第4天 自动化运维利器 Ansible-最佳实战

一、调试

在执行 ad-hoc 或者 playbook 的时候，在后面加上 `-vvv` 参数，就可以看到 Ansible 的详细执行过程，便于排错。

```
[root@qfedu.com ~]# ansible dbservers -i hosts -m ping -vvv

[root@qfedu.com ~]# ansible-playbook -i hosts checkhost.yml -vvv
```


限制授影响的主机

`—limit` 后面跟主机名或者主机组名

```
[root@qfedu ~]# ansible-playbook -i hosts checkhosts.yml --limit dbservers -vvv
```

二、优化 Ansible 执行速度

1. 设置 SSH 为长连接

openssh5.6 版本后支持 Multiplexing

1.1 检查控制机器的 ssh 版本

```
[root@qfedu.com ~]# ssh -V
OpenSSH_7.4p1, OpenSSL 1.0.2k-fips  26 Jan 2017
```

1.2 升级 ssh 客户端程序

假如不是 5.6 版本以上的，可以用下面的办法升级 ssh 客户端程序

- 配置 Centos6 系统的 YUM 源（使用 Centos6）

```
➜  ansible cat /etc/yum.repos.d/openssh.repo
[CentALT]
name=CentALT Packages for Enterprise Linux 6 - $basearch
baseurl=http://mirror.neu.edu.cn/CentALT/6/$basearch/
enable=1
gpgcheck=0
```

- 执行升级命令

```
[root@qfedu.com ~]# yum   update  openssh-clients
```

升级完成后，不必重启任何服务，因为我们的控制机是使用 ssh 的客户端

1.3 设置 ansible 配置文件

```
[root@qfedu.com ~]#  grep sh_args /etc/ansible/ansible.cfg
ssh_args = -C -o ControlMaster=auto -o ControlPersist=10d
# ControlPersist=10d 表示保持长连接 10 天。
# 60s 是 60 秒
```

1.4 建立长连接并测试

设置好后，重新连接一次被控主机，即可让控制主机和被控主机之间建立长连接

```
[root@qfedu.com ~]# ansible webservers -i hosts -m ping
172.18.0.4 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "ping": "pong"
}
```

验证长连接

```
[root@qfedu.com ~]# ss -nta |grep ESTAB
tcp    ESTAB      0      0      172.18.0.2:51864
         172.18.0.4:2222
```

输出中 有 `ESTAB` 状态的就代表是长连接

同时会在主控机当前用户的家目录下的 .ansibl/cp/ 目录下生成对应的 socket 文件

```
[root@qfedu.com ~]# ls -l .ansible/cp/13fe34a1c4
srw------- 1 root root 0 Apr 17 03:36
.ansible/cp/13fe34a1c4
```

2. 开启 pipelining

我们知道默认情况下 Ansible 执行过程中会把生成好的本地 python 脚本文件 PUT 到 远端机器。如果我们开启了 ssh 的 pipelining 特性，这个过程就会在 SSH 的会话中进行。

在不通过实际文件传输的情况下执行ansible模块来使用管道特性,可以减少执行远程服务器上的模块所需的网络操作数量。比如 PUT sftp 等操作都需要建立网络连接。

下面是关闭 Pipeline 的情况下的三步操作。

```
<172.18.0.3> PUT /root/.ansible/tmp/ansible-local-
10883q1xq1u/tmpNbePyo TO /root/.ansible/tmp/ansible-
tmp-1587214813.33-212837305246708/AnsiballZ_ping.py
<172.18.0.3> SSH: EXEC sftp -b - -C -o
ControlMaster=auto -o ControlPersist=600s -o
KbdInteractiveAuthentication=no -o
PreferredAuthentications=gssapi-with-mic,gssapi-
keyex,hostbased,publickey -o
PasswordAuthentication=no -o ConnectTimeout=10 -o
ControlPath=/root/.ansible/cp/553ad38749
'[172.18.0.3]'
<172.18.0.3> (0, 'sftp> put
/root/.ansible/tmp/ansible-local-
10883q1xq1u/tmpNbePyo /root/.ansible/tmp/ansible-tmp-
1587214813.33-212837305246708/AnsiballZ_ping.py\n',
'')

```

如果开启这个设置,将显著提高性能.。

然而当使用”sudo:”操作的时候, 你必须在所有管理的主机的 `/etc/sudoers` 中禁用 `requiretty`.

