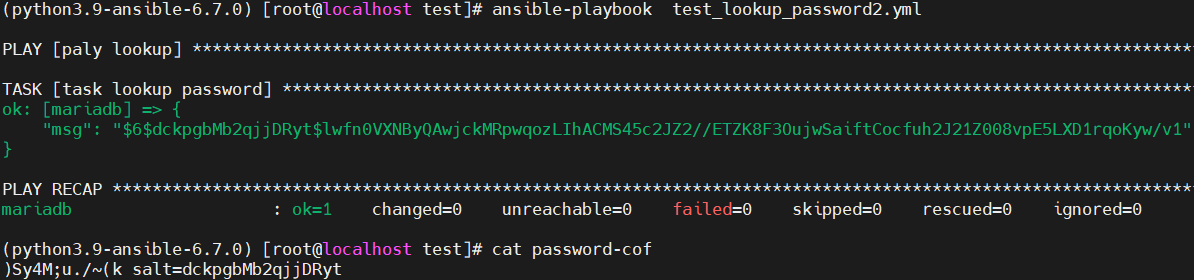
- name: task lookup password

ansible.builtin.debug:

msg: "{{ lookup('ansible.builtin.password', 'password-{{ username }}', chars='digits,ascii\_letters,punctuation', length=12, encrypt='sha512\_crypt' ) }}"

EOF

# ansible-playbook test\_lookup\_password2.yml #执行playbook，效果如下



执行机：# cat password-cof

)Sy4M;u./~(k salt=dckpgbMb2qjjDRyt

**★过滤器**

过滤器是实现某个功能的函数，

**★正则表达式过滤器**

**★regex\_search搜索字符串**

# cat > test\_regex\_search.yml <<EOF

---

- name: paly regex\_search

hosts: mariadb

tasks:

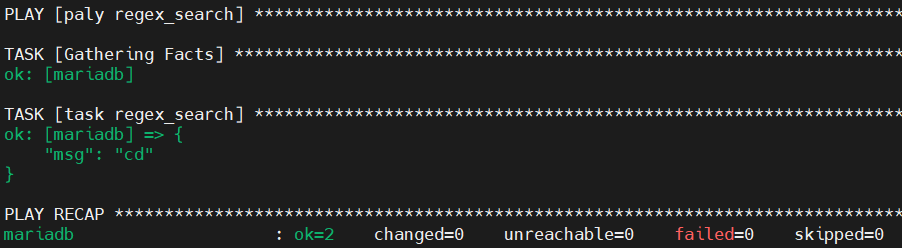
- name: task regex\_search

ansible.builtin.debug:

msg: "{{ 'abCDcdef\ncdefg' | regex\_search('(cd)') }}"

EOF

# ansible-playbook test\_regex\_search.yml #执行playbook，效果如下（默认区分大小写，默认支持换行匹配）



**设置允许多行搜索及忽略大小写（不区分大小写）**

# cat > test\_regex\_search.yml <<EOF

---

- name: paly regex\_search

hosts: mariadb

tasks:

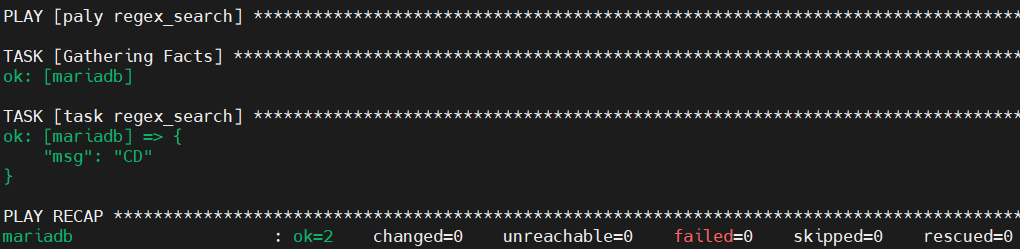
- name: task regex\_search

ansible.builtin.debug:

msg: "{{ 'abCDcdef\ncdefg' | regex\_search('(cd)', multiline=True, ignorecase=True) }}"

EOF

# ansible-playbook test\_regex\_search.yml #执行playbook，效果如下（不区分大小写时优先匹配到了CD）



**★regex\_findall返回所有匹配结果列表**

# cat > test\_regex\_findall.yml <<EOF

---

- name: paly regex\_findall

hosts: mariadb

tasks:

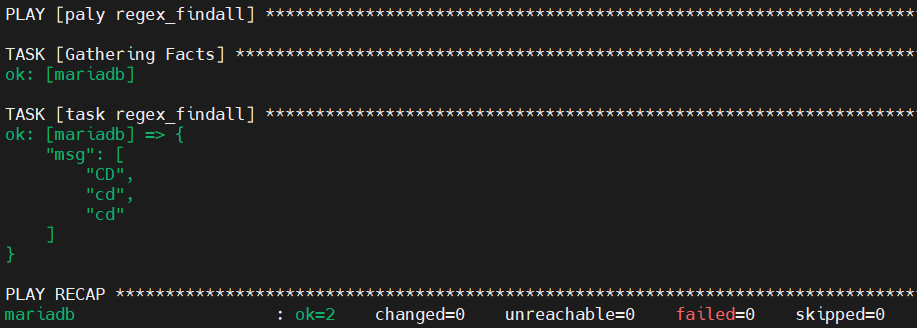
- name: task regex\_findall

ansible.builtin.debug:

msg: "{{ 'abCDcdef\ncdefg' | regex\_findall('(cd)', multiline=True, ignorecase=True) }}"

EOF

# ansible-playbook test\_regex\_findall.yml #执行playbook，效果如下



**★regex\_replace替换匹配的字符串**

默认区分大小写，默认支持换行匹配

# cat > test\_regex\_replace.yml <<EOF

---

- name: paly regex\_replace

hosts: mariadb

tasks:

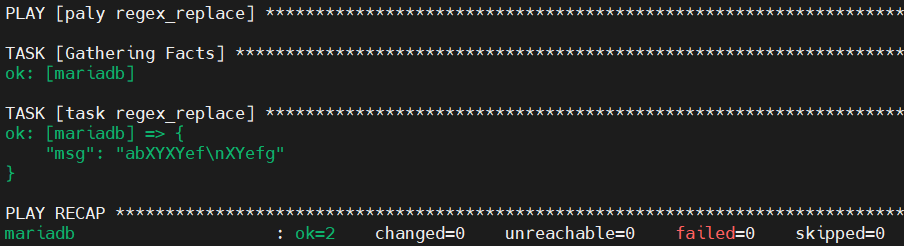
- name: task regex\_replace

ansible.builtin.debug:

msg: "{{ 'abCDcdef\ncdefg' | regex\_replace('(cd)', 'XY', multiline=True, ignorecase=True) }}"

EOF

# ansible-playbook test\_regex\_replace.yml #执行playbook，效果如下（字母cd被替换为XY了）



**★密码hash过滤器**

**★hash()**

计算字符串或文件内容的hash值

# cat > test\_hash.yml <<EOF

---

- name: paly hash

hosts: mariadb

tasks:

- name: task hash

ansible.builtin.debug:

msg:

- "{{ 'hello' | hash('md5') }}"

- "{{ 'hello' | hash('sha1') }}"

- "{{ 'hello' | hash('sha256') }}"

EOF

# ansible-playbook test\_hash.yml #执行playbook，效果如下



**★password\_hash()**

将输入的明文密码转为加密后的密码

# cat > test\_password\_hash.yml <<EOF

---

- name: paly hash

hosts: mariadb

tasks:

- name: task hash

ansible.builtin.debug:

msg:

- "{{ 'hello' | password\_hash('sha256') }}"

- "{{ 'hello' | password\_hash('sha512') }}"

EOF

# ansible-playbook test\_password\_hash.yml #执行playbook，效果如下



**★变量过滤器**

**★mandatory变量强制定义过滤器**

当引用一个未被定义的变量时，ansible默认会报错，

可以通过更改ansible.cfg配置项的方式关闭这种机制，如

# vi ansible.cfg

[defaults]

error\_on\_undefined\_vars = False

在关闭这个机制的情况下，如果还想让ansible强制检查某个变量是否定义，可以使用mandatory过滤器

# cat > test\_mandatory.yml <<EOF

---

- name: paly mandatory

hosts: mariadb

tasks:

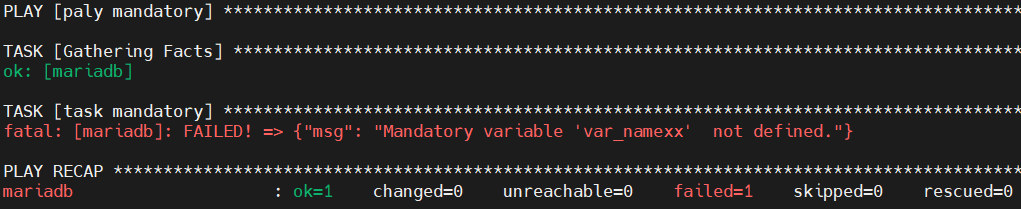
- name: task mandatory

ansible.builtin.debug:

msg: "{{ var\_namexx | mandatory }}"

EOF

# ansible-playbook test\_mandatory.yml #执行playbook，效果如下



**★default()变量默认值过滤器**

default()过滤器可以为未定义变量设置默认值，优先级最低

如果想将变量的值为false、False和空（None）时视为未定义，则必须给defaults过滤器第二参数位置加上true

# cat > test\_default.yml <<EOF

---

- name: paly default

hosts: mariadb

tasks:

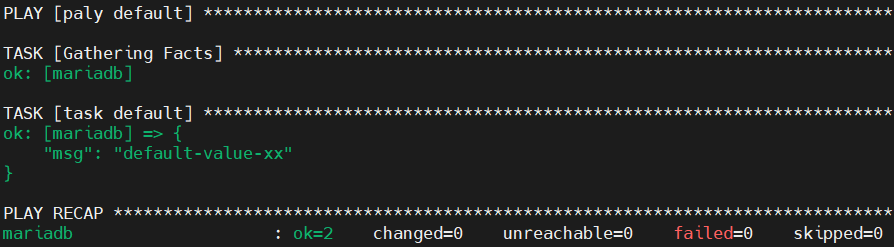
- name: task default

ansible.builtin.debug:

msg: "{{ var\_namexx | default('default-value-xx', true) }}"

EOF

# ansible-playbook test\_default.yml #执行playbook，效果如下



**★default(omit)可删除参数过滤器**

在使用模块时，有些参数若未定义，则不使用此参数

# cat > test\_omit.yml <<EOF

---

- name: paly omit

hosts: mariadb

tasks:

- name: task omit

ansible.builtin.file:

dest: "{{ item.path }}"

state: touch

mode: "{{ item.mode | default(omit) }}"

loop:

- path: /tmp/testfile1

- path: /tmp/testfile2

- path: /tmp/testfile3

mode: "0440"

EOF

# ansible-playbook test\_omit.yml #执行playbook，效果如下（前2个文件未定义mode变量，则不使用mode参数，最后一个文件有mode参数，则模块使用了mode参数）



mariadb# ls -lh /tmp

-rw-r--r--. 1 root root 0 Jan 4 16:33 testfile1

-rw-r--r--. 1 root root 0 Jan 4 16:33 testfile2

-r--r-----. 1 root root 0 Jan 4 16:33 testfile3

**★random随机数过滤器**

# cat > test\_random.yml <<EOF

---

- name: paly random

hosts: mariadb

gather\_facts: no

tasks:

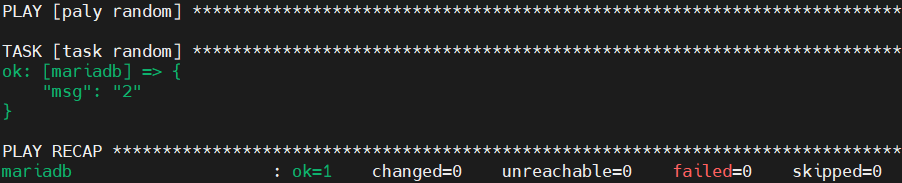
- name: task random

ansible.builtin.debug:

msg: "{{ 5 | random() }}"

EOF

# ansible-playbook test\_random.yml #执行playbook，效果如下（从0到4，整数）



random()等同于random(start=0, step=1)

# cat > test\_random.yml <<EOF

---

- name: paly random

hosts: mariadb

gather\_facts: no

tasks:

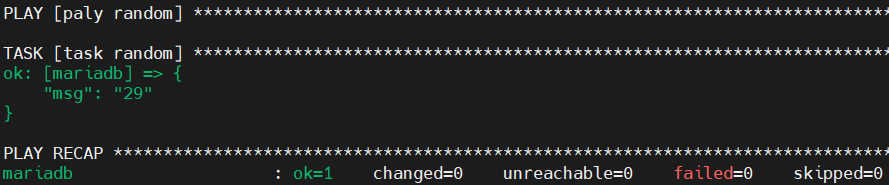
- name: task random

ansible.builtin.debug:

msg: "{{ 51 | random(start=1, step=2) }}"

EOF

# ansible-playbook test\_random.yml #执行playbook，效果如下（从1到50之间的奇数，不取到51）



**★列表元素过滤器**

min过滤器 获取最小值元素；max过滤器 获取最大值元素

unique过滤器 给列表元素去重

sort过滤器 给列表元素排序（升序），reverse过滤器会返回一个顺序与原始顺序相反的列表以达到降序的效果

shuffle过滤器会打乱原列表，返回乱序的列表

# cat > test\_listxx.yml <<EOF

---

- name: paly listxx

hosts: mariadb

gather\_facts: no

tasks:

- name: task listxx

ansible.builtin.debug:

msg:

- "min is: {{ [1,9,5,1,2,4,3,3] | min }}"

- "max is: {{ [1,9,5,1,2,4,3,3] | max }}"

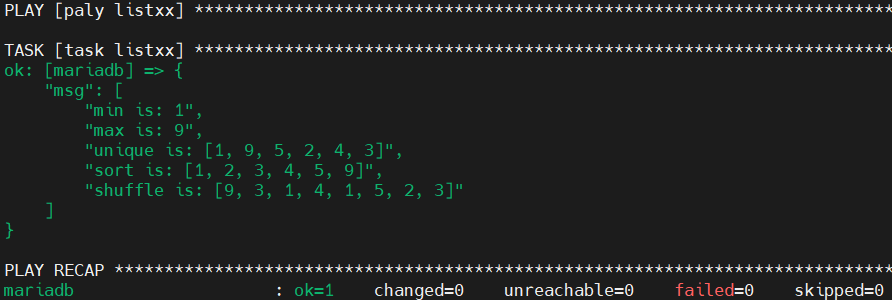
- "unique is: {{ [1,9,5,1,2,4,3,3] | unique }}"

- "sort is: {{ [1,9,5,1,2,4,3,3] | unique | sort }}"

- "shuffle is: {{ [1,9,5,1,2,4,3,3] | shuffle }}"

EOF

# ansible-playbook test\_listxx.yml #执行playbook，效果如下



**★IP地址过滤器**

ipaddr过滤器用于测试是否为IP地址格式（是则返回此ip，否则返回false）

ipaddr过滤器可以用于从cidr（x.x.x.x/24）中提取出一个IP地址

# cat > test\_ipaddr.yml <<EOF

---

- name: paly ipaddr

hosts: mariadb

gather\_facts: no

tasks:

- name: task ipaddr

ansible.builtin.debug:

msg:

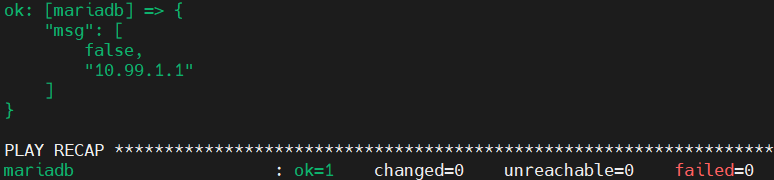
- "{{ '10.99.1.999' | ipaddr }}"

- "{{ '10.99.1.1/24' | ipaddr('address') }}"

EOF

# ansible-playbook test\_ipaddr.yml #执行playbook，效果如下

fatal: [mariadb]: FAILED! => {"msg": "The ipaddr filter requires python's netaddr be installed on the ansible controller"}



**★用regex正则表达式查找ip**

# vi test\_ipaddr.yml

---

- name: paly ipaddr

hosts: mariadb

gather\_facts: no

tasks:

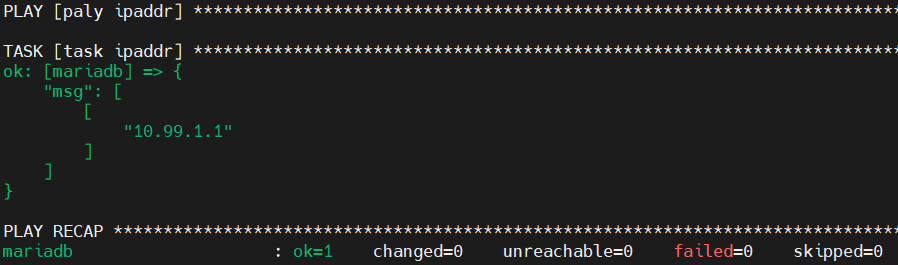
- name: task ipaddr

ansible.builtin.debug:

msg:

- "{{ 'can you find ip 10.99.1.1/24' | regex\_findall('\\b(?:[0-9]{1,3}\\.){3}[0-9]{1,3}\\b') }}"

# ansible-playbook test\_ipaddr.yml



**★格式化字符串过滤器**

使用lower、upper、及capitalize过滤器来转换字符串的大小写

# cat > test\_string.yml <<EOF

---

- name: paly string

hosts: mariadb

gather\_facts: no

tasks:

- name: task string

ansible.builtin.debug:

msg:

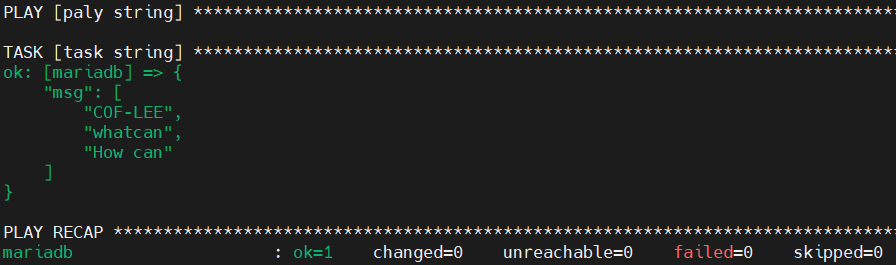
- "{{ 'cof-lee' | upper }}"

- "{{ 'WHATcan' | lower }}"

- "{{ 'how can' | capitalize }}"

EOF

# ansible-playbook test\_string.yml #执行playbook，效果如下



**★base64过滤器**

# cat > test\_base64.yml <<EOF

---

- name: paly base64

hosts: mariadb

gather\_facts: no

tasks:

- name: task base64

ansible.builtin.debug:

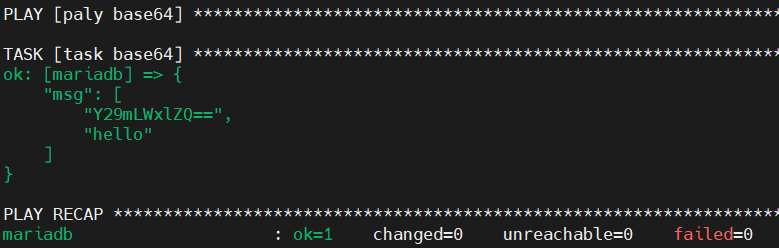
msg:

- "{{ 'cof-lee' | b64encode }}"

- "{{ 'aGVsbG8=' | b64decode }}"

EOF

# ansible-playbook test\_base64.yml #执行playbook，效果如下



**★时间格式化过滤器**

# cat > test\_strftime.yml <<EOF

---

- name: paly strftime

hosts: mariadb

gather\_facts: no

tasks:

- name: task strftime

ansible.builtin.debug:

msg:

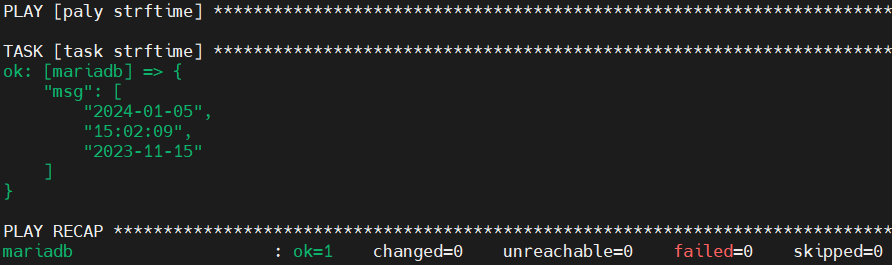
- "{{ '%Y-%m-%d' | strftime }}"

- "{{ '%H:%M:%S' | strftime }}"

- "{{ '%Y-%m-%d' | strftime(1699998888) }}"

EOF

# ansible-playbook test\_strftime.yml #执行playbook，效果如下



**★第9章、role角色**

**★role角色的应用**

ansible的role（翻译为：角色）是ansible自动化项目的一种组织方式，在标准化的目录结构中打包所有任务/变量/文件/模板及其他资源，即把原来一个playbook剧本里的所有内容进行了拆分，例如：

把tasks任务单独写成文件并放到一个名为tasks的子目录中

把vars变量单独写到文件中并放到一个名为vars的子目录中

把jinja2模板文件放到一个名为templates的子目录中

............

**role目录结构如下：**

-- role顶级目录 #可设置为项目名称，role目录名称或者说role名称不能带减号，只能用字母数字

|-- default #目录中的main.yml文件定义角色**变量的默认值**，优先级低，可被覆盖

|-- files #存放角色任务中**引用的静态文件**

|-- handlers #目录中的main.yml文件定义**handler处理任务**

|-- meta #目录中的main.yml文件定义角色的相关信息，如**作者，许可证，依赖等信息**

|-- tasks #目录中的\*.yml文件定义角色的**任务**（入口文件为main.yml，里面引入其他任务.yml文件）

|-- templates #此目录用于存放**jinja2模板文件**

|-- vars #目录中的main.yml文件定义角色**变量的值**，优先级高于default

|-- tests #此目录可包含playbook文件用于测试本role

①创建项目目录

# mkdir -p rolexx/{files,handlers,tasks,templates,vars} #创建role项目目录及需要用的子目录

#也可使用ansible-galaxy init rolexx命令来创建一个项目，自动帮我们创建好相应的子目录及main.yml文件

# tree rolexx/

rolexx/

├── files

├── handlers

├── tasks

├── templates

└── vars

②在vars子目录里写所有需要用到的变量文件

# cat > rolexx/vars/main.yml <<EOF

users:

- cof

- lee

EOF

③在templates子目录里写所有需要用到的j2模板文件

# cat > rolexx/templates/conf-xxx.j2 <<EOF

{% for user in users %}

{{ user }}

{% endfor %}

EOF

④在tasks子目录里写所有任务清单文件

# cat > rolexx/tasks/task-yyy.yml <<EOF

- name: task yyy

ansible.builtin.debug: # ansible<=2.9就只写debug

msg: debug yyy

EOF

# cat > rolexx/tasks/task-xxx.yml <<EOF

- name: task xx

ansible.builtin.template: # ansible<=2.9就只写template

src: conf-xxx.j2 #不需要指定路径，会自动去rolexx/templates/下面找

dest: /tmp/conf-xxx.j2.conf

notify: restart\_sshd

EOF

⑤在handlers子目录里写所有handler任务

# cat > rolexx/handlers/main.yml <<EOF

---

- name: restart\_sshd #须同任务的notify触发的名称，如有其他handlers文件，须在main.yml里引入

ansible.builtin.service: # ansible<=2.9就只写service

name: sshd

state: restarted

EOF

⑥创建tasks入口文件，指定任务顺序（文件名称必须为main.yml）

# cat > rolexx/tasks/main.yml <<EOF

- ansible.builtin.include\_tasks: task-xxx.yml #在入口文件引入其他任务.yml文件，

- ansible.builtin.include\_tasks: task-yyy.yml # ansible<=2.9就只写include\_tasks

EOF

最后创建一个playbook，引用上面创建好的role角色

# role入口文件（文件名称自定义，与role目录同级）就是一个playbook剧本，在剧本里使用上面写好的role

# cat > role-playbook.yml <<EOF

---

- name: play xxx

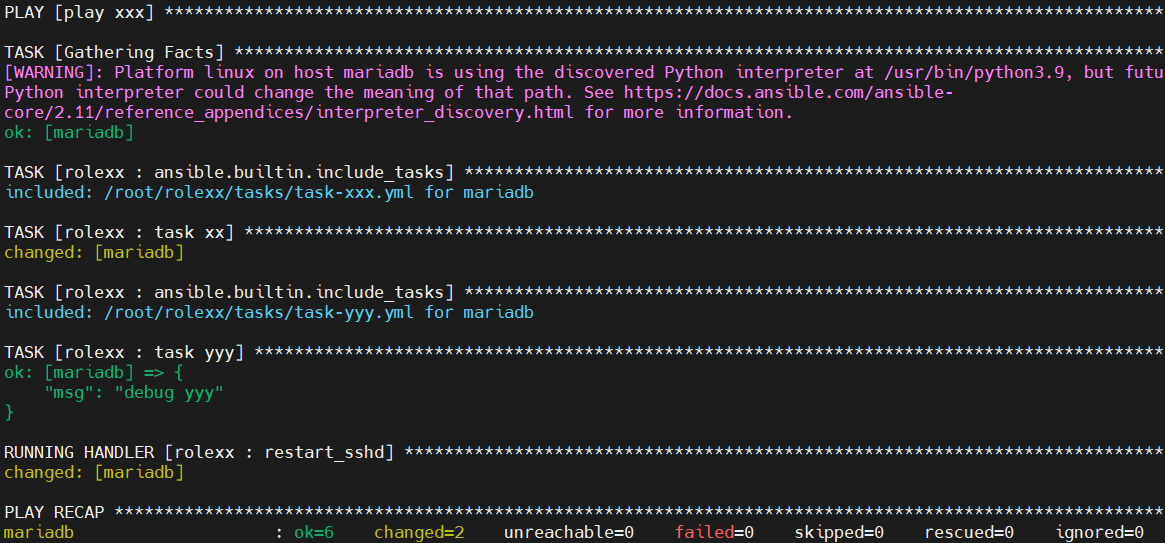
hosts: mariadb

roles: #注意，这个roles有s

- role: rolexx #项目目录名称，也可直接写为 - rolexx ，这个role没有s

EOF

# ansible-playbook -i inventory role-playbook.yml



**★role路径说明**

在role-main.yml的playbook文件里只写了role名称，没有写绝对路径，那么ansible是从哪里找到此角色的呢？

如果ansible.cfg配置文件里未指明**roles\_path=/dir-xxx**，则从以下顺序查找角色目录：

playbook文件所在目录/roles #playbook文件所在目录的roles子目录

~/.ansible/roles #当前用户的家目录下的.ansible/roles子目录

/usr/share/ansible/roles #固定路径

/etc/ansible/roles #固定路径

playbook文件所在目录/ #playbook文件所在目录，本例是在此目录下找的

**# role入口文件里还可写其他不在role里的任务，即一个playbook文件里引用了role后，还可有其他任务**

# cat > role-playbook2.yml <<EOF

---

- name: play xxx

hosts: mariadb

pre\_tasks:

- name: task before roles

ansible.builtin.debug: # ansible<=2.9就只写debug

msg: "在roles之前运行的任务"

roles:

- role: rolexx #项目目录名称（role名称）

tasks: #普通任务默认在roles之后执行

- name: normal tasks after roles

ansible.builtin.debug:

msg: "在roles之后运行的任务"

post\_tasks:

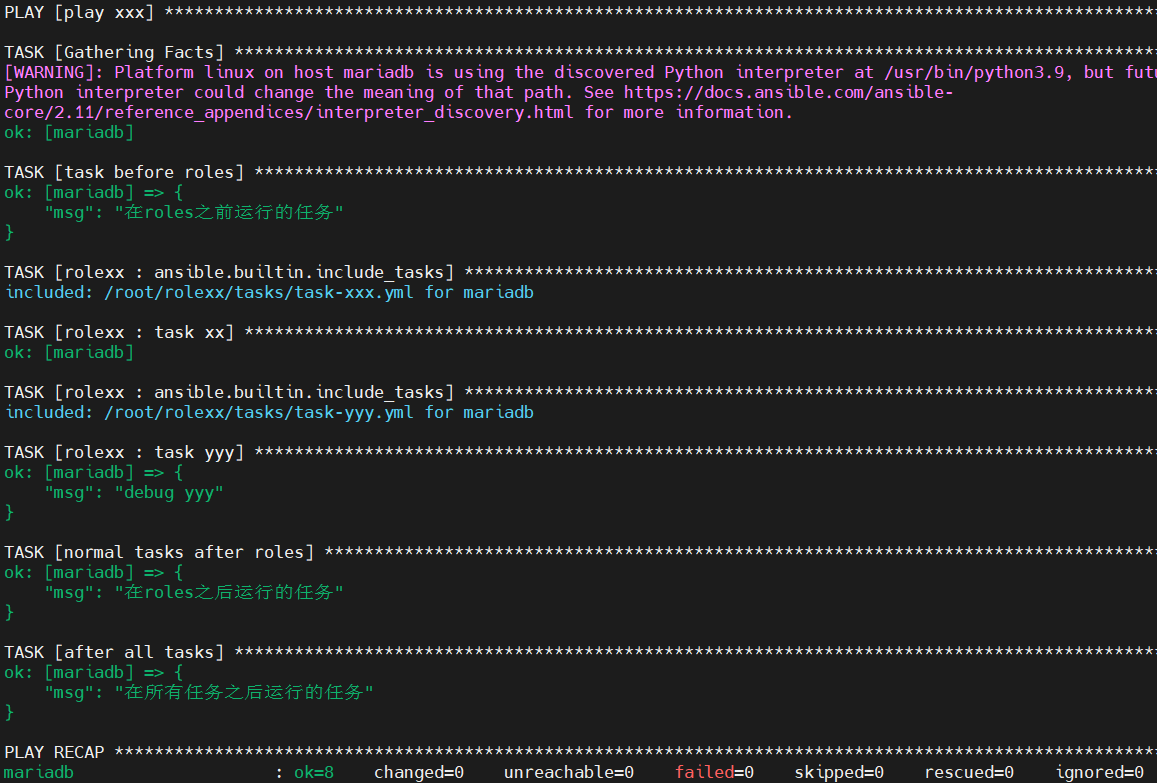
- name: after all tasks

ansible.builtin.debug:

msg: "在所有任务之后运行的任务"

