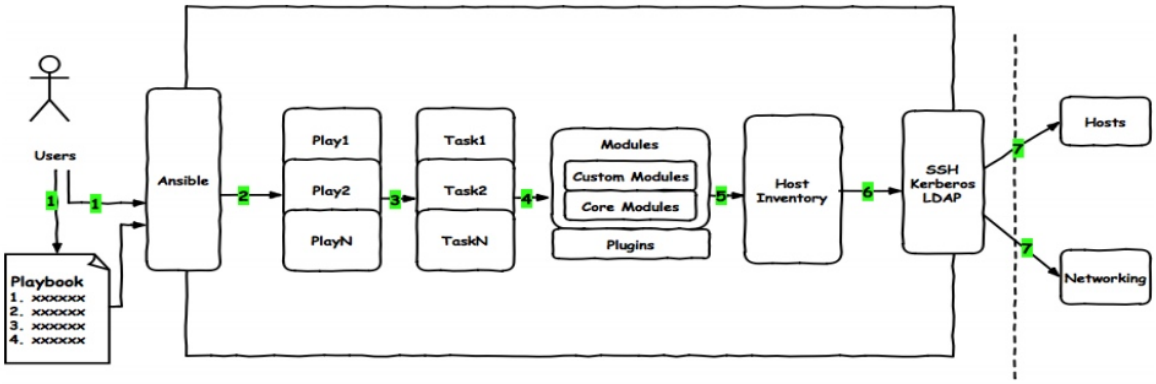
02.Ansible Playbook进阶

1.Ansible Playbook

1.1 什么是Playbook

`playbook` 是一个 由 `yaml` 语法编写的文本文件，它由 `play` 和 `task` 两部分组成。
`play`： 主要定义要操作主机或者主机组
`task`：主要定义对主机或主机组具体执行的任务，可以是一个任务，也可以是多个任务（模块）



总结: `playbook` 是由一个或多个 `play` 组成，一个 `play` 可以包含多个 `task`任务。
可以理解为: 使用多个不同的模块来共同完成一件事情。

1.2 Playbook与Ad-Hoc

- 1) `playbook` 是对 `AD-Hoc` 的一种编排方式。
- 2) `playbook` 可以持久运行，而 `Ad-Hoc` 只能临时运行。
- 3) `playbook` 适合复杂的任务，而 `Ad-Hoc` 适合做快速简单的任务。
- 4) `playbook` 能控制任务执行的先后顺序。

1.3 Playbook书写格式

`playbook` 是由 `yml` 语法书写，结构清晰，可读性强，所以必须掌握 `yml` 语法

语法	描述
缩进	YAML使用固定的缩进风格表示层级结构,每个缩进由两个空格组成, 不能使用tabs
冒号	以冒号结尾的除外，其他所有冒号后面所有必须有空格。
短横线	表示列表项，使用一个短横杠加一个空格。多个项使用同样的缩进级别作为同一列表。

1.4 Playbook案例实战

1.4.1 Ansible部署NFS示例

1.编写安装配置 `nfs` 服务的 `playbook` 文件

```
[root@manager ~]# cat  install_nfs.yml
- hosts: webservers
  tasks:
    - name: Installed NFS Server
      yum:
        name: nfs-utils
        state: present

    - name: Configure NFS Server
      copy:
        src: ./exports.j2
        dest: /etc/exports

    - name: Init NFS Server
      file:
        path: /ops_web
        state: directory
        owner: nfsnobody
        group: nfsnobody

    - name: Systemd NFS Server
      systemd:
        name: nfs
        state: started
        enabled: yes
```

2.准备 `playbook` 依赖的 `exports.j2` 文件

```
[root@m01 playbook]# echo "/data 172.16.1.0/24(rw,sync)" >
exports.j2
```

3.检查 `playbook` 语法

```
[root@m01 playbook]# ansible-playbook nfs.yml --syntax-check

playbook: nfs.yml
```

5.客户端执行命令测试

```
[root@m01 playbook]# showmount -e 172.16.1.8
Export list for 172.16.1.8:
/data 172.16.1.0/24
[root@m01 playbook]# showmount -e 172.16.1.7
Export list for 172.16.1.7:
/data 172.16.1.0/24
```

1.4.2 Ansible部署Httpd示例

1.编写安装配置 `httpd` 服务的 `playbook` 文件

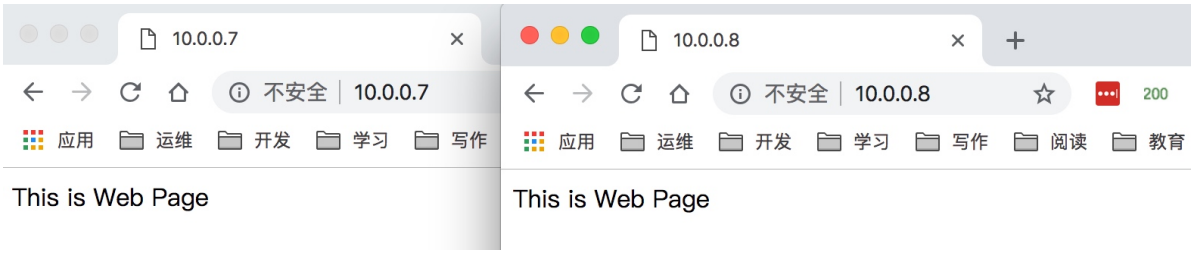
```
[root@manager playbook-httpd]# cat installed_httpd.yml
#1.定义play
#2.定义task、（Installed、Configure、Init、Systemd）


- hosts: webservers
  tasks:
    - name: Installed Httpd Server
      yum:
        name: httpd
        state: present

    - name: Configure Httpd Server
      copy:
        src: ./httpd.conf.j2
        dest: /etc/httpd/conf/httpd.conf
        owner: "root"
        group: "root"
        mode: '0644'
        backup: yes
      notify: Restart Httpd Server

    - name: Init Httpd Server
      copy:
        src: ./index.html.j2
        dest: /var/www/html/test.html

    - name: Systemd Httpd Server
      systemd:
        name: httpd
        state: started
        enabled: yes

  handlers:
```

```
    - name: Restart Httpd Server
      systemd:
        name: httpd
        state: restarted
```

2.检查 `playbook` 语法

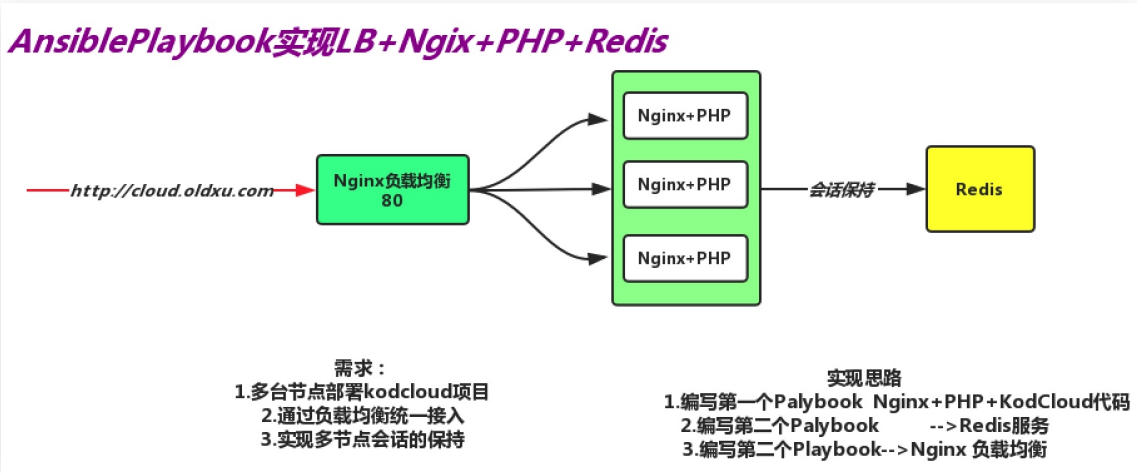
```
[root@m01 playbook]# ansible-playbook installed_httpd.yml --
syntax-check

playbook: installed_httpd.yml

[root@m01 playbook]# ansible-playbook installed_httpd.yml
```

4.访问服务器对应的 `web` 页面测试



1.5 Playbook部署集群架构

1.5.1 项目架构图



1.5.2 项目需求

- 1.使用多台节点部署 `kodcloud` 网盘
- 2.使用 `Nginx` 作为负载均衡统一调度
- 3.使用 `Redis` 实现多台节点会话保持

1.5.3 环境准备

```
[root@ansible-hostname ~]# cat /etc/ansible/hosts
[dbservers]
172.16.1.5

[lbservers]
172.16.1.6

[webservers]
172.16.1.7
172.16.1.8
```

1.5.4 部署Redis

```
[root@manager playbook-cluster]# cat install_redis.yml
- hosts: dbservers
  tasks:
    - name: Installed Redis Server
      yum:
        name: redis
        state: present

    - name: Configure Redis Server
      copy:
        src: conf/redis.conf.j2
        dest: /etc/redis.conf
        owner: 'redis'
        group: 'root'
        mode: '0640'
      notify: Restart Redis Server

    - name: Systemd Redis Server
      systemd:
        name: redis
        state: started
        enabled: yes

  handlers:
    - name: Restart Redis Server
      systemd:
        name: redis
        state: restarted
```

