### 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：2023-11-02

**★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。

★ansible及ansible-core对应关系：

|  |  |
| --- | --- |
| ansible版本 | 依赖的ansible-core版本 |
| 2.8.x | 无 |
| 2.9.x | 无 |
| 2.10.7 | ansible-base-2.10.17 |
| 3.4.0 | ansible-base-2.10.17 |
| 4.10.0 | ansible-core-2.11.12 |
| 5.10.0 | ansible-core-2.12.10 |
| 6.7.0 | ansible\_core-2.13.12 |
| 7.7.0 | ansible\_core-2.14.10 |
| 8.4.0 | ansible\_core-2.15.4 |

**★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）**

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

# yum install python3.9 -y #安装python3.9版本，才支持更高版本的ansible

#创建虚拟环境并激活

# mkdir -p ~/python39-ansible-venv

# python3.9 -m venv ~/python39-ansible-venv

# source ~/python39-ansible-venv/bin/activate #激活python虚拟环境，可以执行deactivate命令退出此虚拟环境

# pip3 install -U pip #先更新pip软件

# pip3 install 'ansible>=4,<5' #安装ansible时指定版本号

# ansible --version #查看ansible版本

ansible [core 2.11.12]

config file = None

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

ansible python module location = /root/python39-ansible-venv/lib64/python3.9/site-packages/ansible

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

executable location = /root/python39-ansible-venv/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 = 3.1.2

libyaml = True

如果退出系统，之后重新登录，需要再次激活此python虚拟环境才有ansible命令：

# source ~/python39-ansible-venv/bin/activate #激活python虚拟环境

**★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

**★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（写同一行）

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：

host1

server2

host3

[group2]

server2

[all:vars] #表示所有主机默认的ssh相关信息（all表示所有主机）

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 --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

# 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），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

**★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.

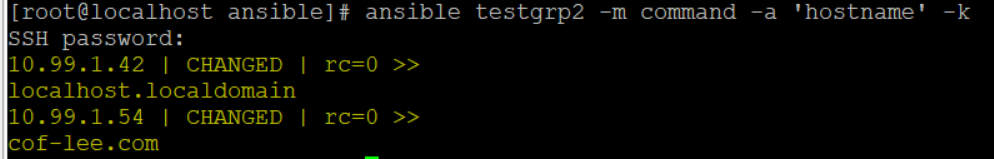
#解决方法为让ansible在远程登录主机时不检查host key，修改配置文件：

# 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



**★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

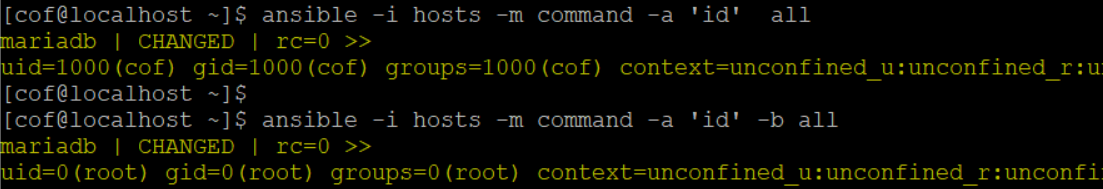
# ansible -i ./inventory *mariadb* -m command -a 'ip addr' #登录mariadb这台主机，执行'ip addr'命令，默认使用当前执行此ansible命令的用户的密钥去登录目标主机

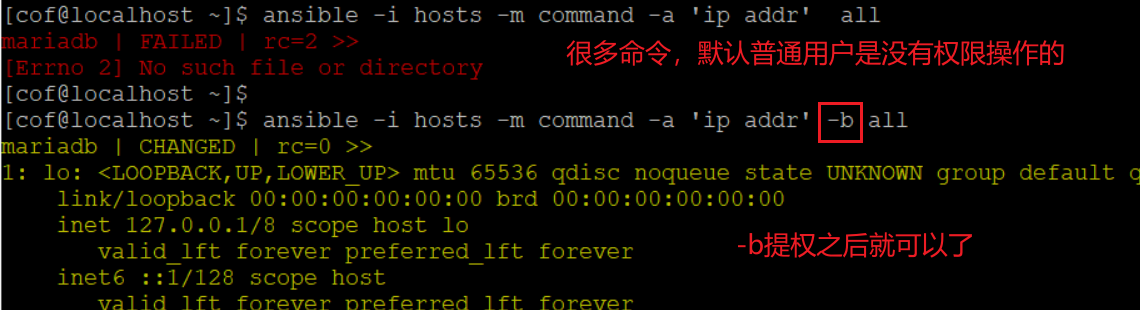
**★sudo提权**

[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

**★ansible用户提权4种方式：**

1、直接用root远程，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

# ansible.cfg配置文件里只设置需要提权，不写具体的方法：

[privilege\_escalation]

become=True

#具体提权方法在inventory里设置，不同主机设置不同的become方法：

# 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

#默认就是ansible\_become\_user=root ansible\_become\_method=sudo 不询问become\_pass

**★例：**

使用普通用户ssh远程登录，再su切换到root用户（非sudo）

# cat ansible.cfg

[defaults]

host\_key\_checking=false

inventory=/home/userxx/playbooks/inventory

[privilege\_escalation]

become=True

become\_method=su

become\_user=root

become\_ask\_pass=false

# cat /home/userxx/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

#或者把python解释器路径信息写到inventory里

# vi inventory

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

}

**★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.）

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

|  |  |
| --- | --- |
| [defaults] |  |
| inventory = /etc/ansible/hosts | 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=/var/log/ansible.log | 指定日志保存文件 |
| roles\_path=~/.ansible/roles/ | 指定role角色存储位置 |
| 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 |
| [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 |
| control\_path\_dir = ~/.ansible/cp |  |
| control\_path = %(directory)s/%%h-%%r |  |
| 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 |

**★第2章、ansible模块及使用**

**★ansible常用模块**

# ansible-doc -l #列出所有模块

# ansible-doc -l | wc -l #查看模块数量

3387 #ansible-2.9默认3387个模块，2.8版本有2834个模块

ansible2.8及以前版本只有role的概念（role角色，详见后面ansible-galaxy的章节）

从ansible-2.9开始，引入了collection的概念（模块的集合），因为模块越来越多，不同厂商及开发者书编写的模块可能会同名，造成冲突，于是将模块进行分组（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模块），将相同类型的模块放到同一个collection集合里。但2.9版本仍然将所有模块（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，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）

**★ansible模块分组：**

|  |  |
| --- | --- |
| ansible2.9及之前版本的模块名 | ansible2.10及之后的模块名（分组了） |
| firewalld | ansible.posix.firewalld |
| selinux | ansible.posix.selinux |
| sysctl | ansible.posix.sysctl |
| parted | community.general.parted |
| lvg | community.general.lvg |
| filesystem | community.general.filesystem |
| script | script（和旧版本相同，未分组） |
| shell | shell（和旧版本相同，未分组） |
|  |  |

★从2.10开始短名称的模块（未分组的）只有60多个了

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

68

2.10版本开始仍使用短名称的模块如下：

（不在以下列出的模块，需要注意区分当前的ansible版本以决定是否写分组后的FQCN名称或仍写短名称）

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

# 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

★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

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

# 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 -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 -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 -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 -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 -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 -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 -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 myrepo.repo

