ansible-playbook 的使用

1. playbook 简介与文件格式

playbook 字面意思是剧本，现实中由演员按照剧本表演，在 Ansible 中，这次由计算机进行表演，由计算机安装、部署应用，提供对外服务，以及组织计算机处理各种各样的事情。

playbook 文件由 YMAL 语言编写。YMAL 格式类似于 JSON 的文件格式，便于理解、阅读和书写。首先学习了解一下 YMAL 的格式，对后面书写 playbook 很有帮助。以下是
playbook 常用的 YMAL 格式规则。

文件的第一行应该以“---”（三个连字符）开始，表明 YMAL 文件的开始；
在同一行中，# 之后的内容表示注释，类似于 shell 、Python 和 Ruby；
YMAL 中的列表元素以“-”开头，然后紧跟着一个空格，后面为元素内容；
同一个列表中的元素应该保持相同的缩进，否则会被当作错误处理；
play 中的 hosts、variables、roles、tasks 等对象的表示方法都是键值中间以“:”分隔表示，“:”后面还要增加一个空格。
首先看下面这个例子：

复制- apple
- banana
- orange

等价于 JSON 的下面这个格式：

复制[
  “apple”,
  “banana”,
  “orange”
]

playbook 文件是通过 ansible-playbook 命令进行解析的，该命令会根据自上而下的顺序依次执行 playbook 文件中的内容。

2. playbook 的组成

playbook 是由一个或多个“play”组成的列表。play 的主要功能在于，将事先合并为一组的主机组合成事先通过 Ansible 定义好的角色。将多个 play 组织在一个 playbook 中就可以让它们联同起来，按事先编排好的机制完成一系列复杂的任务。

playbooks 主要有以下四部分构成，分别如下：

Target 部分，定义将要执行 playbook 的远程主机组；
Variable 部分，定义 playbook 运行时需要使用的变量；
Task 部分，定义将要在远程主机上执行的任务列表；
Handler 部分，定义 task 执行完成以后需要调用的任务。
下面介绍下构成 playbook 的四个组成部分。

（1）Hosts 和 Users

playbook 中的每一个 play 的目的都是为了让某个或某些远程主机以某个指定的用户身份执行任务。

hosts：用于指定要执行任务的远程主机，每个 playbook 都必须指定 hosts，hosts 也可以使用通配符格式。主机或主机组在 inventory 清单（hosts 文件）中指定，可以使用系统默认的 /etc/ansible/hosts，也可以自己编辑，在运行的时候加上 -i 选项，可指定自定义主机清单的位置。

remote_user：用于指定在远程主机上执行任务的用户，可以指定任意用户，也可以使用 sudo，但是用户必须要有执行相应任务的权限。

（2）任务列表

play 的主体部分是 task list，其中的各任务按次序逐个在 hosts 中指定所有远程主机上的执行，即在所有远程主机上完成第一个任务后再开始第二个。在运行自上而下某 playbook 时，如果中途发生错误，则所有已执行任务都将回滚，因此在更正 playbook 后需要重新执行一次。

task 的目的是使用指定的参数执行模块，而在模块参数中可以使用变量。模块执行一个命令，哪怕执行一次或多次，其结果是一样的，这意味着 playbook 多次执行是安全的，因为其结果均一致。tasks 包含 name 和要执行的模块，name 是可选的，只是为了便于用户阅读，建议加上，模块是必需的，同时也要给予模块相应的参数。

定义 tasks 推荐使用 module: options 格式，例如：

复制service: name=httpd state=running
（3）handlers

用于当关注的资源发生变化时采取一定的操作，handlers 和“notify”配合使用。“notify”这个动作可用于在每个 play 执行的最后被触发，这样可以避免当多次有改变发生时，每次都执行指定的操作，通过“notify”，仅在所有的变化发生完成后一次性地执行指定操作。

在 notify 中列出的操作称为 handler，也就是说 notify 用来调用 handler 中定义的操作。

注意：在 notify 中定义的内容一定要和 handlers 中定义的“- name”内容一样，这样才能达到触发的效果，否则会不生效。

（4）tags

用于让用户选择运行或略过 playbook 中的部分代码。Ansible 具有幂等性，因此会自动跳过没有变化的部分；但是当一个 playbook 任务比较多时，一个个的判断每个部分是否发生了变化，也需要很长时间。因此，如果确定某些部分没有发生变化，就可以通过 tags 跳过这些代码片断。