下面的步骤是实现这个特性的步骤

1) 在 ansible.cfg 配置文件中设置 pipelining 为 True

```
[root@qfedu.com ~]# grep pipelining /etc/ansible/ansible.cfg
# Enabling pipelining reduces the number of SSH operations required to
pipelining = True
```

2) 配置被控主机的 `/etc/sudoers` 文件，添加下面的内容(默认没有)

> 关于 Sudo 参考：Sudo（简体中文）

```
# Disable "ssh hostname sudo <cmd>", because it will show the password in clear text.
# You have to run "ssh -t hostname sudo <cmd>".
#
# Defaults    requiretty
```


三、设置 facts 缓存

默认情况下，Ansible 每次执行 playbook 时的第一个 Task 就是获取每台主机的 facts 信息。假如不需要可以设置 `gather_facts = no` 进行关闭，以提高执行 playbook 的效率。

假如想获取 facts 信息，同时又想加速这个 task 的效率，就需要设置 facts 缓存。

缓存 facts 信息可以存档 JSON 文件中，也可以方式 redis 和 memcached 中。

1. 首先是可以在 ansible.cfg 文件中设置

```
grep gathering /etc/ansible/ansible.cfg
gathering = smart
```

ansible的配置文件中可以修改'gathering'的值为 `smart`、`implicit` 或者 `explicit`。

- `smart` --> 表示默认收集facts，但facts已有的情况下不会收集，也就是会使用缓存facts；
- `implicit` --> 表示默认收集facts，要禁止收集，必须使用gather_facts: False；
- `explicit` --> 则表示默认不收集，要显式收集，必须使用gather_facts: Ture

2. 在playbook 中设置


```
---
- hosts: all
  gather_facts: yes     # 显式定义收集
  gather_facts: no      # 显式定义不收集
```

配置缓存的目标

1. 缓存到文件（JSON格式的数据） 在 ansible.cfg 文件中配置缓存到一个普通文件中

   同时还可指定搜集哪些信息，比如只搜集 network，virtual

```
gathering =  smart
gather_subset = network,virtual  # 只搜集 网络信息和虚拟化信息
fact_caching = jsonfile        # 缓存到 json 文件
fact_caching_connection = /dev/shm/ansible_fact_cache
fact_caching_timeout = 86400  # 缓存数据时间是一天

```

`fact_caching_connection` 是一个放置在可写目录(如果目录不存在,ansible会试图创建它)中的本地文件路径,文件名是在 Inventory 中保存的 IP 或者 hostname .

验证

```
[root@qfedu.com ~]# ls /dev/shm/ansible_fact_cache/
172.18.0.3  172.18.0.5
[root@qfedu.com ~]# head -n 3 /dev/shm/ansible_fact_cache/*
==> /dev/shm/ansible_fact_cache/172.18.0.3 <==
{
    "_ansible_facts_gathered": true,
    "ansible_all_ipv4_addresses": [

==> /dev/shm/ansible_fact_cache/172.18.0.5 <==
{
    "_ansible_facts_gathered": true,
    "ansible_all_ipv4_addresses": [
[root@qfedu.com ~]#
```


2. 缓存到 redis

- 在任一机器或者ansible 控制主机上部署 Redis 服务

```
[root@qfedu.com ~]# yum install redis
```

- 假如 Redis 服务不在 ansible 控制主机上，还应该设置 redis 监听地址

```
~ # grep '^bind' /etc/redis.conf
bind 0.0.0.0
```

- 在控制主机 python 的 redis 库

```
[root@qfedu.com ~]# pip install redis
```

- 在 ansible.cfg 文件中配置缓存到 redis

```
gathering = smart
fact_caching = redis        # 缓存到 redis
fact_caching_connection = 192.168.1.37:6379:0

fact_caching_timeout = 86400  # 缓存数据时间是一天

```

- 检查

```
127.0.0.1:6379> keys *
1) "ansible_facts172.18.0.3"
2) "ansible_facts172.18.0.4"
3) "ansible_cache_keys"
127.0.0.1:6379> get ansible_facts172.18.0.3
```