EOF

# ansible-playbook -i inventory role-playbook2.yml



**★role的用法**

不同的主机要执行不同的任务，也有相同的任务应用到若干台主机上

这时需要创建多组任务的role，在一个playbook文件里给不同的主机使用相应的role

# cat test-role-playbook.yml

---

- name: play 1

hosts: web1

role:

- role1

- name: play 2

hosts: web2

role:

- role2

- name: play 3

hosts: web3

role:

- role1

- role2

**★ansible-galaxy的role仓库**

Ansible-Galaxy是一个用于分享ansible的roles及collections的平台，可使用其他人创建好的role，也可发布自己的role，类似helm仓库，ansible-galaxy官网： https://galaxy.ansible.com

ansible>=2.9# ansible-galaxy role --help #查看帮助信息

#2.9版本开始区分type字段，type值有role和collection，

#2.8及以下ansible版本不区分type字段，默认就是role（或者说没有加入collection的功能）

ansible<2.8# ansible-galaxy --help

# ansible-galaxy init roleyy #创建一个名为roleyy的项目（结构目录，子文件），不指定type字段则默认为role

# ansible-galaxy role init roleyy #创建一个名为roleyy的项目，指明type为role（ansible>=2.9）

roleyy

├── defaults

│ └── main.yml

├── files

├── handlers

│ └── main.yml

├── meta

│ └── main.yml

├── README.md

├── tasks

│ └── main.yml

├── templates

├── tests

│ ├── inventory

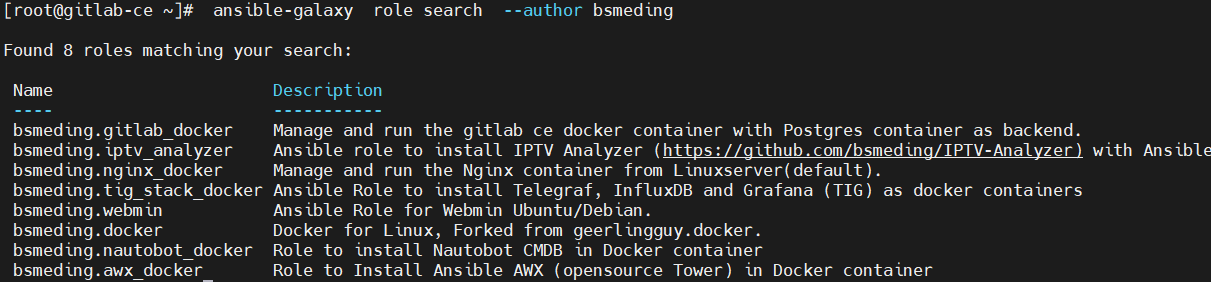
│ └── test.yml

└── vars

└── main.yml

# ansible-galaxy role search --galaxy-tags=database --platforms=EL #在官网搜索role，根据tags及平台搜索

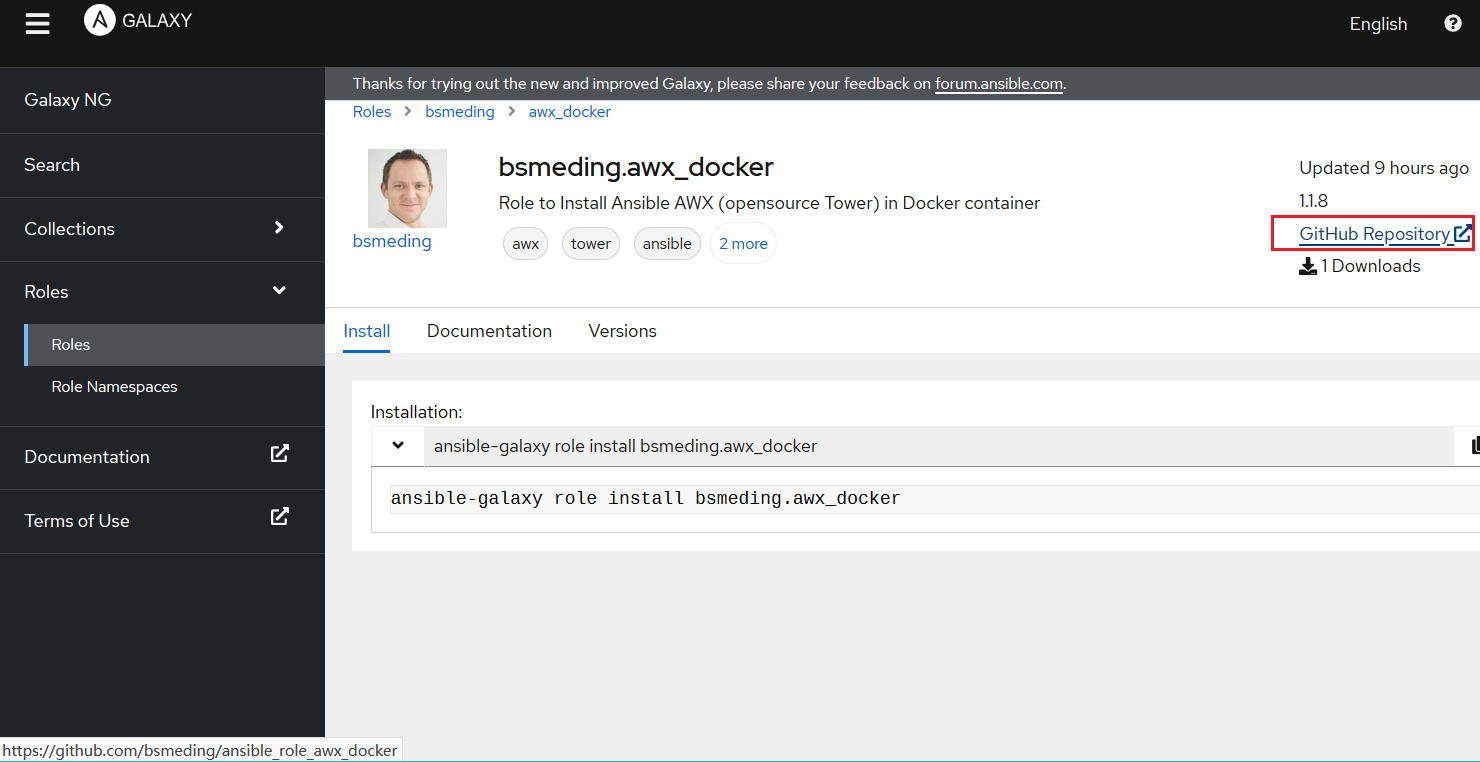
# ansible-galaxy role search --author bsmeding #在官网搜索role，根据作者名称搜索



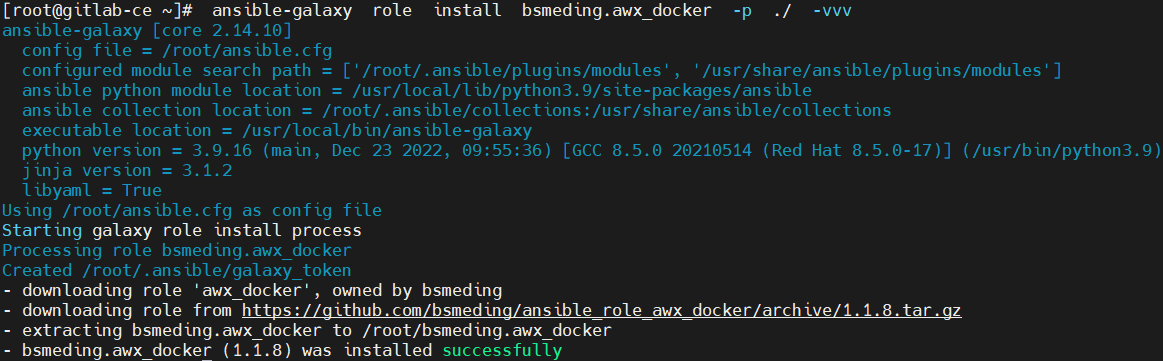
# ansible-galaxy install role-name-xx -p ./ -vvv #下载项目，-p指定保存路径

搜索时是在官网搜索，下载则可能到其他网站去下载了（若服务器无法访问此网站，则下载失败）

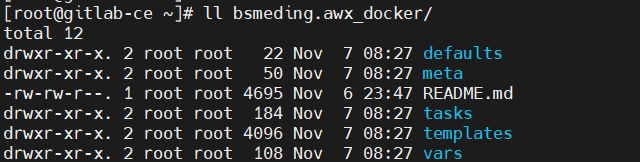
也可直接在官网去搜索，搜索到对应的role里会写有安装命令及仓库地址



# ansible-galaxy role install bsmeding.awx\_docker -p ./ -vvv #ansible>=2.9版本命令



在当前路径下多了一个名为 bsmeding.awx\_docker 的目录（就是刚刚下载的那个role）



若不指定下载路径，则默认保存到：

~/.ansible/roles/bsmeding.awx\_docker

# ansible-galaxy role install rolexx.tar.gz #离线安装role包

Starting galaxy role install process

- extracting rolexx.tar.gz to /root/.ansible/roles/rolexx.tar.gz

- rolexx.tar.gz was installed successfully

# ls -lh /root/.ansible/roles/ #role名称竟然有.tar.gz后缀

drwxr-xr-x. 9 root root 141 Nov 8 19:13 rolexx.tar.gz

# cp rolexx.tar.gz rolexx #将离线压缩包换个名字

# ansible-galaxy role install rolexx #这样子安装才正常

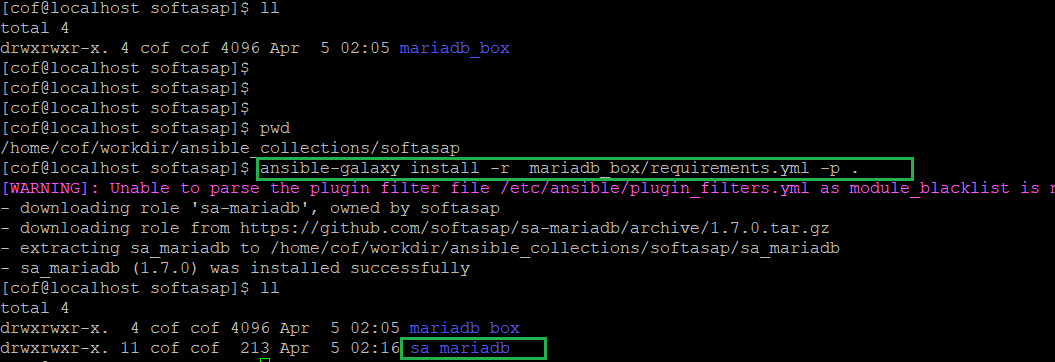
Starting galaxy role install process

- extracting rolexx to /root/.ansible/roles/rolexx

- rolexx was installed successfully

有的role里面有要求安装依赖的其他项目，可先下载其他依赖，比如softasap.mariadb\_box项目里的requirements.yml文件指定了一些依赖，使用以下命令下载依赖的role：

# ansible-galaxy install -r mariadb\_box/requirements.yml -p ./



# ansible-galaxy role list --roles-path=./ #列出指定目录下的role

# ansible-galaxy list --roles-path=./

# /home/cof/workdir #首先去指定的路径下找

- rhel-system-roles.timesync, (unknown version)

- timesync, (unknown version)

# /home/cof/workdir/myroles #最后也还会去ansible配置里指定的路径找

- roles-test, (unknown version)

★识别某目录是否为role角色目录的原理是：

检测此目录下是否有meta子目录及meta/main.yml文件里是否有role描述信息，示例如下：

# cat /root/.ansible/roles/bsmeding.awx\_docker/meta/main.yml

---

dependencies: []

galaxy\_info:

role\_name: awx\_docker

author: bsmeding

description: Role to Install Ansible AWX (opensource Tower) in Docker container

company: "b@rtsmeding.nl"

license: "license (BSD, MIT)"

min\_ansible\_version: '2.10'

platforms:

- name: Fedora

versions:

- all

- name: Debian

versions:

- buster

- bullseye

- bookworm

- name: Ubuntu

versions:

- bionic

- focal

- jammy

- name: Alpine

version:

- all

- name: ArchLinux

versions:

- all

galaxy\_tags:

- awx

- tower

- ansible

- docker

- awx\_version\_17

**★requirements文件内容**

# cat xx-requirements.yml #requirements文件后缀必须为.yml

- src: softasap.sa-mariadb

name: sa\_mariadb

- src: http://test.xxxxx.com/xxxx/yyy.tar

name: yyy-test #下载后的role目录名称可以和下载的tar包名称不一样

**★第10章、collection集合**

role 角色 是把某个任务剧本及用到的参数、模板分类存放到不同子目录下的一个组织结构

collection 集合 是将不同厂商/类型的模块及role分到不同的组（集合）里，是模块及role的组织结构

ansible <= 2.8 ansible-galaxy只有role

ansible ==2.9 ansible-galaxy区分role与collection，只是自带的模块都没有放到collection里

ansible >=2.10 模块被分组到不同的collection里了

**★collection集合内容查看**

# ansible-galaxy collection list #查看所有的collection

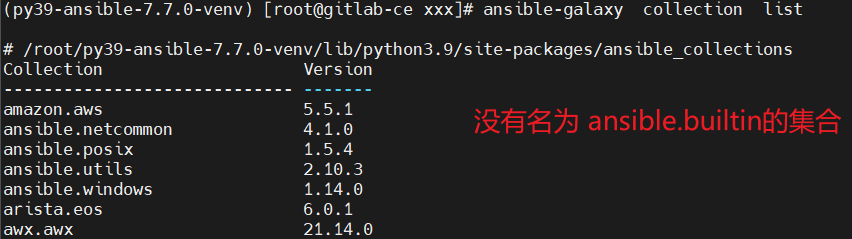
默认在以下路径查找collection（python的site-packages目录下的ansible\_collections子目录）

/usr/local/lib/python3.9/site-packages/ansible\_collections

#如果有python虚拟环境，则在虚拟环境的对应子目录下

path-to-venv/lib/python3.9/site-packages/ansible\_collections

path-to-venv/lib64/python3.9/site-packages/ansible\_collections



ansible>=2.10内置的60多个模块（ansible.builtin.\*）不放在collection里，而是存放在以下路径

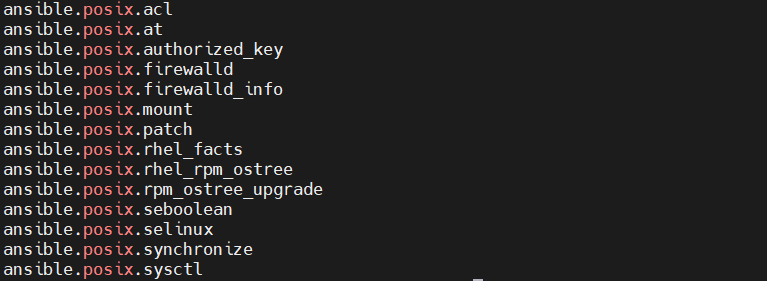
/usr/local/lib/python3.9/site-packages/ansible/modules

#如果有python虚拟环境，则在虚拟环境的对应子目录下

path-to-venv/lib/python3.9/site-packages/ansible/modules/

path-to-venv/lib64/python3.9/site-packages/ansible/modules/

# ansible-doc --list | awk '{print$1}' | grep "ansible\.posix" #查看ansible.posix集合里的模块



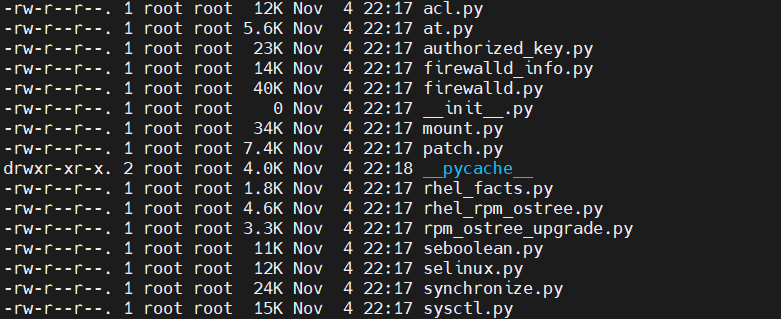
# ls -lh /usr/local/lib/python3.9/site-packages/ansible\_collections/ansible/posix/plugins/modules

#如果有python虚拟环境，则在虚拟环境的对应子目录下

# venv\_dir=~/python3.9-ansible-4.10.0

# ls -lh ${venv\_dir}/lib64/python3.9/site-packages/ansible\_collections/ansible/posix/plugins/modules

#可见ansible的模块就是python脚本，由对应的"模块名称.py"脚本去解析模块的参数并实现相应功能



**★创建并使用collection**

collection的名称必须有2级，且不能有减号，形如：命名空间.collection名称 否则会报错：

ERROR! Invalid collection name 'collection-yy', name must be in the format <namespace>.<collection>.

Please make sure namespace and collection name contains characters from [a-zA-Z0-9\_] only.

# ansible-galaxy collection init cof.collectionyy #创建名为cof.collectionyy的集合，必须指明type为collection

# tree cof #创建集合后，默认在当前路径下创建了cof目录

cof

└── collectionyy

├── docs

├── galaxy.yml

├── plugins

│ └── README.md

├── README.md

└── roles

此集合的信息保存在collection目录下的galaxy.yml文件里

# vi cof/collectionyy/galaxy.yml

namespace: cof #不可修改

name: collectionyy #不可修改

version: 1.0.0 #可自定义

readme: README.md #可自定义，一般不修改

authors:

- your name <example@domain.com> #作者

description: your collection description #描述信息

license:

- GPL-2.0-or-later #许可协议

dependencies: {} #依赖的其他collection

# mkdir cof/collectionyy/meta/ #集合要求有runtime信息，即指定的ansible版本，先创建meta子目录

# cat > cof/collectionyy/meta/runtime.yml <<EOF

---

requires\_ansible: '>=2.9'

EOF

# ansible-galaxy collection list -p ~/ #查看collection

在当前路径下查找，结果没找到我们刚刚创建的collection

原因是ansible找集合时，还要多找一层子目录，-p后指定的路径下层的ansible\_collections这个子目录下去找

所以先创建此目录，再将原先创建的cof目录整个移动到./ansible\_collections下面

# mkdir ~/ansible\_collections

# mv cof ~/ansible\_collections #将整个cof目录移动到./ansible\_collections下面

# ansible-galaxy collection list -p ~/ #查看collection

# /root/ansible\_collections

Collection Version

---------------- -------

cof.collectionyy 1.0.0

**★创建collection的角色**

#下面示例为在集合中创建一个名为rolexx的角色

# cd ~/ansible\_collections/cof/collectionyy/roles

# mkdir -p rolexx/{files,handlers,tasks,templates,vars} #创建role项目目录及需要用的子目录

#在tasks子目录里写所有任务清单文件

# cat > rolexx/tasks/task-yyy.yml <<EOF

- name: task yyy

ansible.builtin.debug: # ansible<=2.9就只写debug

msg: this is cof.collectionyy.rolexx debug yyy

EOF

#创建tasks入口文件，指定任务顺序（文件名称必须为main.yml）

# cat > rolexx/tasks/main.yml <<EOF

- ansible.builtin.include\_tasks: task-yyy.yml # ansible<=2.9就只写include\_tasks

EOF

**★编写playbook剧本，使用cof.collectionyy集合里的角色**

# cd ~/

# cat > ansible.cfg <<EOF

[defaults]

inventory=./inventory

host\_key\_checking=false

deprecation\_warnings=False

#指定集合路径为 ~/ 才找得到我们前面创建的集合cof.collectionyy

collections\_paths=~/

EOF

# cat > inventory <<EOF

mariadb

EOF

# cat > testcollection.yml <<EOF

---

- name: use collection

hosts: mariadb

collections:

- cof.collectionyy #导入collection

tasks:

- name: use role of collection

import\_role: #导入collection里的role

name: rolexx

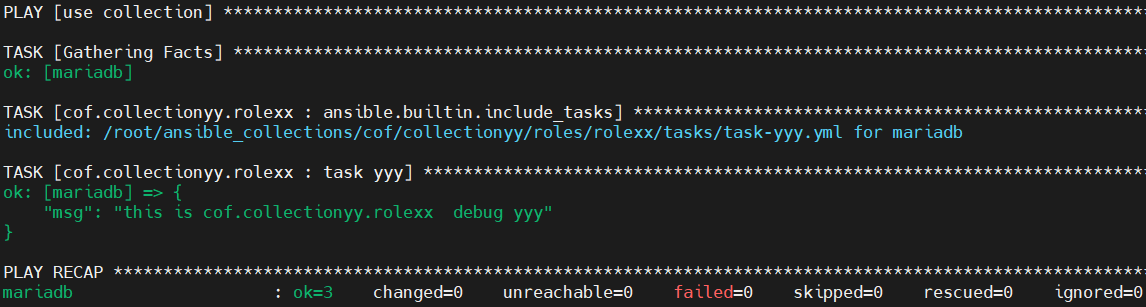
#- name: use module of collection

# 模块名: #直接使用collection里的模块，也可写模块的FQCN名称

# 参数xx: 值xx

EOF

# ansible-playbook testcollection.yml



**★在collection里编写自己的模块**

在自己集合里编写模块，实现某些功能（首先参考上一小节，创建collection）

# mkdir ~/ansible\_collections/cof/collectionyy/plugins/modules/

# vi ~/ansible\_collections/cof/collectionyy/plugins/modules/change\_dns.py #文件名同模块名，内容如下

#!/usr/bin/env python  
# -\*- coding: utf-8 -\*-  
DOCUMENTATION = r'''  
---  
module: change\_dns  
description:   
 - "描述信息1"  
author:   
 - cof-lee  
'''  
EXAMPLES = r'''  
- name: change nameserver  
 cof.collectionyy.change\_dns:  
 resolv\_conf\_file: /etc/resolv.conf  
 nameserver: "114.114.114.114"  
 state: present  
'''  
from ansible.module\_utils.basic import AnsibleModule  
  
module = AnsibleModule(  
 argument\_spec=dict(  
 resolv\_conf\_file=dict(type='str', required=True, aliases=['filename']),  
 nameserver=dict(type='str', required=True),  
 state=dict(type='str', default='present', choices=['absent', 'present']),  
 ),  
 supports\_check\_mode=True,  
)  
  
resolv\_conf\_file = module.params['resolv\_conf\_file']  
nameserver = module.params['nameserver']  
result = {}  
  
try:  
 fileobj = open(resolv\_conf\_file, "w+")  
 fileobj.write("nameserver " + nameserver + "\n") #核心在这里，打开文件并写入内容  
 result['msg'] = "change dns done"  
 result['changed'] = True  
 fileobj.close()  
 module.exit\_json(\*\*result)  
except Exception as err: #如果出错，则任务执行失败  
 error = err  
 result['msg'] = str(err)  
 result['failed'] = True  
 module.fail\_json(\*\*result)

# cat > ~/testcollection2.yml <<EOF

---

- name: use cof.collectionyy

hosts: mariadb

collections:

- cof.collectionyy #导入collection

tasks:

- name: use module of collection

change\_dns: #如果前面不导入collection，则这里写为cof.collectionyy.change\_dns

resolv\_conf\_file: /etc/resolv.conf

nameserver: "114.114.114.114"

register: result\_msg

ignore\_errors: true

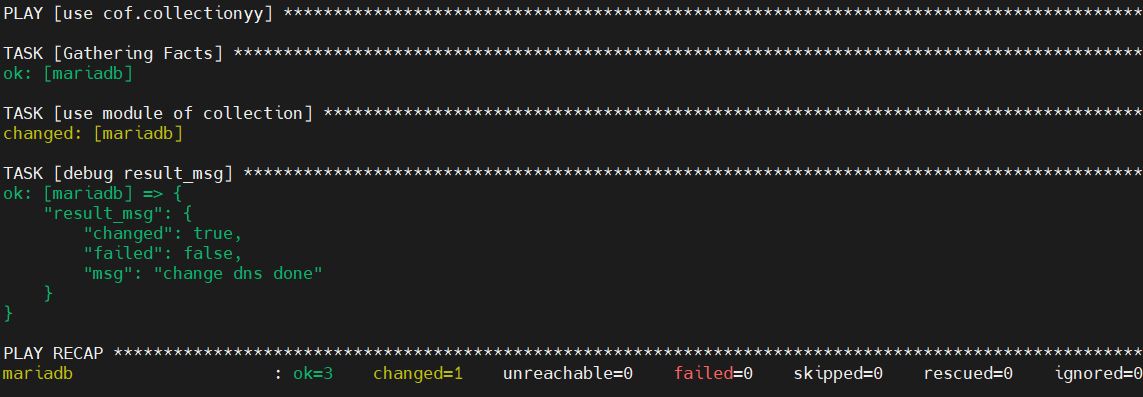
- name: debug result\_msg

ansible.builtin.debug: # ansible<=2.9就只写debug

var: result\_msg

EOF

# ansible-playbook ~/testcollection2.yml

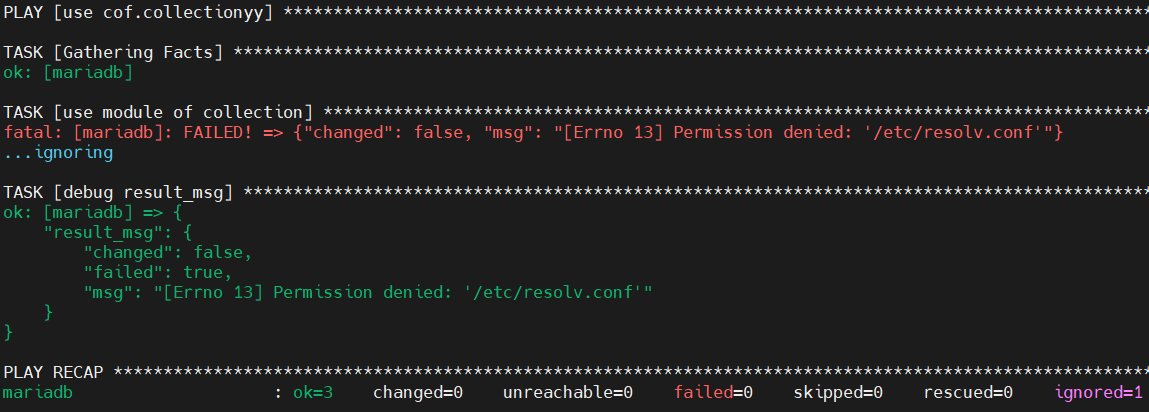




★如果mariadb主机上的/etc/resolv.conf文件没有写权限，则会报错，效果如下：

mariadb# chattr +i /etc/resolv.conf

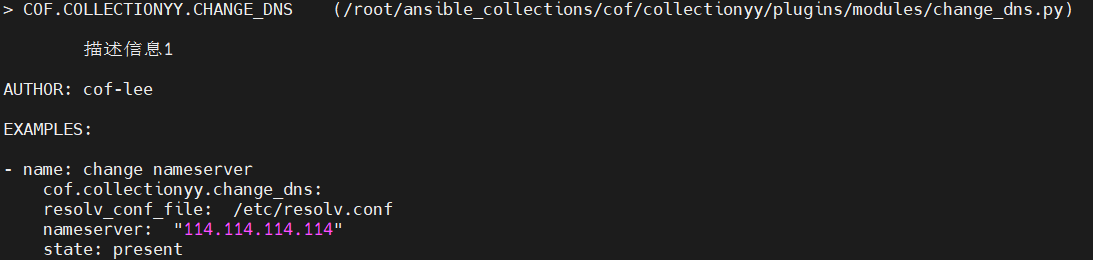
执行机# ansible-playbook ~/testcollection2.yml



# ansible-doc --list | grep change\_dns

cof.collectionyy.change\_dns ...