3. playbook 执行结果解析
使用 ansible-playbook 运行 playbook 文件，输出的内容为 JSON 格式，并且由不同颜色组成，便于识别。一般而言，输出内容中，每个颜色表示的含义如下：

绿色代表执行成功，但系统保持原样；
黄色代表系统状态发生改变，也就是执行的操作生效；
红色代表执行失败，会显示错误信息。
下面是一个简单的 playbook 文件：

复制- name: create user
  hosts: 172.16.213.231
  user: root
  gather_facts: false
  vars:

```
    user1: testuser
  tasks:
   - name: start createuser
     user: name="{{user1}}"
```

上面 playbook 实现的功能是新增一个用户，每个参数含义如下：

name 参数对该 playbook 实现的功能做一个概述，后面执行过程中，会输出 name 的值；
hosts 参数指定了对哪些主机进行操作；
user 参数指定了使用什么用户登录到远程主机进行操作；
gather_facts 参数指定了在执行 task 任务前，是否先执行 setup 模块获取主机相关信息，此参数默认值为 true，表示开启，如果在 task 中使用 facts 信息时，就需要开启此功能；否则设置为 false，这样可以加快 playbook 的执行速度；
vars 参数指定了变量，这里指定一个 user1 变量，其值为 testuser，注意，变量值一定要用引号括起来；
tasks 指定了一个任务，其下面的 name 参数同样是对任务的描述，在执行过程中会打印出来，user 是一个模块，后面的 name 是 user 模块里的一个参数，而增加的用户名调用了上面 user1 变量的值。

4. playbook 中 tasks 语法使用

在 playbook 中，task 部分是整个任务的核心，前面介绍的 ansible 常用模块，如 commands、shell、file、cron、user 等模块，在 playbook 中仍然可用，每个模块所使用的参数以及含义跟命令行模式下也完全一样，只不过写法不同而已。

下面通过几个例子来看看 playbook 中常见功能模块的写法。

（1）playbook 示例

下面是一个 playbook 示例，test.yml 文件内容如下：

```
复制- hosts: hadoophosts
  remote_user: root
  tasks:
   - name: create hadoop user
     user: name=hadoop state=present
   - name: create hadoop directory and chmod/chown
     file: path=/opt/hadoop state=directory mode=0755 owner=hadoop group=hadoop
   - name: synchronize hadoop program
     synchronize: src=/data/hadoop/ dest=/opt/hadoop
```

```
  - name: Setting environment variables
    shell: echo "export JAVA_HOME=/usr/jdk" >> /etc/profile
```

在 playbook 文件中，使用了 user、file、synchronize 和 shell 模块，文件开始定义了一个主机组 hadoophosts，然后设置 root 用户在远程主机上执行操作；接着是 task 任务的开始，“- name”是描述性信息，用来标识任务执行内容和进度，第一个 task 用来创建一个 hadoop 用户，使用了 user 模块。

注意，上面的 user 表示 ansible 的 user 模块，而 user 后面的 name、state 是 user 模块的参数，这些参数的含义上面已经介绍过了。

下面还有 file、synchronize 及 shell 模块，它们的写法跟 user 模块类似，不再过多介绍。

从此文件可以看出，通过 playbook 模式编写的文件更加简洁、易懂，只要设置好了任务的运行策略、顺序，每次需要用到这个操作的话，直接执行就可以了。执行的方式如下：

复制[root@server239 ansible]# ansible-playbook test.yml
除了前面已经介绍过的 ansible 模块，还有一些模块在 playbook 中也经常用到，下面再介绍一些常用的 playbook 模块。
（2）unarchive 模块
该模块用来实现解压缩，也就是将压缩文件解压分发到远程不同节点上，只需记住如下几个参数即可：

src，源文件路径，这个源文件在管理机上；
dest，指定远程主机的文件路径；
mode，设置远程主机上文件权限。
看下面这个例子：

```
复制- hosts: 172.16.213.231
  remote_user: root
  gather_facts: false
  tasks:
   - name: unarchive spark files
     unarchive: src=/src/spark.tar.gz dest=/opt
```