[myrepo] #同name参数值

baseurl = file:///mnt/

enabled = 1

gpgcheck = 1

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

name = my local repo #同description参数值

# 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，如果按2.10之后的标准来写，则sysctl模块的完全写法如下：

# 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：**

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

**★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 #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

★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

# 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**

# vi play-import-testx.yml

---

- name: play test ww # playbook 1

hosts: mariadb

gather\_facts: no

tasks:

- name: task ww test

debug:

msg: "debug-ww-test-1"

- name: import tag playbook # playbook 2

import\_playbook: play-tag-testx.yml #引入其他playbook文件，从这里开始执行此playbook的任务

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

**★文件操作相关模块**

**①file**

# vi play-file.yml

---

- name: play xx

hosts: mariadb

gather\_facts: no

tasks:

- name: task file xx

file: #在目标主机上创建/删除/属性设置某文件

path: /tmp/test.xx

owner: cof

group: cof

mode: 0640

state: touch #touch表示创建文件

- name: stat file

stat:

path: /tmp/test.xx

register: return\_stat

- name: debug stat

debug:

var: return\_stat

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

**★创建目录及设置selinux文件类型**

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**

file:

src: /dirxx #源文件/目录

dest: /tmp/dirxx #链接

state: link #软链接

**②copy**

**★新建文件并写入一行内容**

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

★将执行机本地某文件复制到目标主机上

copy:

src: test.sh #执行机本地的文件

dest: /tmp/test.sh #复制到目标主机上

**③synchronize**

# vi play-synchronize.yml

---

- name: play xx

hosts: mariadb

gather\_facts: no

tasks:

- name: task file xx

synchronize: #使用rsync同步文件，**执行机及目标主机上要安装有rsync命令**

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

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

**④fetch**

# vi play-fetch.yml

---

- name:

hosts: mariadb

gather\_facts: no

tasks:

- name: task file xx

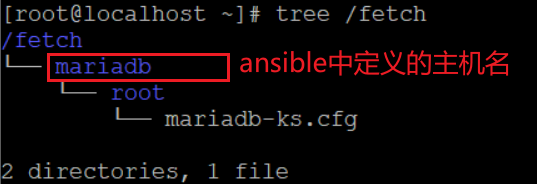
fetch: #将文件从目标主机上复制到执行机，以文件树的格式保存，拷贝完整路径

src: /root/mariadb-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

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



**⑤lineinfile**

# vi play-lineinfile.yml

---

- name: play xx

hosts: mariadb

gather\_facts: no

tasks:

- name: task file xx

lineinfile: #在目标主机上操作某文件（行操作）

path: /etc/hosts

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

#其他操作

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**

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**

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"

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的模块多达三四千个，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

**★挂载光盘**

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

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**

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

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

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**

lvg: #创建vg并加入物理卷

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

lvg:

vg: vg\_xx

pvs: /dev/sdb2,/dev/sdb3

pesize: 32

lvol: #在已创建的vg里创建lv

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**

filesystem:

fstype: ext4

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

**★创建目录**

file:

**path**: /lvpoint

state: directory

owner: cof

group: cof

mode: '0775'

setype: httpd\_sys\_content\_t

**★挂载文件系统**

mount:

path: /lvpoint

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

fstype: ext4

state: present #只写入/etc/fstab，未立即挂载

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**

firewalld:

service: http

permanent: yes #永久配置

state: enabled #允许放通

immediate: yes #立即生效

firewalld:

zone: public # public的zone参数可省略

port: 8200-8210/tcp

permanent: yes

state: enabled

immediate: yes

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**

shell:

cmd: bash test.sh

args:

chdir: /tmp

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

**★cron**

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

查看效果：

# crontab -l -u cof

#Ansible: ping gw

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

**★uri**

uri: #使用uri模块

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**

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**

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') }}"

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，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：**

debug: #模块名

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

- "{{ 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

# vi test\_vars.yml

---

- name: playxx

hosts: mariadb

vars:

varxx: varxxxxxxx

varyy: varyyyyyyy

tasks:

- name: taskxx

debug:

msg:

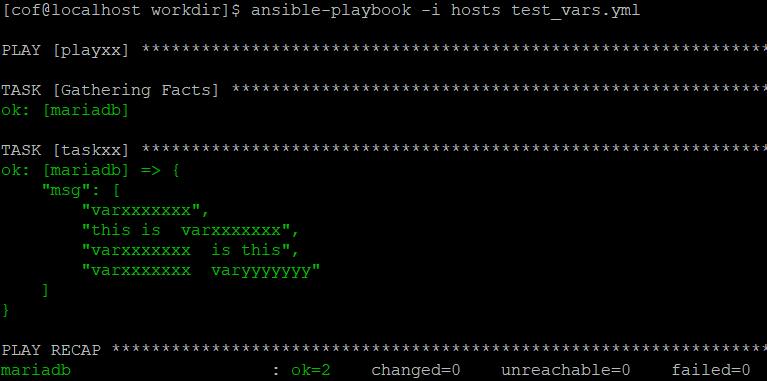
- "{{ varxx }}"

- 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

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



# vi mytask3.yml #内容如下，yaml文件格式

---

- name: playbook name xxx

hosts: mariadb

vars: #在playbook清单文件中定义变量

username: xyz

userid: 1490

tasks:

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

user: #使用user模块

name: "{{username}}"

uid: "{{userid}}"

★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：

# vi mytask3-vars.yml #内容如下，一行一个变量

username: xyz

userid: 1490

# vi mytask3.yml #内容如下，yaml文件格式，以---开头

---

- 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}}

user: #使用user模块

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

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

★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 hosts mytask3.yml -e "username=helo userid=1399"

★命令行手动引入变量文件

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

**★数组变量**

创建一个变量文件：

# vi uservars.yml

users:

user1:

username: user1

userid: 1421

userhome: /home/user1

user2:

username: user2

userid: 1422

userhome: /home/user2

# vi mytask4.yml

---

- 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}}

user: #使用user模块

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

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

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

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

user: #使用user模块

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

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

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

# ansible-playbook mytask4.yml

**★include模块引入变量**

# vi uservars2.yml

users:

username1: user1

username2: user2

# vi mytask5.yml

---

- name: playbook name xxx

hosts: mariadb

tasks:

- name: include users

include\_vars: uservars2.yml #在任务中引入变量文件

- name: debug vars imported

debug:

msg: >

"imported {{ users.username1 }} and {{users['username2']}}"

# ansible-playbook -i hosts mytask5.yml

**★include引入task**

# vi taskxx.yml

- name: new user taskxx

user:

name: "{{username}}"

uid: "{{userid}}"

state: present

# vi playxx.yml

---

- name: my play xx

hosts: mariadb

tasks:

- name: include task from file

include: taskxx.yml

vars:

username: userxx

userid: 1423

# ansible-playbook -i hosts playxx.yml

# vi task6.yml

- 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

service:

name: "{{servicename}}"

state: "{{servicestate}}"

# vi play6.yml

---

- name: my play xx

hosts: mariadb

tasks:

- name: include task from file

include: task6.yml

vars:

servicename: sshd

servicestate: restarted

register: output

- name: debug include task

debug:

var: output

# ansible-playbook -i hosts play6.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 setup #查看所有获取的信息

#查看指定的信息

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

# ansible -i inventory mariadb -m setup -a 'filter="ansible\_default\_ipv4"'

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

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

# ansible -i inventory mariadb -m 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"

}

}

},

# vi facts.yml

---

- name: get facts xx

hosts: mariadb

tasks:

- name: print macaddress

debug:

msg:

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

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

#上面第二级ens33，在查看时为ansible\_ens33，在使用时要去掉ansible\_

# ansible-playbook facts.yml

TASK [print macaddress 10.99.1.49] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [mariadb] => {

"msg": [

"the ip address is 10.99.1.49",

"the macaddress is 00:0c:29:66:75:78"

]

★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

setup: #这里是一个任务（setup任务的内容就这一行，下面不是它的参数了）

- name: print macaddress #接下来是其他任务，可正常获取facts变量

**★同一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刷新facts信息

- name: refresh facts

setup:

filter: ansible\_lvm

**★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# vi /etc/ansible/facts.d/mariadb.fact #内容如下

[test]

var1=value1

var2=value2

然后在ansible执行机查看：

# ansible -i hosts mariadb -m setup -a 'filter="ansible\_local"'

mariadb | SUCCESS => {

"ansible\_facts": {

"ansible\_local": {

"mariadb": {

"test": {

"var1": "value1",

"var2": "value2"

}

}

},

**★playbook及template里写变量的区别**

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

**★setup模块的filter里的facts变量**

**二级全称**，只可用二级过滤，双引号前后不可有空格

# ansible mariadb -m 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）

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

"{{ ansible\_facts.ens33.ipv4.address }}"

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

也可从二级开头，这时二级不可去除“ansible\_”字符串前缀，

"{{ ansible\_ens33.ipv4.address }}"

**★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: ansible\_default\_ipv4["address"] == "10.99.1.43"

**★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，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 }}"

"{{ 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 }}

{{ ansible\_default\_ipv4['address'] }} {{ ansible\_facts.hostname }}

**★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

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

# vi newuser-play.yml

---

- name: create new user

hosts: mariadb

vars\_files:

- userinfo.secret

tasks:

- name: create new user task

user:

name: "{{ item.name }}"

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

with\_items: "{{ newusers }}"

# 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

# vi play-test1.yml

---

- name: add users

hosts: mariadb

tasks:

- name: add user

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

# ansible-playbook -i hosts play-test1.yml

**②字典loop**

# vi play-test2.yml

---

- name: add users

hosts: mariadb

tasks:

- name: add user

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

# ansible-playbook -i hosts play-test2.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

---

- name: debug add users

hosts: mariadb

gather\_facts: no

tasks:

- name: debug add user task

debug:

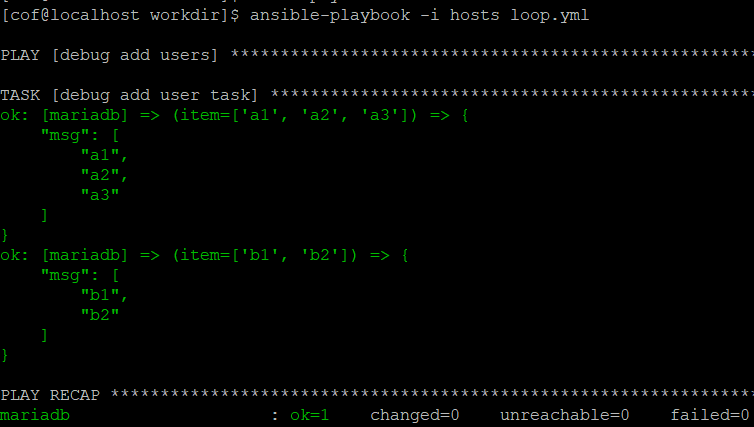
msg: "{{ item }}"

loop:

- [a1,a2,a3]

- [b1,b2]

loop执行结果：



---

- name: debug add users

hosts: mariadb

gather\_facts: no

tasks:

- name: debug add user task

debug:

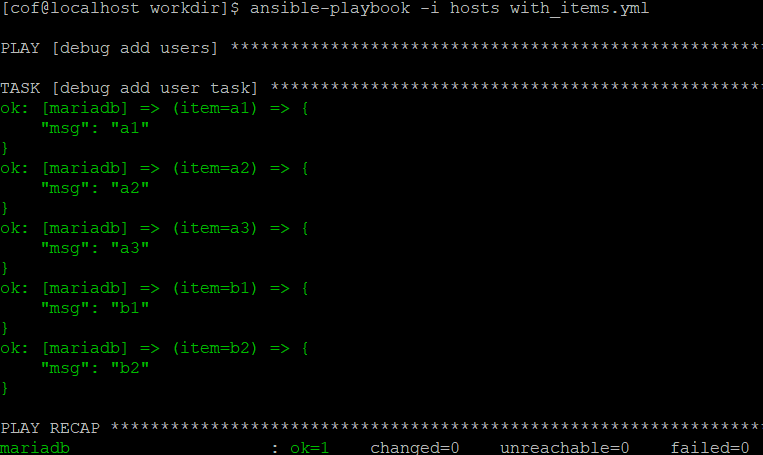
msg: "{{ item }}"

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

- [a1,a2,a3]

- [b1,b2]

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……）

- name: debug add users

hosts: mariadb

gather\_facts: no

tasks:

- name: debug add user task

debug:

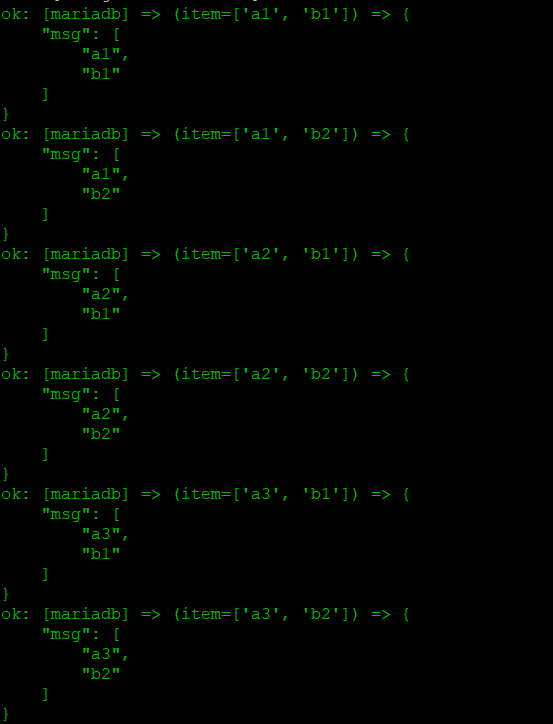
msg: "{{ item }}"

with\_nested:

- [a1,a2,a3]

- [b1,b2]

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**

# vi myvars.yml

user1: [a1,b1,c1,d1]

user2: [a2,b2,c2,d2]

# vi play-test7.yml

---

- name: debug add users

hosts: mariadb

gather\_facts: no

tasks:

- name: include vars

include\_vars: myvars.yml

- name: debug add user task

debug:

msg: "{{ item }}"

loop:

- "{{ user1 }}"

- "{{ user2 }}"