# ansible-doc cof.collectionyy.change\_dns

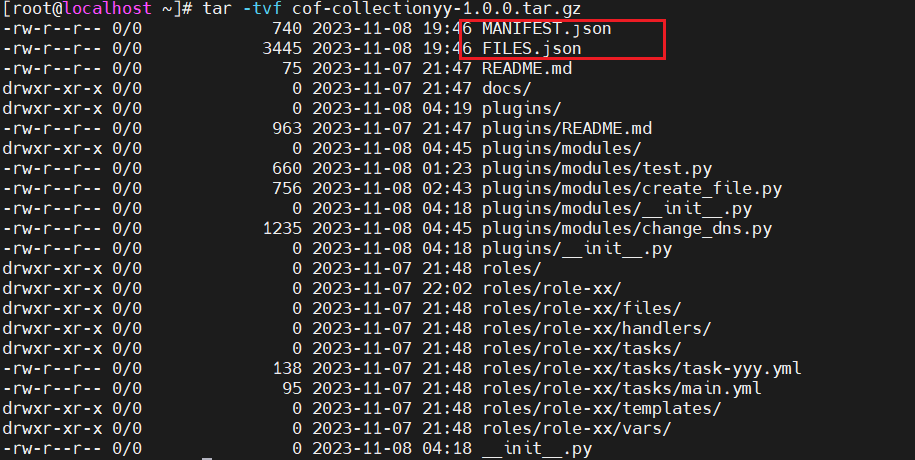


**★构建collection离线包**

前面几小节我们创建的cof.collectionyy集合，只能在本地用，就算打包就.tar.gz压缩包，也无法安装，因为里面没有必须的MANIFEST.json及FILES.json文件。需要将集合发布出去（构建发布后的安装包）

# ansible-galaxy collection build ~/ansible\_collections/cof/collectionyy/ #构建可发布的collection

Created collection for cof.collectionyy at /root/cof-collectionyy-1.0.0.tar.gz #自动生成压缩包



构建后的发布包里不含 collection目录下的galaxy.yml文件！

然后将发布后的集合压缩包复制到其他ansible执行机上去使用

# ansible-galaxy collection install cof-collectionyy-1.0.0.tar.gz #安装collection离线包

Starting galaxy collection install process

Process install dependency map

Starting collection install process

Installing 'cof.collectionyy:1.0.0' to '/root/ansible\_collections/cof/collectionyy'

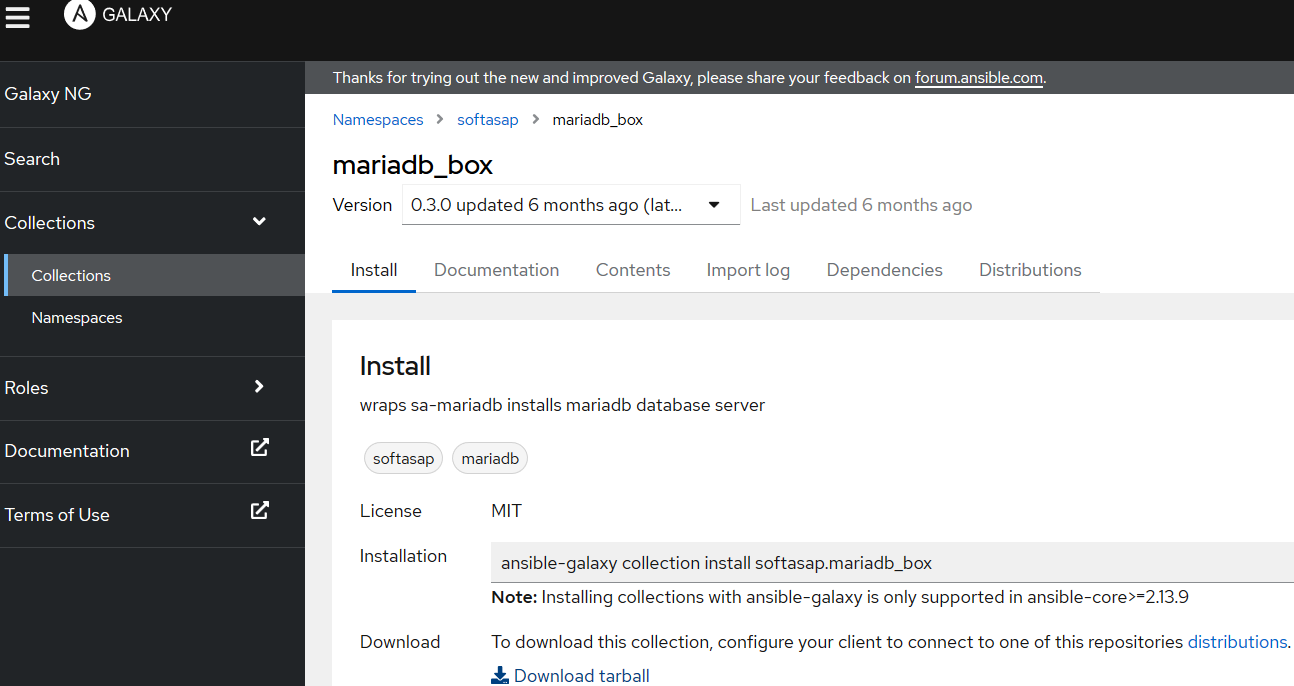
cof.collectionyy:1.0.0 was installed successfully

**★ansible-galaxy的collection仓库**

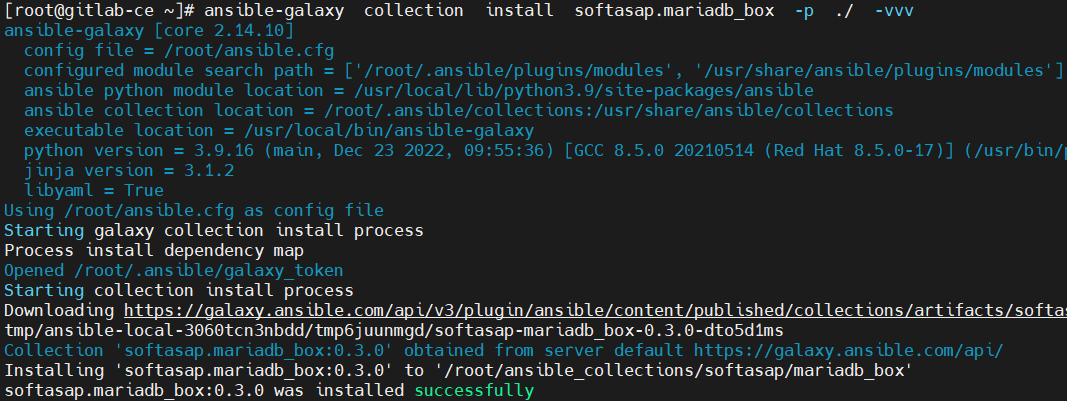
Ansible-Galaxy是一个用于分享ansible的roles及collections的平台，可使用其他人创建好的collection，也可发布自己的collection，类似helm仓库，ansible-galaxy官网： https://galaxy.ansible.com

如果要从galaxy官网下载内容，要求使用ansible-core>=2.13.9的版本（对应ansible>=6.x）

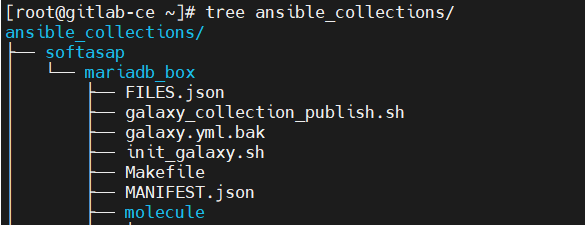
首先在官网搜索需要的collection，搜索到对应的collection里写有安装命令



# ansible-galaxy collection install softasap.mariadb\_box -p ./ -vvv #2.9版本命令



在当前路径下的多了一个ansible\_collections目录，用于存放collection的，其下多了一个名为softasap/mariadb\_box的子目录（就是刚刚下载的那个collection）



若不指定-p下载路径，则默认保存到：

~/.ansible/collections/ansible\_collections/softasap/mariadb\_box

**★requirements文件内容**

有些项目会依赖一些collection，可以将依赖的集合写在requirements.yml文件里

# cat xx-requirements.yml #requirements文件后缀必须为.yml

---

collections:

-name: namespaceYY.collectionYY

version: x.y.z

# ansible-galaxy collection install -r xx-requirements.yml

**★发布collection到ansible-galaxy官网**

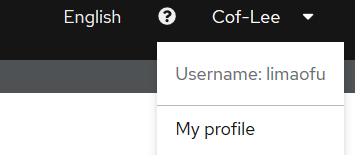
首先在本地创建collection并发布为.tar.gz压缩包

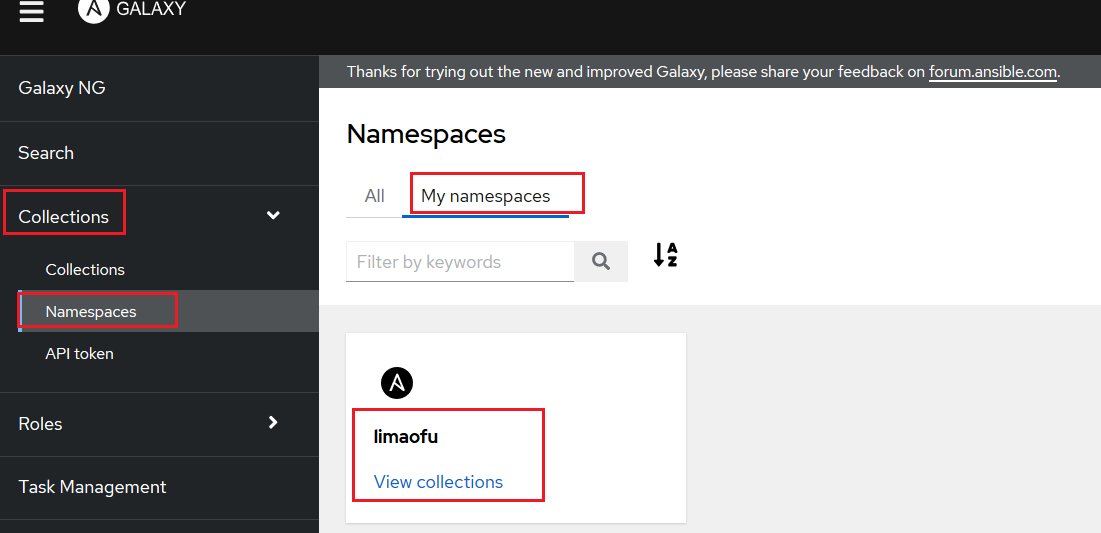
命名空间为自己在ansible-galaxy网站上的命名空间，集合名称随便写，比如作者已创建了一个collection：

# ansible-galaxy collection build ~/ansible\_collections/limaofu/collectionyy/

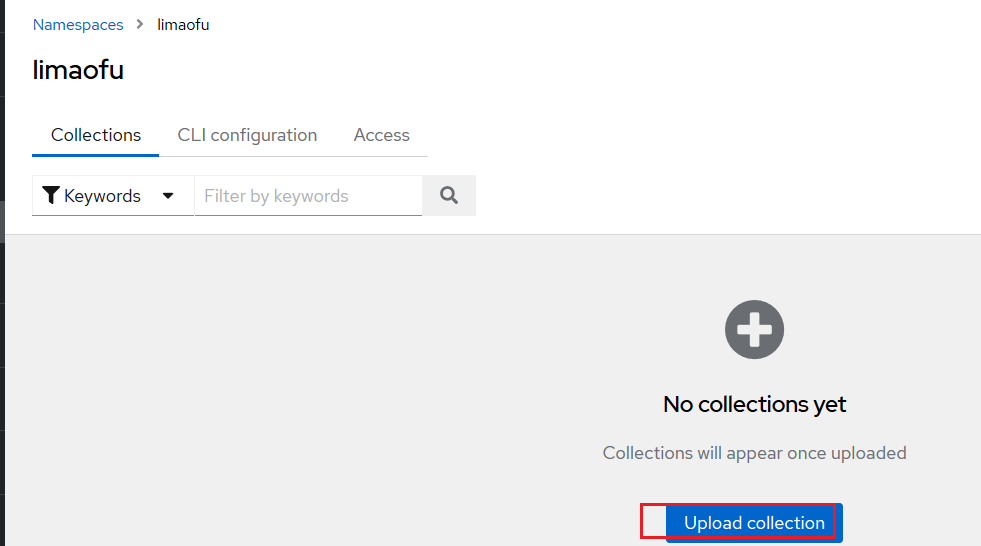
Created collection for limaofu.collectionyy at /root/limaofu-collectionyy-1.0.0.tar.gz

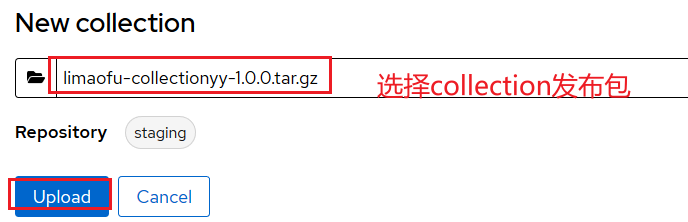
然后登录ansible-galayx官网 https://galaxy.ansible.com ，登录我的账号：limaofu

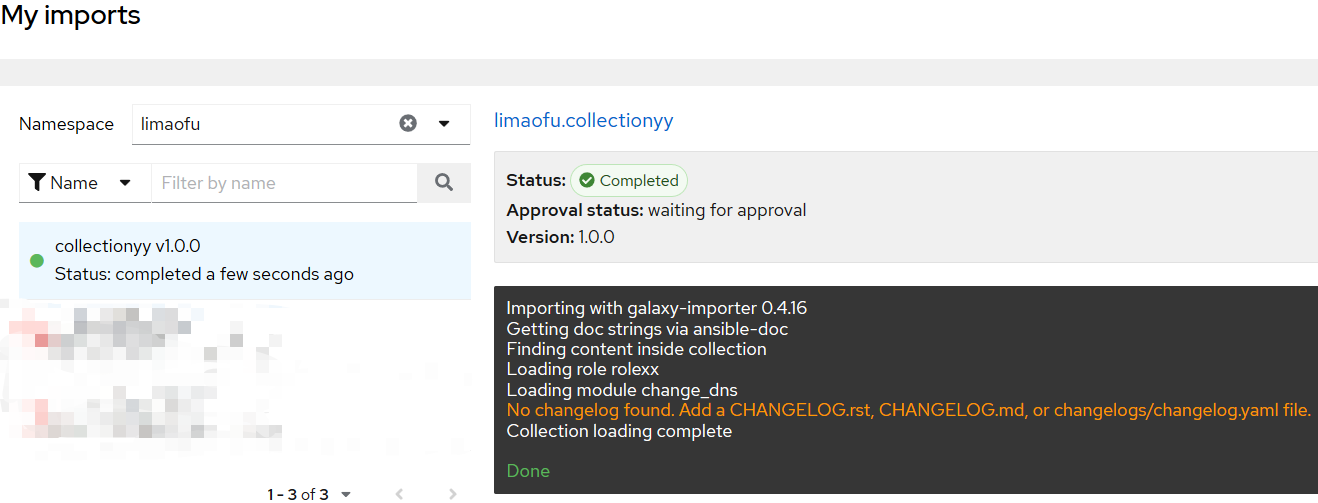


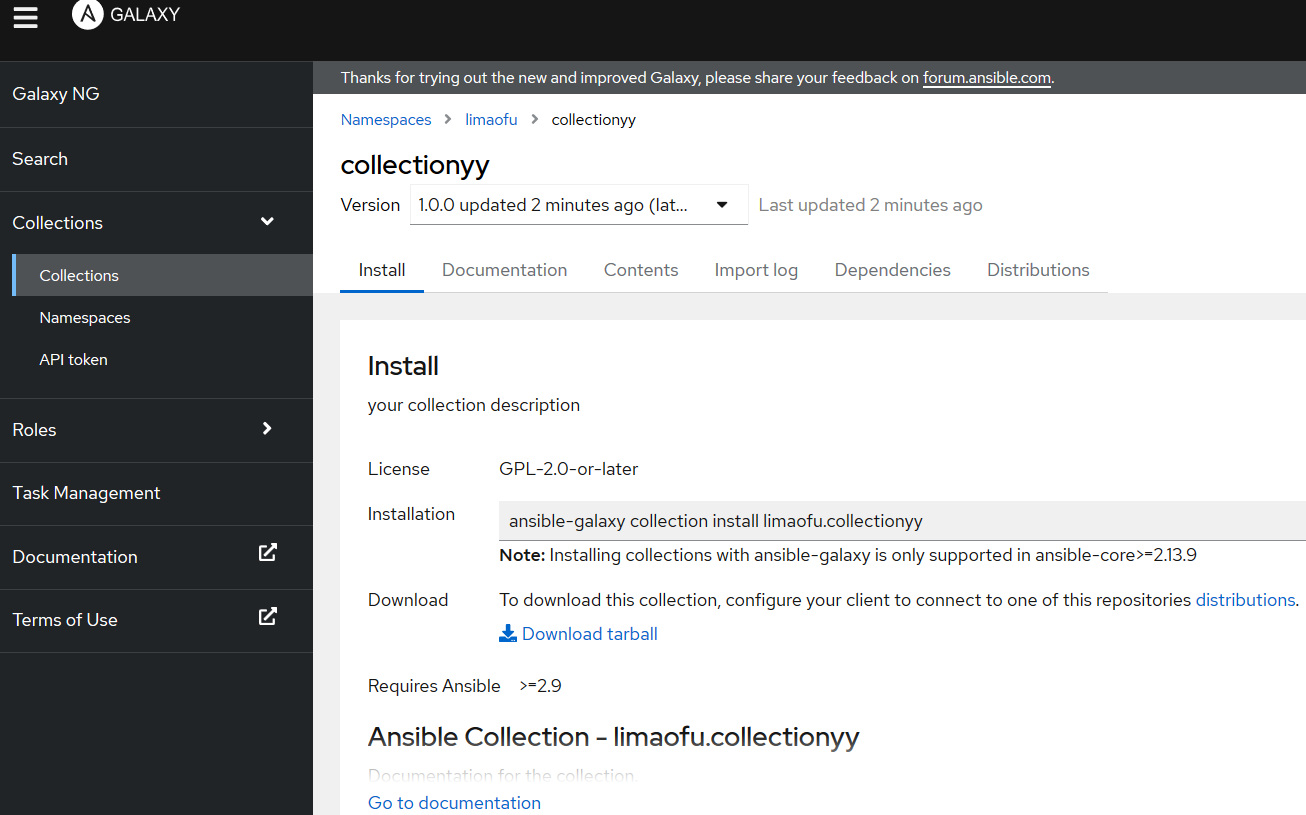


上传collection压缩包到网站上：









发布成功后，所有人都能查找到此集合了

**★collection的使用**

# ansible-galaxy collection install namespaceXX.collectionXX

# ansible-galaxy collection install limaofu.collectionyy #安装集合，此集合已发布到ansible-galaxy官网

# ansible-galaxy collection install limaofu.collectionyy:==1.0.0 #安装集合，指定版本号

Starting galaxy collection install process

Process install dependency map

Starting collection install process

Downloading https://galaxy.ansible.com/api/v3/plugin/ansible/content/published/collections/artifacts/limaofu-collectionyy-1.0.0.tar.gz to /root/.ansible/tmp/ansible-local-2081dkCcfF/tmpxKd\_Lu/limaofu-collectionyy-1.0.0-HVUnm6

Installing 'limaofu.collectionyy:1.0.0' to '/root/.collections/ansible\_collections/limaofu/collectionyy'

limaofu.collectionyy:1.0.0 was installed successfully

# cat > testcollection-yy.yml <<EOF

---

- name: use collection

hosts: mariadb

collections:

- limaofu.collectionyy #导入collection

tasks:

- name: use role of collection

import\_role: #导入collection里的role

name: rolexx

- name: use module of collection

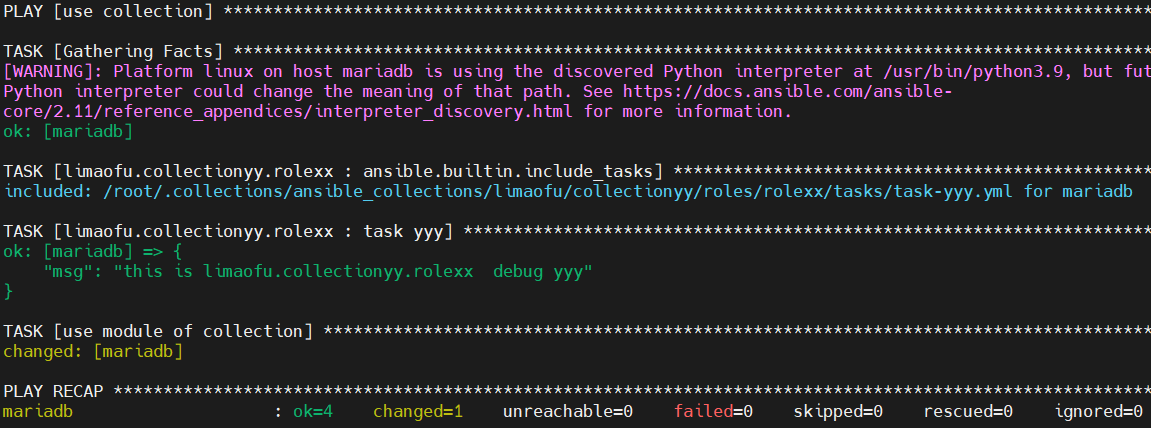
change\_dns: #直接使用collection里的模块，也可写模块的FQCN名称

resolv\_conf\_file: /etc/resolv.conf

nameserver: "114.114.114.114"

EOF

# ansible-playbook testcollection-yy.yml



**★第11章、ansible-navigator**

ansible-navigator是使用了容器的思想，把我们本地的python虚拟环境及相应的ansible模块、集合、角色等内容都打包到一起，做成一个容器镜像，然后在容器里运行ansible-playbook剧本

**★ansible自动化运维技术演进**

ansible --> ansible-navigator --> ansible-AWX

|  |  |
| --- | --- |
| ansible | 是个本地化 的命令行工具 |
| ansible-navigator | 打包整个ansible/python环境到容器镜像里了 |
| ansible-AWX/AAP平台 | 是一个常驻后台的服务，提供可视化界面，可定时/周期自动执行ansible剧本任务，并提供邮件通知等功能 |

**★安装ansible-navigator**

和ansible-core一样，建议使用pip工具安装ansible-navigator

# pip3 install ansible-navigator

# ansible-navigator --version

ansible-navigator 24.2.0

ansible-navigator运行剧本时会启动容器，所以需要安装一个容器运行时

# yum install podman -y

**★ansible-navigator运行playbook剧本**

★在工作目录下创建ansible.cfg及inventory文件

# mkdir ~/test

# cd ~/test

# cat > ansible.cfg <<EOF

[defaults]

inventory = ./inventory

host\_key\_checking = False

[privilege\_escalation]

become=True

become\_method=sudo

become\_user=root

become\_ask\_pass=False

EOF

# cat > inventory <<EOF

mariadb

EOF

# cat > play-testxx.yml <<EOF

---

- name: play testxx

hosts: mariadb

tasks:

- name: task xx

ansible.builtin.debug:

msg: this is test task xx

EOF

★使用ansible-navigator执行剧本

# ansible-navigator run play-testxx.yml -i inventory -m stdout

#run后必须先跟剧本文件，

#-i指定主机清单文件，默认为ansible.cfg里指定的inventory文件，若未指定则默认为/etc/ansible/hosts

#-eei指定执行环境镜像，默认是需要联网去拉取镜像的，默认镜像为：ghcr.io/ansible/creator-ee:v0.22.0

#可以将镜像下载到本地，然后修改ansible-navigator.yml配置为：优先使用本地的镜像

# -m stdout表示以标准输出方式显示结果，默认进入交互模式界面

**★使用本地镜像仓库**

# cat > /etc/containers/registries.conf <<EOF

unqualified-search-registries = [ "cof-lee.com" ]

[[registry]]

prefix = "cof-lee.com"

location = "cof-lee.com"

insecure = true

EOF

#navigator配置文件指定默认使用的镜像及镜像获取方式，以及执行playbook输出模式为stdout非交互式

# cat > ./ansible-navigator.yml <<EOF

---

ansible-navigator:

mode: stdout #指定为非交互模式，默认是交互模式interactive

execution-environment:

image: cof-lee.com/eei/creator-ee:v0.22.0 #指定默认的eei执行环境镜像

pull:

policy: missing #表示优先用要地的，本地没有则网络拉取；always表示只从网络拉取镜像的，默认

EOF

# mode使用 stdout（非交互）配置时，如果要以交互模式运行，需要带上参数 -m interactive

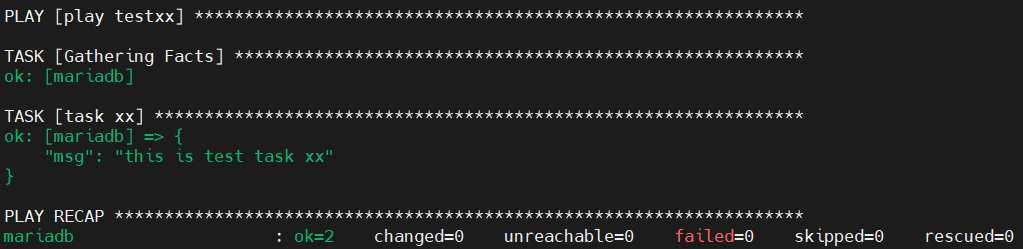
# mode使用 interactive（交互）配置时，如果要以非交互模式运行，需要带上参数 -m stdout

#红帽的镜像有 registry.redhat.io/ansible-automation-platform-22/ee-supported-rhel8:latest

#通用的镜像有 quay.io/ansible/ansible-runner:latest

**★以标准输出模式执行playbook**

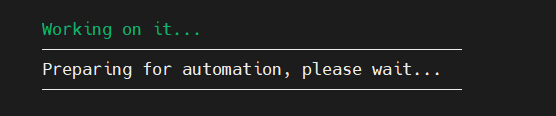
# ansible-navigator run play-testxx.yml -i inventory --eei cof-lee.com/eei/creator-ee:v0.22.0 -m stdout



和执行ansible-playbook命令一样，直接输出运行结果，不需要人工干预

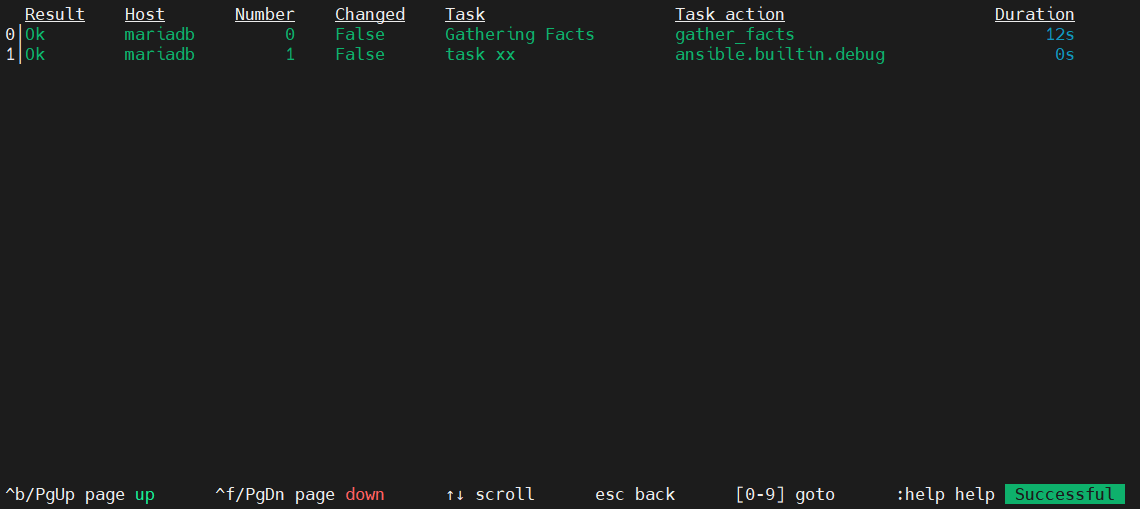
**★以交互模式执行playbook**

# ansible-navigator run play-testxx.yml -i inventory -m interactive #未显示指定eei时，使用配置文件里的eei执行环境镜像cof-lee.com/eei/ee-supported-rhel8:latest

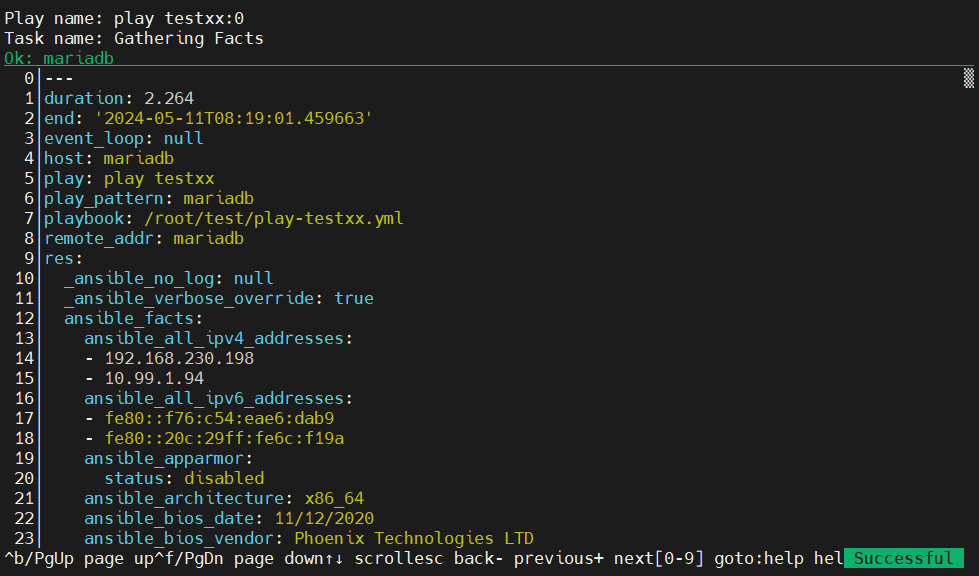




默认进入交互模式，按下数字0，进入序号为0的playbook详情页（0-9可直接按相应数字键进入，10以上需要先按下:冒号再加数字回车）



按下相应数字，查看任务详情，比如按下0查看gather\_facts详情：



可按下加号+查看下一页，按Esc键返回上一级，直到最上级后，再次按Esc则退出ansible-navigator交互界面

**★ansible-navigator运行原理**

ansible-navigator运行剧本时，首先使用podman创建一个临时容器，使用的镜像称为eei（Execution Environment Image）执行环境镜像（镜像里包含一套完整的ansible工具命令）

然后在容器里使用ansible-playbook命令执行指定的playbook剧本，剧本执行结束后，结束相应的容器

# podman ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

ced6a34b9aa8 cof-lee.com/eei/creator-ee:v0.22.0 ansible-playbook ... 3 seconds ago Up 2 seconds ansible\_runner\_3031d0d3-8819-477b-ad2b-f4e9b9c22ee1