1.5.5 部署PHP环境

```
[root@manager playbook-cluster]# cat install_nginx_php.yml
#1.安装nginx
#2.安装php
#3.添加nginx虚拟主机，触发重启
#4.配置php，连接redis；触发重启
#5.部署phpadmin；、


- hosts: webservers
  tasks:
    - name: Installed Nginx Server
      yum:
        name: nginx
        state: present

    - name: Installed PHP Server
      yum:
        name: "{{ packages }}"
      vars:
        packages:
          - php71w
          - php71w-cli
          - php71w-common
          - php71w-devel
          - php71w-embedded
          - php71w-gd
          - php71w-mcrypt
          - php71w-mbstring
          - php71w-pdo
          - php71w-xml
          - php71w-fpm
          - php71w-mysqlnd
          - php71w-opcache
          - php71w-pecl-memcached
          - php71w-pecl-redis
          - php71w-pecl-mongodb
     # state: present

    - name: Create Nginx Process Runtime Group
      group:
        name: www
        gid: 666
```

```
    - name: Create Nginx Process Runtime User
      user:
        name: www
        uid: 666
        create_home: no

    - name: Configure Nginx Nginx.conf
      copy:
        src: ./conf/nginx.conf.j2
        dest: /etc/nginx/nginx.conf
        owner: 'root'
        group: 'root'
        mode: '0644'
      notify: Restart Nginx Server


    - name: Configure php php.ini
      copy:
        src: ./conf/php.ini.j2
        dest: /etc/php.ini
      notify: Restart PHP Server

    - name: Configure php php-fpm.conf
      copy:
        src: ./conf/php-fpm.d.www.conf.j2
        dest: /etc/php-fpm.d/www.conf
      notify: Restart PHP Server

    - name: Systemd Nginx Server
      systemd:
        name: nginx
        state: started
        enabled: yes

    - name: Systemd PHP Server
      systemd:
        name: php-fpm
        state: started
        enabled: yes

# download code

    - name: Create Web Site Directory
      file:
        path: /ansible/admin
```

```
        state: directory
        owner: 'www'
        group: 'www'
        mode: '0755'

    - name: Unarchive Myadmin Code
      unarchive:
        src: file/phpmyadmin.zip
        dest: /ansible/admin
        owner: 'www'
        group: 'www'

    - name: Configure Nginx VHosts ansible.oldxu.com;
      copy:
        src: ./conf/ansible.oldxu.com.conf.j2
        dest: /etc/nginx/conf.d/ansible.oldxu.com.conf
      notify: Restart Nginx Server

  handlers:
    - name: Restart Nginx Server
      systemd:
        name: nginx
        state: restarted

    - name: Restart PHP Server
      systemd:
        name: php-fpm
        state: restarted
```

1.5.6 部署负载均衡

```
[root@manager playbook-cluster]# cat install_proxy.yml
- hosts: proxyservers
  tasks:
    - name: Installed Nginx Proxy
      yum:
        name: nginx
        state: present

    - name: Add Nginx Proxy VHosts Virtual Site
      copy:
        src: conf/proxy_ansible.oldxu.com.conf.j2
        dest: /etc/nginx/conf.d/proxy_ansible.oldxu.com.conf
      notify: Restart Nginx Server
```

```
    - name: Systemd Nginx Server
      systemd:
        name: nginx
        state: started
        enabled: yes

  handlers:
    - name: Restart Nginx Server
      systemd:
        name: nginx
        state: restarted
```

2.Ansible Variables

2.1 什么是变量

变量提供了便捷的方式来管理 `ansible` 项目中的动态值。 比如 `nginx-1.12`，可能后期会反复的使用到这个版本的值，那么如果将此值设置为变量，后续使用和修改都将变得非常方便。这样可以简化项目的创建和维护;

2.2 变量定义的方式

- 在 `Ansible` 中定义变量分为如下三种方式：
  - 1) 通过命令行传递变量参数定义
  - 2) 在play文件中进行定义变量
    - 2.1) 通过vars定义变量
    - 2.2) 通过vars_files定义变量
  - 3) 通过inventory在主机组或单个主机中设置变量
    - 3.1) 通过host_vars对主机进行定义
    - 3.2) 通过group_vars对主机组进行定义
- 问题：如果定义的变量出现重复，造成冲突，如何解决?

2.3 在Playbook中定义变量

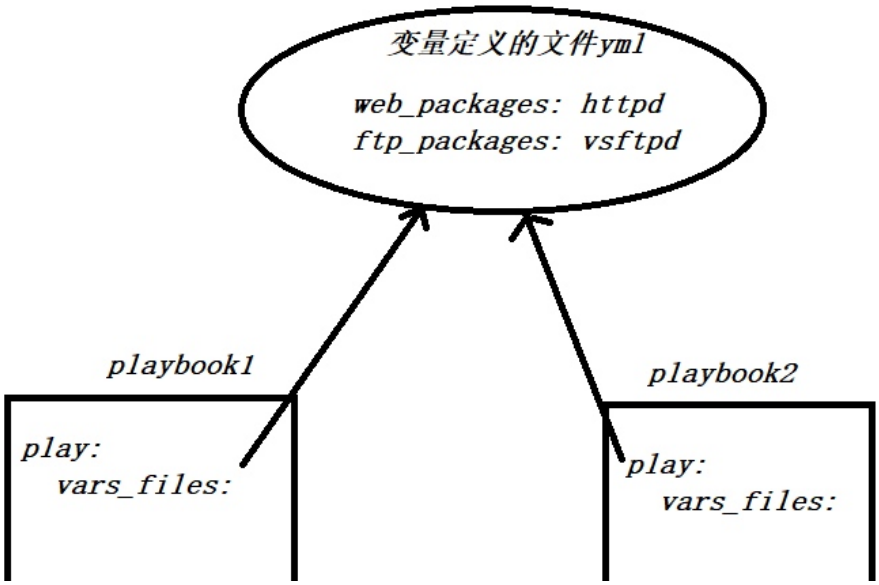
2.3.1 vars方式定义变量

在 `playbook` 的文件中开头通过 `vars` 关键字进行变量定义

```
[root@ansible project1]# cat f1.yml
- hosts: webservers
  vars:
    - web_packages: httpd
    - ftp_packages: vsftpd

  tasks:
    - name: Output Vaiables
      debug:
        msg:
          - "{{ web_packages }}"
          - "{{ ftp_packages }}"
```

2.3.2 vars_files方式定义变量

在 `playbook` 中使用 `vars_files` 指定文件作为变量文件，好处就是其他的 `playbook` 也可以调用；



1.准备一个用于存储变量的文件，后缀为 `.yml` 文件内容：`vars_name: value`

```
[root@ansible project1]# cat vars.yml
web_packages: httpd
ftp_packages: vsftpd
```

2.使用 `Playbook` 调用变量文件

```
[root@ansible project1]# cat f2.yml
- hosts: webservers
  vars_files: ./vars.yml
  tasks:
    - name: Output Vaiables
      debug:
        msg:
          - "{{ web_packages }}"
          - "{{ ftp_packages }}"
```

2.4 在Inventory中定义变量

2.2.3 Inventory文件中定义变量

3、在inventory主机清单中定义变量，但是要注意：主机变量优先级高于主机组变量。（了解即可）

1.设定主机变量的方式

```
[root@m01 project1]# vim /etc/ansible/hosts
[webservers]
172.16.1.3 myid=1 state=master
172.16.1.4 myid=2 state=backup

[webservers:vars]
port=80                          # groups
```

2.`playbook` 调用变量

```
[root@m01 project1]# cat p3.yml
- hosts: webservers
  tasks:
    - name: Output Vaiables
      debug:
        msg:
          - "{{ myid }} {{ state }} {{ port }}"
```

2.2.4 使用host_vars定义变量

1.在项目目录中创建 `host_vars` 目录，然后在创建一个文件，文件的文件名称要与 `inventory` 清单中的主机名称要保持完全一致，如果是ip地址，则创建相同ip地址的文件即可

```
[root@ansible project1]# cat hosts
[webserver]
172.16.1.7
172.16.1.8
[root@ansible project1]# mkdir host_vars
```

2.在 `host_vars` 目录中创建文件，给 `172.16.1.7` 主机定义变量

```
[root@ansible project1]# cat host_vars/172.16.1.7
web_packages: zlib-static
ftp_packages: zmap
```

3.准备一个 `playbook` 文件调用 `host_vars` 目录中定义的主机变量

```
[root@ansible project1]# cat f4.yml
- hosts: 172.16.1.7
  tasks:
    - name: Output Vaiables
      debug:
        msg:
          - "{{ web_packages }}"
          - "{{ ftp_packages }}"

- hosts: 172.16.1.8
  tasks:
    - name: Output Vaiables
      debug:
        msg:
          - "{{ web_packages }}"
          - "{{ ftp_packages }}"
```

2.2.5 使用group_vars定义变量

1.在项目目录中创建 `group_vars` 目录，然后在创建一个文件，文件的文件名称要与 `inventory` 清单中的组名称保持完全一致；

```
[root@ansible project1]# cat hosts
[webservers]
172.16.1.7
172.16.1.8
[root@ansible project1]# mkdir group_vars
```

2.在 `group_vars` 目录中创建 `webservers` 文件，为 `webservers` 主机组设定变量；

```
[root@ansible project1]# cat group_vars/webservers
web_packages: wget
ftp_packages: tree
```

3.编写 `playbook`，只需在 `playbook` 文件中使用变量即可；

```
[root@ansible project1]# cat f4.yml
- hosts: webservers
  tasks:
    - name: Output Vaiables"
      debug:
        msg:
          - "{{ web_packages }}"
          - "{{ ftp_packages }}"
```

4.测试其他组能否使用 `webservers` 组中定义的变量；测试后会发现无法调用；

```
[root@ansible project1]# cat f4.yml
- hosts: lbservers   #使用lbservers
  tasks:
    - name: Output Vaiables
      debug:
        msg:
          - "{{ web_packages }}"
          - "{{ ftp_packages }}"
```

5.但是系统提供了特殊的 `all` 组，也就说在 `group_vars` 目录下创建一个 `all` 文件，定义变量对所有的主机组都生效；

```
[root@ansible project1]# cat group_vars/all
web_packages: wget
ftp_packages: tree

[root@ansible project1]# cat f4.yml
- hosts: lbserver   #无论哪个组使用该变量，都没有任何的问题
  tasks:
    - name: Output Vaiables
      debug:
        msg:
          - "{{ web_packages }}"
          - "{{ ftp_packages }}"
```

