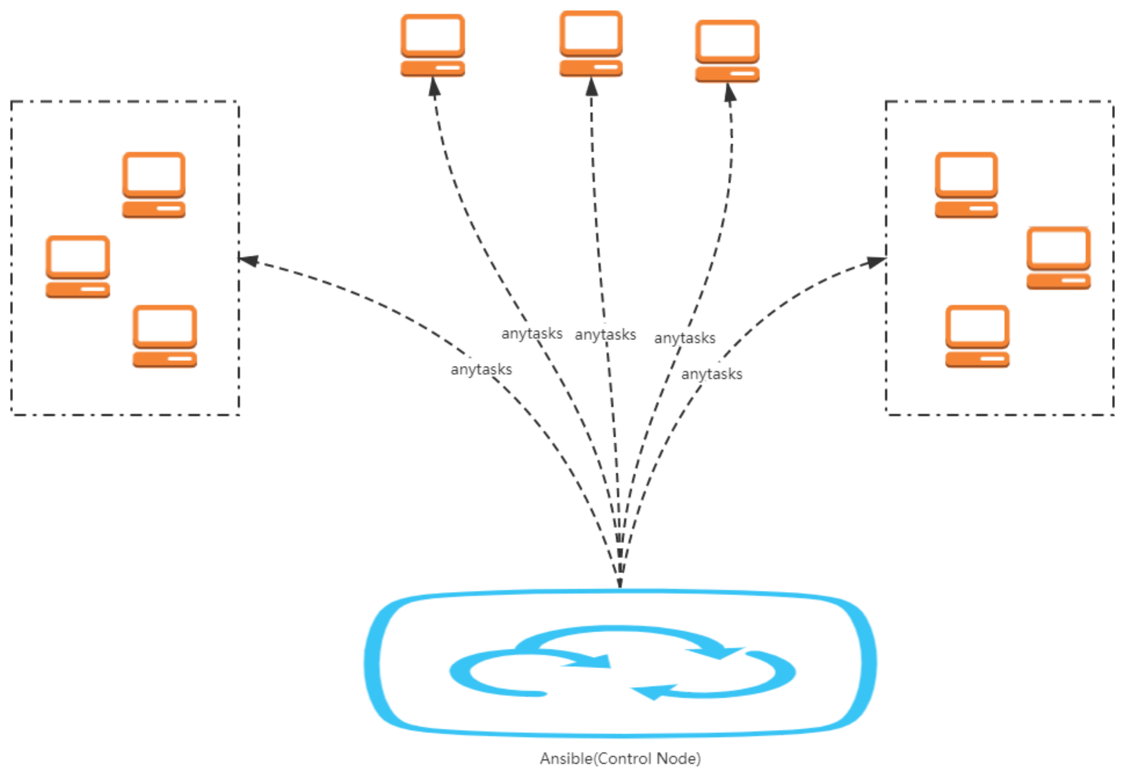
Ansible 自动化运维管理

今日主题：《Ansible 自动化运维管理》

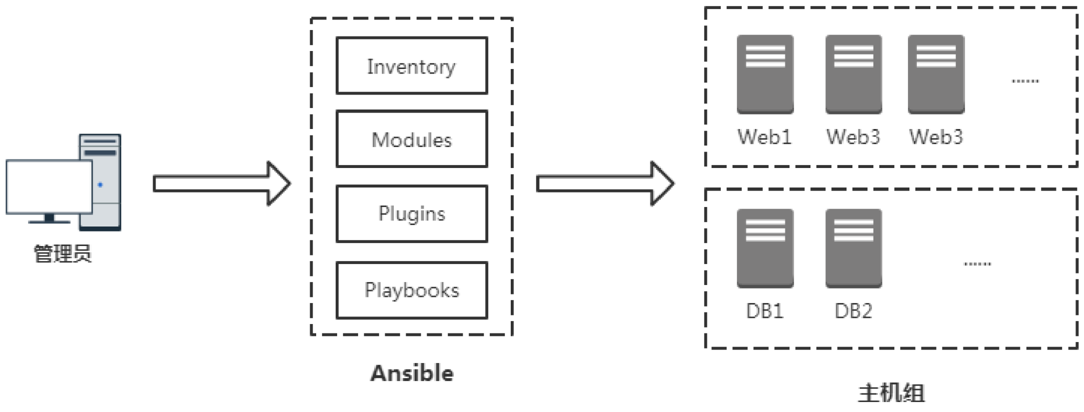
讲师：袁飞

阿良教育：www.aliangedu.cn

1. 自动化运维应用场景



- 文件传输
- 应用部署
- 配置任务
- 任务流编排



1.1 常用运维工具

- Ansible `Python，Agentless，中小型应用环境`
- Saltstack `Python，一般需要agent，执行效率高`
- Puppet `Ruby，功能强大，配置复杂，重型，适合大型环境`