宿主机（执行机）

# podman inspect ced6a34b9aa8 | grep Pid

"Pid": 17039,

"ConmonPid": 17031,

# ln -s /proc/17039/ns/net /var/run/netns/ced6a34b9aa8

# ip netns exec ced6a34b9aa8 ip addr #查看容器的ip

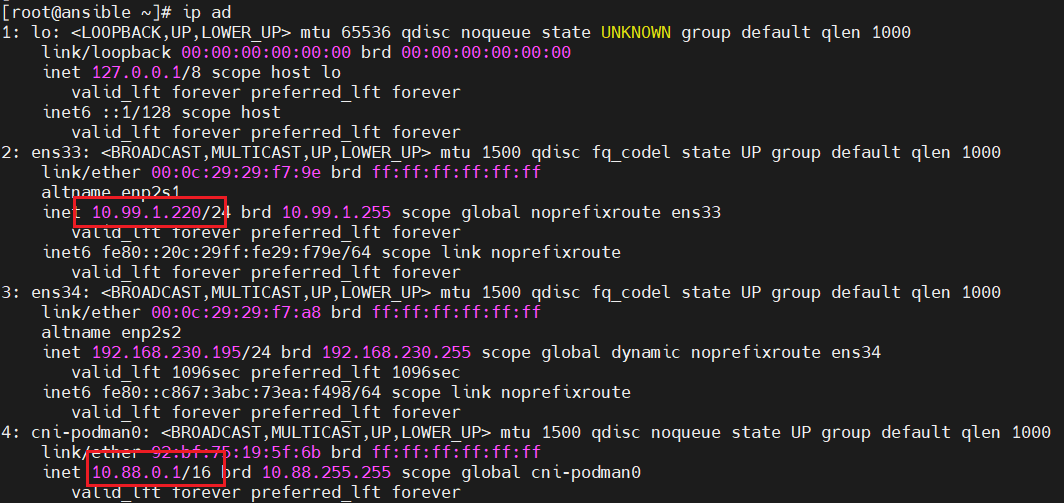
2: eth0@if28: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc noqueue state UP group default

link/ether 92:79:7b:ec:be:85 brd ff:ff:ff:ff:ff:ff link-netnsid 0

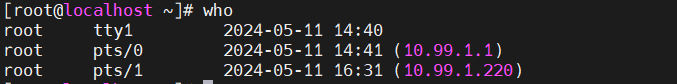
inet 10.88.0.25/16 brd 10.88.255.255 scope global eth0

valid\_lft forever preferred\_lft forever

宿主机（执行机）# ip addr #执行机的ip为10.99.1.86，容器网段10.88.0.0/16



登录到受管节点查看，下图可见当容器访问外部节点时，做了nat地址转换，转为执行机的ip了



ansible-navigator执行时，自动将执行机（宿主节点）的当前工作路径映射到eei容器里，以及当前用户的~/.ssh路径也映射到容器里了

# podman inspect 容器id #查看目录映射情况

"HostConfig": {

"Binds": [

"/tmp/ansible-navigator\_q64mmldm/artifacts:/runner/artifacts/:rw,rprivate,rbind",

"/tmp/ansible-navigator\_q64mmldm:/runner/:rw,rprivate,rbind",

"/root/test:/root/test/:rw,rprivate,rbind", #当前工作路径

"/tmp/ssh-I45OMkJzGDWy:/tmp/ssh-I45OMkJzGDWy/:rw,rprivate,rbind",

"/root/.ssh:/home/runner/.ssh/:rw,rprivate,rbind", #映射ssh密钥路径

"/root/.ssh:/root/.ssh/:rw,rprivate,rbind"

],

#执行机的/etc/hosts文件内容也添加到了容器里的/etc/hosts里

sh-4.4# more /etc/hosts

127.0.0.1 localhost localhost.localdomain localhost4 localhost4.localdomain4

::1 localhost localhost.localdomain localhost6 localhost6.localdomain6

10.99.1.86 controller.cof-lee.com controller

10.99.1.238 hub.cof-lee.com hub

10.99.1.246 harbor.cof-lee.com cof-lee.com cof-net.com

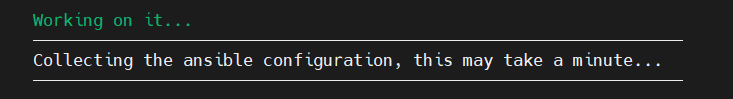
10.99.1.220 host.containers.internal

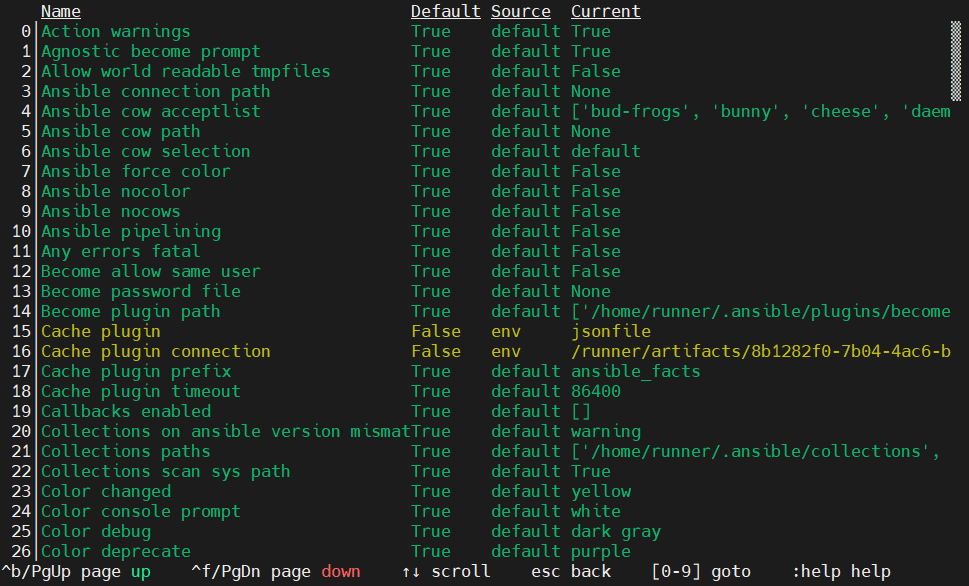
10.0.2.100 b79fdf7c6d36 ansible\_runner\_d09821d2-8146-4122-845e-f545f11efbfb

**★ansible-navigator配置**

# ansible-navigator config dump -m stdout #以标准输出方式列出所有配置项（也会创建临时eei环境）

# ansible-navigator config -m interactive #进入交互界面，查看配置项



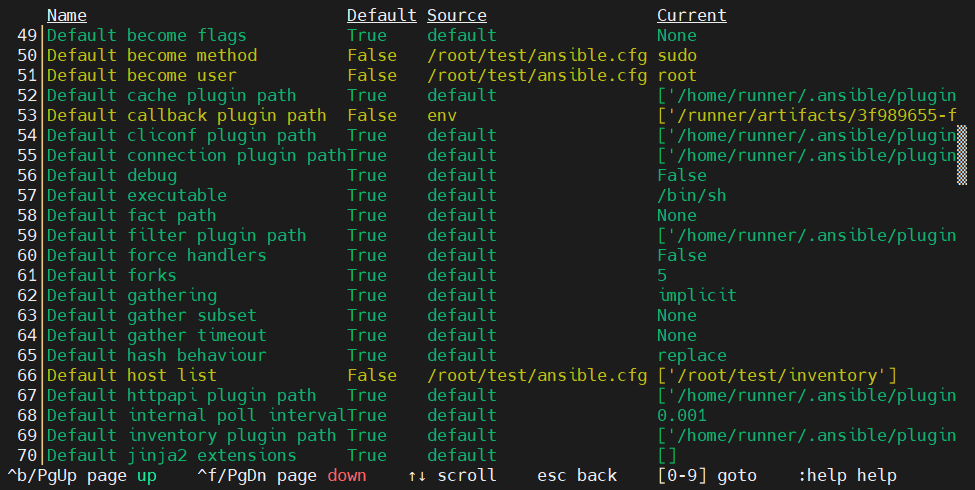


可按下冒号:f 加配置项，查看对应的配置信息，这个f表示filter（过滤，查找）例如：

:f forks #查看含有名为forks的配置项







如上图，最左侧数字表示行数，接下来各列说明：

Name这列表示配置项名称，

Default这列表示是否使用默认配置

Source这列表示配置的来源，值为default表示使用的是默认配置，其他为配置源文件或来源

Current这列表示该配置项的当前生效值

**★使用本地镜像仓库及输出模式为stdout非交互式**

# cat > /etc/containers/registries.conf <<EOF

unqualified-search-registries = [ "cof-lee.com" ]

[[registry]]

prefix = "cof-lee.com"

location = "cof-lee.com"

insecure = true

EOF

#navigator配置文件指定默认使用的镜像及镜像获取方式，以及执行playbook输出模式为stdout非交互式

# cat > ./ansible-navigator.yml <<EOF

---

ansible-navigator:

mode: stdout #指定为非交互模式，默认是交互模式interactive

execution-environment:

image: cof-lee.com/eei/ansible-runner:latest #指定默认的eei执行环境镜像

pull:

policy: missing #表示优先用要地的，本地没有则网络拉取；always表示只从网络拉取镜像的，默认

EOF

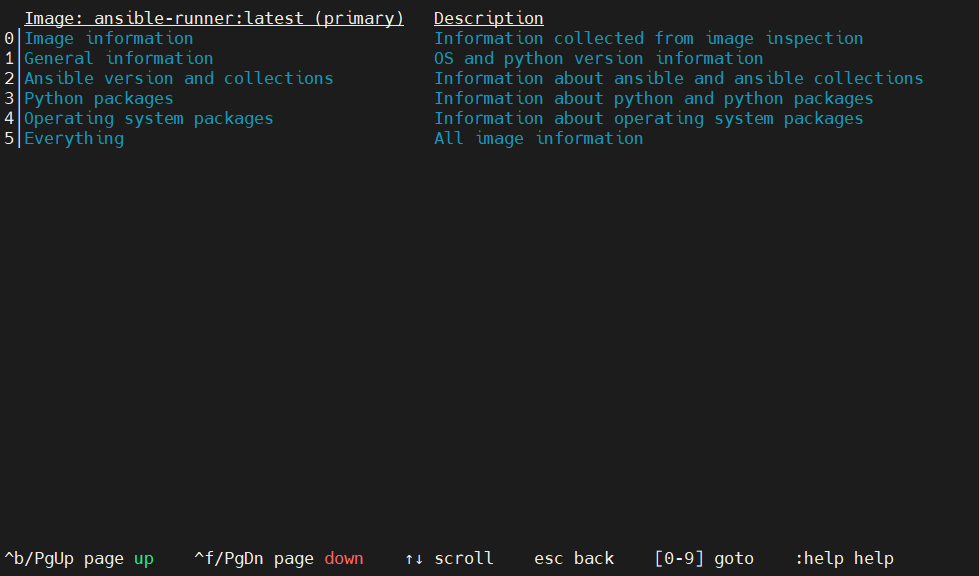
# mode使用 stdout（非交互）配置时，如果要以交互模式运行，需要带上参数 -m interactive

# mode使用 interactive（交互）配置时，如果要以非交互模式运行，需要带上参数 -m stdout

**★构建EEI执行环境镜像**

★假如需要基于'quay.io/ansible/ansible-runner:latest'镜像，安装某个role/collection，再生成新的镜像：

# ansible-navigator images --eei quay.io/ansible/ansible-runner:latest #查看基础镜像的信息



★创建用于构建镜像的清单文件

# mkdir ~/test

# cd ~/test

# cat > execution-environment.yml <<EOF

---

version: 1

build\_arg\_defaults:

EE\_BASE\_IMAGE: 'quay.io/ansible/ansible-runner:latest' #基于此镜像进行自定义添加新功能

EE\_BUILDER\_IMAGE: 'quay.io/ansible/ansible-builder:latest' #此镜像用于builder操作

dependencies:

galaxy: requirements.yml # galaxy表示需要安装ansible-galaxy的内容，可为role及collection【可选参数】

python: requirements.txt # python表示需要安装的python包【可选参数】

system: bindep.txt # system表示需要安装软件包，可为rpm软件包【可选参数】

EOF

# registry.redhat.io/ansible-automation-platform-22/ee-supported-rhel8:latest #红帽的基础镜像

# registry.redhat.io/ansible-automation-platform-22/ansible-builder-rhel8:latest #红帽的builder镜像

# cat > requirements.yml <<EOF

collections:

- name: /build/limaofu-collectionyy-1.0.0.tar.gz #依赖的collection，或者说要安装的collection

EOF

#说明，因为构建镜像时，是将context/\_build/映射到容器里的/build，且将/build设置为容器的WORKDIR，所以依赖的文件路径只能以/build目录开头

# cat > bindep.txt <<EOF

python36 [platform:rpm]

python3-pip [platform:rpm]

EOF

# cat > requirements.txt <<EOF

paramiko

EOF

# ansible-builder create #创建构建配置信息，会自动生成context目录

Complete! The build context can be found at: /root/test/context

# tree context/ #查看context目录结构

context/

├── \_build

│ └── bindep.txt #这个文件就是我们前面写的system软件清单，自动复制过来了

│ └── requirements.txt #这个文件就是我们前面写的python依赖清单，自动复制过来了

│ └── requirements.yml #这个文件就是我们前面写的galaxy依赖清单，自动复制过来了

│ └── scripts

│ ├── assemble

│ ├── check\_ansible

│ ├── check\_galaxy

│ ├── entrypoint

│ ├── install-from-bindep

│ └── introspect.py

└── Containerfile #这个是ansible-builer要用到的配置文件，根据execution-environment.yml生成的

# cat context/Containerfile #查看构建配置文件，局部内容如下

ARG EE\_BASE\_IMAGE="quay.io/ansible/ansible-runner:latest" #基础镜像

ARG EE\_BUILDER\_IMAGE="quay.io/ansible/ansible-builder:latest" #构建镜像

FROM $EE\_BASE\_IMAGE as galaxy

ARG ANSIBLE\_GALAXY\_CLI\_COLLECTION\_OPTS=

USER root

COPY \_build/scripts/ /output/scripts/

COPY \_build/scripts/entrypoint /opt/builder/bin/entrypoint

ADD \_build /build #路径映射，将执行机本地的context/\_build/映射到容器里的/build

WORKDIR /build

#接下来开始进行构建操作，首先安装ansible依赖的role及collection

RUN ansible-galaxy role install -r requirements.yml --roles-path "/usr/share/ansible/roles"

RUN ANSIBLE\_GALAXY\_DISABLE\_GPG\_VERIFY=1 ansible-galaxy collection install $ANSIBLE\_GALAXY\_CLI\_COLLECTION\_OPTS -r requirements.yml --collections-path "/usr/share/ansible/collections"

FROM $EE\_BUILDER\_IMAGE as builder

COPY --from=galaxy /usr/share/ansible /usr/share/ansible

RUN ansible-builder introspect --sanitize --user-pip=requirements.txt --user-bindep=bindep.txt --write-bindep=/tmp/src/bindep.txt --write-pip=/tmp/src/requirements.txt

RUN /output/scripts/assemble

#将本次构建依赖的collection压缩包复制到context/\_build/目录下

# cp ~/limaofu-collectionyy-1.0.0.tar.gz context/\_build/

# podman login hub.cof-lee.com -u admin -p passwdxxhub #登录hub，使我们有权限下载需要用到的镜像，如果要获取的镜像为public公共镜像，则可以不用登录hub

#构建镜像

# ansible-builder build -t ansible-runner-user:latest #大概需要5-10分钟，期间需要联网更新依赖

Running command:

podman build -f context/Containerfile -t ansible-runner-user:latest context

Complete! The build context can be found at: /root/test/context

# podman images

REPOSITORY TAG IMAGE ID CREATED SIZE

localhost/ansible-runner-user latest 4da699fd6611 2 minutes ago 874 MB

# podman tag localhost/ansible-runner-user:latest cof-lee.com/eei/ansible-runner-user:latest

# podman push cof-lee.com/eei/ansible-runner-user:latest

**★附录、playbook清单引入其他文件**

**①playbook引入变量文件**

vars\_files:

- myvars.yml

**②tasks引入变量文件**

tasks:

- name: include vars file

ansible.builtin.include\_vars: myvars.yml # ansible<=2.9就只写include\_vars

**③playbook默认会去读取当前目录下的host\_vars和group\_vars子目录下的变量文件**，主机找host\_vars子目录下同主机名的变量文件，主机组找group\_vars子目录下同主机组名的变量文件（没有.yml后缀）

**以上引入的变量文件内容必须为yaml格式**

**④template引入jinja2模板文件**

tasks:

- name: j2 to config file

ansible.builtin.template: # ansible<=2.9就只写template

src: template-xx.j2

dest: /etc/xxxx.config

**⑤task引入其他task文件**

tasks:

- name: include taskxx

ansible.builtin.include: taskxx.yml # ansible<=2.9就只写include

ansible-core>=2.16不再支持include，建议只使用include\_tasks及import\_tasks模块

**⑥playbook引入其他playbook**

- name: play xx

hosts: all

tasks:

- name: play 1

ansible.builtin.import\_playbook: play-1.yml #引入其他playbook文件，ansible<=2.9就只写import\_playbook

**⑦playbook引入role角色项目**

- name: play xx

hosts: all

roles:

- role: role-namexx

#或者：

- name: play xx

hosts: all

roles:

- role-namexx

**⑧tasks引入role角色项目**

tasks:

- name: include role

ansible.builtin.include\_role: # ansible<=2.9就只写include\_role

name: myrolexx

**⑨playbook引入collection集合**

---

- name: play xx

hosts: mariadb

collections:

- cof.collectionyy

tasks:

**-------------------------------------**

**第12章、ansible-awx自动化管理平台**

RedHat最早开发了Ansible Tower的自动化管理平台，在Ansible Tower 3.8及之前的版本中，执行环境与运行ansible tower的系统紧密耦合，这使得管理运行ansible playbook所需的各种模块的依赖变得困难，对执行环境的更改必须直接在ansible tower系统上完成，如果不同的playbook需要不同的依赖环境，则必须手动设置和更新Python虚拟环境来管理它们。于是后来开发了**AAP**（Ansible Automation Platform）这个自动化控制平台，AAP改进了这一架构，不再直接使用系统的环境，而是使用自动化执行环境（基于容器的一个ansible及python环境）

RedHat的Ansible Tower及Ansible Automation Platform是收费的，社区有开源免费的对应产品：Ansible-AWX

**AWX** provides a web-based user interface, REST API, and task engine built on top of Ansible. It is one of the upstream projects for Red Hat Ansible Automation Platform.

AWX是一个开源免费的自动化运维管理平台，用于管理Ansible项目。借助图形化的AWX，能够方便的编排部署Ansible Playbook，可以通过可视化仪表板，基于角色的访问控制，作业计划，集成通知和图形化管理集中控制it架构。

开源地址：https://github.com/ansible/awx

**★在k8s上部署Ansible-AWX**

AWX 17版本以前可以通过各种方法进行安装，比如通过docker-compose工具直接运行，但18版本以后只能部署在K8S/OpenShift集群中，通过AWX operator进行安装。

17及之前版本参考链接： https://github.com/ansible/awx/blob/17.0.1/INSTALL.md

18及之后版本参考链接： https://ansible.readthedocs.io/projects/awx-operator/en/latest/

本例中安装23.6.0的awx版本，awx-operator版本为2.10.0

**①首先安装k8s集群，并安装helm**

过程省略，可参考作者的k8s相关文档：

https://limaofu.github.io/t3doc/k8s%E6%93%8D%E4%BD%9C%E6%89%8B%E5%86%8C-2.3-back-1229.docx

**②添加helm软件仓库并安装awx-operator**

# helm repo add awx-operator https://ansible.github.io/awx-operator/ #要求仓库地址能访问才添加成功

# helm install ansible-operator awx-operator/awx-operator -n awx --create-namespace #在线安装

#awx-operator要用到以下2个镜像：

gcr.io/kubebuilder/kube-rbac-proxy:v0.15.0

quay.io/ansible/awx-operator:2.10.0

**离线安装awx-operator：**

# helm pull awx-operator/awx-operator #下载chart包到本地

也可直接下载： wget https://github.com/ansible/awx-operator/releases/download/2.7.0/awx-operator-2.7.0.tgz

# helm install awx-operator awx-operator-2.10.0.tgz -n awx --create-namespace #离线安装

# kubectl get pod -n awx

NAME READY STATUS RESTARTS AGE

awx-operator-controller-manager-7b77767859-7p9rw 2/2 Running 0 19s

★如果需要指定使用内部image仓库，则修改awx-operator-2.7.0.tgz里面的deployment文件：

# tar -xf awx-operator-2.7.0.tgz -C ~/

# vi ~/awx-operator/templates/deployment-awx-operator-controller-manager.yaml #修改以下2行

image: gcr.io/kubebuilder/kube-rbac-proxy:v0.15.0

image: quay.io/ansible/awx-operator:2.10.0

#以上2行改为：

image: cof-lee.com/awx/kube-rbac-proxy:v0.15.0

image: cof-lee.com/awx/awx-operator:2.10.0

# cd ~/

# tar -zcf awx-operator-new.tgz awx-operator #再打包为tar.gz压缩包

# helm install awx-operator ~/awx-operator-new.tgz -n awx --create-namespace #离线安装

**③创建sc,pv给ansible-awx的postgres数据库用**

# cat > local-storage-class.yaml <<EOF

apiVersion: storage.k8s.io/v1

kind: StorageClass

metadata:

name: local-storage

namespace: awx

provisioner: kubernetes.io/no-provisioner

#volumeBindingMode: Immediate

volumeBindingMode: WaitForFirstConsumer

EOF

# kubectl create -f local-storage-class.yaml

# kubectl get sc -n awx

# cat > pv-ansible-awx.yaml <<EOF

apiVersion: v1

kind: PersistentVolume

metadata:

name: awx-postgres-pv

namespace: awx

spec:

capacity:

storage: 10Gi

volumeMode: Filesystem

accessModes:

- ReadWriteOnce

persistentVolumeReclaimPolicy: Delete

storageClassName: local-storage

local:

path: /ansible-awx-pg-storage #工作结点上要创建此目录

nodeAffinity:

required:

nodeSelectorTerms:

- matchExpressions:

- key: kubernetes.io/hostname

operator: In

values:

- node1.cof-lee.com #指定node结点的hostname，根据实际情况修改

EOF

# mkdir /ansible-awx-pg-storage #所有工作节点都要创建

# kubectl create -f pv-ansible-awx.yaml

# kubectl get pv -n awx

★注意，本例pv使用后端存储为local“本地存储”，正式环境建议使用nfs等网络存储

pvc不用手动创建，在创建awx实例时，会自动创建pvc，去绑定可用的pv

**★部署AWX实例**

# cat > ansible-awx.yaml <<EOF

---

apiVersion: awx.ansible.com/v1beta1

kind: AWX

metadata:

name: ansible-awx

namespace: awx

spec:

service\_type: nodeport

postgres\_storage\_class: local-storage

#以下手动指定要使用的镜像

postgres\_image: cof-lee.com/awx/postgres

postgres\_image\_version: "13"

redis\_image: cof-lee.com/awx/redis

redis\_image\_version: "7"

image: cof-lee.com/awx/awx

image\_version: "23.6.0"

init\_container\_image: cof-lee.com/awx/awx-ee

init\_container\_image\_version: "latest"

control\_plane\_ee\_image: cof-lee.com/awx/awx-ee:latest

ee\_images:

- name: AWX-EE (latest)

image: cof-lee.com/awx/awx-ee:latest

EOF

# kubectl create -f ansible-awx.yaml #需要等待5到10分钟左右

**#默认要用到以下4个镜像：**

docker.io/library/postgres:13

docker.io/library/redis:7

quay.io/ansible/awx-ee:latest

quay.io/ansible/awx:23.6.0

**★创建web-svc以供主机外部访问（这里可选，前面指定service\_type: nodeport时已创建好了）**

# kubectl expose deployment ansible-awx-web --name ansible-awx-web-svc --type NodePort -n awx

# kubectl get svc ansible-awx-web-svc -n awx

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

ansible-awx-postgres-13 ClusterIP None <none> 5432/TCP 10m

ansible-awx-service NodePort 10.7.87.150 <none> 80:30518/TCP 9m41s

ansible-awx-web-svc NodePort 10.7.181.237 <none> 8052:31801/TCP 109m（可选）

awx-operator-controller-manager-metrics-service ClusterIP 10.7.158.115 <none> 8443/TCP 129m

#后面那个3万多的端口号

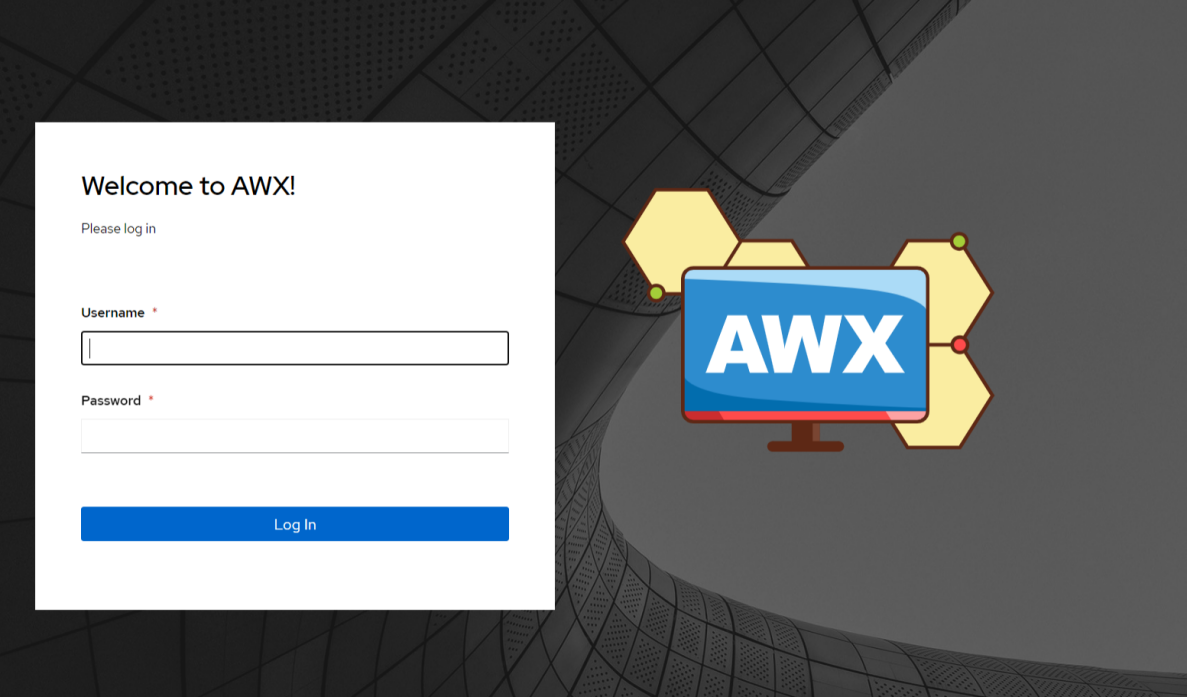
#查看admin密码

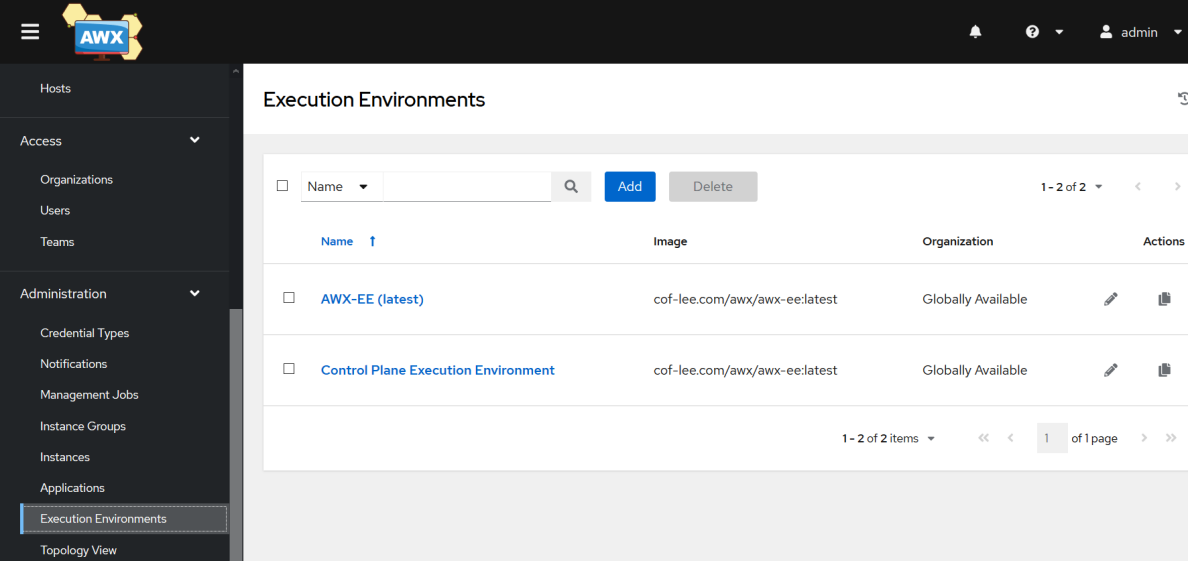
# kubectl get secrets -n awx ansible-awx-admin-password -ojsonpath="{.data.password}" | base64 -d

Hc25UcHSfBC4FB0WCLU02YDYiR8Nd4mg

★使用浏览器登录 http://node1地址:30518/ 或http://node1地址:31801/

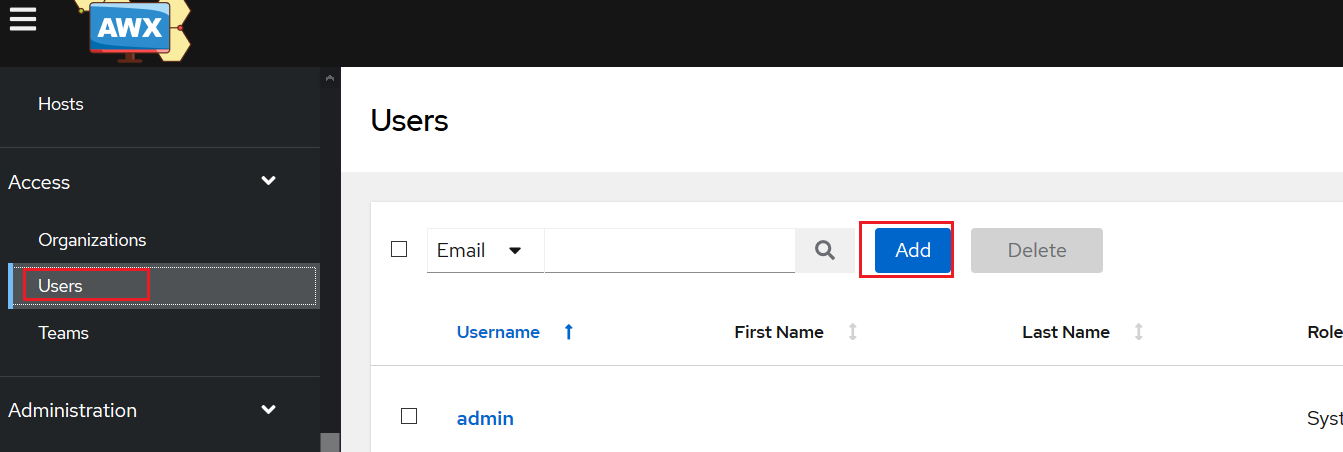
账号为admin，密码为上面查看到的。如果浏览器访问不了，可能要一会时间，大概15-30分钟，初始化时间较长。