四、设置 Ansible 的执行策略

1. 策略介绍

默认的执行策略是按批并行处理的，假如总共 15 台主机，每次并发 5 个线程执行的策略如下：

```
h1/h2/h3/h4h5 -------> h6/h7/h8/h9/h10  -----> h11/h12/h13/h14/h15
                全部执行完后，进入下一批                依次类推
```

从 asible2.0 开始，可以通过在 playbook 中设置 strategy 的值改变这策略，也可以在 `ansible.cfg` 配置文件中设置一个默认的策略:

```
[defaults]
strategy = free
```

改变后的策略，可以前赴后继的对主机进行执行 task，执行模式如下:

假如 h4 主机先执行完，会及时的让 下一个排队的主机进入到 执行的队列中。

```
h1/h2/h3/h4/h5  ------> h1/h2/h3/h6/h5 -------> h1/h2/h3/h6/h7  -----> ...
```


2 环境准备

准备多台机器

可以使用如下方式给一台主机添加多个 IP 达到拥有多个主机的效果。

```
[root@web-server ~]# ip addr add 172.18.0.5/16 dev eth0
[root@web-server ~]# ip addr add 172.18.0.6/16 dev eth0
[root@web-server ~]# ip addr add 172.18.0.7/16 dev eth0
[root@web-server ~]# ip addr add 172.18.0.8/16 dev eth0
[root@web-server ~]# ip addr add 172.18.0.9/16 dev eth0
[root@web-server ~]# ip addr add 172.18.0.10/16 dev eth0
```

添加到资产中

```
[dbservers]
172.18.0.3

[webservers]
172.18.0.4
172.18.0.5
172.18.0.6
172.18.0.7
172.18.0.8
172.18.0.9
172.18.0.10

[allservers:children]
dbservers
webservers

[allservers:vars]
user=tomcat
```

`strategy` 默认的值的是 `linear` ，就是按批并行处理，下面是配置为 `free` 的方式实例：

```
- hosts: webservers
  strategy: free
  tasks:
    - name: ping hosts
      ping:
```

执行

> 默认 Ansible 的执行队列有一个，就是并行执行，假如控制节点的机器有多个 CPU，并且性能较好，可以打开多个执行队列，就是并发。

- 方式一: 在 ansible.cfg 中设置

```
[defaults]
forks = 30
```

- 方式二： 在命令行里使用

```
ansible-playbook -f 3  my_playbook.yml
```

- 实例演示

```
[root@qfedu.com ~]# ansible-playbook -i hosts checkhost.yml -f 3

PLAY [webservers] **********

```

```
TASK [Gathering Facts] ***********************************************************
ok: [172.18.0.5]
ok: [172.18.0.4]
ok: [172.18.0.6]
ok: [172.18.0.7]
ok: [172.18.0.8]
ok: [172.18.0.9]

TASK [ping hosts] ****************************************************************
ok: [172.18.0.4]
ok: [172.18.0.5]
ok: [172.18.0.6]
ok: [172.18.0.7]
ok: [172.18.0.8]
ok: [172.18.0.9]

TASK [Gathering Facts] ***********************************************************
ok: [172.18.0.10]

TASK [ping hosts] ****************************************************************
ok: [172.18.0.10]

PLAY RECAP ***********************************************************************
```

```
172.18.0.10                : ok=2    changed=0
 unreachable=0    failed=0    skipped=0    rescued=0
   ignored=0
172.18.0.4                 : ok=2    changed=0
 unreachable=0    failed=0    skipped=0    rescued=0
   ignored=0
172.18.0.5                 : ok=2    changed=0
 unreachable=0    failed=0    skipped=0    rescued=0
   ignored=0
172.18.0.6                 : ok=2    changed=0
 unreachable=0    failed=0    skipped=0    rescued=0
   ignored=0
172.18.0.7                 : ok=2    changed=0
 unreachable=0    failed=0    skipped=0    rescued=0
   ignored=0
172.18.0.8                 : ok=2    changed=0
 unreachable=0    failed=0    skipped=0    rescued=0
   ignored=0
172.18.0.9                 : ok=2    changed=0
 unreachable=0    failed=0    skipped=0    rescued=0
   ignored=0

```


五、 异步和轮询

默认情况下，执行 ansible 的时候，这个命令会一直处于阻塞状态，直到在每个节点上完成任务为止。这可能并不总是合乎需要的，或者您运行的操作所花费的时间超过了SSH超时。