# ansible-playbook -i hosts play-test7.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**

# vi play-test1.yml

---

- 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

debug:

msg: running task now

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

# ansible-playbook -i hosts play-test1.yml

**②判断变量是否已定义**

# vi play-test2.yml

---

- 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

debug:

msg: running task now

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

# ansible-playbook -i hosts play-test2.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

# vi play-test3.yml

---

- name: test when

hosts: mariadb

vars:

surpportarch:

- x86\_64

- amd64

- x64

tasks:

- name: test when run my task

debug:

msg: running task now

when: ansible\_architecture in surpportarch

# ansible-playbook -i hosts play-test3.yml

**★register返回任务运行结果及判断**

# vi play-test4.yml

---

- name: test when

hosts: mariadb

tasks:

- name: test run my task

shell: "cat /etc/resolv.conf"

**register**: ret\_msg #返回运行状态结果的字典

ignore\_errors: true

- name: debug ret\_msg succeeded

debug:

msg: running task debug ret\_msg succeeded

when: ret\_msg is succeeded

- name: debug ret\_msg failed

debug:

msg: running task debug ret\_msg failed

when: ret\_msg is failed

- name: debug ret\_msg skipped

debug:

msg: running task debug ret\_msg skipped

when: ret\_msg is skipped

- name: debug ret\_msg changed

debug:

msg: running task debug ret\_msg changed

when: ret\_msg is changed

# ansible-playbook -i hosts play-test4.yml

**★changed\_when**

# vi play-test1.yml

---

- name: play test1

hosts: mariadb

tasks:

- name: task 1

shell: 'echo haha'

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

debug:

msg: "debug changed"

# ansible-playbook -i hosts play-test1.yml

**★failed\_when**

# vi play-test2.yml

---

- name: play test2

hosts: mariadb

tasks:

- name: task 1

shell: 'setenforce 0'

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

debug:

msg: "debug changed"

# 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**

# vi ignore\_errors.yml

---

- name: play ig

hosts: test66

tasks:

- name: txx

shell: init 6

async: 1

poll: 0

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

- name: tdeb

debug:

msg: after init task

# 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**

# vi fail.yml

---

- name: pxx

hosts: test66

tasks:

- name: txx

fail:

msg: "sdd is not exist"

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

# ansible-playbook fail.yml

#fail模块用来创建一个失败的任务，起到中断playbook的作用

**★block-rescue-always流程处理**

# vi play-block-testx.yml

---

- 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

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

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

debug:

msg: "always test"

# 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

# vi play-testhandlers.yml

---

- 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

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

service:

name: sshd

state: restarted

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

**★给任务打tags**

# vi play-tag-testx.yml

---

- name: play test xx

hosts: mariadb

gather\_facts: no

tasks:

- name: task deb1 test

debug:

msg: "debug-test-1 deb1"

tags:

- deb1

- name: deb2 test

debug:

msg: "xxx debug 2 test deb2"

tags:

- deb2

- name: deb3 test

debug:

msg: "always debug 3 test deb-always"

tags:

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

# ansible-playbook play-tag-testx.yml #正常执行所有任务

# ansible-playbook play-tag-testx.yml --tags=deb1 #只执行tag值为deb1的任务

# 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 --list-tags #查看所有tag

# ansible-playbook play-tag-testx.yml --skip-tags=deb2 #跳过此tag的任务

# ansible-playbook play-tag-testx.yml --tags=tagged #只执行所有带tag的任务

# 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 #}

# vi template-xx.j2

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

# vi play-j2.yml

---

- name: play xx

hosts: all

tasks:

- name: task xx

template: #是template模块负责调用jinja2

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:

shell: cat /tmp/test\_config

register: return\_test\_config

- name:

debug:

var: return\_test\_config

# ansible-playbook -i hosts 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 %} |  |

# vi play-j2-for.yml

---

- name: play j2 for

hosts: mariadb

vars:

users:

- cof

- lee

- tom

tasks:

- name: task j2 for

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

# ansible-playbook -i hosts 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 hosts 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

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

**★第7章、ansible任务处理**

**★ansible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

# vi play-delegate\_to.yml

---

- name: play xx

hosts: 10.99.1.248

tasks:

- name: task1 restart httpd

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

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

# ansible-playbook -i hosts 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

**★ansible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**

# vi play-async.yml

---

- name: play xx async

hosts: mariadb

tasks:

- name: task sleep xx

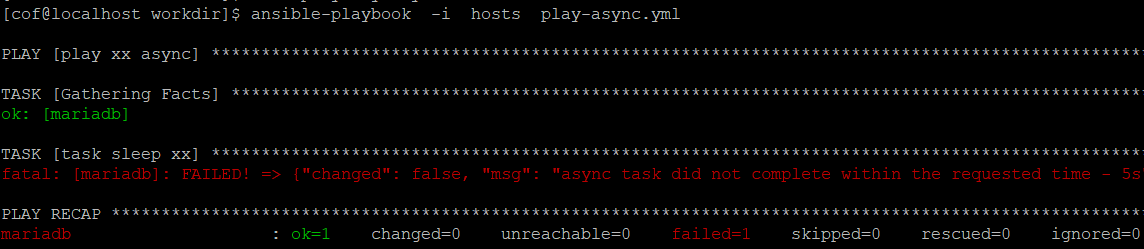
shell:

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

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



**②async\_status模块**

# vi play-async.yml

---

- name: play xx async

hosts: mariadb

tasks:

- name: task sleep xx

shell:

cmd: "sleep 100" #执行长时间的任务

async: 600 #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

async\_status: #所以判断任务最好放在最后执行

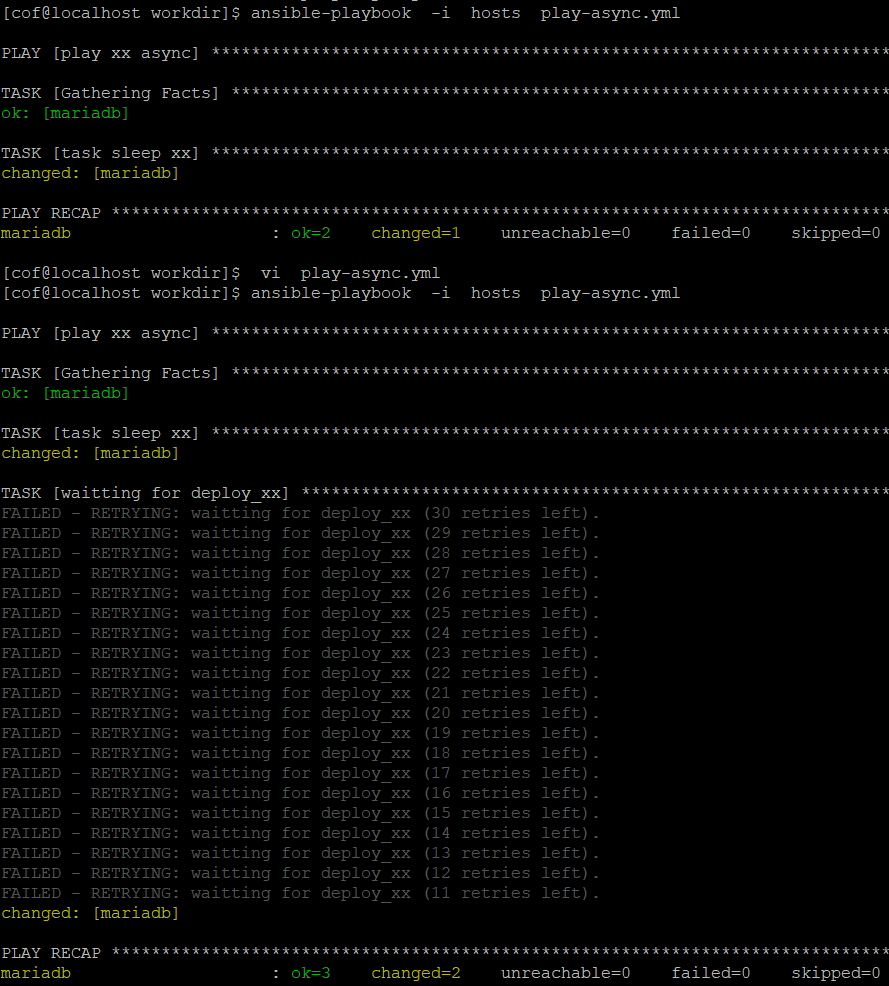
jid: "{{ deploy\_xx.ansible\_job\_id }}"