这个操作是将管理机上的 /src/spark.tar.gz 文件传输到远程主机上后进行解压缩，并将解压缩后的文件放到远程主机的 /opt 目录下。注意，这个例子设置了 gather_facts 选项为 false，这是因为下面的 task 中，没有用到 facts 信息。

（3）lineinfile、replace 模块

在自动化运维中，对文件进行内容替换是一个非常常见的场景，比如修改、删除、添加操作系统的某些参数等。Ansible 中虽然提供了 shell 模块结合 sed 命令来达到替换的效果，但经常会遇到需要转义的问题，并且考虑到可读性和可维护性等多方面因素，使用 Ansible 自带的替换模块是一个不错的选择。Ansible 常用的替换模块为 replace 和 lineinfile。

replace 模块可以根据指定的正则表达式替换远程主机下某个文件中的内容，常用的参数如下表所示：

看下面这个例子：

```
复制- hosts: 172.16.213.231
  remote_user: root
  tasks:
   - name: modify selinux
     replace: path=/etc/selinux/config regexp="enforcing" replace=disabled
backup=yes
```

这个操作是对远程主机上 /etc/selinux/config 文件中的 enforcing 字符串进行替换，替换为 disabled，替换前进行备份，其实就是关闭远程主机上 selinux 服务。

再介绍一下 lineinfile，此模块也可以实现 replace 的功能，但 lineinfile 功能更加强大，支持的参数也比较多，常用参数含义如下表所示：

下面来看一个基于 lineinfile 的 playbook 任务：

```
复制- hosts: 172.16.213.231
  remote_user: root
  tasks:
```

```
  - lineinfile: dest=/etc/profile insertafter='ulimit(.*)' line="ulimit -c
unlimited"
  - lineinfile: dest=/etc/profile line="export JAVA_HOME=/usr/jdk"
  - lineinfile: dest=/etc/selinux/config regexp='SELINUX=(.*)'
line='SELINUX=disabled'
  - lineinfile: dest=/etc/resolv.conf regexp='search(.*)' state=absent
```

在 playbook 任务中，调用了四次 lineinfile 替换操作，第一次是在 /etc/profile 文件中找到以 ulimit 开头的行，并在后面添加一行内容“ulimit -c unlimited”；第二次是在 /etc/profile 文件的最后添加一个 JAVA_HOME 路径；第三次是修改 /etc/selinux/config 文件中以“SELINUX=”开头的行，将其替换为“SELINUX=disabled”，其实就是关闭 selinux；最后一个操作是在 /etc/resolv.conf 文件找查找以 search 开头的行，然后将其删除掉。

（4）register、set_fact、debug 模块

Ansible 中定义变量的方式有很多种，可以将模块的执行结果注册为变量，也可以在 roles 中的文件内定义变量，还可以使用内置变量等，而 register、set_fact 都可用来注册一个变量。

使用 register 选项，可以将当前 task 的输出结果赋值给一个变量，看下面这个例子：

```
复制- hosts: 172.16.213.231
  remote_user: root
  tasks:
   - name: ps command
     shell: hostname
     register: host_result
   - debug: var=host_result
```

此例子是将在远程主机上执行的 shell 命令“hostname”的输出结果赋值给变量 host_result，然后再将变量引用并使用 debug 模块输出，输出结果是 json 格式的。注意，此例子最后还使用了 debug 模块，此模块用于在调试中输出信息。

下面是 playbook 的 debug 输出结果：

```
复制TASK [debug] ********************************************************
ok: [172.16.213.231] => {
    "host_result": {
```

```
        "changed": true,
        "cmd": "hostname",
        "delta": "0:00:00.007228",
        "end": "2020-04-01 04:42:34.254587",
        "failed": false,
        "rc": 0,
        "start": "2020-04-01 04:42:34.247359",
        "stderr": "",
        "stderr_lines": [],
        "stdout": "server231.localdomain",
        "stdout_lines": [
            "server231.localdomain"
        ]
    }
}
```


可以看出，此输出是一段 json 格式的数据，最顶端的 key 为 host_result，大括号内还有多个二级 key，我们想要的结果是输出远程主机的主机名即可，不需要其他这些额外的二级 key 信息，如何实现这个需求呢？如果想要输出 json 数据的某二级 key 项，可以使用 "key.dict" 或 "key['dict']" 的方式引用即可。