可以在后台运行长时间运行的操作，以后再查看它们的状态。

1 执行临时命令是使用异步

比如如下示例是 执行一个任务持续运行 5 秒钟，超时 10 秒(-B 10)，并且不等待任务返回结果(-P 0)

```
[root@qfedu.com ~]# ansible dbservers -B 10 -P 0 -i hosts -a "sleep 5"
172.18.0.3 | CHANGED => {
    "ansible_job_id": "191465439990.2210",
    "changed": true,
    "finished": 0,
    "results_file": "/root/.ansible_async/191465439990.2210",
    "started": 1
}
```

- 查看保存结果集的文件

执行结果会存放的**被控节点**主机的 `/root/.ansible_async/191465439990.2210` 文件中

```
[root@db-server /]# cat /root/.ansible_async/191465439990.2210
{"changed": true, "end": "2020-04-19 02:06:23.716226", "stdout": "", "cmd": ["sleep", "5"], "start": "2020-04-19 02:06:18.310407", "delta": "0:00:05.405819", "stderr": "", "rc": 0, "invocation": {"module_args": {"warn": true, "executable": null, "_uses_shell": false, "strip_empty_ends": true, "_raw_params": "sleep 5", "removes": null, "argv": null, "creates": null, "chdir": null, "stdin_add_newline": true, "stdin": null}}}[root@db-server /]#
```

- 获取结果

应该使用 `async_status` 模块来获取结果，需要传递 job id，就是返回信息中字段 `ansible_job_id` 的值，这里是 `191465439990.2210`。

```
[root@qfedu.com ~]# ansible dbservers -i hosts -m async_status -a "jid=191465439990.2210"
172.18.0.3 | CHANGED => {
    "ansible_job_id": "191465439990.2210",
    "changed": true,
    "cmd": [
        "sleep",
        "5"
    ],
    "delta": "0:00:05.405819",
    "end": "2020-04-19 02:06:23.716226",
    "finished": 1,
    "rc": 0,
    "start": "2020-04-19 02:06:18.310407",
    "stderr": "",
    "stderr_lines": [],
    "stdout": "",
    "stdout_lines": []
}
```

> 假如 -P 的值大于 0 就会起到同步执行的效果，整个Ansible 命令还是阻塞状态的。并且此时 -B 等待结果集的超时时间必须大于，命令实际执行消耗的时间。否则报错。 如下所示：

```
[root@qfedu.com ~]# ansible dbservers -B 3 -P 1 -i hosts -a "sleep 5"
172.18.0.3 | FAILED | rc=-1 >>
async task did not complete within the requested time - 3s
[WARNING]: Failure using method (v2_runner_on_failed) in callback plugin
(<ansible.plugins.callback.mail.CallbackModule object at 0x7f9796792c90>):
'CallbackModule' object has no attribute 'itembody'
```

2 Playbook 中使用异步

这里讲介绍如何异步执行 playbook。

下面演示一个异步任务，这个异步任务执行时长 5 秒左右，等待超时时间是 10 秒钟， 之后需把返回结果注册到变量 `job` 中，这样才能获取到每个备课主机的 job id。最后使用 `debug` 模块打印出来。

async.yml


```
---
- hosts: dbservers
  remote_user: root

  tasks:
    - name: simulate long running op (5 sec), wait for up to 6 sec, poll every 0 sec
      shell: /bin/sleep 5;hostname -i
      async: 6
      poll: 0
      register: job
    - name: show  job id
      debug:
```

```
13          msg: "Job id is {{ job }}"
14
```

执行playbook

```
1  [root@qfedu.com ~]# ansible-playbook -i hosts
   async.yaml
2
3  PLAY [dbservers]
   ****************************************************
   ***********
4
5  TASK [simulate long running op (5 sec), wait for up
   to 6 sec, poll every 0 sec] ***
6  changed: [172.18.0.3]
7
8  TASK [show  job id]
   ****************************************************
   ********
9  ok: [172.18.0.3] => {
10     "msg": "Job id is {u'ansible_job_id':
   u'801565582767.3551', u'started': 1, 'changed':
   True, 'failed': False, u'finished': 0,
   u'results_file':
   u'/root/.ansible_async/801565582767.3551'}"
11 }
12
13 PLAY RECAP
   ****************************************************
   *****************
14 172.18.0.3                 : ok=2    changed=1
    unreachable=0    failed=0    skipped=0    rescued=0
      ignored=0
```