2.5 通过执行Playbook传递变量

在执行Playbook时，可以通过命令行 `--extra-vars` 或 `-e` 外置传参设定变量；

1.准备 `playbook` 文件

```
[root@ansible project1]# cat f5.yml
- hosts: webserver
  tasks:
    - name: Output Vaiables
      debug:
        msg:
          - "{{ web_packages }}"
          - "{{ ftp_packages }}"
```

2.执行 `playbook` 时进行变量的传递

```
[root@ansible project1]# ansible-playbook f5.yml -i hosts -e
"web_packages=GeoIP"
```

3.传递多个外置变量的方式

```
[root@ansible project1]# ansible-playbook f5.yml -i hosts -e
"web_packages=GeoIP" -e "ftp_packages=telnet"
```

2.6 变量优先级测试

- 定义相同的变量不同的值，来测试变量的优先级。操作步骤如下
  - 1）在plabook中定义vars变量
  - 2）在playbook中定义vars_files变量
  - 3）在host_vars中定义变量
  - 4）在group_vars中定义变量
  - 5）通过执行命令传递变量
- 结果报告:
  - 命令行变量-->play中的vars_files-->play中的vars-->inventory-hosts-->
  - host_vars-->group_vars/group_name-->group_vars/all-->inventory-group-->

2.7 变量注册Register

2.7.1 什么是Register

`register` 关键字可以将某个 `task` 任务结果存储至变量中，最后使用 `debug` 输出变量内容，可以用于后续排障；

2.7.2 Register示例1

```
[root@manager ~]# cat f5.yml
---
- hosts: all
  tasks:
    - name:
      shell: netstat -lntp
      register: System_Status

    - name: Get System Status
      debug: msg={{System_Status.stdout_lines}}

#playbook执行结果
[root@manager ~]# ansible-playbook f5.yml
PLAY [all]
*********************************************************************

TASK [Gathering Facts]
*********************************************************************
ok: [10.0.0.30]

TASK [shell]
*********************************************************************
changed: [10.0.0.30]

TASK [Get System Status]
*********************************************************************
ok: [10.0.0.30] => {
    "msg": [
        "tcp    0   0 0.0.0.0:22        0.0.0.0:*   LISTEN
925/sshd",
        "tcp6   0   0 :::22             :::*        LISTEN
925/sshd"
    ]
}
```

```
PLAY RECAP
*********************************************************************
10.0.0.30                  : ok=3    changed=1    unreachable=0
 failed=0
```

2.7.2 Register示例2

- 使用 `register` 关键字完成 `jumpserver key` 的创建；

```
[root@mananger playbook]# cat f9.yaml
- hosts: webservers
  tasks:
    - name: Run Shell Command Random string
      shell:
        cmd: 'if ! grep "SECRET_KEY" ~/.bashrc; then
                SECRET_KEY=`cat /dev/urandom | tr -dc A-Za-z0-9 |
head -c 50`;
                echo "SECRET_KEY=$SECRET_KEY" >> ~/.bashrc;
                echo $SECRET_KEY;
              else
                 echo $SECRET_KEY;
              fi'
      register: SECRET_KEY


    - name: Run Shell Command BOOTSTRAP_TOKEN
      shell:
        cmd: 'if ! grep "BOOTSTRAP_TOKEN" ~/.bashrc; then
                BOOTSTRAP_TOKEN=`cat /dev/urandom | tr -dc A-Za-
z0-9 | head -c 16`;
                echo "BOOTSTRAP_TOKEN=$BOOTSTRAP_TOKEN" >>
~/.bashrc;
                echo $BOOTSTRAP_TOKEN;
             else
                echo $BOOTSTRAP_TOKEN;
             fi'
      register: BOOTSTRAP_TOKEN

    - name: Copy Jms Configure
      template:
        src: ./j-config.yml
        dest: /tmp/jms_config.yml

    - name: Copy Koko Configure
      template:
```

```
        src: ./k-config.yml
        dest: /tmp/koko_config.yml


# 配置文件中：
SECRET_KEY: {{ SECRET_KEY.stdout.split('=')[1] }}
BOOTSTRAP_TOKEN: {{ BOOTSTRAP_TOKEN.stdout.split('=')[1] }}
```

3.Ansible Facts Variables

3.1 什么是facts

`Ansible facts` 用来自动采集，”被控端主机“ 自身的状态信息。
比如: 主机名、IP地址、系统版本、CPU数量、内存状态、磁盘状态等等。

3.2 facts使用场景

- 1.通过facts变量检查被控端硬件CPU信息，从而生成不同的Nginx配置文件。
- 2.通过facts变量检查被控端内存状态信息，从而生成不同的memcached的配置文件。
- 3.通过facts变量检查被控端主机IP地址信息，从而生成不同的redis配置文件。
- 4.通过facts变量.....

3.3 facts语法示例

1.通过 `facts` 获取被控端的主机名与IP地址，然后通过 `debug` 输出；

```
[root@m01 ~]# cat facts.yml
- hosts: webservers
  tasks:
    - name: Output variables ansible facts
      debug:
        msg: >
         this default IPv4 address "{{ ansible_fqdn}}" is "{{
ansible_default_ipv4.address }}"
```

2.facts开启后会影响Ansible主机的性能，如果没有采集被控端主机需求可选择关闭

```
[root@m01 ~]# cat facts.yml
- hosts: web
  gather_facts: no  #关闭信息采集
  tasks:
```

3.如何获取facts的变量，需要使用filter进行过滤

3.4 facts实践案例

3.4.1 案例1-根据主机IP地址生成Redis配置

1）准备两台物理主机

172.16.1.7

172.16.1.8

2）编写redis服务playbook

```
[root@manager ansible_variables]# cat redis.yml
- hosts: webservers
  tasks:
    - name: Installed Redis Server
```

```
      yum:
        name: redis
        state: present

    - name: Configure Redis Server
      template:
        src: ./redis.conf.j2
        dest: /etc/redis.conf
      notify: Restart Redis Server

    - name: Started Redis Server
      systemd:
        name: redis
        state: started
        enabled: yes

  handlers:
    - name: Restart Redis Server
      systemd:
        name: redis
        state: restarted
```

3) 配置文件如下

```
[root@manager ansible_variables]# cat redis.conf.j2
...
bind 127.0.0.1 {{ ansible_eth1.ipv4.address }}
...
```

3.4.2 案例2-根据主机CPU生成Nginx配置

1）准备两台物理CPU核心不一样的主机
172.16.1.7 1核
172.16.1.8 2核

2）编写nginx服务playbook

```
[root@manager ansible_variables]# cat nginx.yml
- hosts: webservers
  tasks:
    - name: Install Nginx Server
      yum:
        name: nginx
        state: present
```

```
    - name: Configure  Nginx.conf
      template:
        src: ./nginx.conf.j2
        dest: /tmp/nginx.conf
      notify: Restart Nginx Server

    - name: Started Nginx Server
      systemd:
        name: nginx
        state: started
        enabled: yes

  handlers:
    - name: Restart Nginx Server
      systemd:
        name: nginx
        state: restarted
```

3）nginx配置文件如下
ansible_processor_cores: 4 # 核心数（每颗物理CPU的核心数)
ansible_processor_count: 2 # 颗数 （有几个CPU）
ansible_processor_vcpus: 8 # 总核心数

```
[root@manager ansible_variables]# cat nginx.conf.j2
user www;
worker_processes {{ ansible_processor_vcpus * 2 }};

error_log  /var/log/nginx/error.log warn;
pid        /var/run/nginx.pid;

events {
    worker_connections  1024;
}

http {
    include       /etc/nginx/mime.types;
    default_type  application/octet-stream;

    log_format  main  '$remote_addr - $remote_user [$time_local]
"$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent"
"$http_x_forwarded_for"';

    access_log  /var/log/nginx/access.log  main;
```

```
    sendfile        on;
    tcp_nopush     on;
    keepalive_timeout  65;
    gzip  on;
    include /etc/nginx/conf.d/*.conf;
}
```

3.4.3 案例3-根据主机内存生成Memcached配置

1）准备两台物理内存不一样的主机
172.16.1.7 1G
172.16.1.8 2G

2）编写memcached服务playbook

```
[root@ansible project1]# cat memcached.yml
- hosts: webserver
  tasks:
    - name: Installed Memcached Server
      yum: name=memcached state=present

    - name: Configure Memcached Server
      template: src=./memcached.j2 dest=/etc/sysconfig/memcached

    - name: Service Memcached Server
      service: name=memcached state=started enabled=yes
```

3）memcached配置文件如下

```
[root@ansible project1]# cat memcached.j2
PORT="11211"
USER="memcached"
MAXCONN="1024"
#根据内存状态生成不同的配置(支持+-*/运算)
CACHESIZE="{{ ansible_memtotal_mb //2 }}"
OPTIONS=""
```

3.6 facts变量优化方案

3.6.1 方式1-关闭facts采集加速执行

1.编写 `TASK` 任务 `sleep10` 秒，针对 `15` 台机器同时执行，需要消耗的时间大概是 `1m54.980s`

```
[root@m01 ~]# cat ansible_facts.yml
- hosts: all
  tasks:
    - name: sleep 10
      command: sleep 10
```

2.使用 `gather_facts: no` 关闭 `facts` 信息采集，发现仅花费了 `0m38.164s` ，整个速度提升了3倍，如果调整 `forks` 操作的主机的数量，也可以得到非常大的提升；

```
[root@m01 ~]# cat ansible_facts.yml
- hosts: all
  gather_facts: no
  tasks:
    - name: sleep 10
      command: sleep 10
```