从上面输出可以看到，我们需要的二级 key 是 stdout 项，所以要仅仅输出此项内容，可以将变量引用改为 host_result.stdout 即可，也就是将上面的 playbook 任务改成如下内容：

复制
```
- hosts: 172.16.213.231
  remote_user: root
  tasks:
   - name: hostname command
     shell: hostname
     register: host_result
   - debug: var=host_result.stdout
   - debug: 'msg="output: {{host_result.stdout}}"'
```

在 playbook 中，又增加了一个 debug 参数，debug 模块常用的参数有两个，即 msg 和 var，它们都可以引用变量输出信息，但有一点小区别，msg 可以输出自定义信息，并且变量需要双大括号包含起来；而 var 参数只能输出变量，并且不需要双大括号。

修改后的 playbook 执行 debug 输出结果如下：

```
复制TASK [debug] ********************************************************
ok: [172.16.213.231] => {
    "host_result.stdout": "server231.localdomain"
}

TASK [debug] ************************************************************
ok: [172.16.213.231] => {
    "msg": "output: server231.localdomain"
}
```

从输出可知，这个才是我们想要的结果。

set_fact 和 register 的功能很类似，它也可以将 task 输出赋值给变量。set_fact 更像 shell 中变量的赋值方式，可以将某个变量的值赋值给另一个变量，也可以将字符串赋值给变量。看下面这个例子：

```
复制- hosts: 172.16.213.231
  remote_user: root
  tasks:
   - name: hostname command
     shell: hostname
     register: host_result
   - set_fact: var1="{{host_result.stdout}}"
   - set_fact: var2="This is a string"
   - debug: msg="{{var1}}  {{var2}}"
```

这个例子是将 hostname 的输出结果赋值给 host_result 变量，然后通过 set_fact 将 host_result 变量赋值给 var1 变量，接着又将一个字符串赋值给 var2 变量，最后，通过 debug 模块输出这些变量信息。注意这些模块的使用方式和书写格式。

这个 playbook 的输出结果为：

```
复制TASK [debug] ********************************************************
ok: [172.16.213.231] => {
```

```
    "msg": "server231.localdomain  This is a string"
}
```

（5）delegate_to、connection 和 local_action 模块

Ansible 默认只会对远程主机执行操作，但有时候如果需要在管理机本机上执行一些操作，该如何实现呢？其实现的方法有很多，可以通过 delegate_to（任务委派）来实现，也可通过 connection:local 方法，还可通过 local_action 关键字来实现。

下面来看一个例子，说明它们的用法。

```
复制- hosts: 172.16.213.231
  remote_user: root
  gather_facts: true
  tasks:
   - name: connection
     shell: echo "connection . {{inventory_hostname}} $(hostname) ." >>
/tmp/local.log
     connection: local
   - name: delegate_to
     shell: echo "delegate_to . {{inventory_hostname}} $(hostname) ." >>
/tmp/local.log
     delegate_to: localhost
   - name: local_action
     local_action: shell echo "local_action. {{inventory_hostname}} $(hostname)"
>> /tmp/local.log
```

这个例子依次使用了 connection、delegate_to 和 local_action 三种方式，还使用了一个变量 {{inventory_hostname}}，这是 Ansible 的一个内置变量，用来获取远程主机的主机名，说到主机名，其实用到了 facts 信息，所以，需要设置 gather_facts 选项为 true。另外，$(hostname) 是 shell 里面的变量，也是用来获取主机名，此例子实现的功能是将远程主机的主机名依次输出到管理机的 /tmp/local.log 文件中。

大数据运维环境下 ansible-playbook 应用案例

1. 批量更改主机名并生成本地解析

在大数据运维环境下，对主机名要求比较严格，所以对大数据节点的主机名要进行统一规划，然后集中设置。如果本地没有建立 DNS 解析服务器，还需要对每个节点添加本地解

析，也就是将每个节点的 IP 和主机名的对应关系添加到 /etc/hosts 文件中。要解决这两个问题，只需要两个 playbook 脚本即可自动完成。

要批量更改每个节点的主机名，首先需要修改 ansible 中 /etc/ansible/hosts 文件内容，添加如下配置：

复制[hostall]
172.16.213.229 hostname=namenodemaster
172.16.213.230 hostname=slave001
172.16.213.231 hostname=slave002
这里定义了一个名为 hostall 的主机组，组中有三台主机，每个主机 IP 后面跟了一个 hostname 变量，变量后面就是定义好的主机名，而这个变量可以在 playbook 脚本中直接引用。