- 获取结果

可以那其中的一个，查看任务结果

getJobResult.yml

```
- hosts: dbservers
  tasks:
  - name: Get job result
    async_status:
      jid: "801565582767.3551"
    register: job_result

  - name: debug job result
    debug:
      var: job_result
```

执行 playbook

```
[root@qfedu.com ~]# ansible-playbook -i hosts getJobResult.yml

PLAY [dbservers] ***********************************************************

TASK [Get job result] ******************************************************
changed: [172.18.0.3]

TASK [debug job result] ****************************************************
ok: [172.18.0.3] => {
    "job_result": {
        "ansible_job_id": "801565582767.3551",
```

```
        "changed": true,
        "cmd": "/bin/sleep 5;hostname -i",
        "delta": "0:00:05.421222",
        "end": "2020-04-19 03:21:47.090911",
        "failed": false,
        "finished": 1,
        "rc": 0,
        "start": "2020-04-19 03:21:41.669689",
        "stderr": "",
        "stderr_lines": [],
        "stdout": "172.18.0.3",
        "stdout_lines": [
            "172.18.0.3"
        ]
    }
}

PLAY RECAP *********************************************************************
172.18.0.3                 : ok=2    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

```


3 注意事项

不应通过将轮询值指定为0来进行需要排他锁的操作（例如yum事务）来尝试异步运行任务。

安装多个包 YUM 模块本身就支持

yum_tasks.yaml

```yaml
- name: install tree vim
  yum:
    name: [tree, vim]
    state: present
```

> 命令行中使用英文逗号隔开: `-m yum -a "name=tree,vim state=present"`

六、使用多个 Inventory 文件

通过从命令行提供多个清单参数或通过配置多个清单参数，可以同时定位多个清单源（目录，动态清单脚本或清单插件支持的文件）

这对于具有多环境的状态下非常有帮助，比如生产环境和开发环境。

1 从命令行定位两个源，如下所示：

```
ansible-playbook get_logs.yml -i development -i production
```

2 使用目录汇总清单源

可以通过组合目录下的多个清单来源和来源类型来创建清单。 这对于组合静态和动态主机并将它们作为一个清单进行管理很有用。

目录中仅支持如下扩展名

```
.yaml   .yml   .json
```

以下清单结合了清单插件源，动态清单脚本和具有静态主机的文件

```
inventory/
  aliyun.yml           # 清单插件，获取阿里云的主机
  dynamic-inventory.py    # 使用动态脚本添加额外的主机
  static-inventory        # 添加静态主机和组
  group_vars/
    all.yml               # 给所有的主机指定变量
```

> 以上的组合可以去掉自己环境中不需要的

- 命令行里使用这个清单目录

```
ansible-playbook example.yml -i inventory

```

- 可以在配置文件中配置

假设这个清单目录的绝对路径是：/etc/ansible/inventory

应该这样配置：

```
[defaults]
inventory      = /etc/ansible/inventory
```

- 要注意变量覆盖

如果存在与其他库存来源之间的变量冲突或组依赖关系，则控制库存来源的合并顺序可能很有用。

根据文件名按字母顺序合并清单，因此可以通过在文件前添加前缀来控制结果：

```
inventory/
  01-aliyun.yml           # 清单插件，获取阿里云的主机
  02-dynamic-inventory.py    # 使用动态脚本添加额外的主机
  03-static-inventory        # 添加静态主机和组
  group_vars/
    all.yml                  # 给所有的主机指定变量
```

重复定义变量导致变量被覆盖，是应该避免的，也可以避免的。

- 测试：

目录结构

```
[root@qfedu.com ~]# tree inventory
inventory
|-- 01-static.yml
`-- 02-static.yml

0 directories, 2 files
```

*文件内容l**

```
# inventory/01-static.yml
[webservers]
172.18.0.[4:10]

[allservers:children]
webservers

[allservers:vars]
name = xiguatian
```

```
# inventory/02-static.yml
[dbservers]
172.18.0.3

[allservers:children]
dbservers

[allservers:vars]
name = shark
```