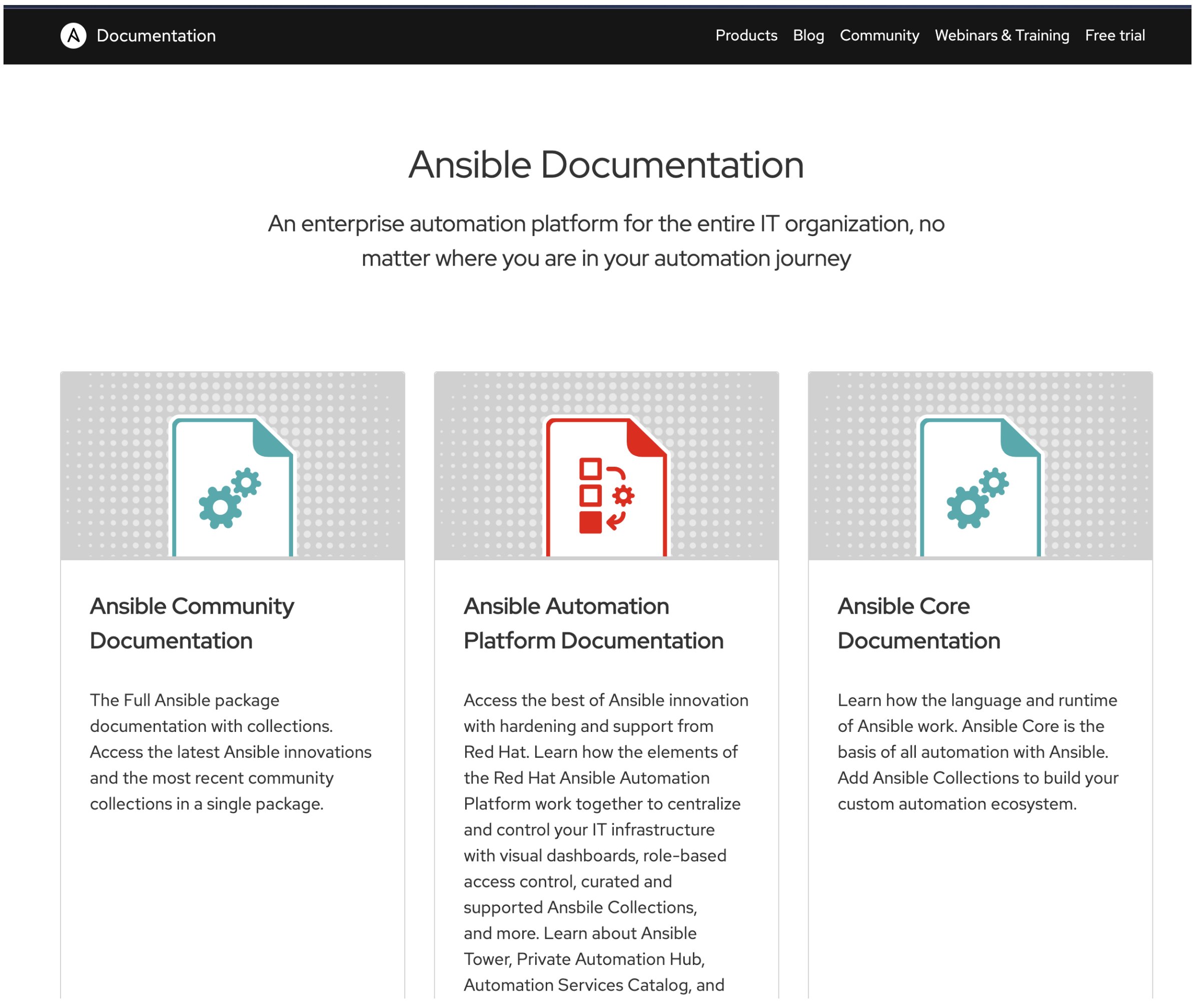
1.2 思考

> 公司计划在年底做一次大型市场促销活动，全面冲刺下交易额，为明年上市做准备。公司要求各业务组对年底大促做好准备，运维部要求所有业务容量进行三倍扩容，并搭建出多套环境可以供开发和测试人员测试，运维老大为了在年底有所表现，要求运维部门同学尽快实现，如果是你接到这个任务时，有没有更快，更便捷的解决方案？

2. Ansible发展史

ansible的名称来源于科幻小说《安德的游戏》中跨越时空的即时通讯工具，可以在相距数光年的距离，远程实时控制前线舰队战斗。

于2012年3月9日发布0.0.1版，2015年10月17日，Red Hat宣布以1.5亿美元收购。



Ansible Documentation

An enterprise automation platform for the entire IT organization, no matter where you are in your automation journey

Ansible Community Documentation

The Full Ansible package documentation with collections. Access the latest Ansible innovations and the most recent community collections in a single package.