接下来就可以编写 playbook 脚本了，内容如下：

```
复制- hosts: hostall
  remote_user: root
  tasks:
    - name: change name
      shell: "echo {{hostname}} > /etc/hostname"
    - name:
      shell: hostname {{hostname}}
```

其中，变量 {{hostname}} 及值就是在 /etc/ansible/hosts 文件中定义的“hostname=namenodemaster”这部分内容。通过使用 shell 模块，实现将定义好的主机名添加到每个远程主机的 /etc/hostname 文件中（限于 RHEL/Centos7/8 系统），然后执行 hostname 命令使其生效。

每个主机名修改完毕后，还需要构建一个本地解析文件（IP 和主机名对应的文件），然后传到每个远程主机上，要实现这个功能，可以编写如下 playbook 脚本，内容如下：

```
复制- hosts: hostall
  remote_user: root
  roles:
  - roles
```

```
  tasks:
   - name: add localhost
     local_action: shell echo "127.0.0.1   localhost" >
{{AnsibleDir}}/roles/templates/hosts.j2
     run_once: true
   - set_fact: ipaddress=
{{hostvars[inventory_hostname].ansible_default_ipv4.address}}
   - set_fact: hostname={{hostvars[inventory_hostname].ansible_facts.hostname}}
   - name: add host record
     local_action: shell echo {{ipaddress}} {{hostname}} >>
{{AnsibleDir}}/roles/templates/hosts.j2
   - name: copy hosts.j2 to allhost
     template: src={{AnsibleDir}}/roles/templates/hosts.j2 dest=/etc/hosts
```

在 playbook 中，使用了角色中的变量，所以要了解 ansible 的默认目录结构，如下图所示：

我们的程序安装在 /etc/ansible 命令下，在这个目录中有三个子目录，分别是 files、templates 和 roles。files 目录主要存放一些要拷贝的远程主机的程序文件；templates 目录下存放了一些配置好的模板文件，该模板文件会统一拷贝到远程主机中；roles目录下创建了一个 main.yml 文件，用来定义角色变量，main.yml 中变量定义方式如下：

复制server1_hostname: 172.16.213.229
server2_hostname: 172.16.213.230
server3_hostname: 172.16.213.231
AnsibleDir: /etc/ansible
BigdataDir: /opt/bigdata
hadoopconfigfile: /etc/hadoop
其中，每行内容中冒号前面的是变量名，后面的是变量的值，定义变量后，就可以在playbook中进行引用了。

最后，再回到上面这个 playbook 文件中，由于要使用角色变量，所以引入了 roles 关键字。接下来，在 tasks 任务中，首先使用了 local_action 模块，在管理机上生成了一个模板文件 hosts.j2，注意这里面的变量 {{AnsibleDir}} 就是在 main.yml 中定义好的，

run_once 表示本地 shell 仅仅执行一次；接着通过 set_fact 定义了两个变量 ipaddress 和 hostname，这两个变量都从 ansible 内置变量中获取具体的值，然后将获取到的 ipaddress 和 hostname 值写入管理机上的 hosts.j2 文件中；最后一个操作步骤是通过 template 模块，将 hosts.j2 模板文件拷贝到远程主机的 /etc/ 目录下并重命名为 hosts 文件。

将此脚本放到 /etc/ansible 目录下，并命名为 hosts.yml，然后执行如下命令：

复制[root@server239 ansible]# ansible-playbook hosts.yml
如果执行成功，会有绿色、浅黄色输出提示；如果执行失败，可以看红色输出内容，判断检查问题。

2. 主机自动建立 SSH 信任
大数据环境下，为了安装、配置和维护的方便，一般会设置管理机（安装 Ansible 的机器）和每个集群节点之间的无密码登录（单向信任），而无密码登录最简单的方式是通过设置 SSH 公私钥认证机制。