register: deploy\_xx\_result

until: deploy\_xx\_result.finished

retries: 30 #检查30次，默认5秒检查一次

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



默认每5秒检查一次

**③wait\_for模块**

# vi play-wait\_for.yml

---

- name: play xx

hosts: mariadb

tasks:

- name: task reboot os

shell: init 6

async: 1

poll: 0

ignore\_errors: true

- name: waitting for os start

wait\_for: #判断某tcp端口是否可连接

host: "{{ inventory\_hostname }}"

port: 22

connect\_timeout: 2

timeout: 5

search\_regex: "OpenSSH"

register: check\_ssh\_port

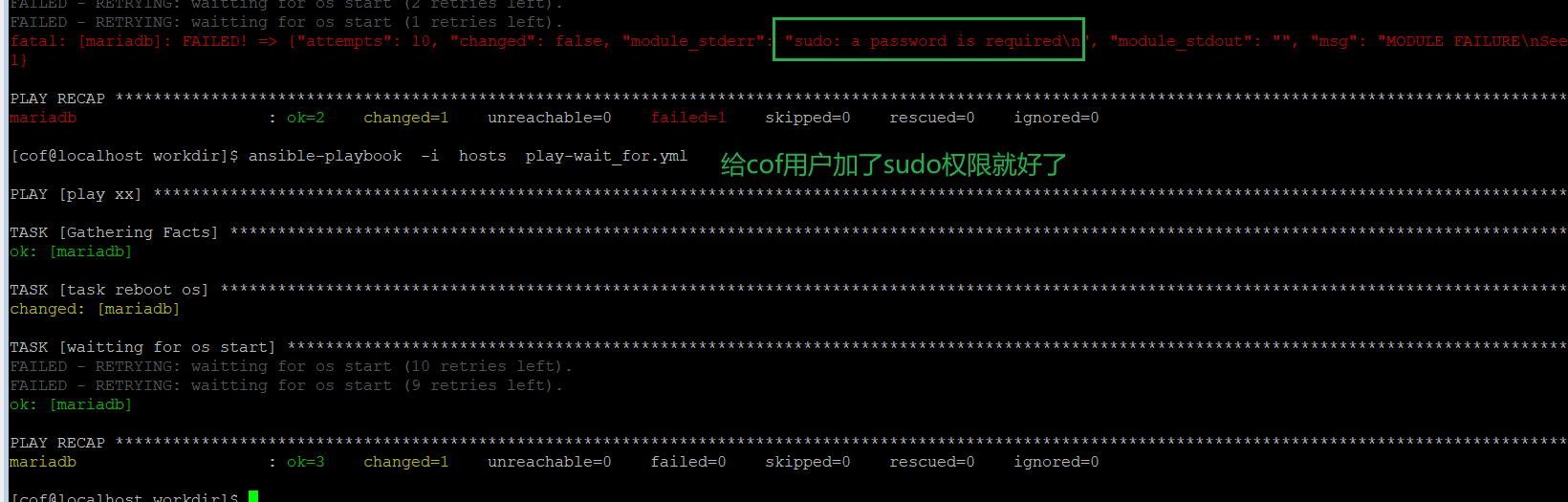
until: check\_ssh\_port is success

retries: 10 #检测10次，默认每5秒一次

delay: 6

delegate\_to: localhost #委托给ansible执行机去检测

# ansible-playbook -i hosts play-wait\_for.yml #执行者要有sudo权限



**④wait\_for\_connection模块**

# vi wait\_for\_connection.yml

---

- name: pxx

hosts: test66

tasks:

- name: restart managed host

shell: init 6

async: 1

poll: 0

ignore\_errors: true

- name: wait for managed host to come back

wait\_for\_connection:

delay: 2 #延迟检测，默认0秒

connect\_timeout: 5 #连接超时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 wait\_for\_connection

debug:

msg: "after wait\_for\_connection"

# ansible-playbook wait\_for\_connection.yml

**★ansible串行处理playbook**

默认情况，playbook在执行一个任务时，是同时下发给多台主机执行的，整个任务完成后，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

# vi play-serial.yml

---

- 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

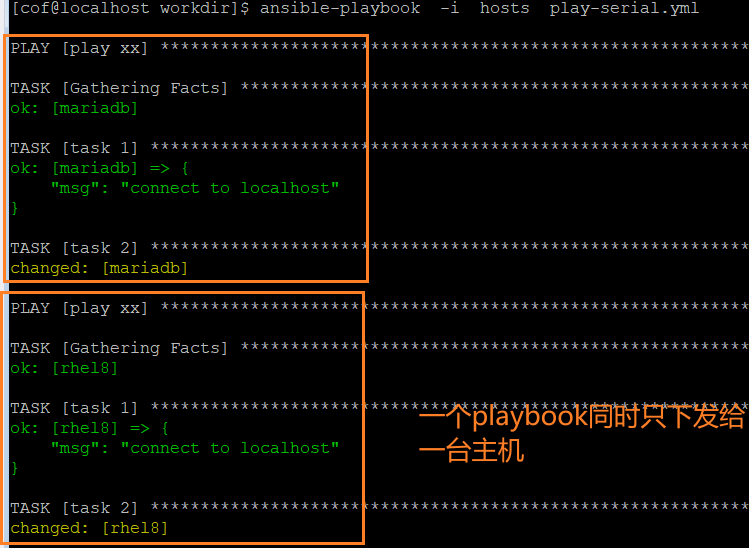
debug:

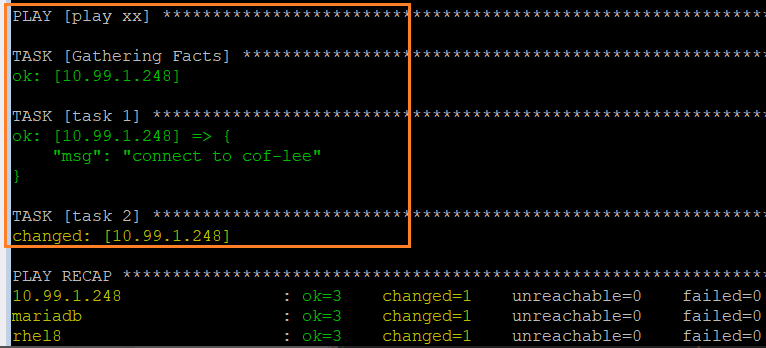
msg: connect to {{ inventory\_hostname }}