Ansible Automation Platform Documentation

Access the best of Ansible innovation with hardening and support from Red Hat. Learn how the elements of the Red Hat Ansible Automation Platform work together to centralize and control your IT infrastructure with visual dashboards, role-based access control, curated and supported Ansbile Collections, and more. Learn about Ansible Tower, Private Automation Hub, Automation Services Catalog, and

Ansible Core Documentation

Learn how the language and runtime of Ansible work. Ansible Core is the basis of all automation with Ansible. Add Ansible Collections to build your custom automation ecosystem.

官方地址: https://www.ansible.com

官方文档: https://docs.ansible.com/ansible/latest

2.1 Ansible功能

- 批量执行远程命令，可以对远程的多台主机同时进行命令的执行
- 批量安装和配置软件服务，可以对远程多台主机进行自动化方式配置和管理各种服务
- 编排高级的企业复杂的IT架构任务，Ansible的Playbook和role可以轻松实现大型的IT复杂架构
- 提供自动化运维工具的开发API，有很多运维工具，如jumpserver就是基于ansible实现自动化管理

2.2 特性

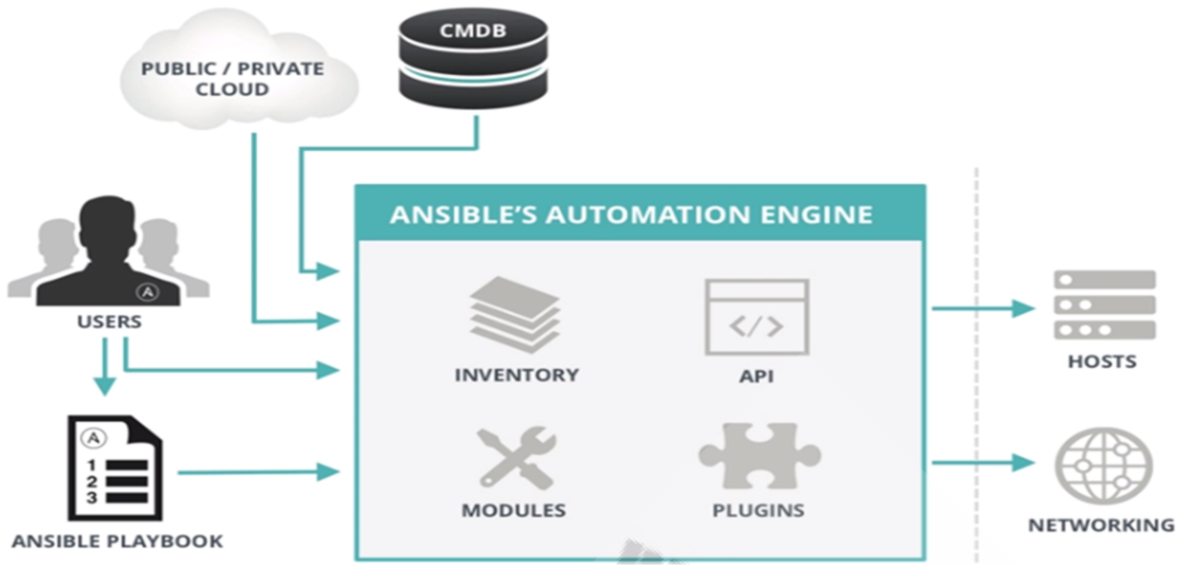
- 基于Python语言实现

- 模块化：调用特定模块完成特定任务，支持自定义模块，可使用任意编程语言编写模块
- 部署简单：基于python和ssh，agentless，无需代理不依赖PKI(无需ssl)
- 安全，基于OpenSSH
- 幂等性：一个任务执行一遍和执行n遍效果一样，不因重复执行带来意外情况，此特性非绝对
- 支持playbook编排任务， YAML格式，编排任务，支持丰富的数据结构
- 较强大的多层解决方案role