3.6.2 方式2-Redis缓存facts加速执行

1.当我们使用 `gather_facts: no` 关闭 `facts`，确实能加速 `Ansible` 执行，但是有时候又需要使用 `facts` 中的内容，还希望执行的速度快一点，这时候可以设置 `facts` 的缓存；

```
[root@m01 ~]# yum install python2-redis
[root@m01 ~]# pip install --upgrade pip
[root@m01 ~]# pip install redis

[root@m01 ~]# cat /etc/ansible/ansible.cfg
[defaults]
# smart 表示默认收集 facts，但 facts 已有的情况下不会收集，即使用缓存 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。

gathering = smart            #在使用 facts 缓存时设置为smart
fact_caching_timeout = 86400
fact_caching = redis
fact_caching_connection = 172.16.1.51:6379

# 若 redis 设置了密码
```

```
# fact_caching_connection = localhost:6379:0:admin
```

2.编写 `Playbook` 测试；

```
[root@m01 ~]# cat ansible_facts.yml
- hosts: all
  gather_facts: no
  tasks:
    - name: sleep 10
      command: sleep 10
```

- 3.测试结果如下：
  - 执行第一次花费了 `1m49.881s` 因为第一次需要将 `facts` 信息缓存至`Redis`
  - 执行第二次花费了 `0m38.130s` 可以看出使用 `Redis` 缓存 `facts`变量，整体执行时间提高了3倍

4.Ansible Task Control

4.1 when条件语句

`when` 判断在 `Ansible` 中的使用频率非常高；比如 `yum` 模块可以自动检测软件包是否已被安装，而无需人为干涉，但对于有些任务则是需要进行判断才可以实现的。

- 比如： `web` 节点都需要配置 `nginx` 仓库，但其他节点并不需要，此时就会用到 `when` 判断。
- 比如: `Centos` 与 `Ubuntu` 都需要安装 `Apache`，而 `Centos` 系统软件包为 `httpd`，而 `Ubuntu`系统软件包为`httpd2`，那么此时就需要判断主机系统，然后为不同的主机系统安装不同的软件包。

4.1.1 案例1-根据不同操作系统安装相同的软件

- 为所有主机安装 `Apache` 软件
  - 系统为`CentOS`： 安装 `httpd`
  - 系统为`Ubuntu`： 安装 `httpd2`

```
[root@m01 playbook]# cat when_system.yml
- hosts: web
  tasks:
        #通过fact变量判断系统为centos才会安装httpd
    - name: Centos Install httpd
      yum: name=httpd state=present
      when: (ansible_distribution == "CentOS")

        #通过fact变量判断系统为ubuntu才会安装httpd2
    - name: Ubuntu Install httpd
      yum: name=httpd2 state=present
      when: (ansible_distribution == "Ubuntu")
```

2.执行 `playbook`

```
[root@m01 playbook]# ansible-playbook when_system.yml

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

TASK [Gathering Facts]
*****************************************************************
***************
ok: [172.16.1.8]
ok: [172.16.1.7]

TASK [Centos Install httpd]
*****************************************************************
**********
ok: [172.16.1.8]
ok: [172.16.1.7]

TASK [Ubuntu Install httpd]
*****************************************************************
**********
skipping: [172.16.1.7]
skipping: [172.16.1.8]

PLAY RECAP
*****************************************************************
**************************
172.16.1.7                 : ok=2    changed=0    unreachable=0
failed=0
```

```
172.16.1.8                 : ok=2    changed=0    unreachable=0
failed=0
```

4.1.2 案例2-为特定的主机添加Nginx仓库

- 为所有主机添加 `Nginx` 仓库
  - 主机名为`web`：添加 `Nginx` 仓库
  - 主机名不为`web`：不做任何处理

```
[root@m01 playbook]# cat when_yum.yml
- hosts: all
  tasks:
  - name: Add Nginx Yum Repository
    yum_repository:
      name: nginx
      description: Nginx Repository
      baseurl: http://nginx.org/packages/centos/7/$basearch/
      gpgcheck: no
    when: (ansible_hostname is match("web*"))


#当然when也可以使用and与or方式
#when: (ansible_hostname is match("web*")) or
#      (ansible_hostname is match("lb*"))
```

2.执行 `playbook`

```
[root@m01 playbook]# ansible-playbook when_yum.yml

PLAY [all]
*********************************************************************
**************************

TASK [Gathering Facts]
*********************************************************************
***************
ok: [172.16.1.7]
ok: [172.16.1.6]
ok: [172.16.1.8]
ok: [172.16.1.5]

#如果主机名不为web相关，则会跳过该tasks
```

```
TASK [Add Nginx Yum Repository]
*********************************************************************
******
skipping: [172.16.1.5]
skipping: [172.16.1.6]
ok: [172.16.1.8]
ok: [172.16.1.7]

PLAY RECAP
*********************************************************************
**************************
172.16.1.5                 : ok=1    changed=0    unreachable=0
failed=0
172.16.1.6                 : ok=1    changed=0    unreachable=0
failed=0
172.16.1.7                 : ok=2    changed=0    unreachable=0
failed=0
172.16.1.8                 : ok=2    changed=0    unreachable=0
failed=0
```

4.1.2 案例3-判断服务是否正常运行

- 判断 `httpd` 服务是否处于运行状态
  - 已运行：则重启服务
  - 未运行：则不做处理

1.通过 `register` 将命令执行结果保存至变量，然后通过 `when` 语句进行判断

```
[root@m01 playbook]# cat when_service.yml
- hosts: webservers
  tasks:
    - name: Check Httpd Server
      command: systemctl is-active httpd
      ignore_errors: yes
      register: check_httpd

    - name: debug outprint       #通过debug的var输出该变量的所有内容
      debug: var=check_httpd

    - name: Httpd Restart #如果check_httpd执行命令结果等于0，则执行重启
httpd，否则跳过
      service: name=httpd state=restarted
      when: check_httpd.rc == 0
```

2.执行 `playbook`

```
[root@m01 playbook]# ansible-playbook when_service.yml

PLAY [web]
*********************************************************************
**************************

TASK [Gathering Facts]
*********************************************************************
***************
ok: [172.16.1.8]
ok: [172.16.1.7]

TASK [Check Httpd Server]
*********************************************************************
************
fatal: [172.16.1.8]: FAILED! => {"changed": true, "cmd":
["systemctl", "is-active", "httpd"], "delta": "0:00:00.023433",
"end": "2019-01-31 03:17:23.781113", "msg": "non-zero return
code", "rc": 3, "start": "2019-01-31 03:17:23.757680", "stderr":
"", "stderr_lines": [], "stdout": "inactive", "stdout_lines":
["inactive"]}
...ignoring
changed: [172.16.1.7]


TASK [Httpd Restart]
*********************************************************************
*****************
skipping: [172.16.1.8]
changed: [172.16.1.7]

PLAY RECAP
*********************************************************************
***************************
172.16.1.7                 : ok=4    changed=2    unreachable=0
failed=0
172.16.1.8                 : ok=3    changed=1    unreachable=0
failed=0
```

4.2 loop 循环语句

在写 playbook 的时候发现了很多 task 都要重复引用某个相同的模块，比如一次启动10个服务，或者一次拷贝10个文件，如果按照传统的写法最少要写10次，这样会显得 playbook 很臃肿。如果使用循环的方式来编写 playbook，这样可以减少重复编写 task 带来的臃肿；

4.2.1 案例1-使用循环批量启动服务

®1.在没有使用循环的场景下，启动多个服务需要写多条 task 任务。

```
[root@m01 playbook]# cat loop-service.yml
- hosts: web
  tasks:
    - name: Installed Httpd Mariadb Package
      yum: name=httpd,mariadb state=latest

    - name: Start Httpd Server
      service: name=httpd state=started enabled=yes

    - name: Start Mariadb Server
      service: name=mariadb state=started enabled=yes
```

2.我们将如上的 playbook 修改为循环的方式，减少重复编写多条 task 任务。

```
[root@m01 playbook]# cat loop-service.yml
- hosts: web
  tasks:
    - name: Installed Httpd Mariadb Package
      yum: name=httpd,mariadb-server state=latest

    - name: Start Httpd Mariadb Server
      service: name={{ item }} state=started enabled=yes
      loop:
        - httpd
        - mariadb
```

3.执行 playbook

```
[root@m01 playbook]# ansible-playbook loop.yml

PLAY [web]
**********************************************************************
**************************
```

```
TASK [Gathering Facts]
*********************************************************************
***************
ok: [172.16.1.7]
ok: [172.16.1.8]

TASK [Installed Httpd Mariadb Package]
*********************************************************************
ok: [172.16.1.7]
ok: [172.16.1.8]

TASK [Start Httpd Mariadb Server]
*********************************************************************
****
ok: [172.16.1.7] => (item=httpd)
ok: [172.16.1.8] => (item=httpd)
ok: [172.16.1.8] => (item=mariadb)
ok: [172.16.1.7] => (item=mariadb)

TASK [Start Mariadb Server]
*********************************************************************
**********
ok: [172.16.1.7]
ok: [172.16.1.8]

PLAY RECAP
*********************************************************************
***************************
172.16.1.7                 : ok=4    changed=0    unreachable=0
 failed=0
172.16.1.8                 : ok=4    changed=0    unreachable=0
 failed=0
```

4.2.2 案例2-使用循环批量安装软件

1.批量安装软件

```
[root@m01 playbook]# cat loop-service-v2.yml
- hosts: web
  tasks:
    - name: Installed Httpd Mariadb Package
      yum: name={{ pack }} state=latest
      vars:
       pack:
         - httpd
         - mariadb-server
```