- name: task 2

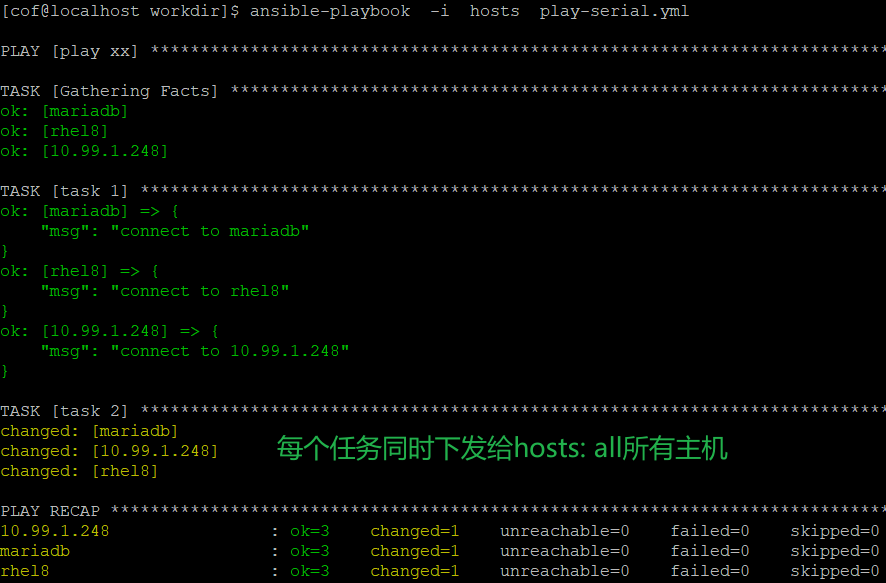
shell: sleep 5

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

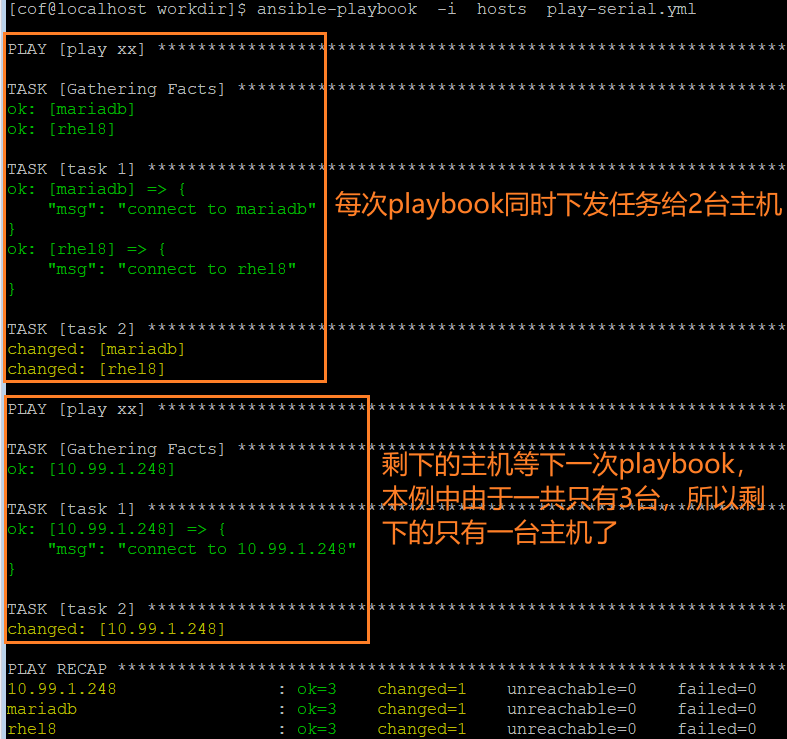




**注释掉 serial: 1 后，效果如下：**



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，效果如下：



forks并行是基于任务的，serial串行是基于playbook的

**★第8章、ansible-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

role目录结构如下：

-- role顶级目录 #可设置为项目名称

|-- 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文件定义处理程序

|-- 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

|-- tasks #目录中的\*.yml文件定义角色的任务，入口文件为main.yml

|-- 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

# mkdir -p role-xx/{files,handlers,tasks,templates,vars} #创建role项目目录

# tree role-xx/

role-xx/

├── files

├── handlers

├── tasks

├── templates

└── vars

# cat > role-xx/vars/main.yml <<EOF

users:

- cof

- lee

EOF

# cat > role-xx/templates/tmplaxx.j2 <<EOF

{% for user in users %}

{{ user }}

{% endfor %}

EOF

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

- name: task yyy

debug:

msg: debug yyy

EOF

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

- name: task xx

template:

src: tmplaxx.j2 #不需要指定路径，会自动去role-xx/templates/下面找

dest: /tmp/xxxxx.conf

notify: restart\_sshd

EOF

# cat > role-xx/handlers/main.yml <<EOF

---

- name: restart\_sshd #须同任务的notify名称，其他handlers文件须在main.yml里引入

service:

name: sshd

state: restarted

EOF

# tasks入口文件，指定任务顺序（文件名称必须为main.yml）

# cat > role-xx/tasks/main.yml <<EOF

- include: task-xxx.yml

- include: task-yyy.yml

EOF

# role入口文件（文件名称自定义，与role目录同级）

# cat > role-main.yml <<EOF

---

- name: play xxx

hosts: mariadb

roles:

- role: role-xx #项目目录名称，也可直接写为 - role-xx

EOF

**★role路径说明**

在role-main.yml的playbook文件里只写了role名称，ansible是从哪里找此角色目录的呢？如果ansible.cfg配置文件里未指明roles\_path=路径，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**

# cat > role-main.yml <<EOF

---

- name: play xxx

hosts: all

pre\_tasks:

- name: xx

debug:

msg: "在roles之前运行的任务"

roles:

- role: role-xx #项目目录名称

tasks: #普通任务默认在roles之后执行

- name: yy

debug:

msg: "在roles之后运行的任务"

post\_tasks:

- name: zz

debug:

msg: "在所有任务之后运行的任务"

EOF

# ansible-playbook -i hosts role-main.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

---

- 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**

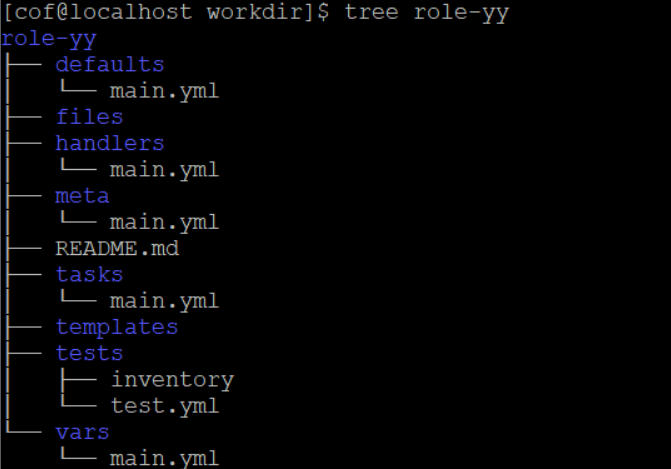
Ansible-Galaxy是一个用于分享ansible role角色的平台，可使用其他人创建好的role，也可发布自己的role，类似helm仓库，ansible-galaxy官网： https://galaxy.ansible.com

# ansible-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（或者说没有加入collection）

#2.8用 ansible-galaxy --help

# ansible-galaxy init role-yy #创建一个名为role-yy的项目（结构目录，子文件）

- Role role-yy was created successfully



# ansible-galaxy search --galaxy-tags=database --platforms=EL #在官网搜索role

Found 635 roles matching your search:

Name Description

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

1it.riak Installs and configures Riak KV and TS, a distributed, highly available NoSQL and TimeSeries database.