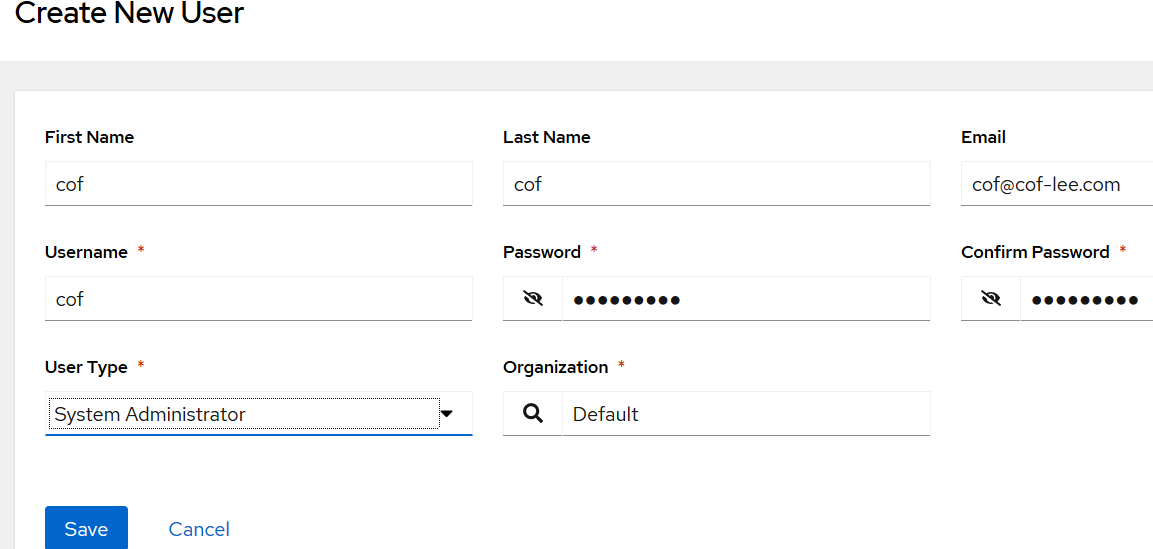


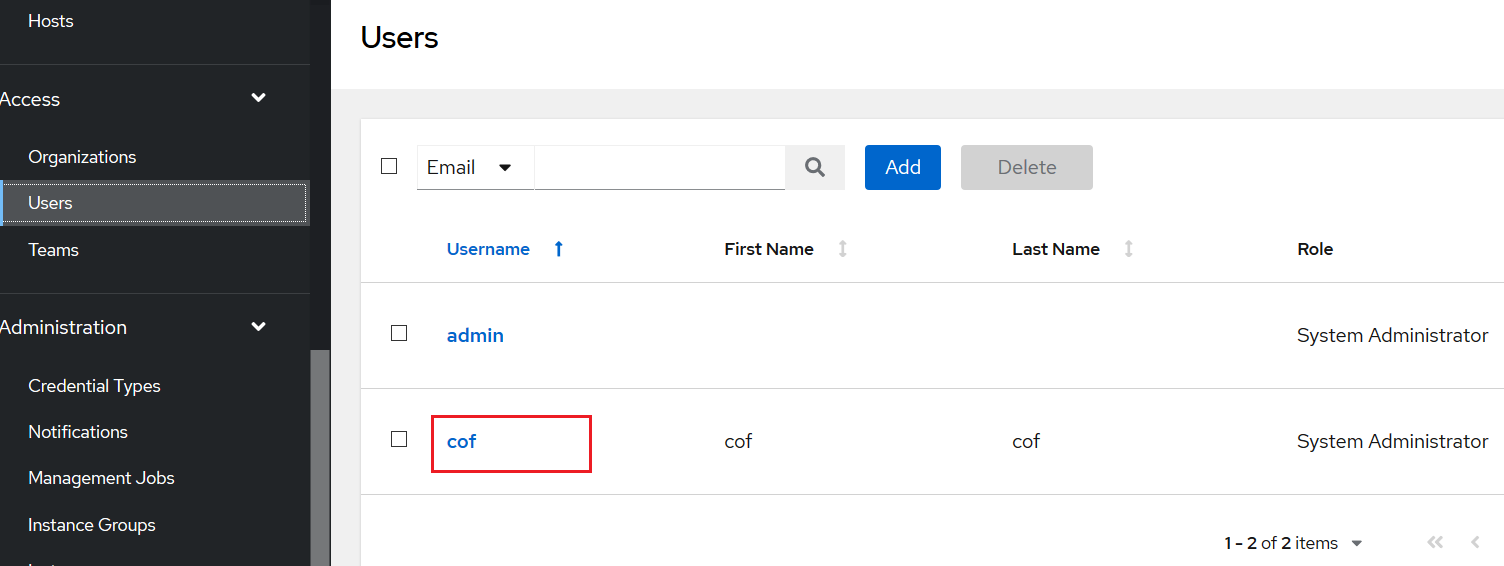


**★创建AWX用户账号**

默认已有admin账号，可再创建其他账号，Access→Users→Add





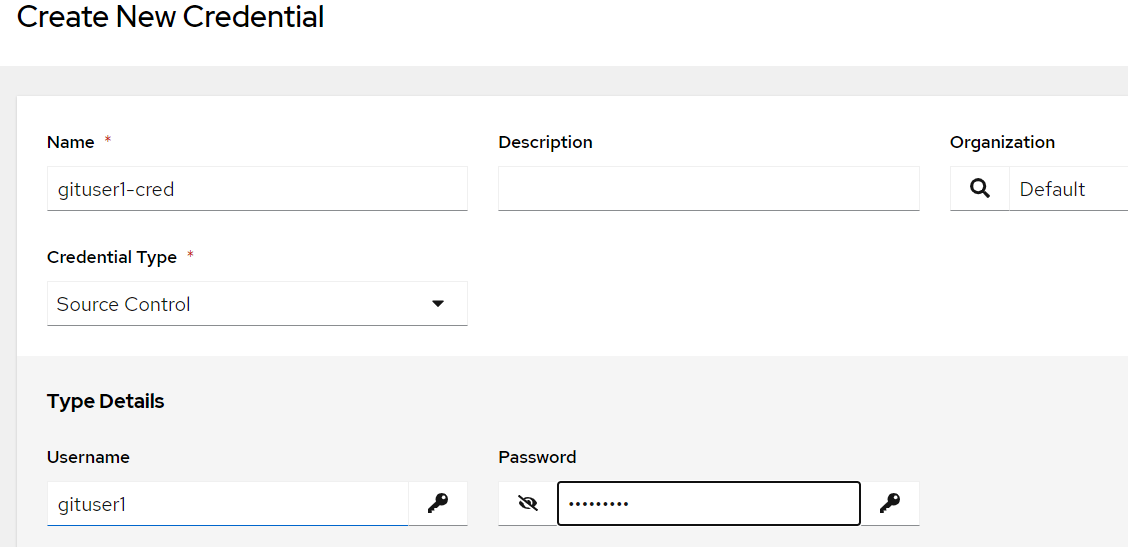


**★创建凭据**

**①创建git用户凭据**

Resources→Credentials→Add



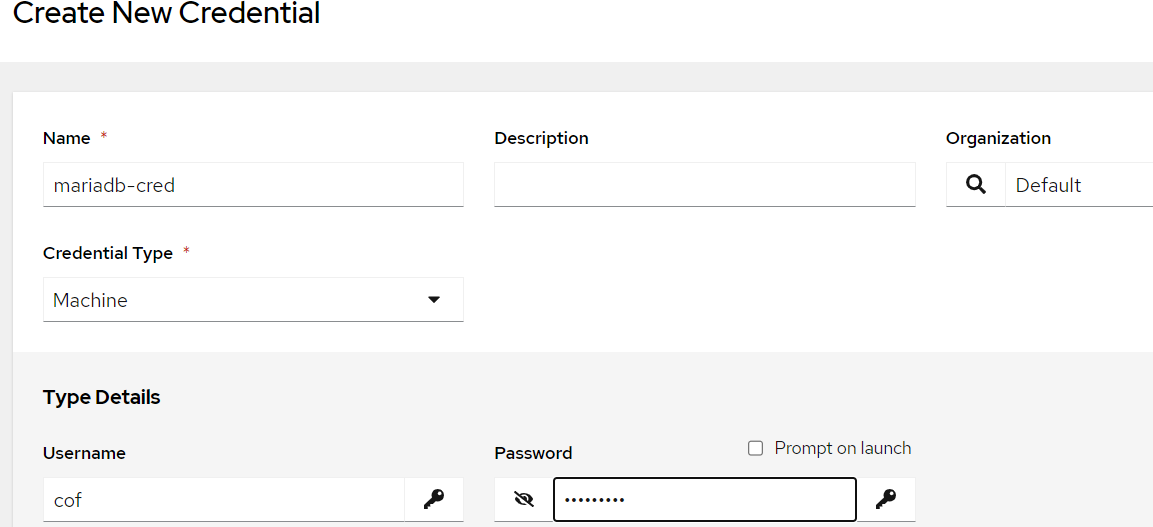


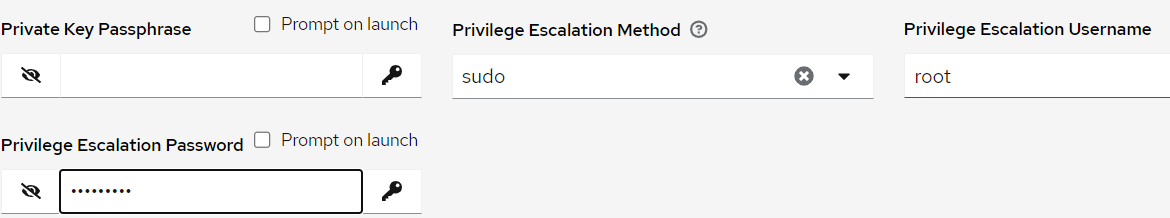
点击“Save”保存

**②创建受管主机登录凭据**

Resources→Credentials→Add

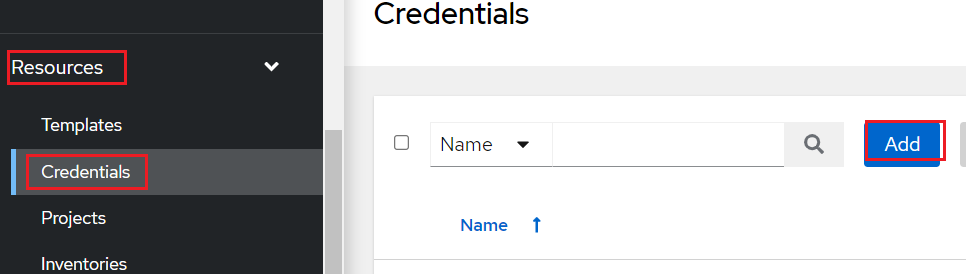




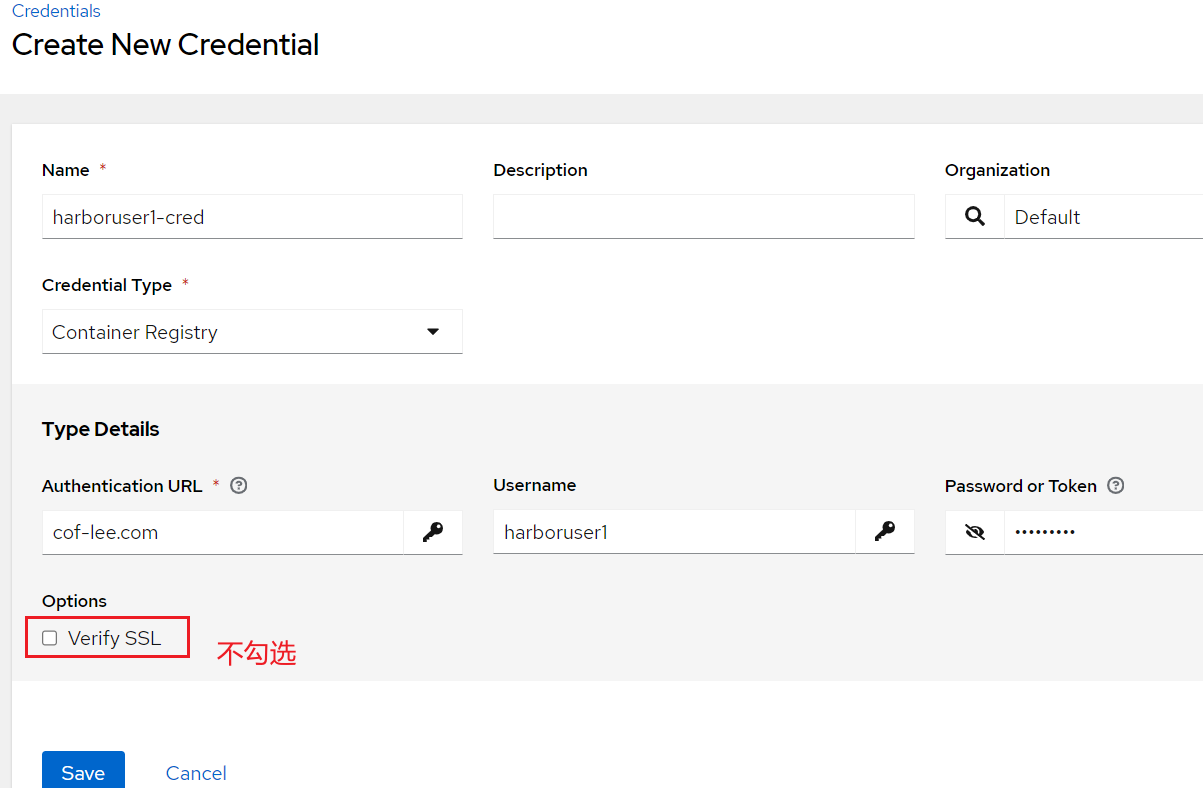


点击“Save”保存

**③创建docker-registry用户凭据**



Resources→Credentials→Add

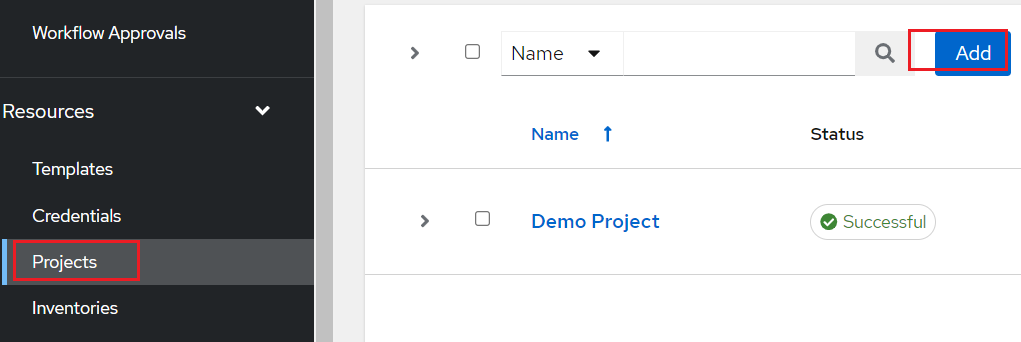


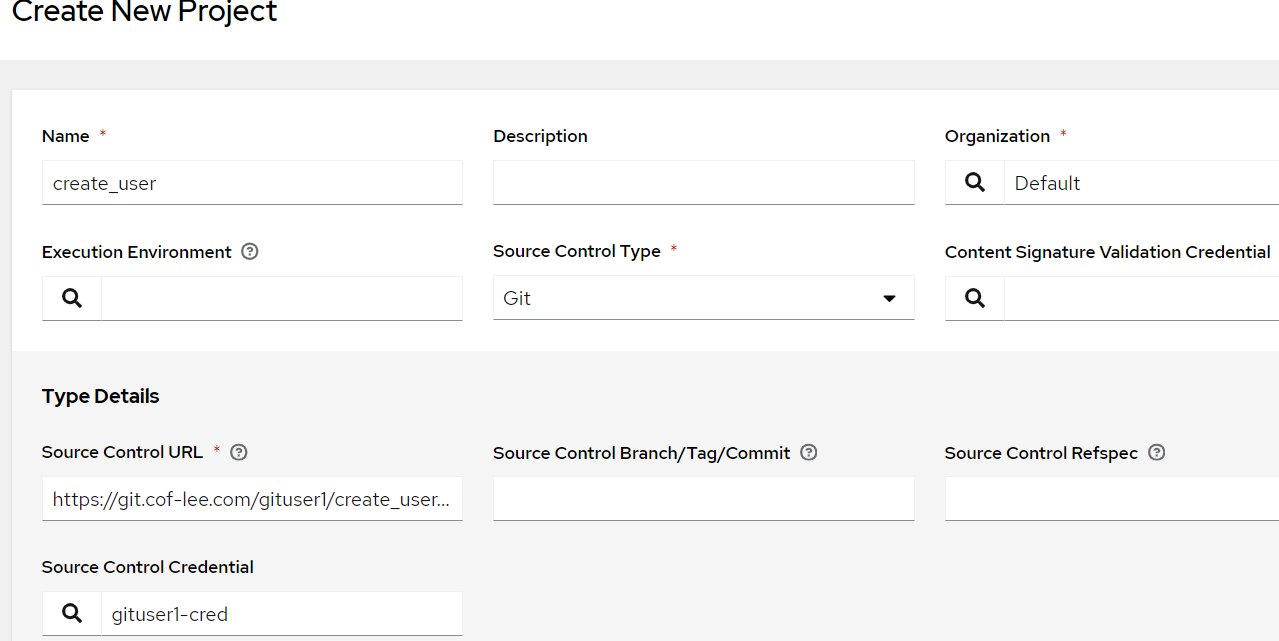
点击“Save”保存

**★创建项目**

创建ansible自动化运维项目，将某个任务的ansible相关文件都放在一个git仓库里，创建项目时指定源为此git仓库即可

Resources→Projects→Add





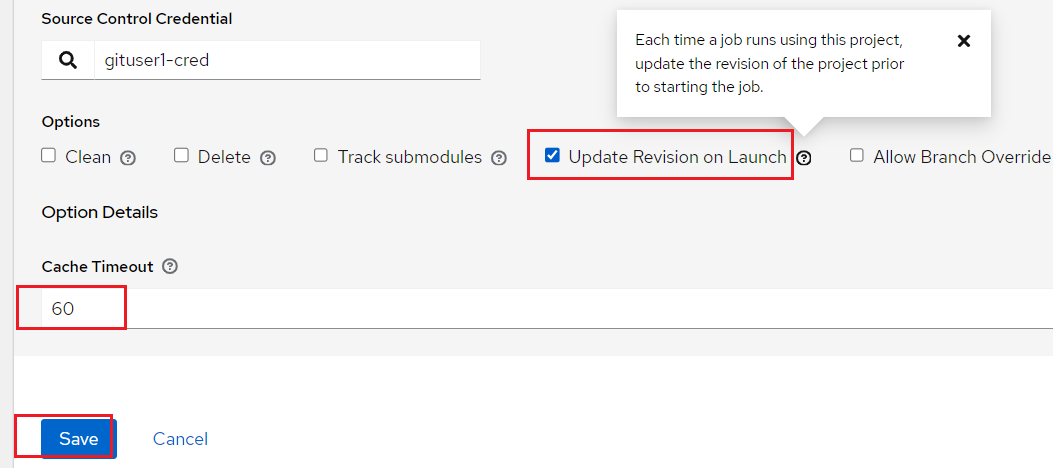
Source Control URL为git仓库地址： https://git.cof-lee.com/gituser1/create\_user.git

Source Control Credential为git仓库的用户凭据

**★设置每次执行作业时自动更新此项目（缓存60秒）**

勾选“Update Revision on Launch”，表示当有作业模板使用此项目时，先自动更新此项目再去执行作业模板

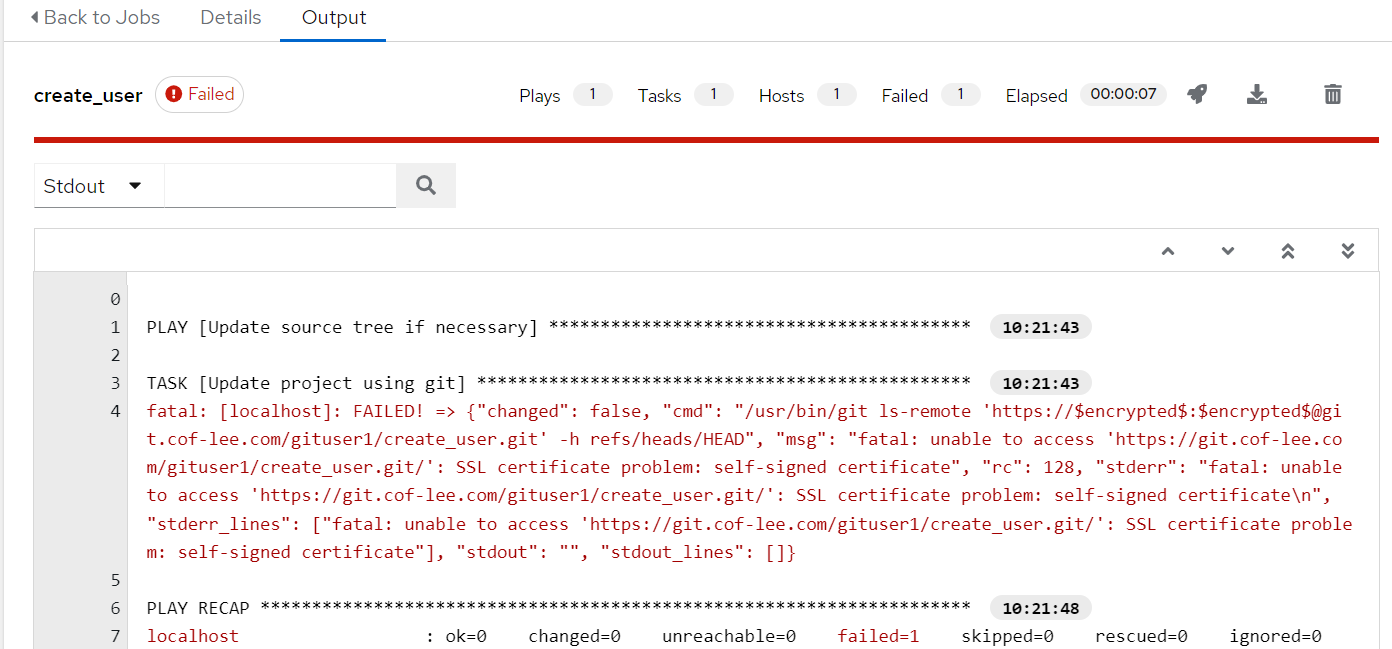
Cache Timeout填写60（表示当执行作业模板的时间比更新项目时间大于60秒才会去更新项目，否则不更新项目



点击“Save”保存

创建完项目后，AWX平台默认会运行一次此项目进行各项检查

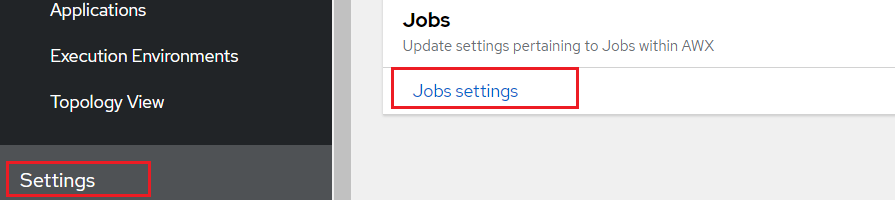
如果出现以下报错，则是使用的git仓库为私有仓库，其ssl证书是自签名的



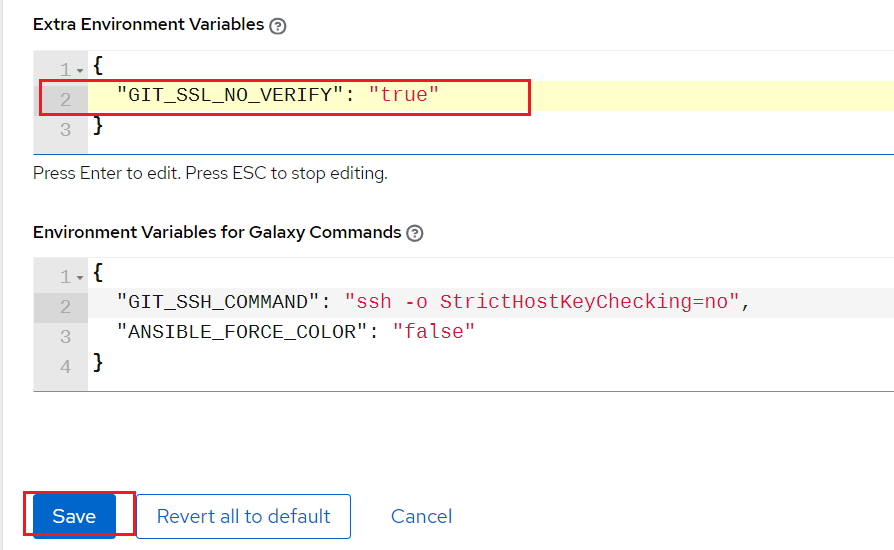
**解决方法:**

**★信任私有git仓库**

Settings→Jobs settings



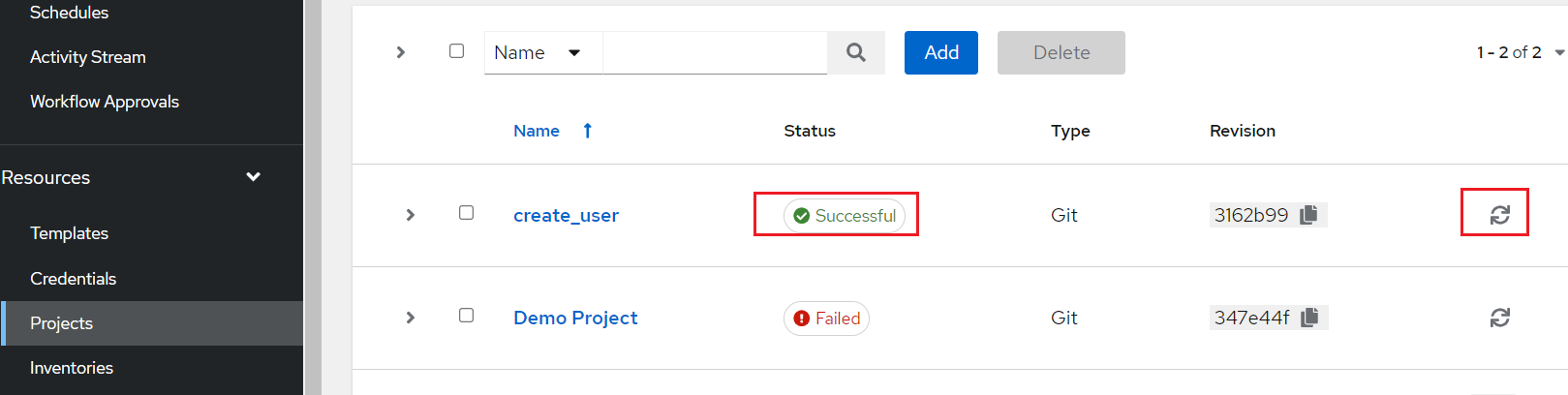
编辑设置：



在“Extra Environment Variables”里新增一条配置："GIT\_SSL\_NO\_VERIFY": "true" 表示不校验git仓库的ssl证书