2.执行 `playbook`

```
[root@m01 playbook]# ansible-playbook loop-service-v2.yml

PLAY [web]
*********************************************************************
**************************

TASK [Gathering Facts]
*********************************************************************
***************
ok: [172.16.1.8]
ok: [172.16.1.7]

TASK [Installed Httpd Mariadb Package]
*********************************************************************
ok: [172.16.1.7]
ok: [172.16.1.8]

PLAY RECAP
*********************************************************************
**************************
172.16.1.7                 : ok=2    changed=0    unreachable=0
 failed=0
172.16.1.8                 : ok=2    changed=0    unreachable=0
 failed=0
```

4.2.3 案例3-使用循环批量创建用户

1.批量创建用户，使用 `key values` 字典的方式

```
[root@manager ~]# cat loop-user.yml
- hosts: web
  tasks:
    - name: Add Users
      user:
        name: {{ item.name }}
        groups: {{ item.groups }}
        state: present
      loop:
        - { name: 'testuser1', groups: 'bin' }
        - { name: 'testuser2', groups: 'root' }
```

2.执行 `playbook`

```
[root@m01 playbook]# ansible-playbook loop-user.yml

PLAY [web]
*********************************************************************
***************************

TASK [Gathering Facts]
*********************************************************************
***************
ok: [172.16.1.8]
ok: [172.16.1.7]

TASK [Add Users]
*********************************************************************
*********************
changed: [172.16.1.7] => (item={u'name': u'testuser1', u'groups':
u'bin'})
changed: [172.16.1.8] => (item={u'name': u'testuser1', u'groups':
u'bin'})
changed: [172.16.1.8] => (item={u'name': u'testuser2', u'groups':
u'root'})
changed: [172.16.1.7] => (item={u'name': u'testuser2', u'groups':
u'root'})

PLAY RECAP
*********************************************************************
***************************
172.16.1.7                 : ok=2    changed=1    unreachable=0
 failed=0
```

```
172.16.1.8                 : ok=2    changed=1    unreachable=0
 failed=0
```

4.2.4 案例4-使用循环批量拷贝文件

```
[root@manager ~]# cat loop-file.yml
- hosts: all
  tasks:
    - name: Configure Rsync Server
      copy: src={{ item.src }} dest=/etc/{{ item.dest }} mode={{
item.mode }}
      with_items:
        - {src: "rsyncd.conf", dest: "rsyncd.conf", mode: "0644"}
        - {src: "rsync.passwd", dest: "rsync.passwd", mode:
"0600"}
```

4.3.Handlers与Notify

`Handlers` 是一个触发器，同时是一个特殊的 `tasks`，它无法直接运行，它需要被 `tasks` 通知后才会运行。比如：`httpd` 服务配置文件发生变更，我们则可通过 `Notify` 通知给指定的 `handlers` 触发器，然后执行相应重启服务的操作，如果配置文件不发生变更操作，则不会触发 `Handlers` 任务的执行；

4.3.1 案例1-变更服务配置触发重启

1.使用 `Ansible` 的 `playbook` 部署 `httpd` 服务

```
[root@m01 ~]# cat webserver.yml
- hosts: web
  remote_user: root
#1.定义变量，在配置文件中调用
  vars:
    http_port: 8881

#2.安装httpd服务
  tasks:
    - name: Install Httpd Server
      yum: name=httpd state=present

#3.使用template模板，引用上面vars定义的变量至配置文件中
    - name: Configure Httpd Server
      template: src=./httpd.conf dest=/etc/httpd/conf/httpd.conf
      notify:   #调用名称为Restart Httpd Server的handlers(可以写多个)
        - Restart Httpd Server
```

```
#4.启动Httpd服务
    - name: Start Httpd Server
      service: name=httpd state=started enabled=yes

#5.如果配置文件发生变化会调用该handlers下面的对应名称的task
  handlers:
    - name: Restart Httpd Server
      service: name=httpd state=restarted
```

2.只有当我们修改配置文件才会触发 `handlers`

```
[root@m01 playbook]# ansible-playbook webserver.yml

PLAY [web]
*********************************************************************
**************************

TASK [Gathering Facts]
*********************************************************************
***************
ok: [172.16.1.8]
ok: [172.16.1.7]

TASK [Install Httpd Server]
*********************************************************************
**********
ok: [172.16.1.8]
ok: [172.16.1.7]

TASK [Configure Httpd Server]
*********************************************************************
********
changed: [172.16.1.8]
changed: [172.16.1.7]

TASK [Start Httpd Server]
*********************************************************************
************
ok: [172.16.1.8]
ok: [172.16.1.7]

RUNNING HANDLER [Restart Httpd Server]
*********************************************************************
changed: [172.16.1.8]
```

```
changed: [172.16.1.7]

PLAY RECAP
*********************************************************************
***************************
172.16.1.7                 : ok=5    changed=2    unreachable=0
failed=0
172.16.1.8                 : ok=5    changed=2    unreachable=0
failed=0
```

4.3.3 Handlers注意事项与说明

- `handlers` 注意事项
  - 1.无论多少个 `task` 通知了相同的 `handlers`， `handlers` 仅会在所有 `tasks` 结束后运行一次。
  - 2.只有 `task` 发生改变了才会通知 `handlers` ，没有改变则不会触发 `handlers`
  - 3.不能使用 `handlers` 替代 `tasks`、因为 `handlers` 是一个特殊的`tasks`

4.4 tags 任务标签

默认情况下， `Ansible` 在执行一个 `playbook` 时，会执行 `playbook` 中所有的任务。而标签功能是用来指定要运行 `playbook` 中的某个特定的任务；

- 1.为 `playbook` 添加标签的方式有如下几种：
  - 对一个 `task` 打一个标签
  - 对一个 `task` 打多个标签
  - 对多个 `task` 打一个标签
- 2.`task`打完标签使用的几种方式
  - `-t` 执行指定tag标签对应的任务
  - `--skip-tags` 执行除 `--skip-tags` 标签之外的所有任务

4.4.1 案例1-指定执行某个tags

- 使用 `-t` 执行指定的 `tags` 标签对应的任务

```
[root@manager ~]# cat nfs.yml
---
- hosts: nfs
  remote_user: root
  tasks:
    - name: Install Nfs Server
      yum: name=nfs-utils state=present
```

```
      tags:
        - install_nfs
        - install_nfs-server

    - name: Service Nfs Server
      service: name=nfs-server state=started enabled=yes
      tags: start_nfs-server
```

2.执行 `playbook`

```
[root@manager ~]# ansible-playbook  f10.yml

PLAY [all]
*********************************************************************
*********************************************************************

TASK [Gathering Facts]
*********************************************************************
*********************************************************
ok: [172.16.1.31]

TASK [Install Nfs Server]
*********************************************************************
*********************************************
ok: [172.16.1.31]

TASK [Service Nfs Server]
*********************************************************************
*********************************************
ok: [172.16.1.31]

PLAY RECAP
*********************************************************************
*********************************************************************
172.16.1.31                : ok=3    changed=0    unreachable=0
   failed=0
```

3.使用 `-t` 指定 `tags` 标签对应的任务， 多个 `tags` 使用逗号隔开即可

```
[root@manager ~]# ansible-playbook -t install_nfs-server nfs.yml

PLAY [nfs]
*********************************************************************
*********************************************************************

TASK [Gathering Facts]
*********************************************************************
*********************************************************
ok: [172.16.1.31]

TASK [Install Nfs Server]
*********************************************************************
*********************************************
ok: [172.16.1.31]

PLAY RECAP
*********************************************************************
*********************************************************************
172.16.1.31                : ok=2    changed=0    unreachable=0
   failed=0
```

4.4.2 案例2-指定排除某个tags

- 使用 `--skip-tags` 排除不执行的 `tags`

```
[root@manager ~]# ansible-playbook --skip-tags install_nfs-server
nfs.yml

PLAY [all]
*********************************************************************
*********************************************************************

TASK [Gathering Facts]
*********************************************************************
*********************************************************
ok: [172.16.1.31]

TASK [Service Nfs Server]
*********************************************************************
*********************************************
ok: [172.16.1.31]

PLAY RECAP
*********************************************************************
*********************************************************************
172.16.1.31                : ok=2    changed=0    unreachable=0
   failed=0
```

4.5 include任务复用

有时，我们发现大量的 `Playbook` 内容需要重复编写，各 `Tasks` 之间功能需相互调用才能完成各自功能，`Playbook` 庞大到维护困难，这时我们需要使用`include`
比如：A项目需要用到重启 `httpd`，B项目需要用到，重启 `httpd`，那么我们可以使用 `Include`来减少重复编写。

4.5.1 案例1-多个项目调用相同task

1.编写 `restart_httpd.yml` 文件

```
#注意这是一个tasks所有没有play的任何信息
[root@ansible project1]# cat restart_httpd.yml
- name: Restart Httpd Server
  service: name=httpd state=restarted
```

2.`A Project` 的 `playbook` 如下

```
[root@ansible project1]# cat a_project.yml
- hosts: webserver
  tasks:
    - name: A Project command
      command: echo "A"

    - name: Restart httpd
      include: restart_httpd.yml
```

3.`B Project` 的 `playbook` 如下

```
[root@ansible project1]# cat b_project.yml
- hosts: webserver
  tasks:
    - name: B Project command
      command: echo "B"

    - name: Restart httpd
      include: restart_httpd.yml
```

4.`A Project` 和 `B Project` 执行后的测试结果如下

```
[root@ansible project1]# ansible-playbook a_project.yml
PLAY [webserver]
*********************************************************************
************************************

TASK [Gathering Facts]
*********************************************************************
**********************************
ok: [172.16.1.8]
ok: [172.16.1.7]

TASK [A Project command]
*********************************************************************
*******************************
changed: [172.16.1.7]
changed: [172.16.1.8]

TASK [Restart Httpd Server]
*********************************************************************
***************************
changed: [172.16.1.8]
changed: [172.16.1.7]
```

```
PLAY RECAP
*********************************************************************
*********************************************
172.16.1.7                 : ok=3    changed=2    unreachable=0
 failed=0    skipped=0    rescued=0    ignored=0
172.16.1.8                 : ok=3    changed=2    unreachable=0
 failed=0    skipped=0    rescued=0    ignored=0
```