5003.mariadb SQL Server relational database engine by using a MariaDB

5u623l20.pgsql PostgreSQL server for \*nix alike OS.

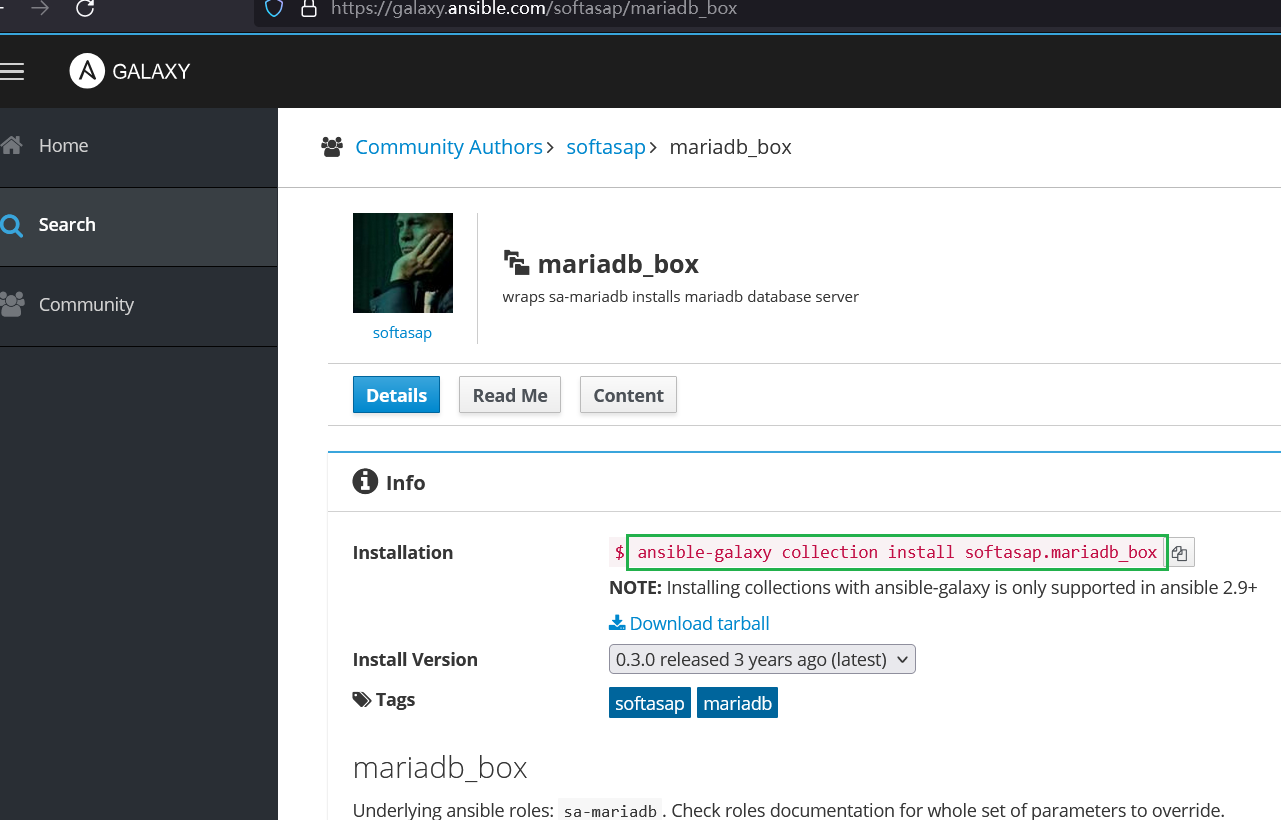
abelboldu.zookeeper Zookeeper cluster role for MidoNet deployments.

abessifi.sqlplus Install and configure SQLPLUS tool

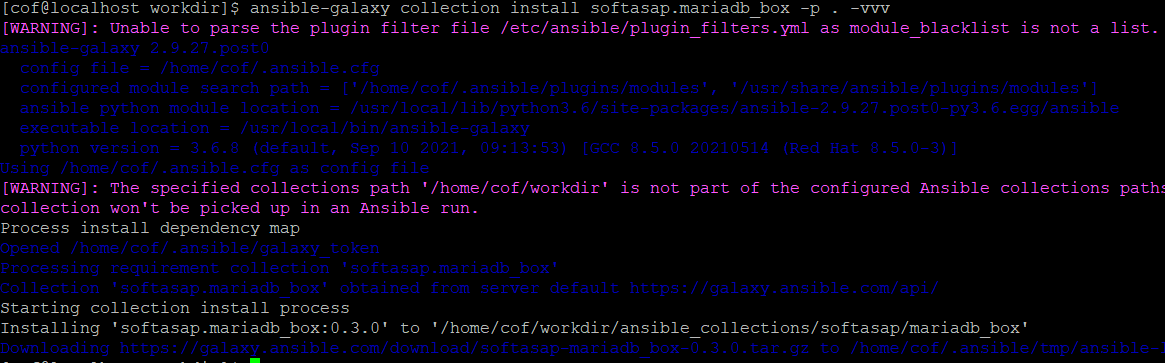
# ansible-galaxy install 5003.mariadb -p ./ -vvv #下载项目，-p指定保存路径



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，会写有下载命令的



# ansible-galaxy collection install softasap.mariadb\_box -p ./ -vvv #2.9版本命令

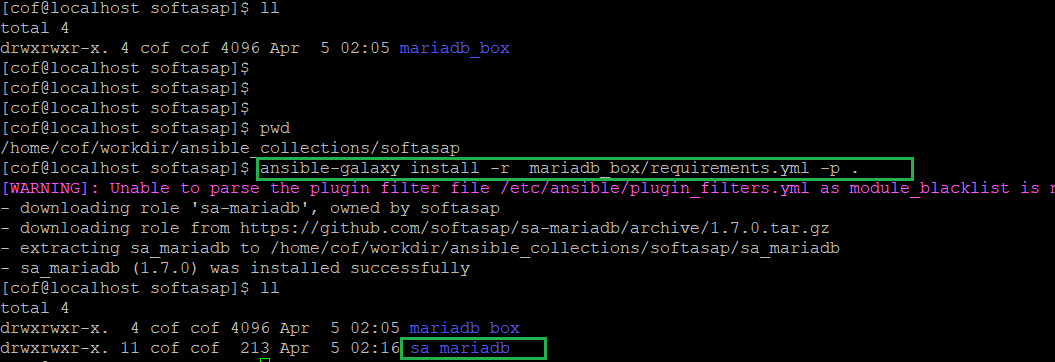


若不指定-p下载路径，则默认保存到：

~/.ansible/collections/ansible\_collections/softasap/mariadb\_box

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 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