点击“Save”保存

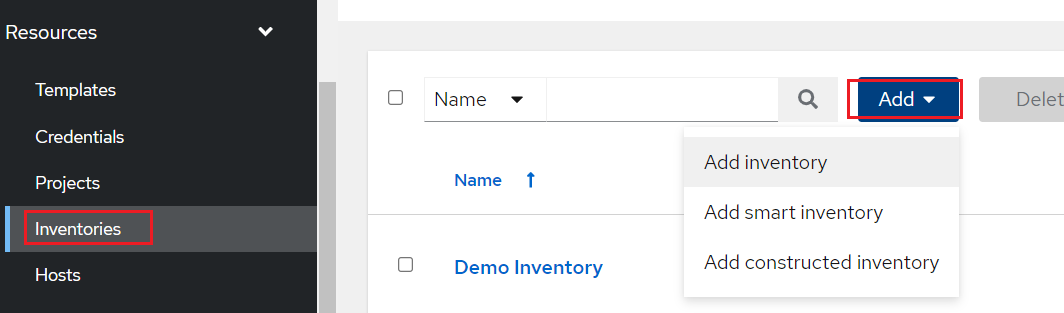
然后再去更新项目

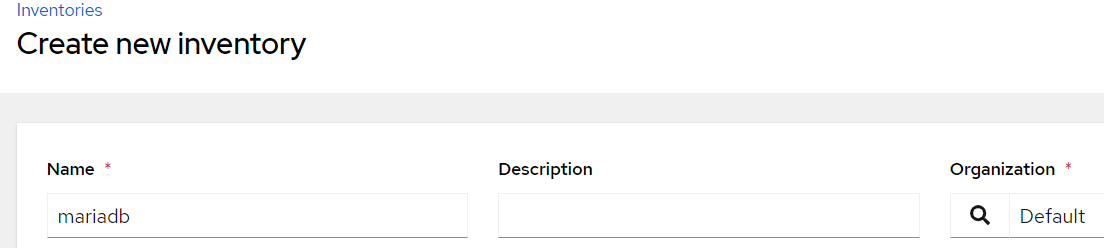


★**创建主机清单**

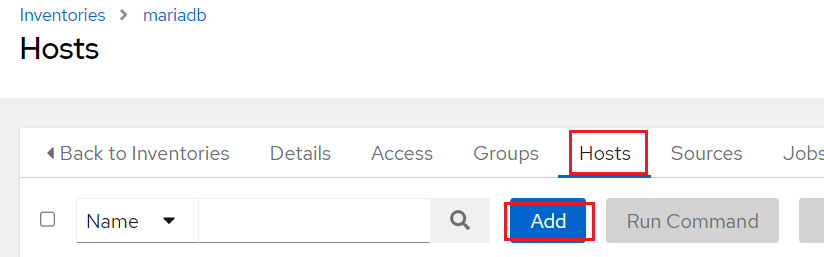
①创建静态主机清单

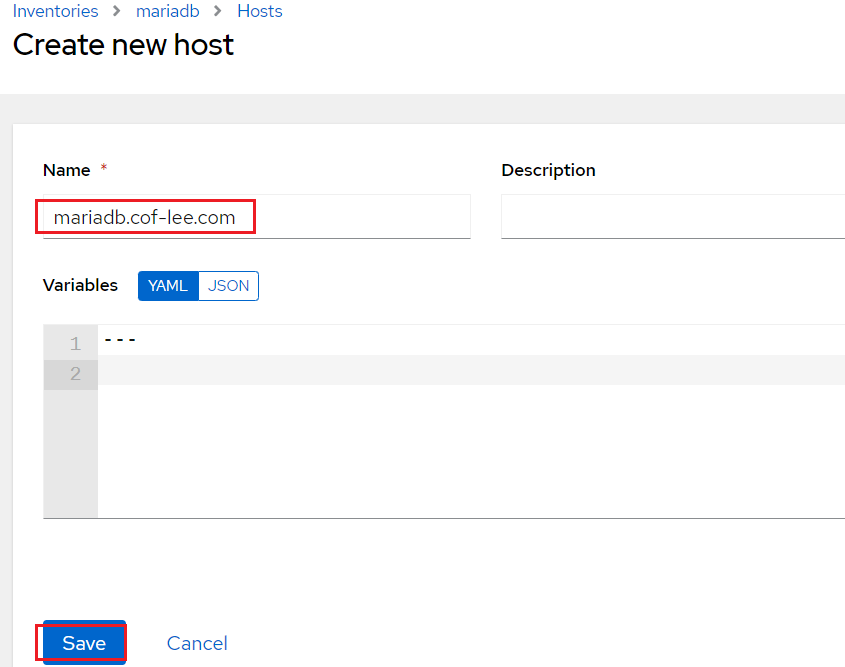
Resources→Inventories→Add→Add inventory





Save保存后可添加groups及hosts：

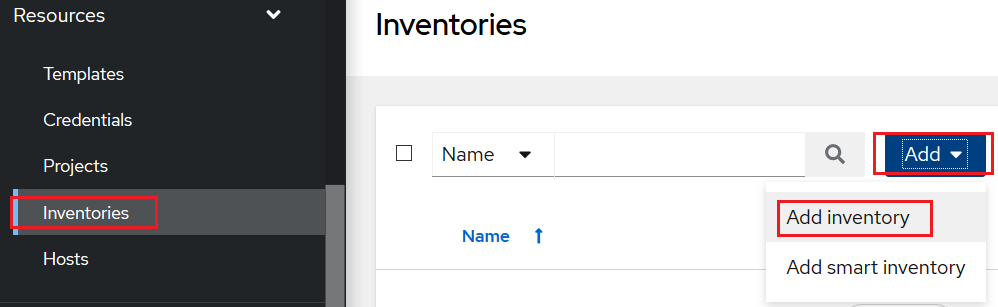




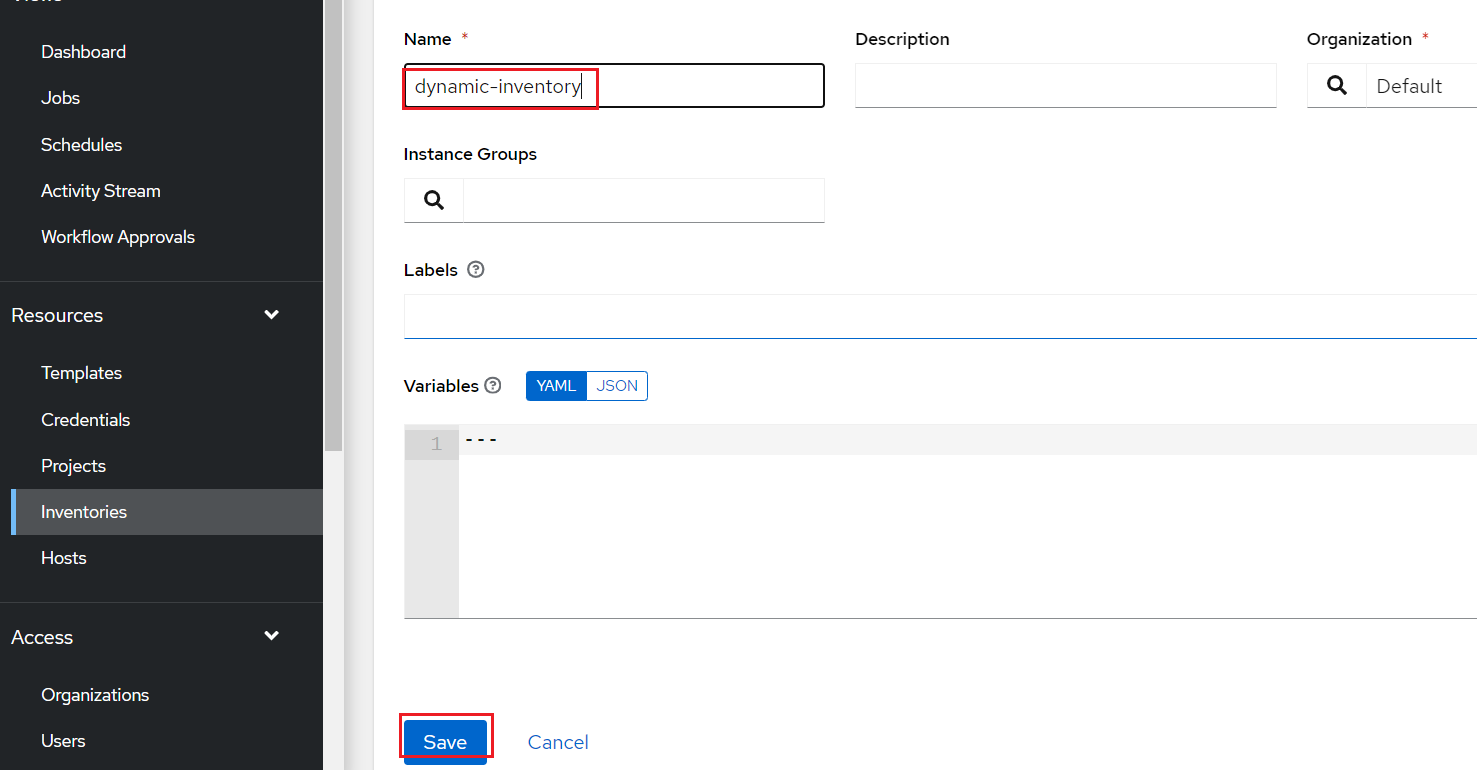
主机name可写主机名称，也可写ip地址

点击“Save”保存

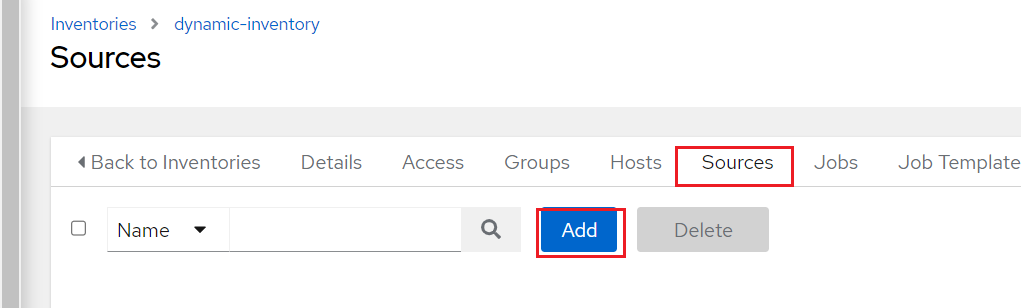
**②创建动态主机清单**

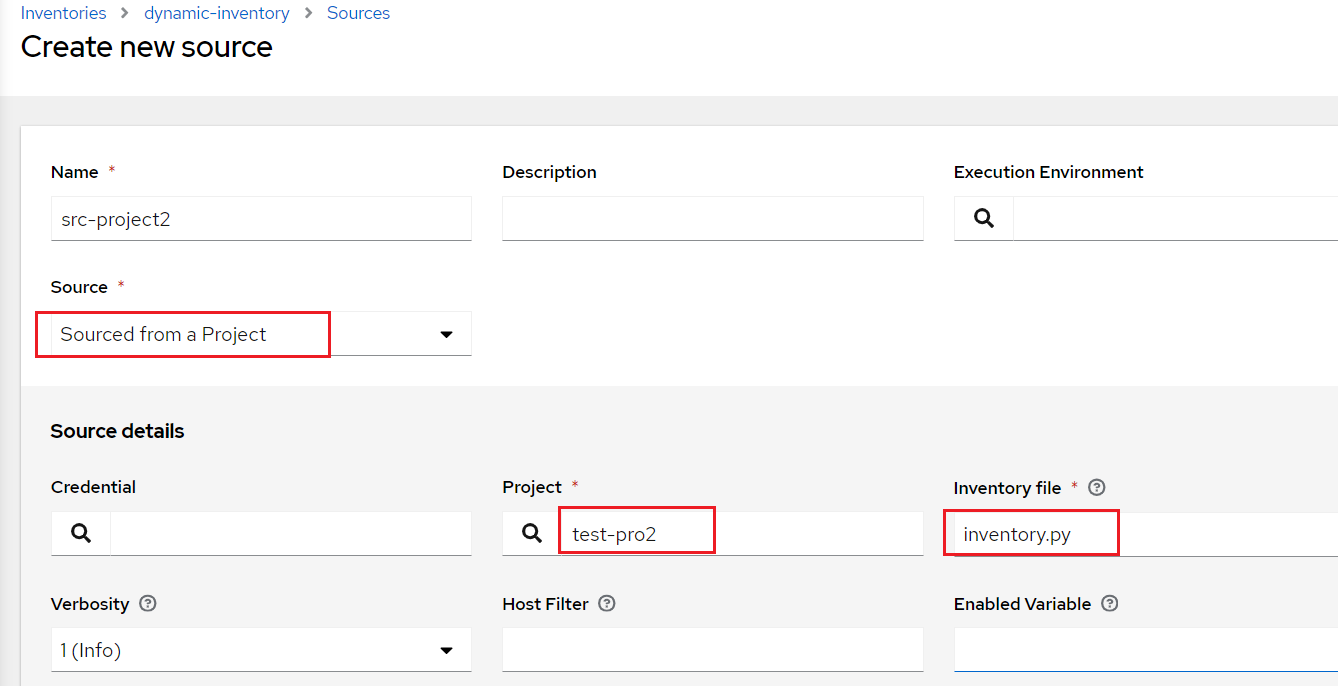


Resources→Inventories→Add→Add inventory

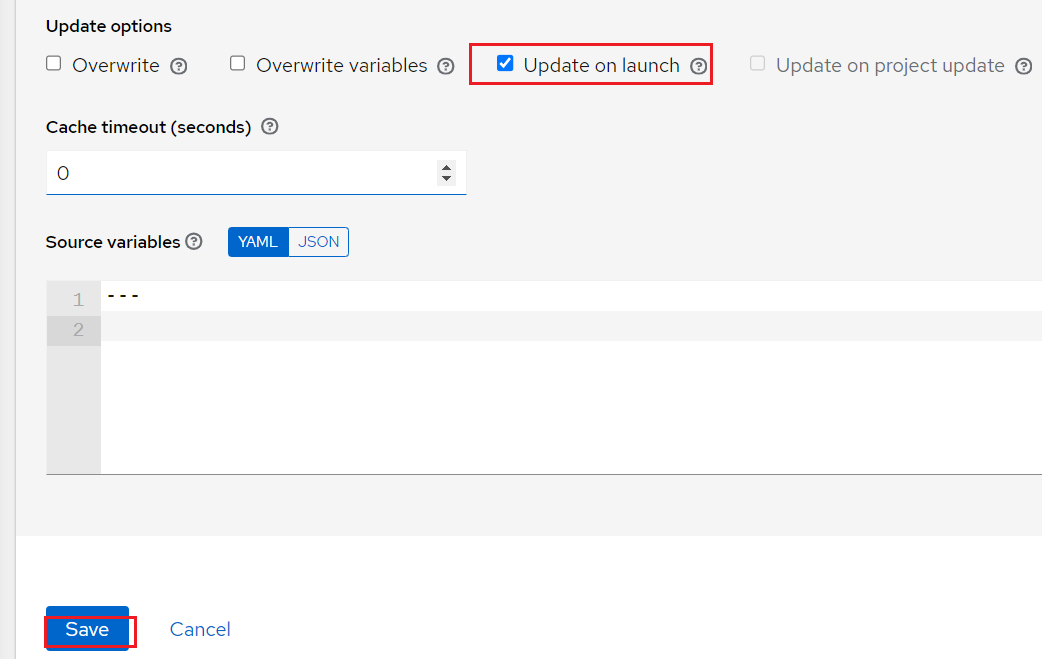


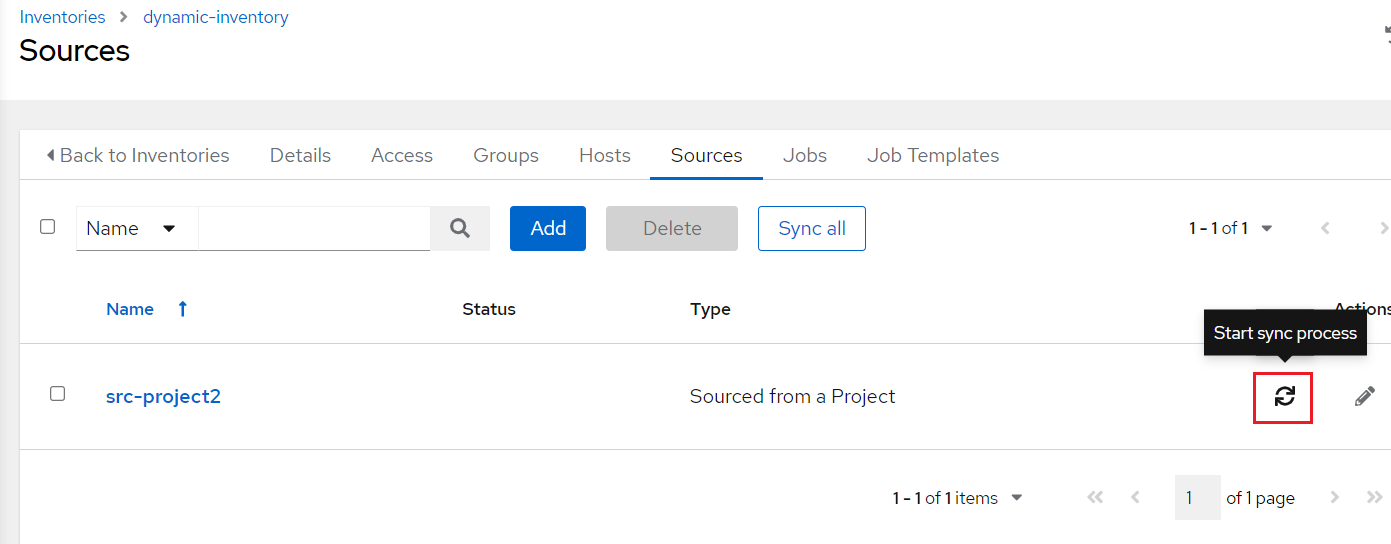
创建之后，在Sources里添加源



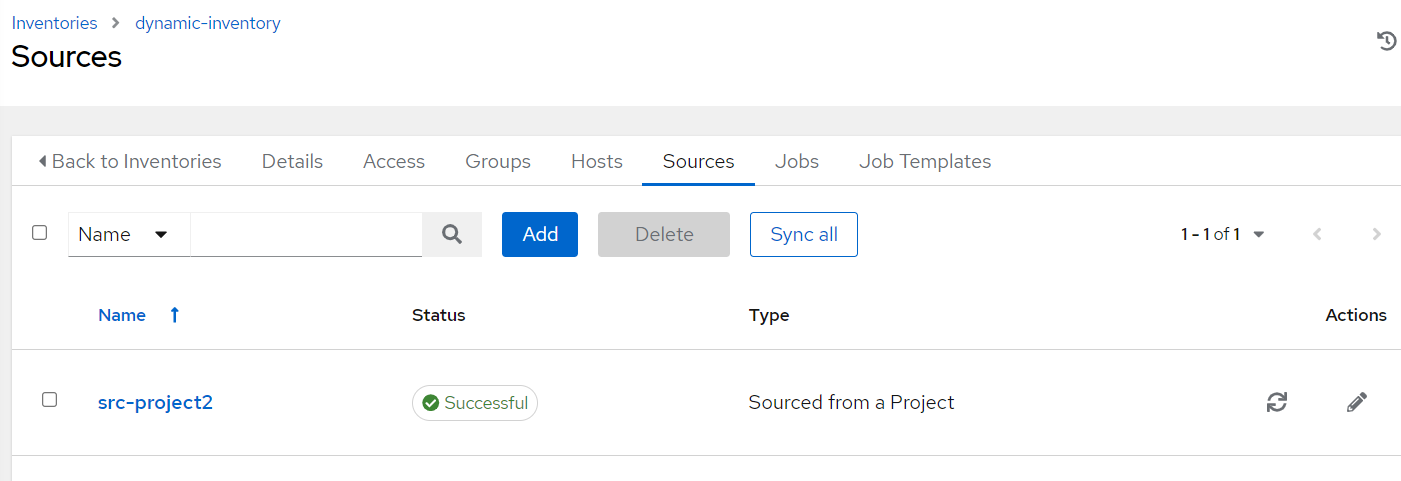


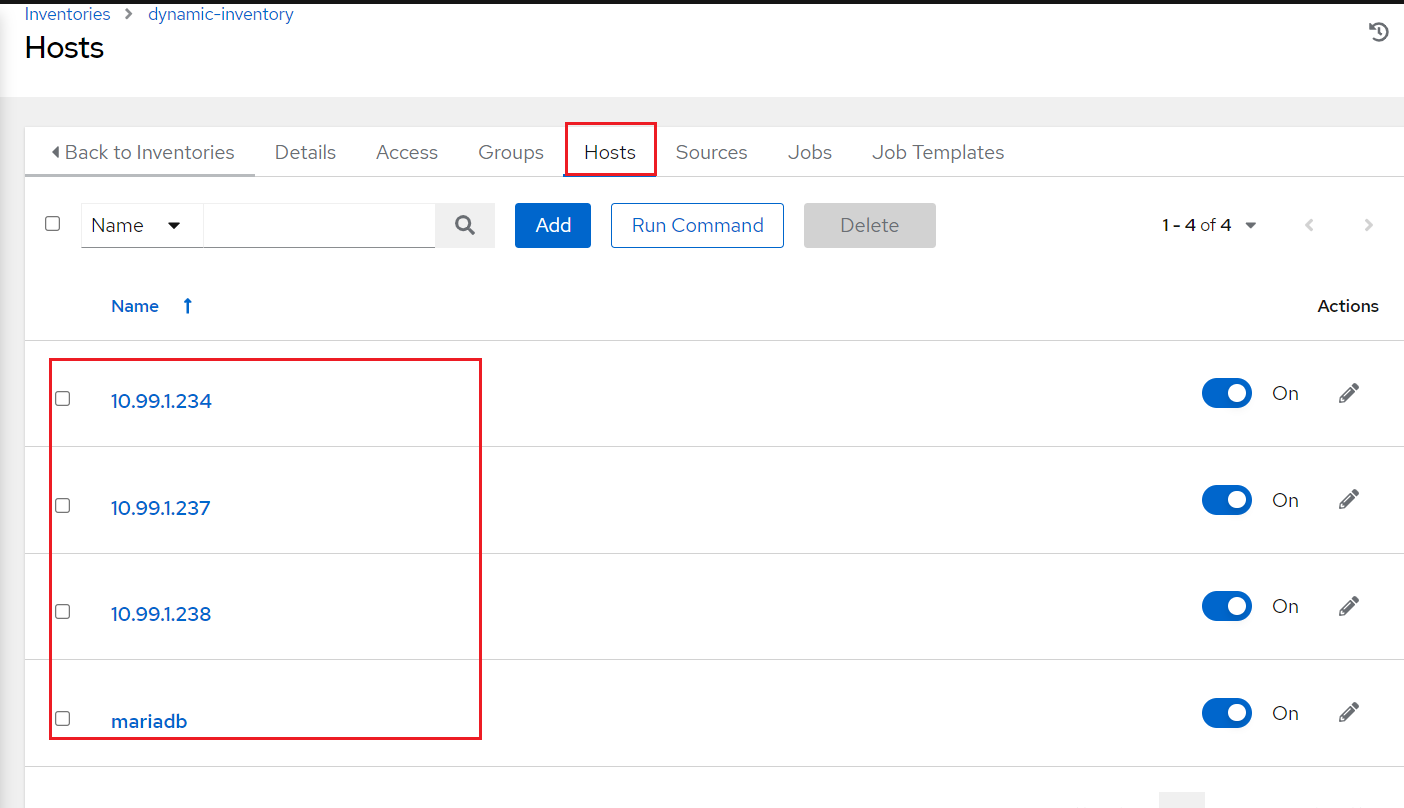
从项目里获取动态inventory文件





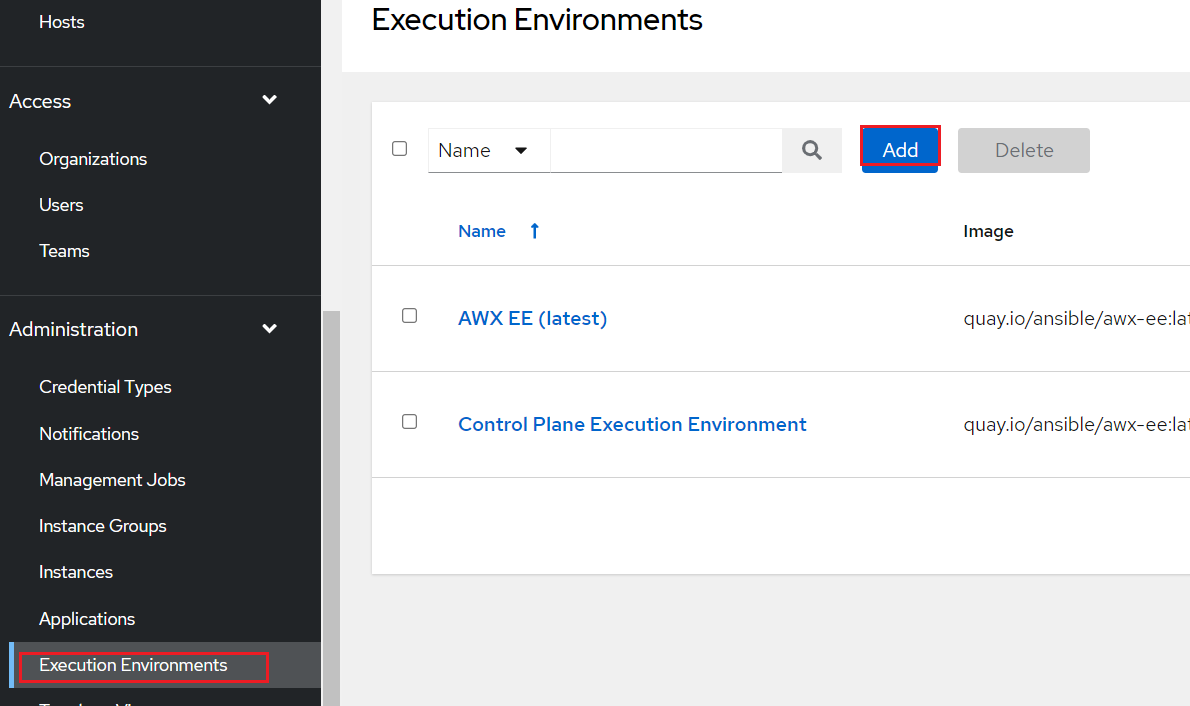
添加源之后，需要同步一下，同步成功才会有主机信息

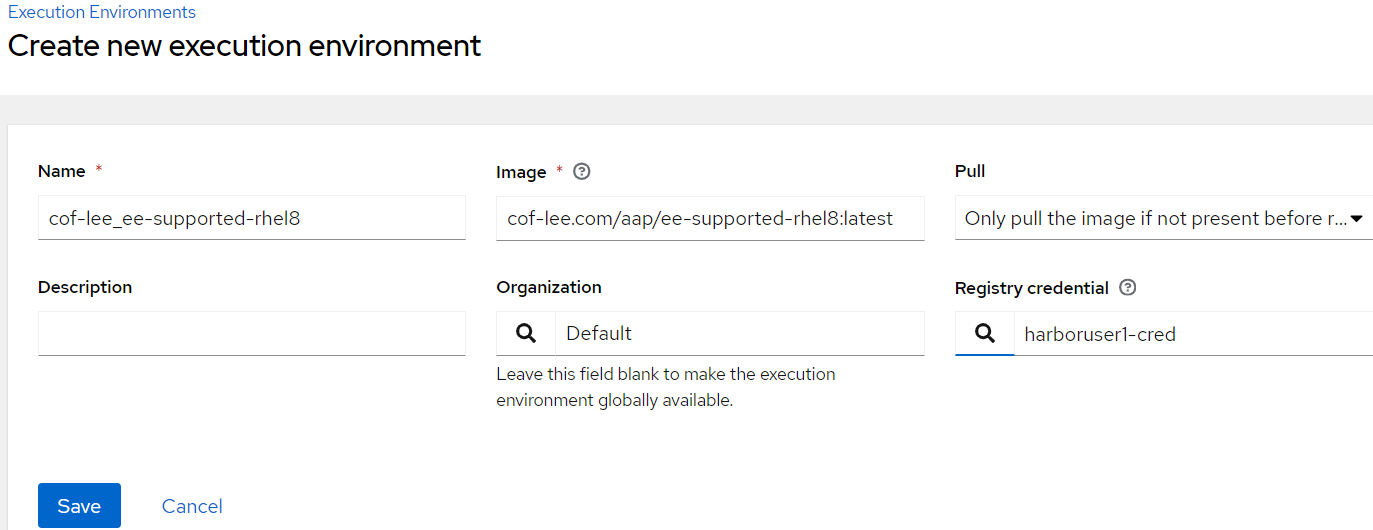




**★新增eei执行环境镜像**

Administration→Execution Environments→Add





Image镜像为： cof-lee.com/aap/ee-supported-rhel8:latest

Registry credential：指定要使用的registry认证凭据，为可选参数（若镜像仓库为public则不用指定）

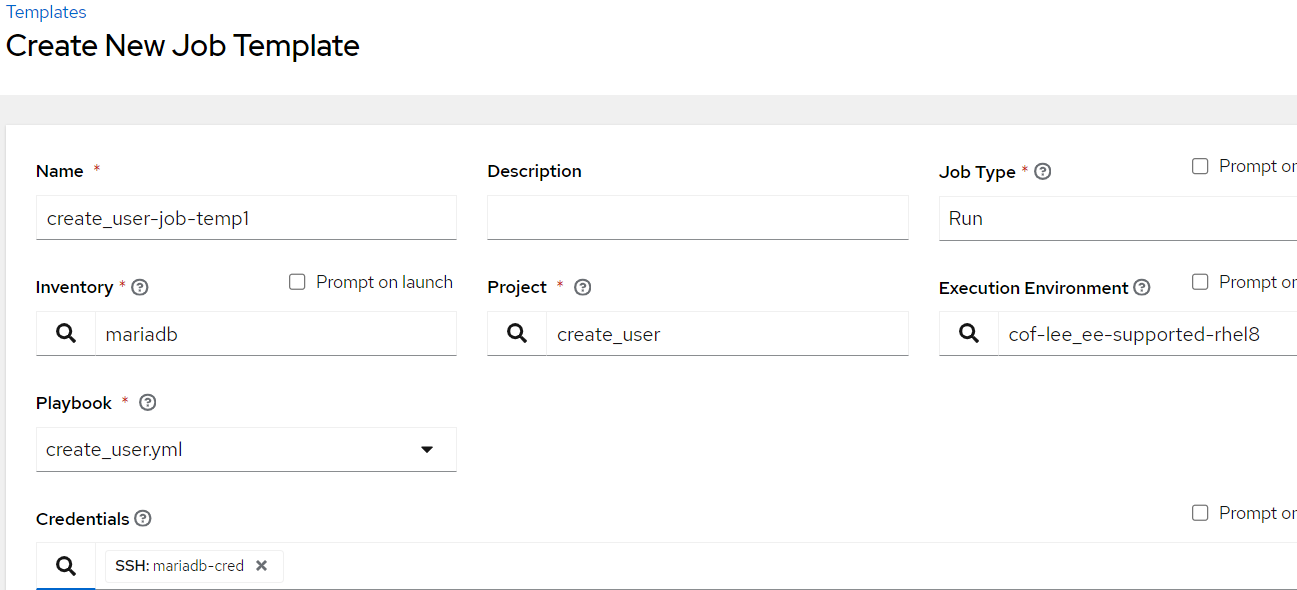
点击“Save”保存

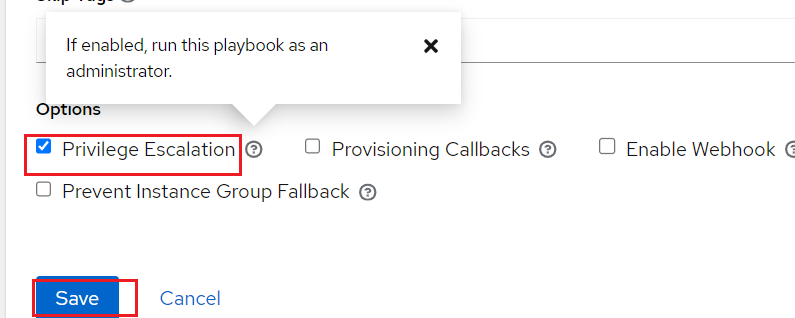
若镜像仓库为私有镜像仓库，使用了自签名ssl证书，则需要在k8s集群所有节点配置docker/containerd信任此registry仓库