4.5.2 案例2-Inlcude结合tags应用

- 通过指定标签 `tags`，来说明是安装 `tomcat8` 还是 `tomcat9`
  - 1.准备入口 `main.yml` 文件，然后包含 `install_tomcat8.yml` 以及 `install_tomcat9.yml`
  - 2.在执行 `main.yml` 时，需要通过 `--tags` 指明要安装的版本

1.编写 `main.yml` 入口文件

```
[root@ansible ~]# cat main.yml
- hosts: localhost
  tasks:
    - name: Installed Tomcat8 Version
      include: install_tomcat8.yml
      tags: tomcat8

    - name: Installed Tomcat9 Version
      include: install_tomcat9.yml
      tags: tomcat9
```

2.编写 `install_tomcat8.yml`

```
[root@ansible ~]# cat install_tomcat8.yml
- hosts: webservers
  vars:
    - tomcat_version: 8.5.63
    - tomcat_install_dir: /usr/local

  tasks:
    - name: Install jdk1.8
      yum:
        name: java-1.8.0-openjdk
        state: present

    - name: Download tomcat
```

```
      get_url:
        url: http://mirrors.hust.edu.cn/apache/tomcat/tomcat-
8/v{{ tomcat_version }}/bin/apache-tomcat-{{ tomcat_version
}}.tar.gz
        dest: /tmp

    - name: Unarchive tomcat-{{ tomcat_version }}.tar.gz
      unarchive:
        src: /tmp/apache-tomcat-{{ tomcat_version }}.tar.gz
        dest: "{{ tomcat_install_dir }}"
        copy: no

    - name: Start tomcat
      shell: cd {{ tomcat_install_dir }} &&
             mv apache-tomcat-{{ tomcat_version }} tomcat8 &&
             cd tomcat8/bin && nohup ./startup.sh &
```

3.编写 `install_tomcat9.yml`

```
[root@ansible ~]# cat install_tomcat9.yml
- hosts: webservers
  vars:
    - tomcat_version: 9.0.43
    - tomcat_install_dir: /usr/local

  tasks:
    - name: Install jdk1.8
      yum:
        name: java-1.8.0-openjdk
        state: present

    - name: Download tomcat
      get_url:
        url: http://mirrors.hust.edu.cn/apache/tomcat/tomcat-
9/v{{ tomcat_version }}/bin/apache-tomcat-{{ tomcat_version
}}.tar.gz
        dest: /tmp

    - name: Unarchive tomcat-{{ tomcat_version }}.tar.gz
      unarchive:
        src: /tmp/apache-tomcat-{{ tomcat_version }}.tar.gz
        dest: "{{ tomcat_install_dir }}"
        copy: no

    - name: Start tomcat
```

```
      shell: cd {{ tomcat_install_dir }} &&
             mv apache-tomcat-{{ tomcat_version }} tomcat9 &&
             cd tomcat9/bin && nohup ./startup.sh &
```

4.执行 `main.yml` 文件，然后通过 `--tags` 执行对应的版本

```
[root@ansible ~]# ansible-playbook main.yml --tags tomcat8
[root@ansible ~]# ansible-playbook main.yml --tags tomcat9
```

4.7 Playbook异常处理

4.7.1 案例1-Playbook错误忽略

在 `playbook` 执行的过程中，难免会遇到一些错误。由于 `playbook` 遇到错误后，不会执行之后的任务，不便于调试，此时，可以使用 `ignore_errors` 来暂时忽略错误，使得 `playbook` 继续执行。

1.编写 `playbook`，当有 `task` 执行失败则会立即终止后续 `task` 运行

```
[root@manager ~]# cat ignore.yml
---
- hosts: all
  remote_user: root
  tasks:
    - name: Ignore False
      command: /bin/false
      ignore_errors: yes

    - name: touch new file
      file: path=/tmp/oldxu_ignore state=touch
```

2.执行 `playbook`，会报错，后续的任务也没有执行。

```
[root@m01 playbook]# ansible-playbook ignore.yml

PLAY [web]
*********************************************************************
***************************

TASK [Gathering Facts]
*********************************************************************
***************
ok: [172.16.1.7]
```

```
ok: [172.16.1.8]

TASK [Ignore False]
*********************************************************************
******************
fatal: [172.16.1.8]: FAILED! => {"changed": true, "cmd":
["/bin/false"], "delta": "0:00:00.021502", "end": "2019-01-31
20:27:51.206530", "msg": "non-zero return code", "rc": 1,
"start": "2019-01-31 20:27:51.185028", "stderr": "",
"stderr_lines": [], "stdout": "", "stdout_lines": []}
fatal: [172.16.1.7]: FAILED! => {"changed": true, "cmd":
["/bin/false"], "delta": "0:00:00.022049", "end": "2019-01-31
20:27:51.206340", "msg": "non-zero return code", "rc": 1,
"start": "2019-01-31 20:27:51.184291", "stderr": "",
"stderr_lines": [], "stdout": "", "stdout_lines": []}
    to retry, use: --limit @/etc/ansible/playbook/ignore.retry

PLAY RECAP
*********************************************************************
***************************
172.16.1.7                 : ok=1    changed=0    unreachable=0
 failed=1
172.16.1.8                 : ok=1    changed=0    unreachable=0
 failed=1
```

3.此时我们给对应的 `task` 任务添加忽略错误

```
[root@m01 playbook]# cat ignore.yml
- hosts: web
  tasks:
    - name: Ignore False
      command: /bin/false #该命令会返回非0,代表命令执行失败
      ignore_errors: yes  #忽略错误

    - name: touch new file
      file: path=/tmp/oldxu_ignore state=touch
```

4.再次执行 `playbook` 如果碰到指定的 `tasks` 错误，会自动忽略，继续执行剩下的 `tasks`

```
[root@m01 playbook]# ansible-playbook ignore.yml

PLAY [web]
*********************************************************************
***************************
```

```
TASK [Gathering Facts]
*********************************************************************
***************
ok: [172.16.1.8]
ok: [172.16.1.7]

TASK [Ignore False]
*********************************************************************
******************
fatal: [172.16.1.7]: FAILED! => {"changed": true, "cmd":
["/bin/false"], "delta": "0:00:00.019128", "end": "2019-01-31
20:30:45.710746", "msg": "non-zero return code", "rc": 1,
"start": "2019-01-31 20:30:45.691618", "stderr": "",
"stderr_lines": [], "stdout": "", "stdout_lines": []}
...ignoring
fatal: [172.16.1.8]: FAILED! => {"changed": true, "cmd":
["/bin/false"], "delta": "0:00:00.020302", "end": "2019-01-31
20:30:45.715142", "msg": "non-zero return code", "rc": 1,
"start": "2019-01-31 20:30:45.694840", "stderr": "",
"stderr_lines": [], "stdout": "", "stdout_lines": []}
...ignoring

TASK [touch new file]
*********************************************************************
***************
changed: [172.16.1.8]
changed: [172.16.1.7]

PLAY RECAP
*********************************************************************
***************************
172.16.1.7                 : ok=3    changed=2    unreachable=0
 failed=0
172.16.1.8                 : ok=3    changed=2    unreachable=0
 failed=0
```

4.7.1 案例2-task执行失败强制调用handlers

通常情况下，当 `task` 失败后，`play` 将会终止，任何在前面已经被 `tasks notify` 的 `handlers` 都不会被执行。如果你在 `play` 中设置了 `force_handlers: yes` 参数，被通知的 `handlers` 就会被强制执行。(有些特殊场景可能会使用到)

1.编写 `playbook`

```
[root@m01 playbook]# cat igno_handlers.yml
- hosts: web
  force_handlers: yes #强制调用handlers

  tasks:
    - name: Touch File
      file: path=/tmp/bgx_handlers state=touch
      notify: Restart Httpd Server

    - name: Installed Packages
      yum: name=sb state=latest

  handlers:
    - name: Restart Httpd Server
      service: name=httpd state=restarted
```

2.执行 `playbook`

```
[root@m01 playbook]# ansible-playbook igno_handlers.yml

PLAY [web]
*********************************************************************
***************************

TASK [Gathering Facts]
*********************************************************************
***************
ok: [172.16.1.8]
ok: [172.16.1.7]

TASK [Touch File]
*********************************************************************
********************
changed: [172.16.1.8]
changed: [172.16.1.7]

TASK [Installed Packages]
*********************************************************************
************
fatal: [172.16.1.7]: FAILED! => {"changed": false, "msg": "No
package matching 'sb' found available, installed or updated",
"rc": 126, "results": ["No package matching 'sb' found available,
installed or updated"]}
```

```
fatal: [172.16.1.8]: FAILED! => {"changed": false, "msg": "No
package matching 'sb' found available, installed or updated",
"rc": 126, "results": ["No package matching 'sb' found available,
installed or updated"]}

#前者task报错，也不影响handlers的调用
RUNNING HANDLER [Restart Httpd Server]
********************************************************************
changed: [172.16.1.8]
changed: [172.16.1.7]
    to retry, use: --limit
@/etc/ansible/playbook/igno_handlers.retry

PLAY RECAP
********************************************************************
****************************
172.16.1.7                 : ok=3    changed=2    unreachable=0
 failed=1
172.16.1.8                 : ok=3    changed=2    unreachable=0
 failed=1
```