**★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9章、ansible-collection集合**

role 角色 是把实现一个任务的剧本及各个参数及配置模板分类存放到不同子目录下的一个组织结构

collection 集合 是将不同类型的模块分到不同的组（集合）里（有的集合也带有role）

ansible 2.8及之前版本的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

从ansible 2.10及之后版本开始，模块被分组到不同的collection里了

**★collection集合的应用**

**★ansible-galaxy的collection仓库**

**★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

include\_vars: myvars.yml

**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

template:

src: template-xx.j2

dest: /etc/xxxx.config

**⑤task引入其他task文件**

tasks:

- name: include taskxx

include: taskxx.yml

**⑥playbook引入其他playbook**

- name: play xx

hosts: all

tasks:

- name: play 1

import\_playbook: play-1.yml #引入其他playbook文件

**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

include\_role:

name: myrolexx

**-------------------------------------**

**第10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发放进行安装，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://github.com/ansible/awx-operator/

本例中安装23.3.0的awx版本，awx-operator版本为2.7.0

**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-1020.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

离线安装awx-operator：

# helm pull awx-operator/awx-operator #下载chart包到本地

也可直接wget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-2.7.0.tgz -n awx --create-namespace #离线安装

#awx-operator要用到以下2个镜像：

gcr.io/kubebuilder/kube-rbac-proxy:v0.14.1

quay.io/ansible/awx-operator:2.7.0

# 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:v0.14.1

image: quay.io/ansible/awx-operator:2.7.0

以上2行改为：

image: cof-lee.com/awx/kube-rbac-proxy:v0.14.1

image: cof-lee.com/awx/awx-operator:2.7.0

# 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,pvc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: postgres-pv

namespace: awx

spec:

capacity:

storage: 10Gi

volumeMode: Filesystem

accessModes:

- ReadWriteOnce

persistentVolumeReclaimPolicy: Delete

storageClassName: local-storage

local:

path: /ansible-awx-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-storage #所有工作节点都要创建

# kubectl create -f pv-ansible-awx.yaml

# cat > pvc-ansible-awx.yaml <<EOF

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: postgres-13-ansible-awx-postgres-13-0

namespace: awx

spec:

storageClassName: local-storage

accessModes:

- ReadWriteOnce

resources:

requests:

storage: 10Gi

EOF

# kubectl create -f pvc-ansible-awx.yaml

# kubectl get pv -n awx

# kubectl get pvc -n awx

**★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

EOF

# kubectl create -f ansible-awx.yaml #需要等待一段时间

**#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:23.3.0

**★创建web-svc以供主机外部访问**

# kubectl expose deployment ansible-awx-web --name ansible-awx-web-svc --type NodePort -n awx

# kubectl get svc ansible-awx-web-svc -n awx

ansible-awx-web-svc NodePort 10.7.15.252 <none> 8052:31164/TCP 22s #后面那个3万多的端口号

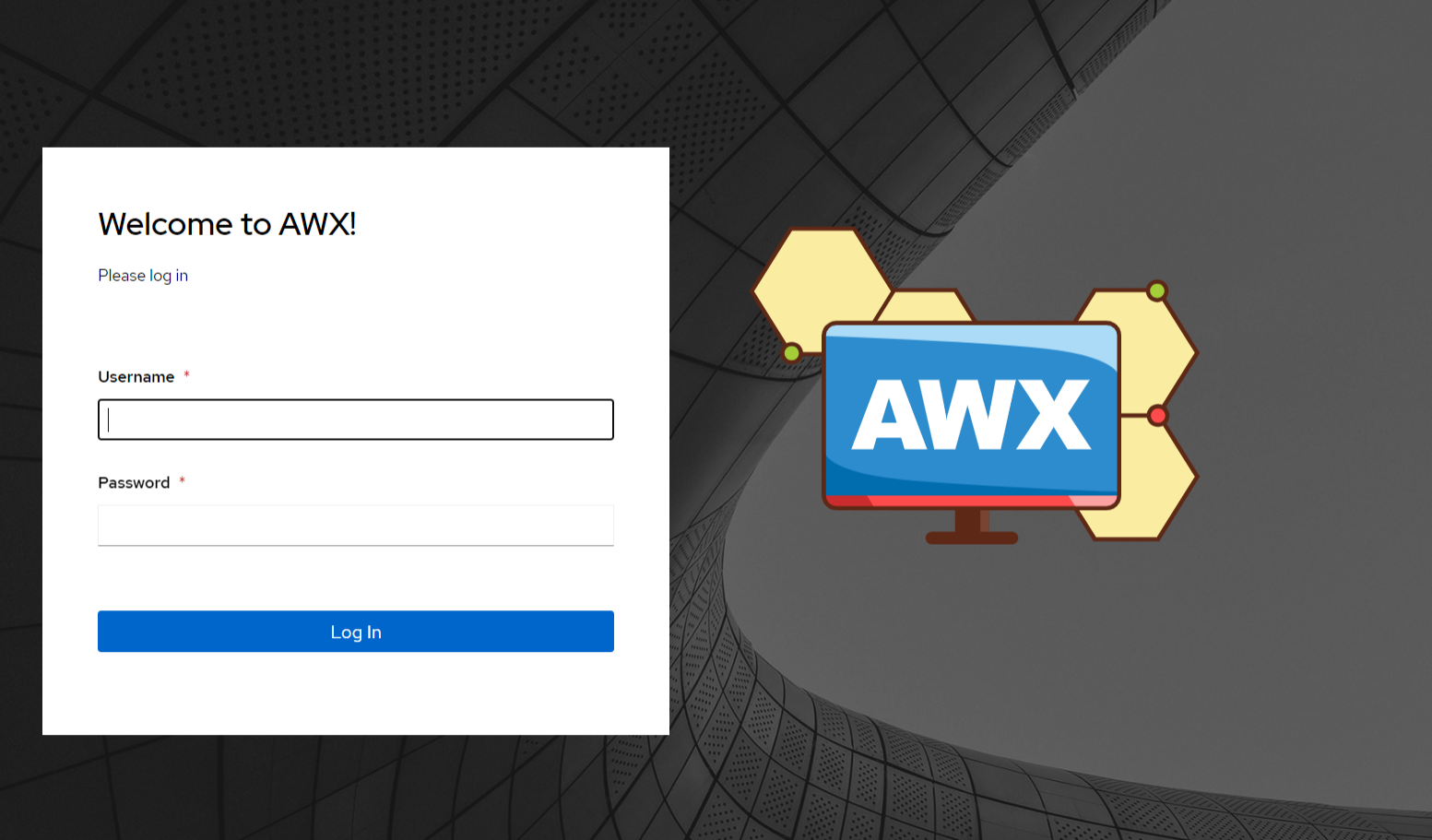
#查看admin密码

# kubectl get secrets -n awx ansible-awx-admin-password -ojsonpath="{.data.password}" | base64 -d

UceLnSHkYylvFOcRo7EdsuhDTED47DTq

★使用浏览器登录 http://x.x.x.x:31164/ 账号为admin，密码为上面查看到的

#如果浏览器访问不了，可能要等很久，大概30分钟，原因未知（可能是deployment给pod分配的cpu/内存资源太少，有待研究）



**★ansible-awx平台使用**