**★创建作业模板**

Resources→Templates→Add→Add job template

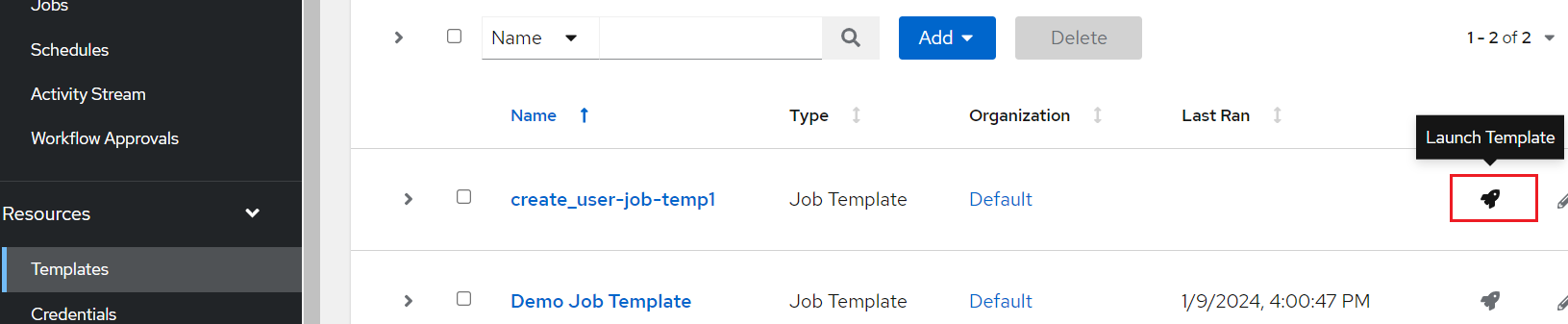


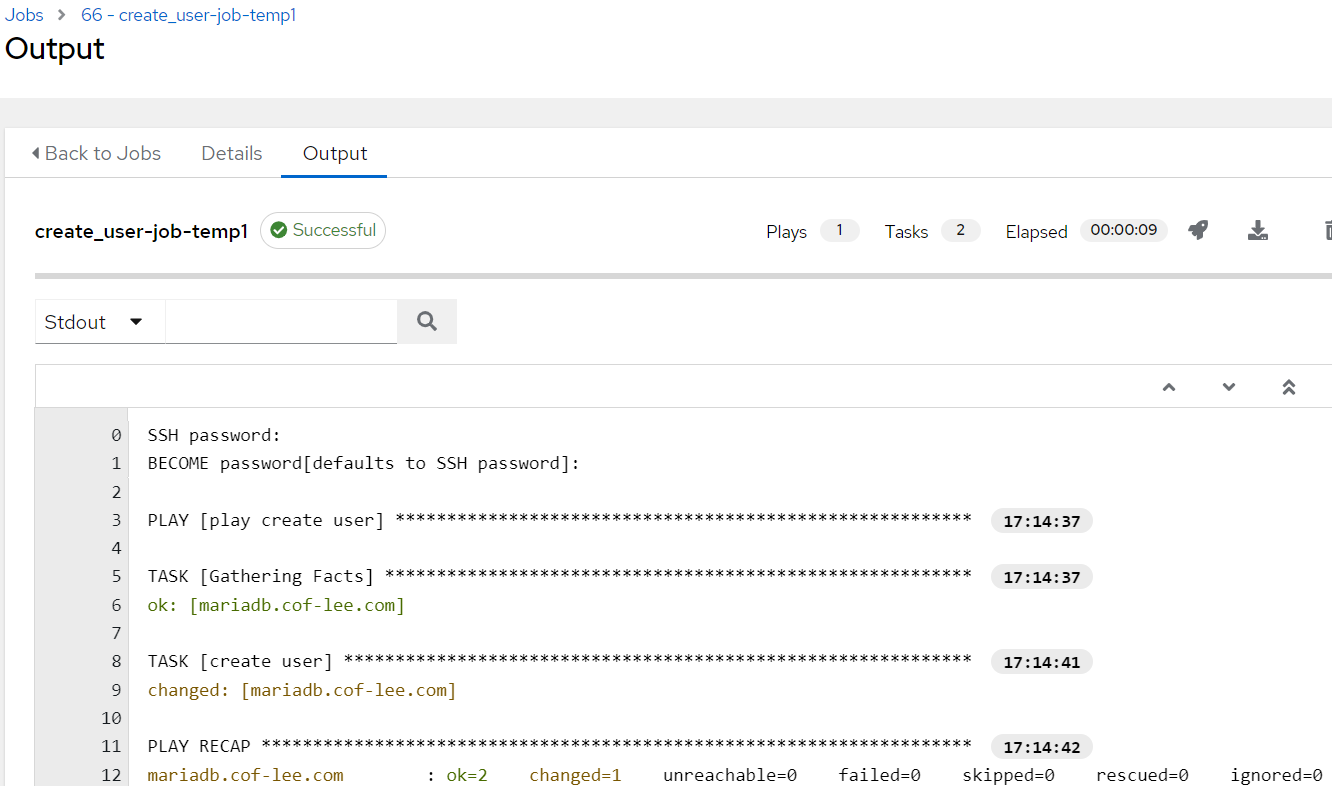




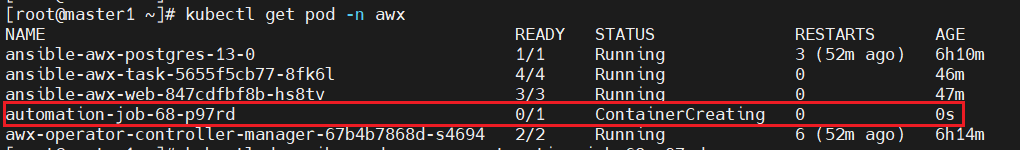
点击“Save”保存

**★运行作业模板**

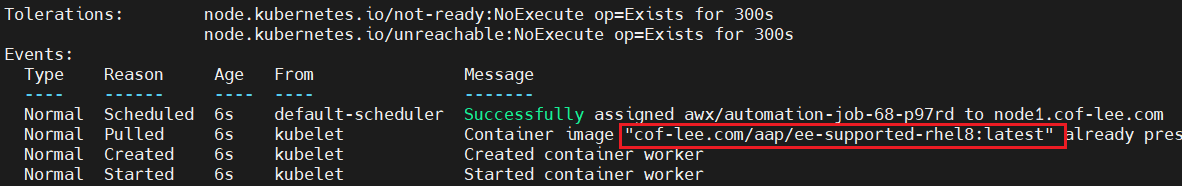




运行作业时，原理是在k8s集群里创建一个临时的容器：

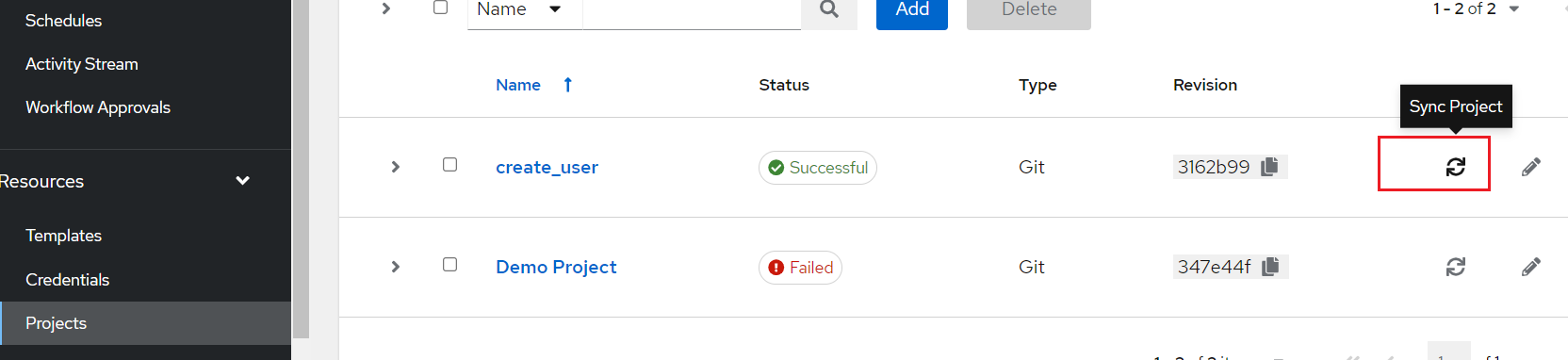


使用的镜像就是模板里指定的eei执行环境镜像：cof-lee.com/aap/ee-supported-rhel8:latest



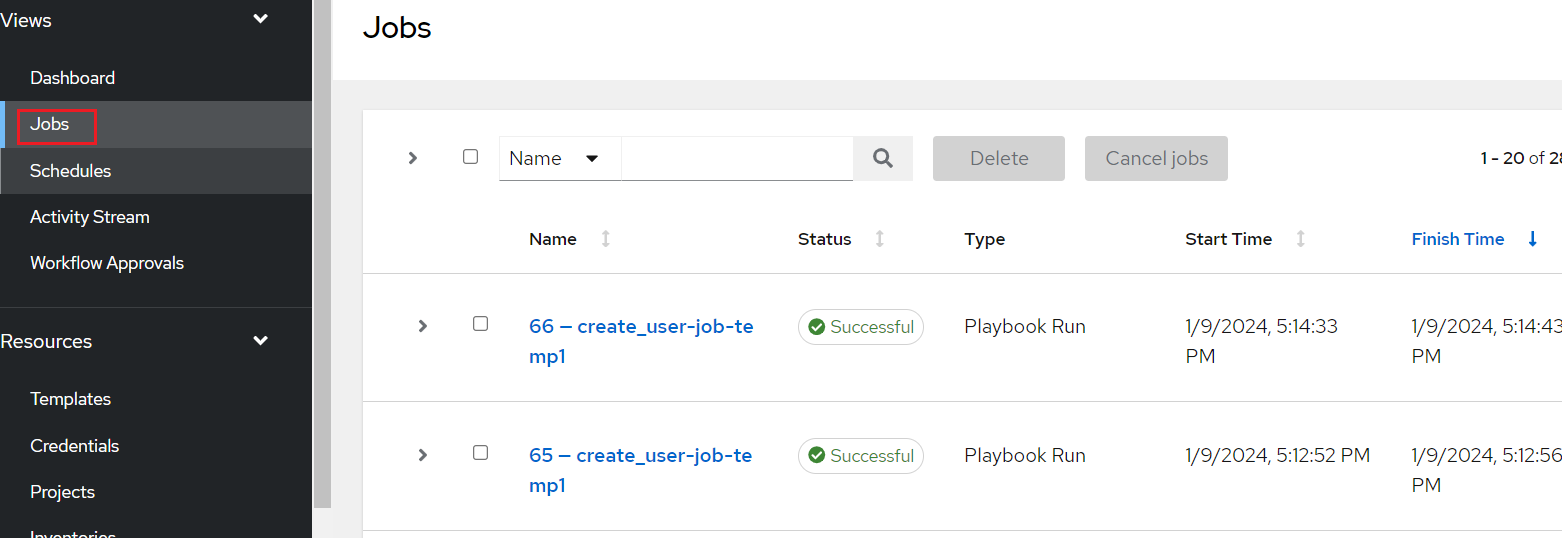
在pod容器里执行完ansible-playbook任务后就自动清除此pod了

若更新了git仓库，需要在AWX平台同步此项目



再去执行作业模板：

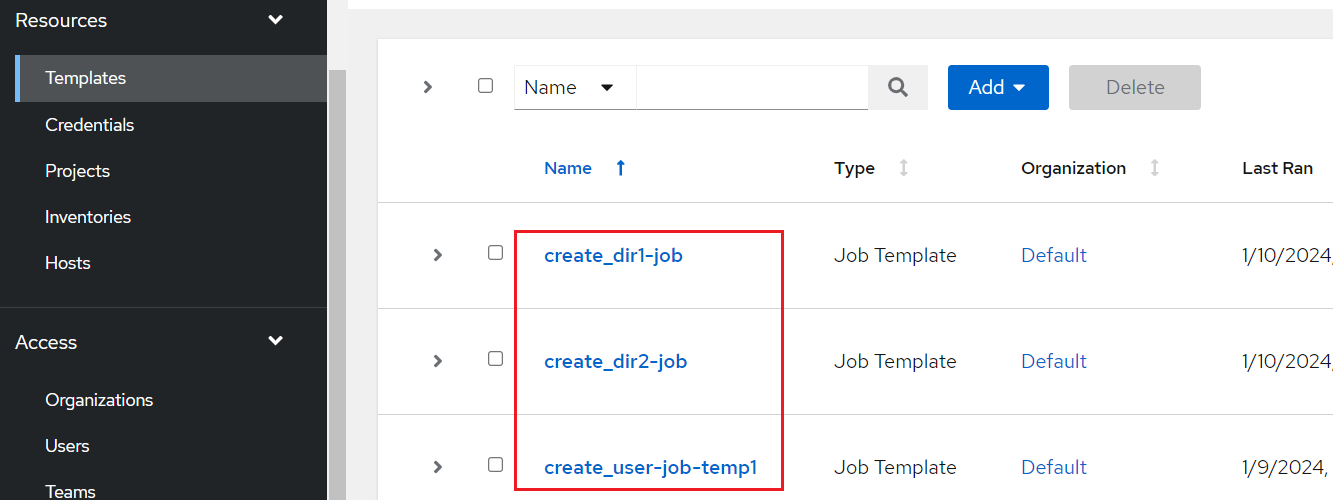
每次运行作业模板，都会生成一个job，可在以下地方查看：Views→Jobs



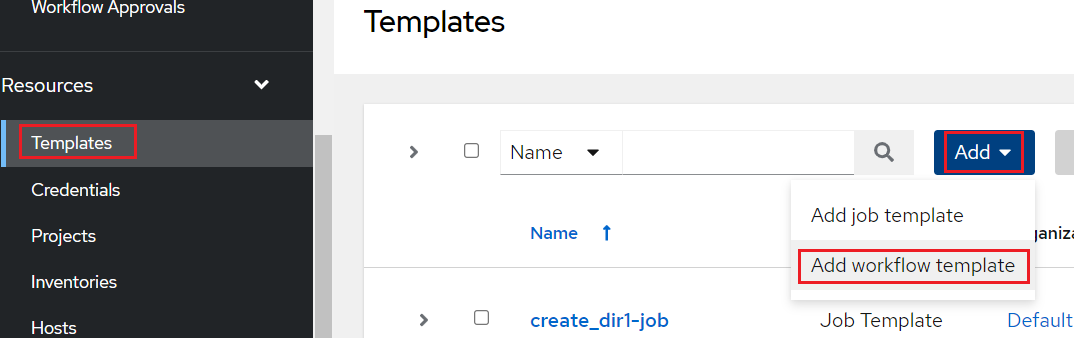
**★创建工作流模板**

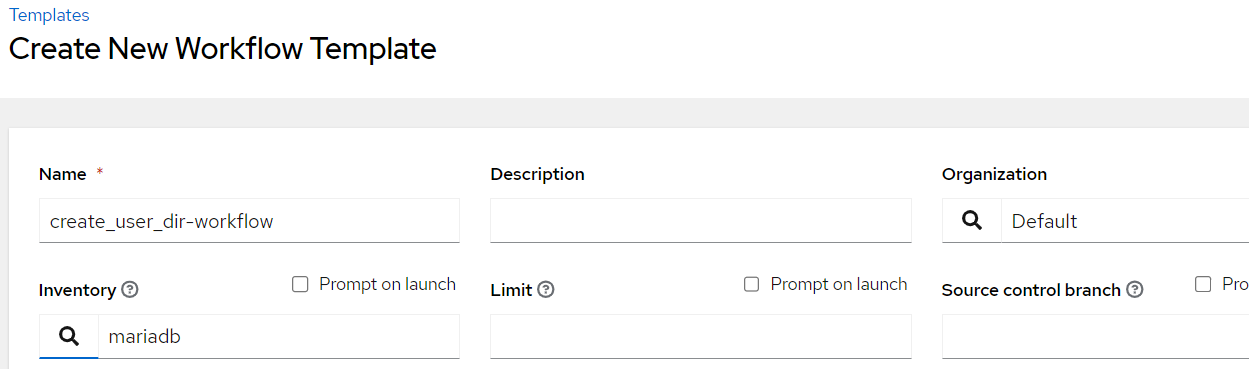
前面创建的工作模板是针对单个作业任务的，假如要执行一系列的作业任务，相互之间有依赖或者说需要按顺序及某些条件判断执行这些作业，这时可创建工作流（workflow）模板

现在有3个作业模板，想创建一个工作流程，首先执行“create\_user-job-temp1”，如果此作业运行成功则继续运行“create\_dir1-job”；如果“create\_user-job-temp1”运行失败，则继续运行“create\_dir2-job”