验证变量的值

```
[root@qfedu.com ~]# ansible all -i inventory -m debug -a "var=name"
172.18.0.4 | SUCCESS => {
    "name": "shark"
}
172.18.0.5 | SUCCESS => {
    "name": "shark"
}
172.18.0.6 | SUCCESS => {
    "name": "shark"
}
172.18.0.7 | SUCCESS => {
    "name": "shark"
}
172.18.0.8 | SUCCESS => {
    "name": "shark"
}
172.18.0.9 | SUCCESS => {
    "name": "shark"
}
172.18.0.10 | SUCCESS => {
    "name": "shark"
```

```
}
172.18.0.3 | SUCCESS => {
    "name": "shark"
}
```

七、使用 Inventory scripts

清单脚本不限制语言

脚本限制条件:

- 脚本必须接受`--list`和`--host <hostname>`参数
- 当使用单个`--list`参数调用脚本时，脚本必须输出到标准输出，就是输出到终端，其中包含要管理的所有组的JSON编码的哈希或字典。

每个组的值应该是包含每个主机列表，任何子组和潜在组变量的哈希或字典，或者仅是主机列表：

```
{
    "group001": {
        "hosts": ["host001", "host002"],
        "vars": {
            "var1": true
        },
        "children": ["group002"]
    },
    "group002": {
        "hosts": ["host003","host004"],
        "vars": {
            "var2": 500
        },
        "children":[]
    }
```

```
16
17 }
```

> 如果组中的任何元素为空，则可以从输出中将其省略。

- 当使用 `host <hostname>` 参数（其中是上面的主机）进行调用时，脚本必须打印一个空的JSON哈希/字典或含有这个主机变量的哈希/字典，以使其可用于模板和剧本。例如：

```
{
    "VAR001": "VALUE",
    "VAR002": "VALUE",
}
```

> 打印变量是可选的。如果脚本不执行此操作，则应打印一个空的哈希或字典。

- 一个简单的示例

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import sys
import json
import argparse

def lists():
    """
    indent 定义输出时的格式缩进的空格数
    """
    dic = {}
    host_list = [ '192.168.2.{}'.format(str(i) ) for i in range(20,23) ]
    hosts_dict = {'hosts': host_list}
```

```
    dic['computes'] = hosts_dict  # 静态文件中的组，在这里定义了主机信息


    return json.dumps(dic,indent=4)

def hosts(name):
    dic = {'ansibl_ssh_pass': '12345'}

    return json.dumps(dic)

if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('-l', '--list', help='host list', action='store_true')
    parser.add_argument('-H', '--host', help='hosts vars')
    args = vars(parser.parse_args())

    if args['list']:
        print( lists() )
    elif args['host']:
        print( hosts(args['host']) )
    else:
        parser.print_help()
```

- 改变文件权限为可执行

```
[ansible@ansible ~]$ sudo chmod 655 /etc/ansible/hosts.py
```

八、项目录结构

使用官方建议的目录结构来组织很多 role 和 playbook 文件是个很棒的建议。

假如你用 role 封装了 playbook，并且任务依赖文件或者依赖其他的任务时，建议使用目录结构管理。

假如是一个简单的独立任务， 只使用 playbook 文件即可，这样会方便我们在其他地方进行引用。

下面是官网最佳实战中推荐的目录结构

```
production                # 关于生产环境服务器的资产清单文件
develop                   # 关于开发环境的清单文件

group_vars/
   group1                 # 组 group1 的变量文件
   group2                 # 组 group2 的变量文件
host_vars/
   hostname1              # hostname1 定义的变量文件
   hostname2              # hostname2 定义的变量文件

library/                  # 如果有自定义的模块,放在这里(可选)
filter_plugins/           # 如果有自定义的过滤插件,放在这里(可选)

site.yml                  # 执行 playbook 的统一入口文件
webservers.yml            # 特殊任务的 playbook
dbservers.yml             # 还是特殊任务的 playbook

roles/                    # role 存放目录
    common/               # common 角色的目录
        tasks/
```

```
            main.yml
        handlers/
            main.yml
        templates/
            ntp.conf.j2
        files/
            bar.txt
            foo.sh
        vars/
            main.yml       # common 角色定义的变量文件
        defaults/
            main.yml       # common 角色定义的默认变量文件（优先级低）
        meta/
            main.yml       #  common 角色的依赖关系文件

    webtier/               # 下面这些都是和 common 同级的目录，是另外的一些角色
    monitoring/
    fooapp/
```