下面 playbook 脚本可以完成管理机到远程主机组 hostall 的无密码登录，脚本内容如下：

复制- hosts: hostall
  gather_facts: no
  roles:
   - roles
  tasks:
   - name: close ssh yes/no check
     lineinfile: path=/etc/ssh/ssh_config regexp='(.*)StrictHostKeyChecking(.*)' line="StrictHostKeyChecking no"
   - name: delete /root/.ssh/
     file: path=/root/.ssh/ state=absent
   - name: create .ssh directory
     file: dest=/root/.ssh mode=0600 state=directory
   - name: generating local public/private rsa key pair
     local_action: shell ssh-keygen -t rsa -b 2048 -N '' -y -f /root/.ssh/id_rsa
   - name: view id_rsa.pub

```
    local_action: shell cat /root/.ssh/id_rsa.pub
    register: sshinfo
  - set_fact: sshpub={{sshinfo.stdout}}
  - name: add sshkey
    local_action: shell echo {{sshpub}} >
{{AnsibleDir}}/roles/templates/authorized_keys.j2
  - name: copy authorized_keys.j2 to all hosts
    template: src={{AnsibleDir}}/roles/templates/authorized_keys.j2
dest=/root/.ssh/authorized_keys mode=0600
    tags:
    - copy sshkey
```

这个 playbook 稍微复杂一些，它仍然用到了角色变量，所以此脚本要放在 /etc/ansible 目录下，脚本一开始通过 lineinfile 模块对远程主机上的 sshd 配置文件 ssh_config 进行文件内容替换，这个替换是关闭 SSH 第一次登录时给出的“yes/no”提示。

接着在远程主机上删除 /root/.ssh 目录，并重新创建此目录，该操作的目的是确保远程主机 /root/.ssh 目录是干净的、权限正确。

然后通过 local_action 模块在管理机上生成一对公私钥，同时将生成的公钥文件内容作为变量 sshinfo 的值，并通过 set_fact 模块重新定义一个变量 sshpub，此变量引用 sshinfo 变量的 stdout 输出，也就是最终的公钥值，紧接着，将变量 sshpub 的内容写入管理机 authorized_keys.j2 模板文件中。

最后，使用 template 模块将 authorized_keys.j2 模板文件拷贝到每个远程主机的 /root/.ssh 目录下，并重命名为 authorized_keys，同时给文件授于属主读、写权限。

将此脚本放到 /etc/ansible 目录下，并命名为 ssh.yml，然后执行如下命令：

```
复制[root@server239 ansible]# ansible-playbook  ssh.yml
```

3. 自动化安装 JDK

自动化安装 JDK 是大数据运维中最常见的一个场景，一般下载二进制版本解压即可使用，所以安装 JDK 的过程是把下载好的 JDK 程序拷贝到远程主机的过程，安装完成后，还要添加 JAVA_HOME 到系统环境变量中，以让系统识别安装的 JDK。

下面的 playbook 文件是自动化安装 JDK 的整个过程，内容如下：

复制- hosts: hostall
  remote_user: root
  roles:
  - roles
  tasks:
   - name: mkdir jdk directory
     file: path={{BigdataDir}} state=directory mode=0755
   - name: copy and unzip jdk
     unarchive: src={{AnsibleDir}}/roles/files/jdk.tar.gz dest={{BigdataDir}}
   - name: set env
     lineinfile: dest=/etc/profile line="{{item.value}}" state=present
     with_items:
     - {value: "export JAVA_HOME={{BigdataDir}}/jdk"}
     - {value: "export PATH=$JAVA_HOME/bin:$PATH"}
   - name: chmod bin
     file: dest={{BigdataDir}}/jdk/bin mode=0755 recurse=yes
   - name: enforce env
     shell: source /etc/profile

此脚本中的 BigdataDir、AnsibleDir 都是角色变量，前面已经定义过具体的路径了，其中，jdk.tar.gz 位于管理机上，脚本最后通过 item.value 变量将 JDK 环境变量写入到了 /etc/profile 文件的最后，这个变量的定义和引用方式需要注意。

将此脚本放到 /etc/ansible 目录下，并命名为 jdk.yml，然后执行如下命令：

复制[root@server239 ansible]# ansible-playbook jdk.yml
脚本执行成功的话，那么JDK 和环境变量就都配置好了。