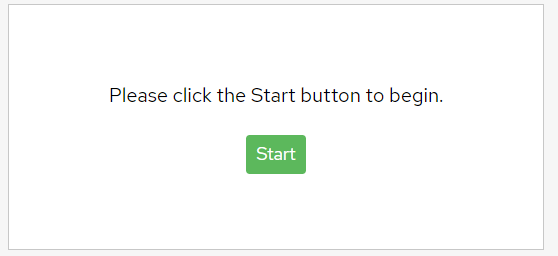


Resources→Templates→Add→Add workflow template

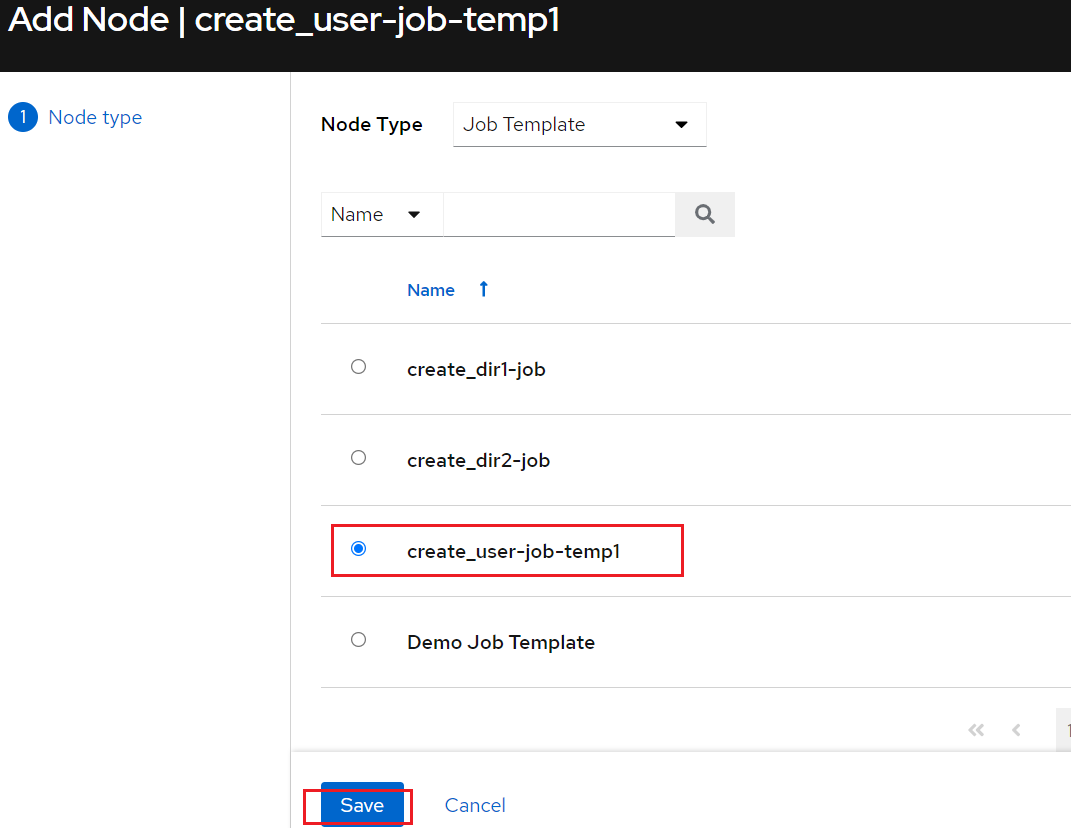


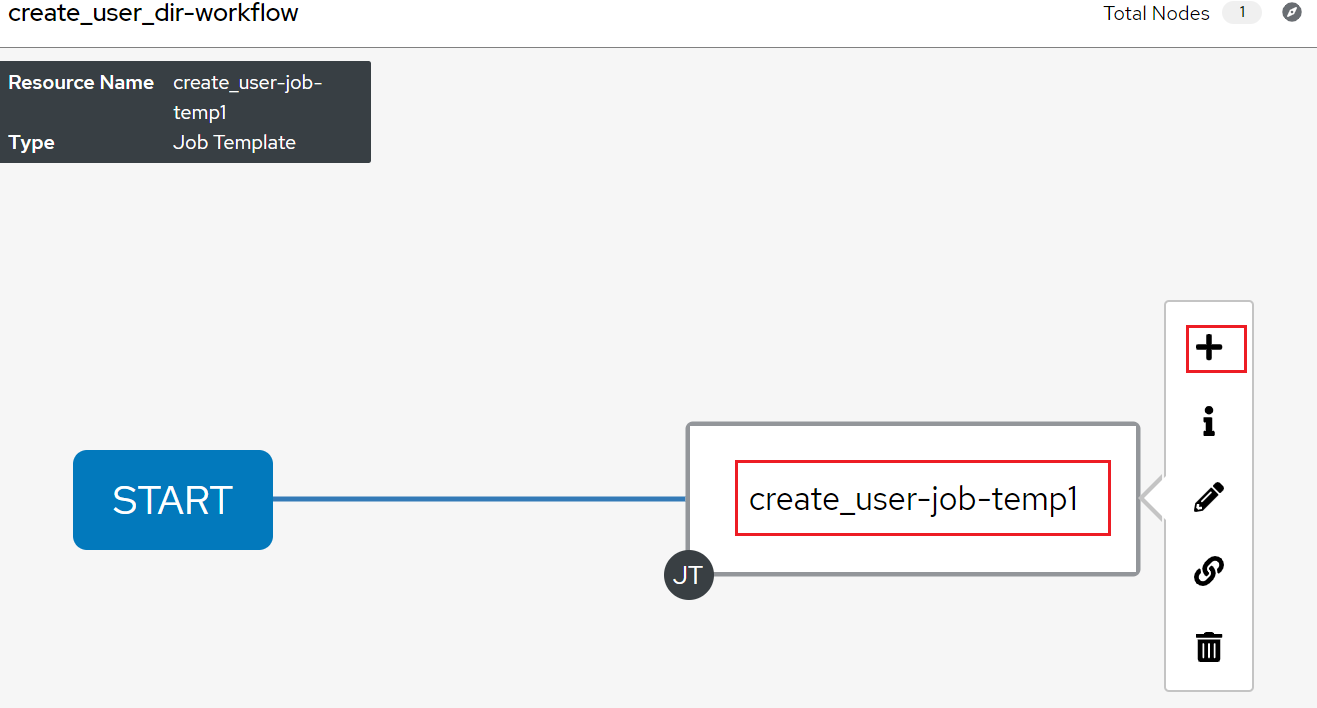


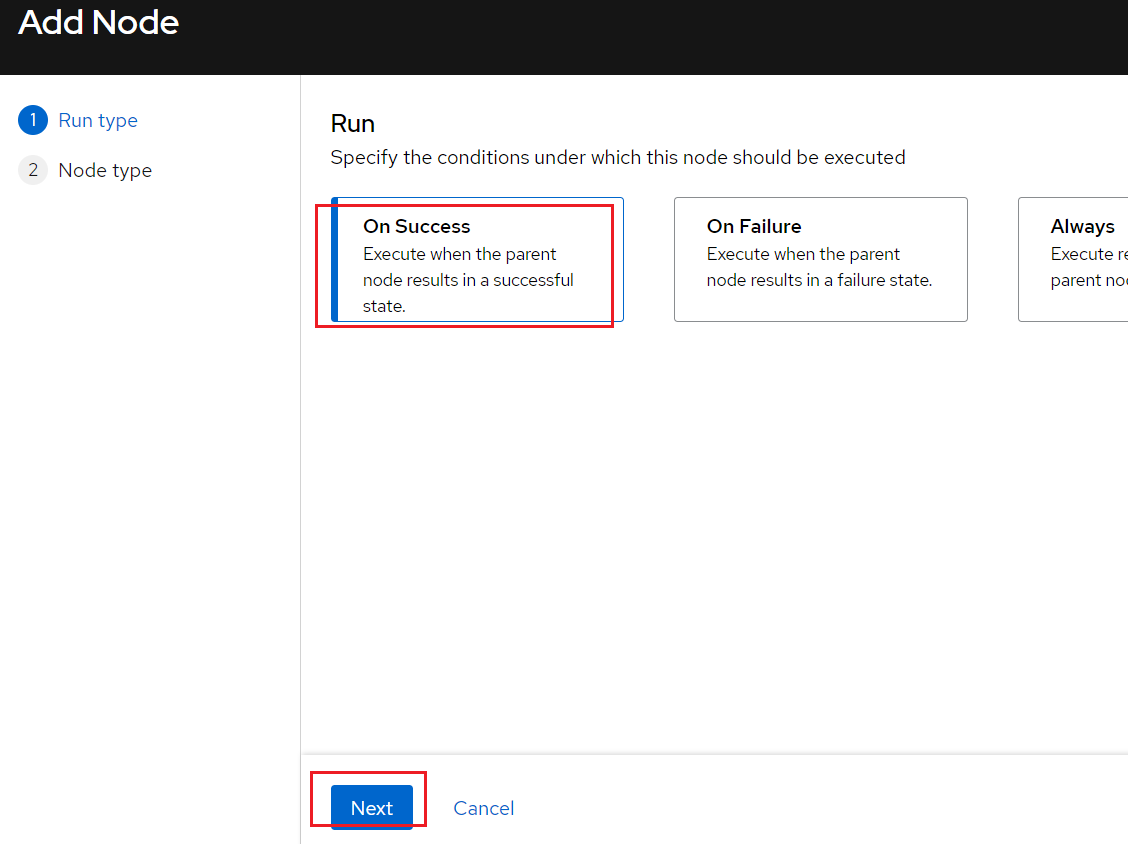
点击“Save”后：

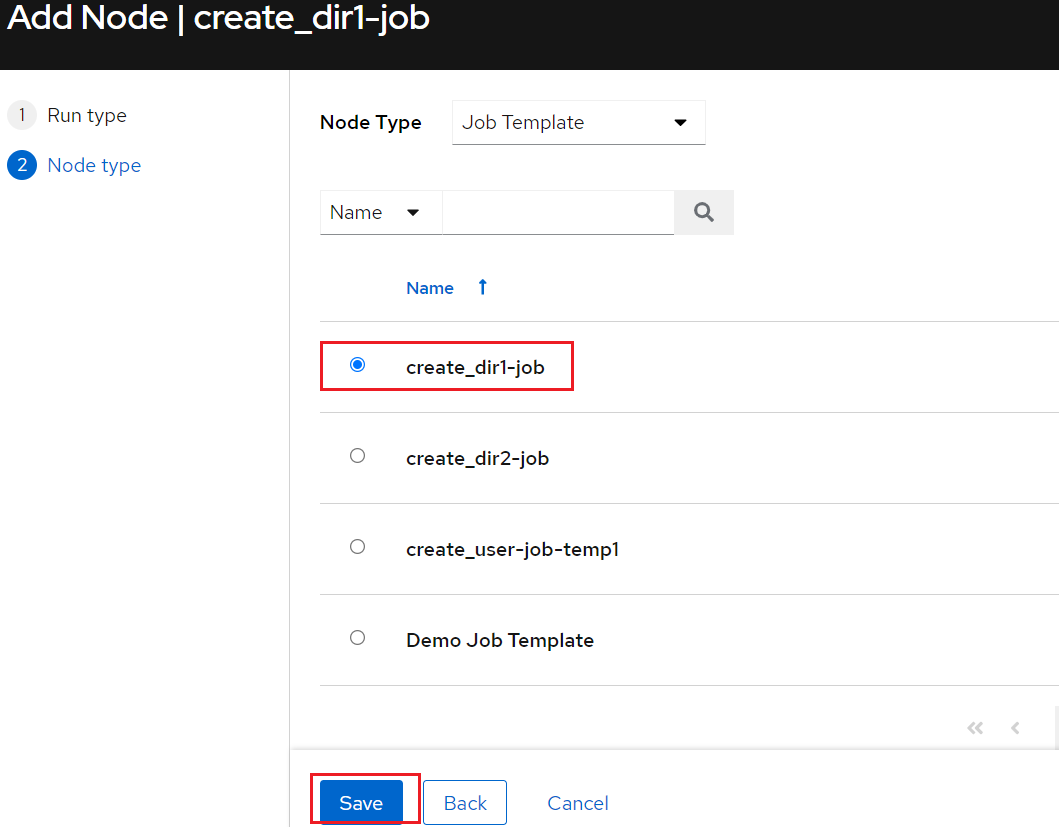


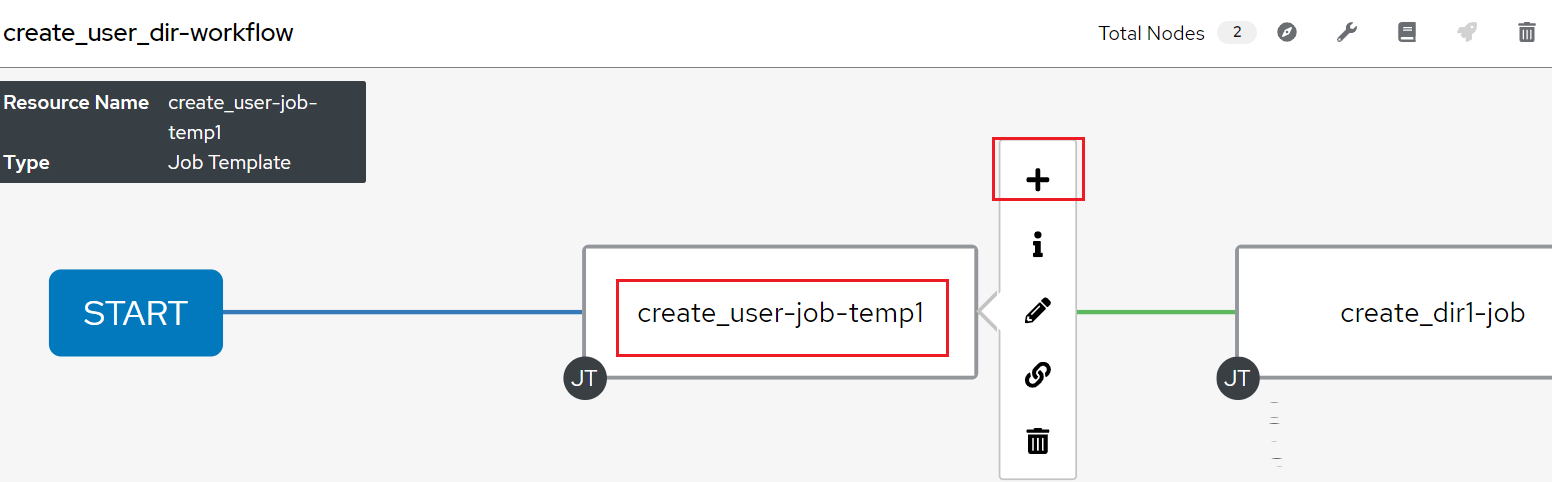
点击“Start”后进入工作流编辑界面：

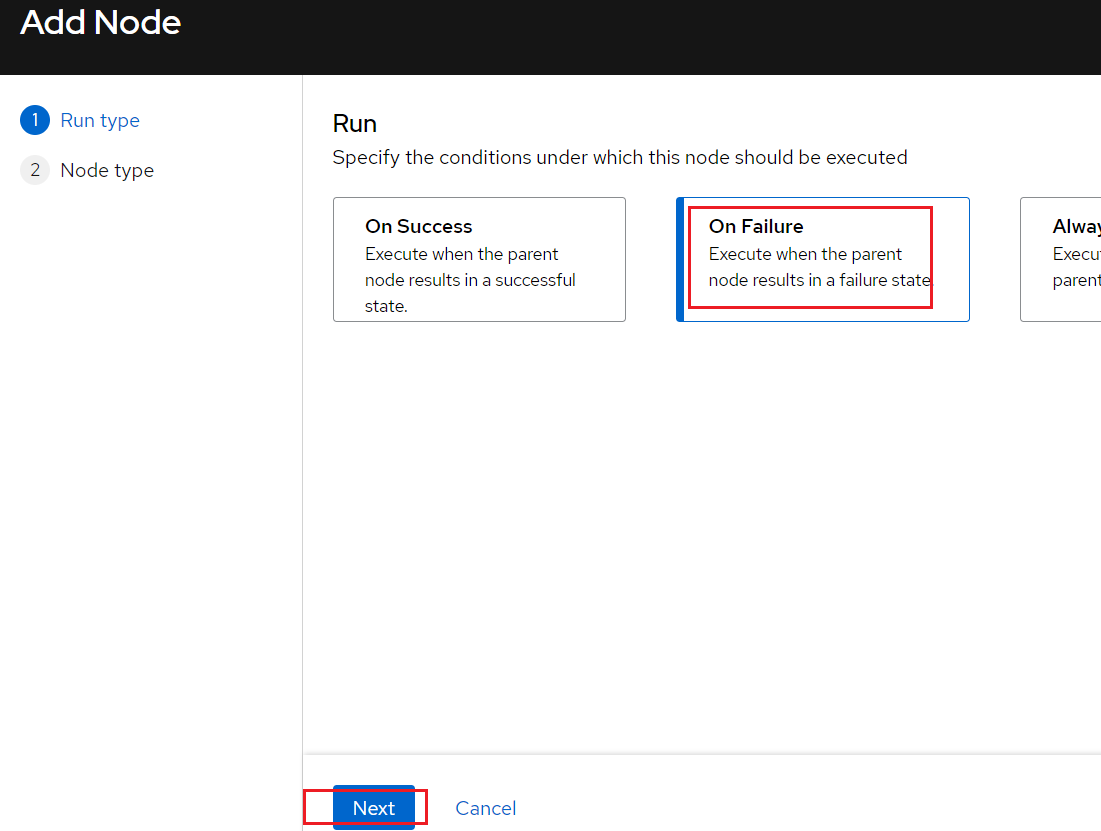


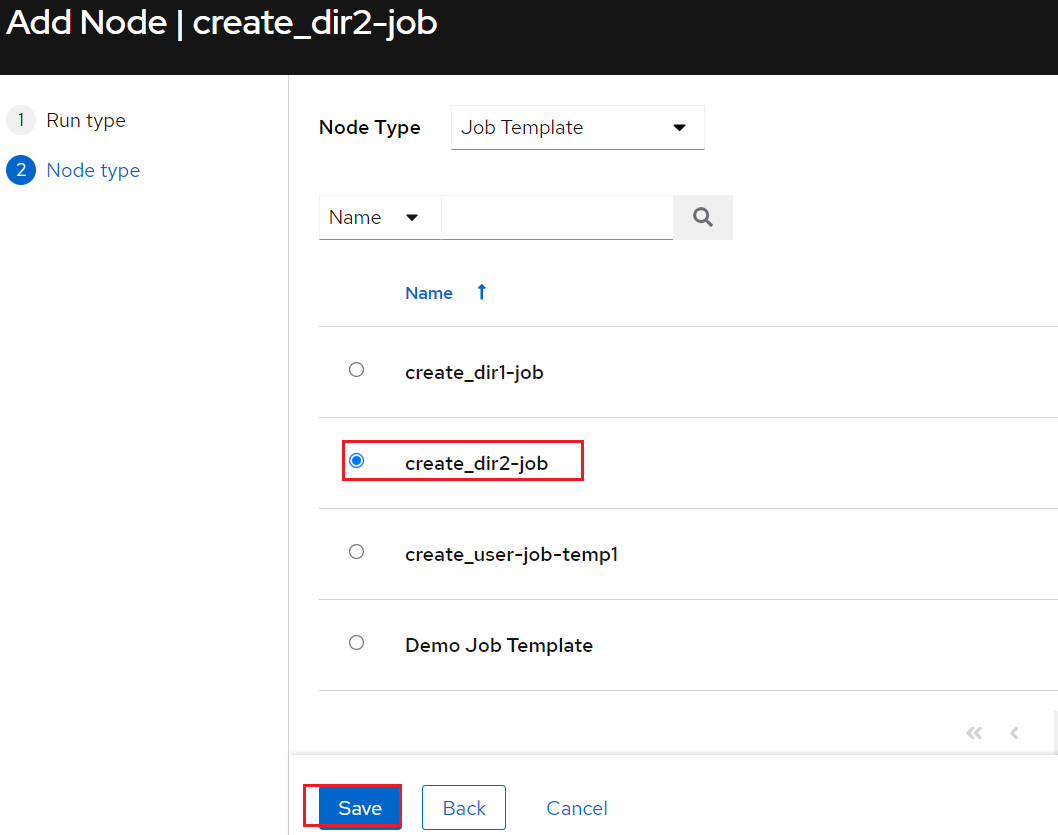


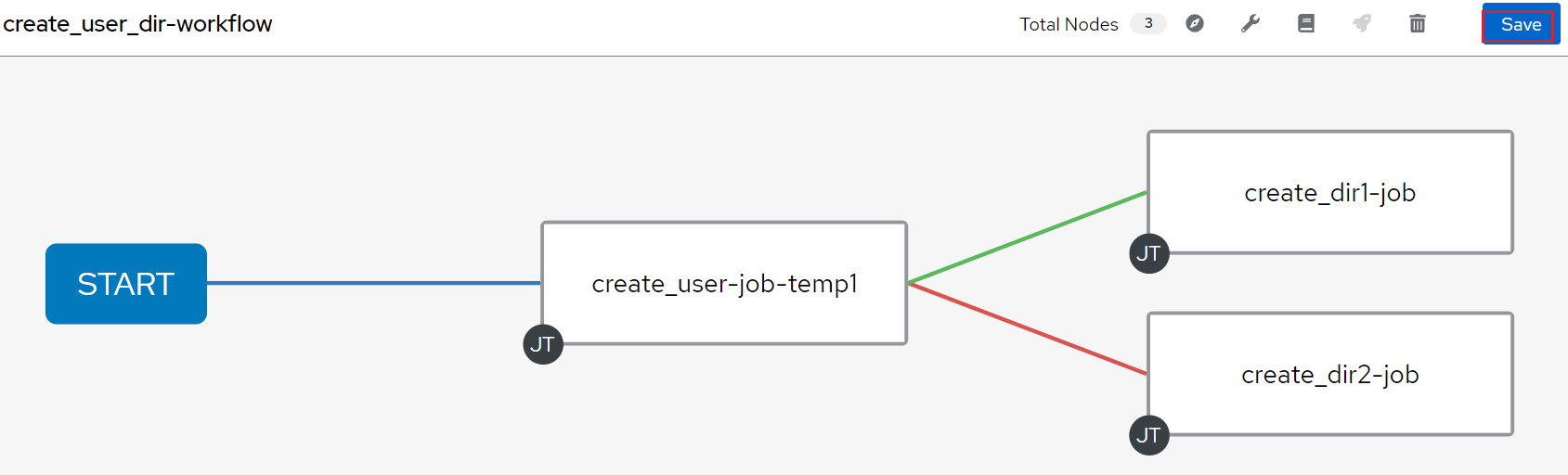




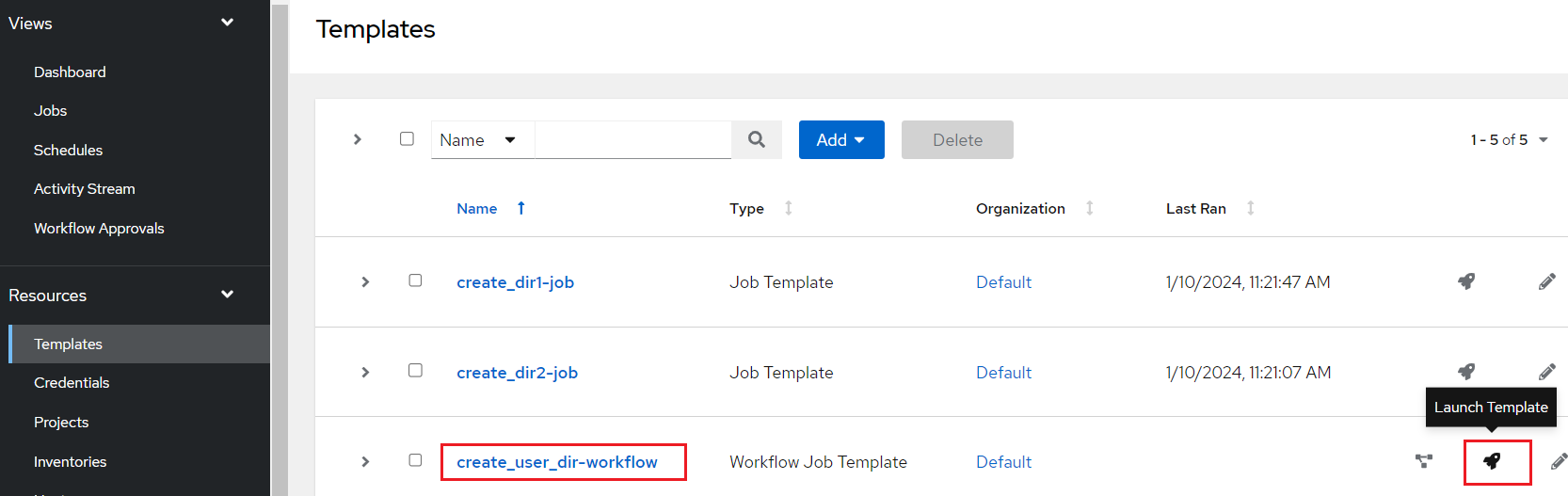


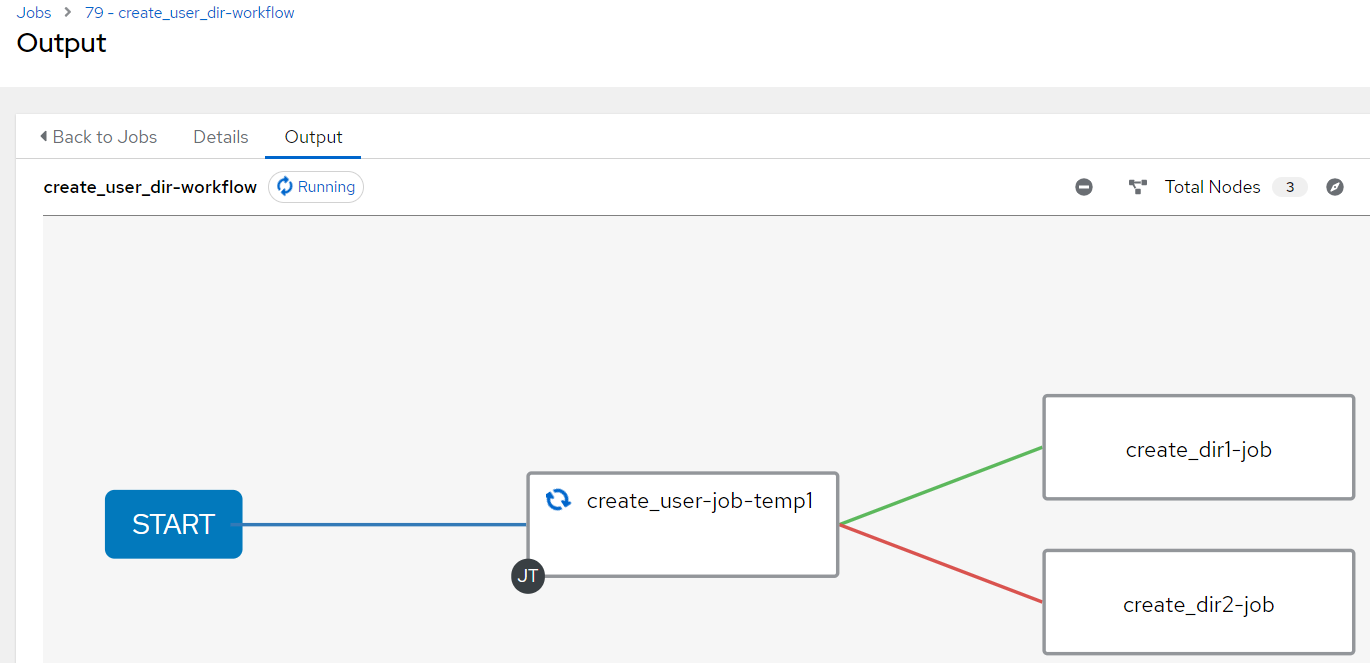




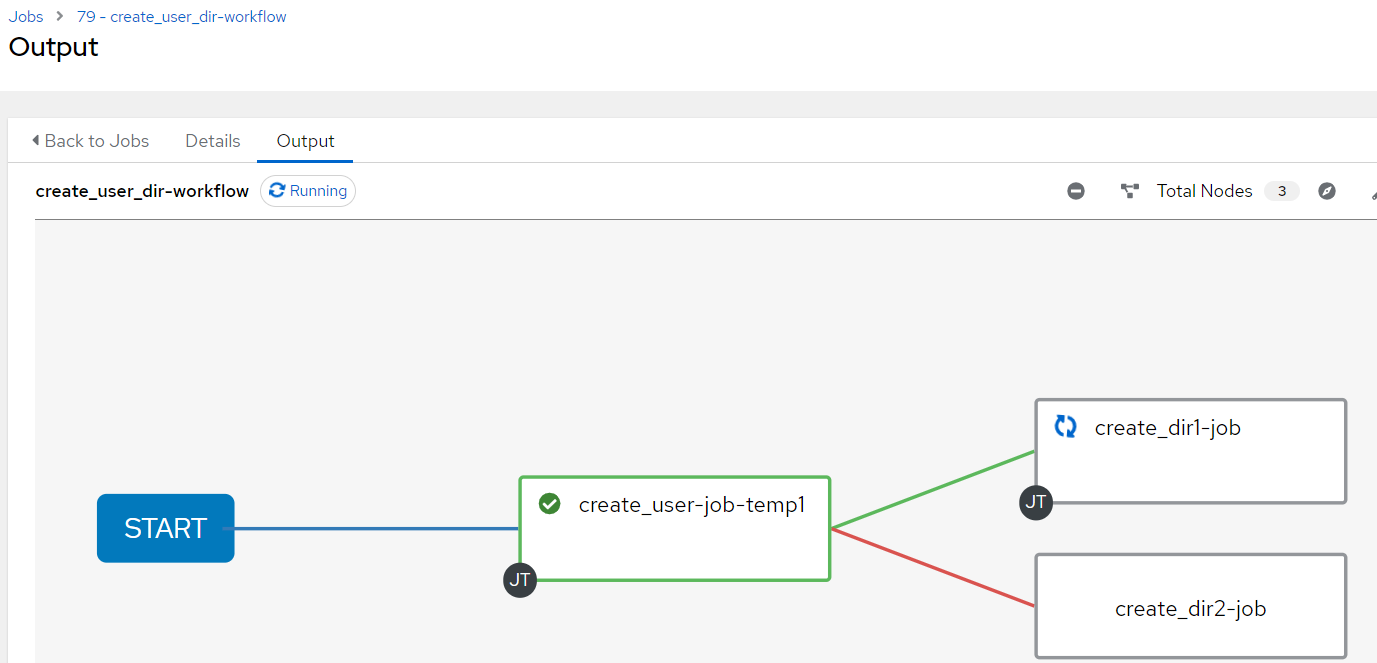


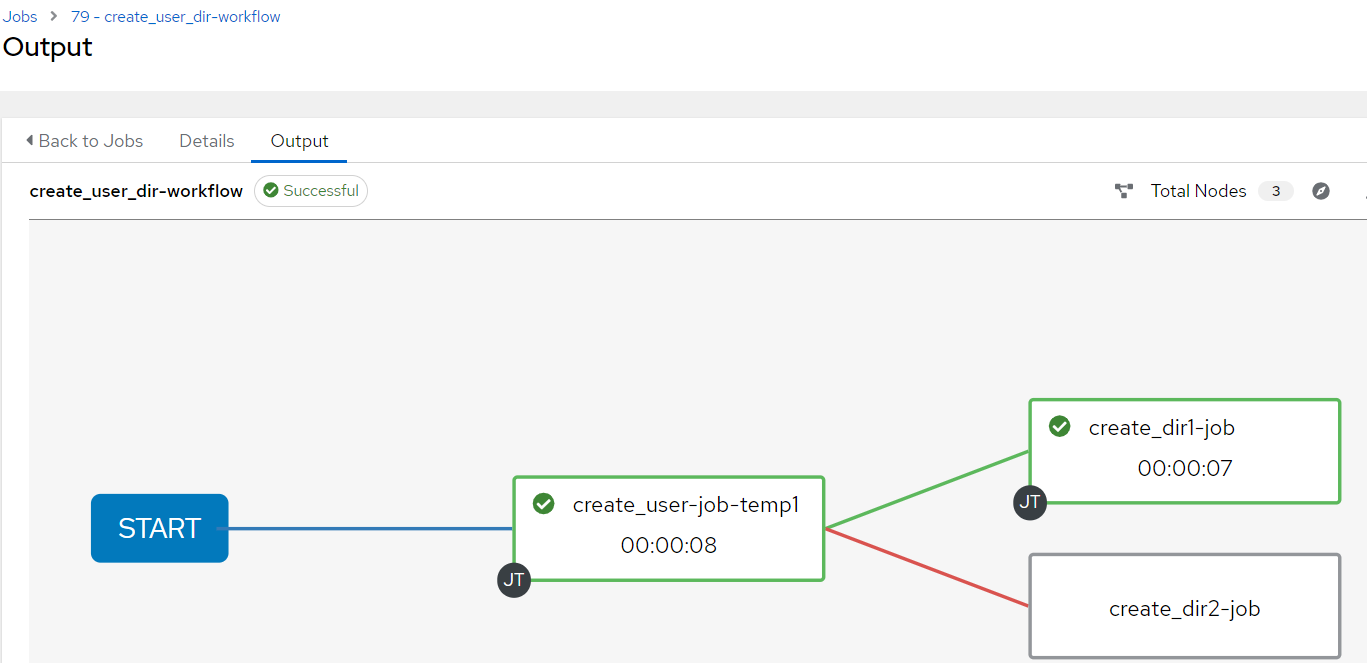
**执行工作流：**





首先执行“create\_user-job-temp1”，成功后再执行“create\_dir1-job”





**★开启Webhook功能**

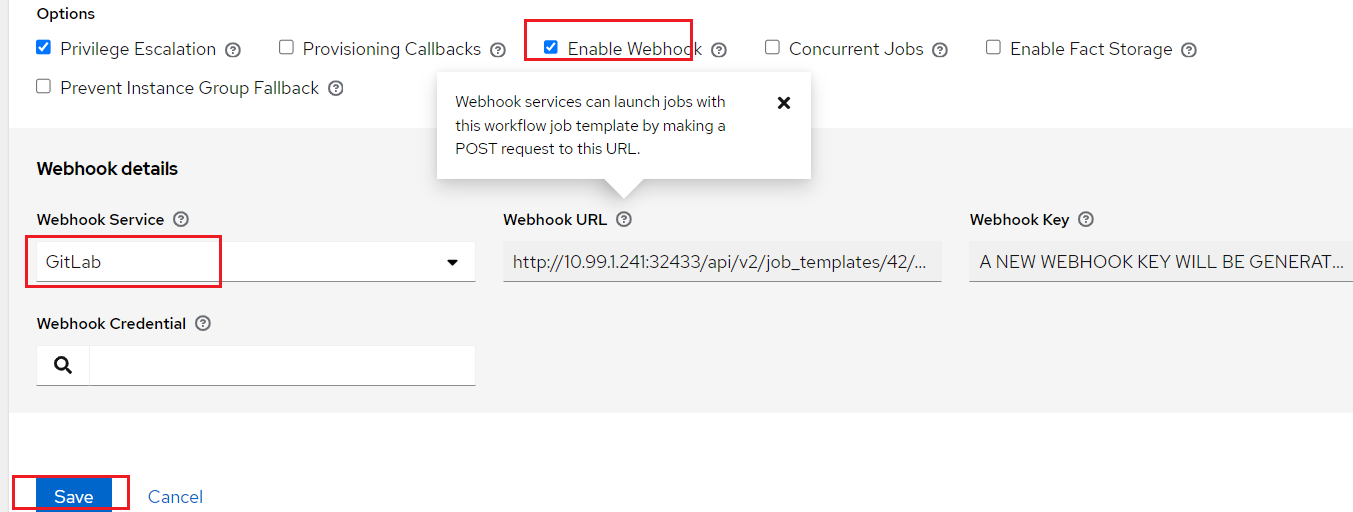
webhook功能表示当某项目关联的git仓库有更新时，由git服务器发送一个通知到本awx服务，然后触发此作业模板/工作流模板的执行

在创建作业模板/工作流模板时，勾选“Enable Webhook”，

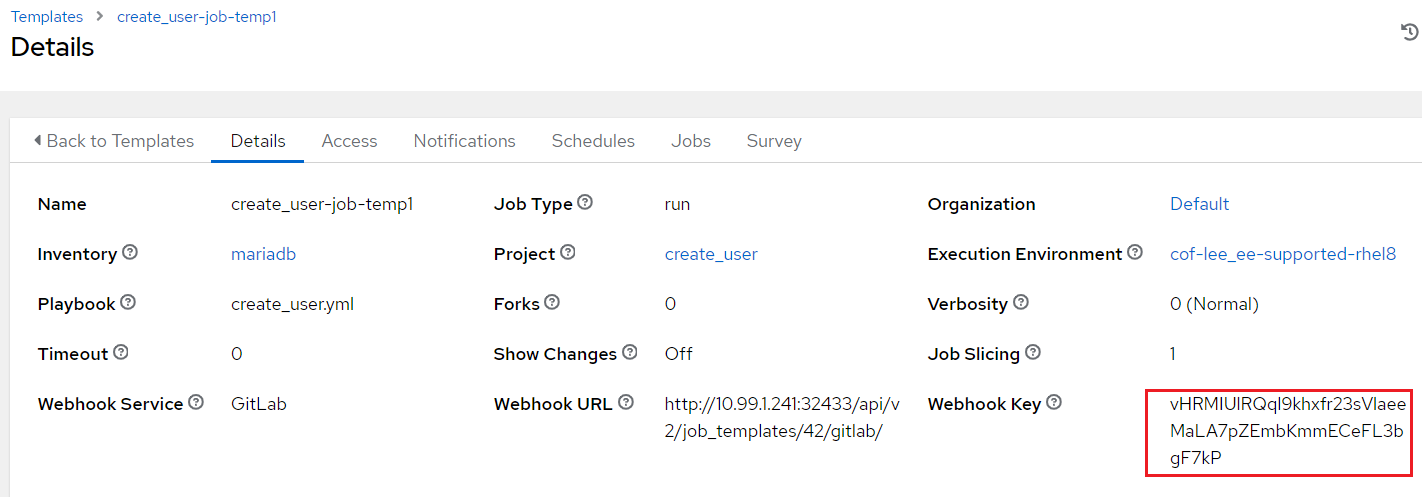
Webhook Service选择相应的git仓库类型，本例为“GitLab”，

Webhook URL是自动生成的，此信息是提供给gitlab服务器用的

Webhook Key是点击“Save”保存模板后自动生成，此信息也是提供给gitlab服务器用的



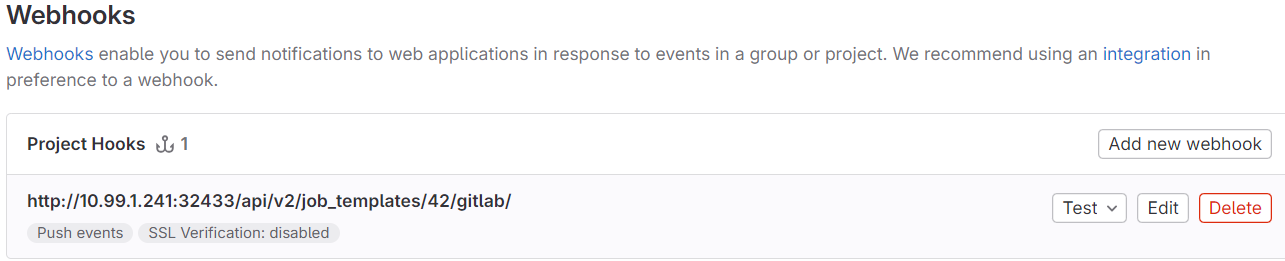
点击“Save”保存后查看详情，复制Webhook URL及Webhook Key信息：



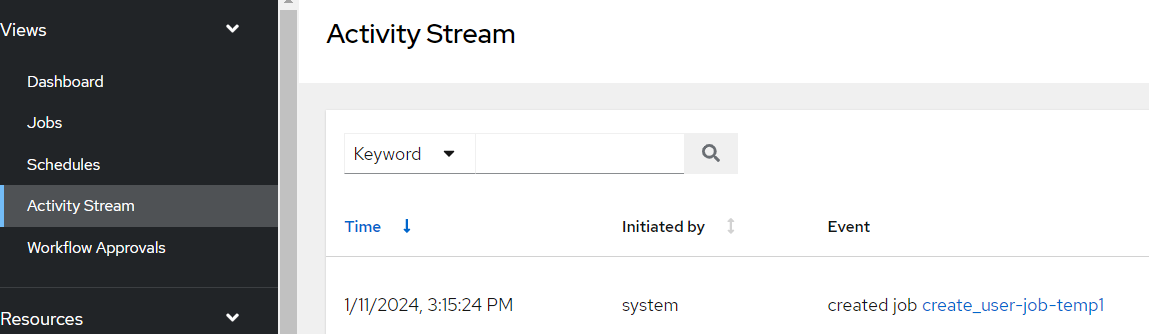
Webhook URL： http://10.99.1.241:32433/api/v2/job\_templates/42/gitlab/

Webhook Key： vHRMIUlRQql9khxfr23sVlaeeMaLA7pZEmbKmmECeFL3bgF7kP

然后在gitLab上配置相应的webhook：

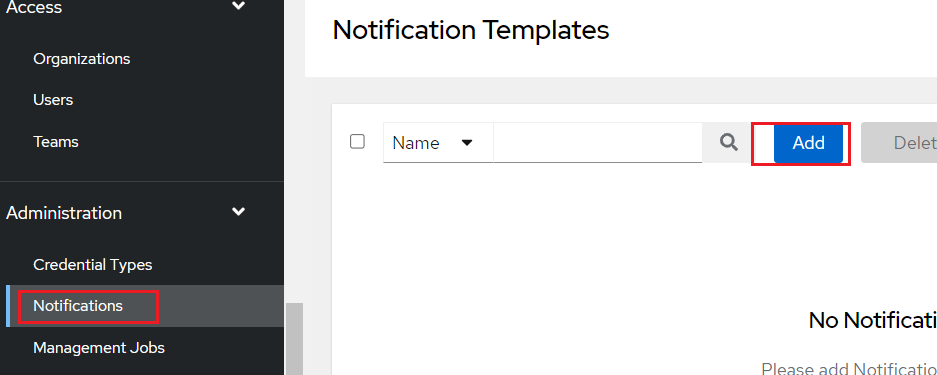


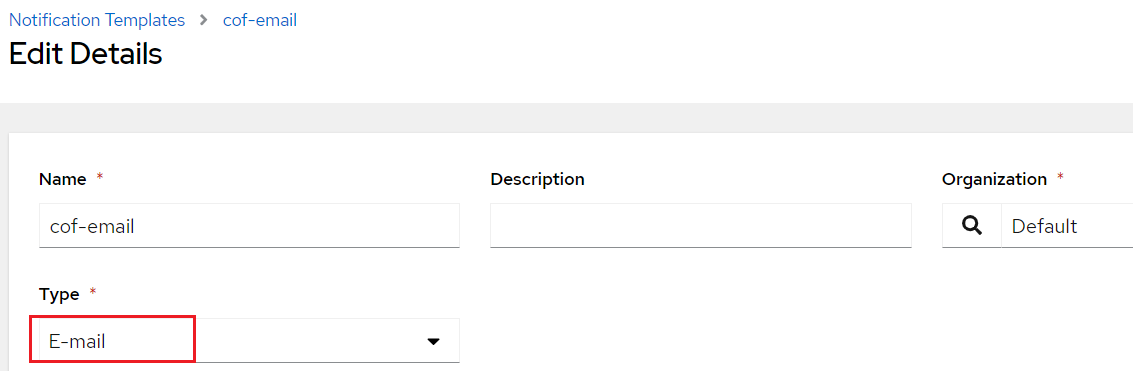
更新代码触发webhook后，此作业模板会收到触发通知而自动运行：





**★添加通知邮箱**







如果邮箱服务器未启用tls，则不勾选UseTLS（无论是否启用tls，端口都是25）

且“Use SSL”和“Use TLS”只能二选一，不可同时勾选