4.7.2 案例3-控制Tasks报告状态为OK

1.编辑 `playbook`

```
[root@m01 playbook]# cat change.yml
- hosts: web
  tasks:

#获取系统httpd服务启动状态,将其结果写入Httpd_Port变量中
    - name: Get Httpd Server Port
      shell: netstat -lntp|grep httpd
      register: Httpd_Port

#输出Httpd_Port变量中的内容
    - name: Out Httpd Server Status
      debug: msg={{ Httpd_Port.stdout_lines }}
      ignore_errors: yes
```

2.执行 `playbook` 会发现第一个 `task` 运行 `shell` 模块报告的改变，即使它没有真正的在远端系统做出改变，如果你一直运行，它会一直处在改变状态。

```
[root@m01 playbook]# ansible-playbook change.yml
```

```
PLAY [web]
*********************************************************************
**************************

TASK [Gathering Facts]
*********************************************************************
***************
ok: [172.16.1.7]
ok: [172.16.1.8]

TASK [Get Httpd Server Port]
*********************************************************************
*********
changed: [172.16.1.8]
changed: [172.16.1.7]

TASK [Out Httpd Server Status]
*********************************************************************
*******
ok: [172.16.1.7] => {
    "msg": [
        "tcp6       0      0 :::80                   :::*
           LISTEN      2486/httpd          "
    ]
}
ok: [172.16.1.8] => {
    "msg": [
        "tcp6       0      0 :::80                   :::*
           LISTEN      12600/httpd         "
    ]
}

PLAY RECAP
*********************************************************************
***************************
172.16.1.7                 : ok=3    changed=1    unreachable=0
 failed=0
172.16.1.8                 : ok=3    changed=1    unreachable=0
 failed=0
```

3.`shell` 任务不应该每次都报告 `changed` 状态，因为它没有在被管理主机执行后发生变化。添加 `changed_when: false` 来抑制这个改变

```
[root@m01 playbook]# cat change.yml
- hosts: web
  tasks:

    - name: Get Httpd Server Port
      shell: netstat -lntp|grep httpd
      register: Httpd_Port
      changed_when: false   #该task不发生changed提示

    - name: Out Httpd Server Status
      debug: msg={{ Httpd_Port.stdout_lines }}
      ignore_errors: yes
```

4.再次执行 `playbook`

```
[root@m01 playbook]# ansible-playbook change.yml

PLAY [web]
*********************************************************************
**************************

TASK [Gathering Facts]
*********************************************************************
***************
ok: [172.16.1.8]
ok: [172.16.1.7]

TASK [Get Httpd Server Port]
*********************************************************************
*********
ok: [172.16.1.7]
ok: [172.16.1.8]

TASK [Out Httpd Server Status]
*********************************************************************
*******
ok: [172.16.1.7] => {
    "msg": [
        "tcp6       0      0 :::80                   :::*
           LISTEN      2486/httpd          "
    ]
}
ok: [172.16.1.8] => {
    "msg": [
```

```
        "tcp6       0      0 :::80                   :::*
           LISTEN      12600/httpd         "
    ]
}

PLAY RECAP
*********************************************************************
***************************
172.16.1.7                 : ok=3    changed=0    unreachable=0
failed=0
172.16.1.8                 : ok=3    changed=0    unreachable=0
failed=0
```

4.7.3 案例4-changed_when检查任务结果

```
[root@m01 project2]# cat changed_when.yml
- hosts: webservers
  tasks:
    - name: configure httpd server
      template: src=./httpd.conf.j2
dest=/etc/httpd/conf/httpd.conf
      notify: Restart Httpd Server

    - name: Check HTTPD
      shell: /usr/sbin/httpd -t
      register: httpd_check
      changed_when:
        - httpd_check.stdout.find('OK') #查找变量返回的结果是否有ok，如
不存在则终止该tasks
        - false #被管理主机没有发生变化状态则为OK

    - name: start httpd server
      service: name=httpd state=started enabled=yes

  handlers:
    - name: Restart Httpd Server
      systemd: name=httpd state=restarted
```

编写 `Nginx` 服务的 `playbook` 配置管理，该 `playbook` 需要具备修改配置文件自动检查功能?

```
[root@ansible project1]# cat f21.yml
- hosts: webserver
  tasks:
```

```
    - name: Install Nginx Server
      yum: name=nginx state=present

    - name: Configure Nginx Server
      template: src=./nginx.conf.j2 dest=/etc/nginx/nginx.conf
      notify: Restart Nginx Server

    - name: Check Nginx Server
      shell: /usr/sbin/nginx -t
      register: check_nginx
      changed_when:
        - check_nginx.stdout.find('successful')
        - false

    - name: Start Nginx Server
      service: name=nginx state=started enabled=yes

  handlers:
    - name: Restart Nginx Server
      systemd: name=nginx state=restarted
```

5.Ansible Advanced

5.1 Ansible Jinja2

5.1.1 什么是jinja2

`Jinja2` 是 `Python` 的全功能模板引擎
`Ansible` 需要使用 `Jinja2` 模板来修改被管理主机的配置文件。

- 场景1：给10台主机装上Nginx服务，但是要求每台主机的端口都不一样，如何解决?
- 场景2:

5.1.2 Ansible如何使用jinja2

`ansible` 使用 `jinja2` 模板需要借助 `template` 模块实现，那 `template` 模块是用来做什么的?
`template` 模块和 `copy` 模块完全一样，都是拷贝文件至远程主机，区别在于 `template` 模块会解析要拷贝的文件中变量的值，而 `copy` 则是原封不动的将文件拷贝至被控端。

5.1.3 jinja模板基本语法

- 1）要想在配置文件中使用`jinj2`，`playbook`中的`tasks`必须使用`template`模块
- 2）配置文件里面使用变量，比如`{{ PORT }}`或使用`{{ facts 变量 }}`

5.1.4 jinja模板逻辑关系

1.循环表达式

```
{% for i in EXPR %}...{% endfor %}
```

2.判断表达式

```
{% if  EXPR %}...{% elif EXPR %}...{% endif %}
```

3.注释

```
{# COMMENT #}
```

5.1.5 jinja模板示例

1.使用`Playbook`推送文件

```
[root@m01 playbook]# cat jinja2.yml
- hosts: web
  tasks:
    - name: Copy Template File /etc/motd
      template: src=./motd.j2 dest=/etc/motd
```

2.准备`motd.j2`文件

```
[root@m01 playbook]# cat motd.j2
Welcome to {{ ansible_hostname }}
This system total Memory is: {{ ansible_memtotal_mb }} MB
This system free Memory is: {{ ansible_memfree_mb }} MB
```

3.执行`playbook`

```
[root@m01 playbook]# ansible-playbook jinja2.yml
```

```
PLAY [web]
*********************************************************************
**************************

TASK [Gathering Facts]
*********************************************************************
***************
ok: [172.16.1.8]
ok: [172.16.1.7]

TASK [Copy Template File /etc/motd]
*********************************************************************
**
changed: [172.16.1.8]
changed: [172.16.1.7]

PLAY RECAP
*********************************************************************
***************************
172.16.1.7                 : ok=2    changed=1    unreachable=0
failed=0
172.16.1.8                 : ok=2    changed=1    unreachable=0
failed=0
```

4.检查执行后的状态

```
[root@m01 playbook]# ssh root@172.16.1.7
Welcome to web01
This system total Memory is: 470 MB
This system free Memory is: 193 MB

[root@m01 playbook]# ssh root@172.16.1.8
Welcome to web02
This system total Memory is: 470 MB
This system free Memory is: 198 MB
```

上面的例子展示了如何使用 `facts` 变量，当 `playbook` 被执行后，`ansible_hostname` 和 `ansible_memtotal_mb` 将会被替换成被管理主机上搜集的 `facts` 变量的值

5.1.5 案例1-Jinja2管理Nginx

- `ansible` 使用 `jinja2` 的 `for` 循环表达式渲染出 `nginx` 负载均衡的配置文件

1.使用 `Playbook` 推送文件

```
[root@m01 playbook]# cat proxy.yml
- hosts: web
  vars:
    http_port: 80
    server_name: www.bgx.com

  tasks:
    - name: Copy Template Nginx Configure
      template: src=blog.conf.j2
dest=/etc/nginx/conf.d/blog.bgx.com.conf
      notify: Reload Nginx Server

  handlers:
     - name: Reload Nginx Server
       service: name=nginx state=reloaded
```

2.准备 `blog.conf.j2` 配置文件

```
[root@manager playbook-cluster]# cat
conf/proxy_ansible.oldxu.com.conf.j2
upstream ansible_php {
{% for i in groups['webservers'] %}
    server {{i}}:80;
{% endfor %}


}

server {
    listen 80;
    server_name ansible.oldxu.com;

    location / {
        proxy_pass http://ansible_php;
        proxy_set_header Host $http_hosts;
    }
}
```

3.执行 `playbook`

```
[root@m01 playbook]# ansible-playbook proxy.yml

PLAY [web]
*********************************************************************
**************************

TASK [Gathering Facts]
*********************************************************************
***************
ok: [172.16.1.7]
ok: [172.16.1.8]

TASK [Copy Template Nginx Configure]
*********************************************************************
*
changed: [172.16.1.7]
changed: [172.16.1.8]

PLAY RECAP
*********************************************************************
****************************
172.16.1.7                 : ok=2    changed=1    unreachable=0
 failed=0
172.16.1.8                 : ok=2    changed=1    unreachable=0
 failed=0
```

4.检查 `jinja` 模板渲染出来的配置文件

```
[root@web02 ~]# cat /etc/nginx/conf.d/blog.bgx.com.conf
upstream www.bgx.com {
#设置变量，并进行循环赋值,渲染配置
      server 172.16.1.7:80;
      server 172.16.1.8:80;
      server 172.16.1.9:80;
  }

server {
    listen  80;
    server_name www.bgx.com;
    location / {
        proxy_pass http://www.bgx.com;
        proxy_set_header Host $http_host;
    }
```

```
}
```

5.1.6 案例2-Jinja2管理Keepalived

- ansible使用jinja2的if判断表达式渲染出keepalived的Master和Slave的配置文件。并推送至lb组，实现方案如下：
  - 1.设定 `Inventory` 中 `host_vars` 然后根据不同主机设定不同的变量
  - 2.在 `Playbook` 中使用 `when` 判断主机名称，然后分发不同的配置文件。
  - 3.使用 `jinja2` 的方式渲染出不同的配置文件。

1.使用 `playbook` 推送 `keeplaived` 配置文件

```
[root@m01 playbook]# cat keepalived.yml
- hosts: lb
  tasks:
    - name: Copy Template Keepalived Configure
      template: src=keepalived.j2
dest=/etc/keepalived/keepalived.conf
      notify: Restart Keepalived Server

  handlers:
     - name: Restart Keepalived Server
       service: name=keepalived state=restarted
```

2.准备 `keepalived.j2` 配置文件

```
[root@m01 playbook]# cat keepalived.j2
global_defs {
    router_id {{ ansible_fqdn }}
}
vrrp_instance VI_1 {
{% if ansible_fqdn  == 'lb01' %}
#如果主机名为lb01则使用如下配置
    state MASTER
    priority 150
{% elif ansible_fqdn  == 'lb02' %}
#如果主机名为lb02则使用如下配置
    state Backup
    priority 100
{% endif %}
#相同配置
    interface eth0
    virtual_router_id 51
    advert_int 1
```

```
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    virtual_ipaddress {
        10.0.0.3
    }
}
```

3.执行 `playbook`

```
[root@m01 playbook]# ansible-playbook keepalived.yml

PLAY [lb]
*********************************************************************
***************************

TASK [Gathering Facts]
*********************************************************************
***************
ok: [172.16.1.5]
ok: [172.16.1.6]

TASK [Copy Template Keepalived Configure]
***************************************************************
changed: [172.16.1.6]
changed: [172.16.1.5]

RUNNING HANDLER [Restart Keepalived Server]
*********************************************************
changed: [172.16.1.6]
changed: [172.16.1.5]

PLAY RECAP
*********************************************************************
***************************
172.16.1.5                 : ok=3    changed=2    unreachable=0
failed=0
172.16.1.6                 : ok=3    changed=2    unreachable=0
failed=0
```

4.检查 `lb01 Master` 的 `keepalived` 配置文件

```
[root@lb01 ~]# cat /etc/keepalived/keepalived.conf
 global_defs {
```

```
    router_id lb01
}
vrrp_instance VI_1 {
#如果主机名为lb01则使用如下配置
    state MASTER
    interface eth0
    virtual_router_id 51
    priority 150

  #相同配置
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    virtual_ipaddress {
        10.0.0.3
    }
}
```

5.检查 `lb02 Backup` 的 `keepalived` 配置文件

```
[root@lb02 ~]# cat /etc/keepalived/keepalived.conf
global_defs {
    router_id lb02
}
vrrp_instance VI_1 {
    #如果主机名为lb02则使用如下配置
    state Backup
    interface eth0
    virtual_router_id 51
    priority 100
#相同配置
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    virtual_ipaddress {
        10.0.0.3
    }
}
```

5.2 Ansible Roles

5.2.1 Roles基本概述

前面已经学过 `tasks` 和 `handlers`，那怎样组织 `playbook` 才是最好的方式呢？简单的回答就是：使用 `Roles`
`Roles` 基于一个已知的文件结构，去自动的加载 `vars，tasks` 以及 `handlers` 以便 `playbook` 更好的调用。
`roles` 相比 `playbook` 的结构更加的清晰有层次，但 `roles` 显然要比 `playbook` 更加复杂难理解！
比如：我们无论安装什么软件都会安装时间同步服务，那么每个 `playbook` 都要编写时间同步服务的 `task`。此时我们可以将时间同步服务 `task` 任务编写好，等到需要使用的时候进行调用就行了。

5.2.2 Roles目录结构

`roles` 官方目录结构，必须按如下方式定义。在每个目录中必须有 `main.yml` 文件，这些属于强制要求

```
[root@m01 ~]# cd /etc/ansible/roles
[root@m01 roles]# mkdir
{nfs,rsync,web}/{vars,tasks,templates,handlers,files,meta} -p
[root@m01 roles]# tree
.
├── nfs                 #角色名称
|   ├── files           #存放文件
|   ├── handlers        #触发任务
|   ├── tasks           #具体任务
|   ├── templates       #模板文件
|   └── vars            #定义变量
|   └── meta            #依赖关系
```

5.2.3 Roles依赖关系

- `roles` 允许在使用时自动引入其他 `role`，`role` 依赖关系存储在 `meta/main.yml` 文件中。
    - 例如: 安装 `wordpress` 项目时:
        - 1.需要先确保 `nginx` 与 `php-fpm` 的 `role` 都能正常运行
        - 2.然后在 `wordpress` 的 `role` 中定义，依赖关系
        - 3.依赖的 `role` 有 `nginx` 以及 `php-fpm`

```
#wordpress依赖nginx与php-fpm的role
[root@m01 playbook]# cat /root/roles/wordpress/meta/main.yml
---
dependencies:
  - { role: nginx }
  - { role: php-fpm }
```

`wordpress` 的 `role` 会先执行 `nginx、php-fpm` 的 `role`，最后在执行`wordpress` 本身

5.2.4 Roles案例实战

- Roles三步走:
  - 1.创建 `roles` 目录结构，手动创建或使用 `ansible-galaxy init test roles`
  - 2.编写 `roles` 的功能，也就是 `tasks`
  - 3.最后 `playbook` 引用 `roles` 编写好的 `tasks`

5.2.4.2 案例1-Roles部署NFS

1.目录结构如下

```
[root@m01 roles]# tree /etc/ansible/roles

├── group_vars
│   └── all
├── hosts
├── nfs
│   ├── files
│   ├── handlers
│   │   └── main.yml
│   ├── tasks
│   │   └── main.yml
│   ├── templates
│   │   └── exports
│   └── vars
├── site.yml
```

2.定义 `roles` 主机清单

```
[root@m01 roles]# cat /etc/ansible/roles/hosts
[nfs]
172.16.1.31
```

3.查看 `nfs` 角色的 `tasks` 任务

```
[root@m01 roles]# cat /etc/ansible/roles/nfs/tasks/main.yml
- name: Install Nfs-Server
  yum: name=nfs-utils state=present

- name: Configure Nfs-Server
  template: src=exports dest=/etc/exports
  notify: Restart Nfs-Server

- name: Create Directory Data
  file: path={{ share_dir }} state=directory owner=www group=www
mode=0755

- name: Start Nfs-Server
  service: name=nfs state=started enabled=yes
```

4.查看 `nfs` 角色的 `handlers`

```
[root@m01 roles]# cat /etc/ansible/roles/nfs/handlers/main.yml
- name: Restart Nfs-Server
  service: name=nfs state=restarted
```

5.查看 `nfs` 角色的 `files` 目录

```
[root@m01 roles]# cat /etc/ansible/roles/nfs/templates/exports
{{ share_dir }} {{ share_ip }}
(rw,sync,all_squash,anonuid=666,anongid=666)
```

6.`nfs` 对应的变量定义

```
[root@m01 roles]# cat /etc/ansible/roles/group_vars/all
#nfs
share_dir: /data
share_ip: 172.16.1.31
```

7.在 `playbook` 中使用 `role`，指定 `nfs` 主机组，执行 `nfs` 服务的`roles`

```
[root@m01 roles]# cat /etc/ansible/roles/site.yml
- hosts: nfs
```

```
  remote_user: root
  roles:
    - nfs

[root@m01 roles]# ansible-playbook -i hosts site.yml
PLAY [nfs]
*********************************************************************
**************************

TASK [Gathering Facts]
*********************************************************************
***************
ok: [172.16.1.31]

TASK [nfs : Install Nfs-Server]
*********************************************************************
******
ok: [172.16.1.31]

TASK [nfs : Configure Nfs-Server]
*********************************************************************
****
ok: [172.16.1.31]

TASK [nfs : Create Directory Data]
*********************************************************************
***
ok: [172.16.1.31]

TASK [nfs : Start Nfs-Server]
*********************************************************************
********
ok: [172.16.1.31]

PLAY RECAP
*********************************************************************
**************************
172.16.1.31                : ok=5    changed=0    unreachable=0
 failed=0
```

5.2.4.3 案例2-Roles部署Memcached

1.目录结构如下

```
[root@m01 memcached]# cd /etc/ansible/roles/
[root@m01 memcached]# tree memcached/
.
├── tasks
│   ├── main.yml
│   ├── start.yml
│   ├── template.yml
│   └── yum.yml
└── templates
    └── memcached.j2
```

2.查看 `memcached` 的 `tasks`

```
[root@m01 memcached]# cat tasks/main.yml
- include: yum.yml
- include: template.yml
- include: start.yml

[root@m01 ~]# cat tasks/yum.yml
- name: install memcached package
  yum: name=memcached

[root@m01 ~]# cat tasks/template.yml
- name: Copy memcahed conf
  template: src=memcached.j2 dest=/etc/sysconfig/memcached

[root@m01 ~]# cat templates/memcached.j2
PORT="11211"
USER="memcached"
MAXCONN="{{ ansible_memtotal_mb//4 }}"
CACHESIZE="64"
OPTIONS=""

[root@m01 ~]# cat tasks/start.yml
- name: start memcached
  service: name=memcached state=started enabled=yes
```

3.在 `playbook` 中使用 `role`，执行 `memcached` 服务的 `roles`

```
[root@m01 ~]# cat site.yml
- hosts: "{{ host }}"
  remote_user: root
  roles:
    - role: memcached

# 执行playbook
[root@m01 ~]# ansible-playbook site.yml -e "host=10.0.0.51"
```