2.3 架构

组合INVENTORY、API、MODULES、PLUGINS，为ansible命令工具，其为核心执行工具

- INVENTORY: Ansible管理主机的清单，一般默认路径为 /etc/ansible/hosts
- MODULES: Ansible执行命令的功能模块，多数为内置核心模块，也可自定义
- PLUGINS: 模块功能的补充，如连接类型插件，循环插件，变量插件，过滤插件等
- API: 供第三方程序调用的应用程序编程接口



2.4 注意事项

- 执行ansible的主机一般称为管理端，主控端，中控，master或堡垒机
- 主控端Python版本需要2.6及以上

- 被控端Python版本小于2.4，需要安装python-simplejson
- 被控端如开启SELinux，需要安装libselinux-python
- windows不能作为主控端

3. Ansible入门

3.1 安装Ansible

- 官方文档：`https://docs.ansible.com/ansible/latest/installation_guide/intro_installation.html#installing-ansible-on-specific-operating-systems`

3.1.1 下载

- 下载地址：`https://releases.ansible.com/ansible/`
- pip下载：`https://pypi.org/project/ansible/`

3.1.2 安装

- 包安装

```
# 注意：先要安装epel源
$ yum install -y ansible
```

  - 示例

```
# 查看ansible版本
$ yum info ansible
Loaded plugins: fastestmirror
Repodata is over 2 weeks old. Install yum-cron? Or run: yum
makecache fast
Determining fastest mirrors
Available Packages
Name        : ansible
Arch        : noarch
Version     : 2.9.23
```

```
Release     : 1.el7
Size        : 17 M
Repo        : epel/x86_64
Summary     : SSH-based configuration management, deployment, and
task execution system
URL         : http://ansible.com
License     : GPLv3+
Description : Ansible is a radically simple model-driven
configuration management,
            : multi-node deployment, and remote task execution
system. Ansible works
            : over SSH and does not require any software or
daemons to be installed
            : on remote nodes. Extension modules can be written
in any language and
            : are transferred to managed machines automatically.
```

- pip安装

```
$ yum install -y python-pip
$ pip install --upgrade pip -i http://pypi.douban.com/simple/ --
trusted-host pypi.douban.com  # 可以忽略
$ pip install ansible --upgrade -i http://pypi.douban.com/simple/
--trusted-host pypi.douban.com
```

- 确认安装

```
$ ansible --version
ansible 2.9.25
  config file = /etc/ansible/ansible.cfg
  configured module search path =
[u'/root/.ansible/plugins/modules',
u'/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python2.7/site-
packages/ansible
  executable location = /usr/bin/ansible
  python version = 2.7.5 (default, Nov 16 2020, 22:23:17) [GCC
4.8.5 20150623 (Red Hat 4.8.5-44)]
```

3.2 相关文件

3.2.1 配置文件

- /etc/ansible/ansible.cfg `ansible主配置文件，配置ansible工作特性`

- /etc/ansible/hosts 主机清单文件，注意：不是/etc/hosts文件
- /etc/ansible/roles 角色存放目录

3.2.2 主配置文件

```
# Ansible的配置文件可以存放在多个不同地方，优先级从高到底
ANSIBLE_CONFIG          # 环境变量
./ansible.cfg           # 当前目录下的ansible.cfg
~/ansible.cfg           # 当前用户家目录下的asible.cfg
/etc/ansible/ansibe.cfg # 系统默认配置文件
```

- 配置文件解析 注意：绝大多数配置项无需修改

```
[defaults]
#inventory      = /etc/ansible/hosts         # 主机列表配置文件
#library        = /usr/share/my_modules/     # 库文件存放目录
#remote_tmp     = ~/.ansible/tmp             # 临时py命令文件存放在远程主机
的目录路径
#local_tmp      = ~/.ansible/tmp             # 本机的临时命令执行目录
#forks          = 5                          # 默认并发数
#sudo_user      = root                       # 默认sudo用户
#ask_sudo_pass = True                        # 每次执行ansible命令是否询问
SSH密码
#ask_pass      = True
#remote_port    = 22                         # 默认ssh远程端口
#roles_path    = /etc/ansible/roles          # 默认角色存放路径
#host_key_checking = False                   # 检查对应服务器的host_key，
推荐取消注释，实现第一次连接自动信任目标主机
#log_path = /var/log/ansible.log             # ansible执行日志文件，推荐开
启
#module_name = command                       # 默认使用的模块，可以修改为
shell等
#private_key_file = /path/to/file            # 配置远程ssh私钥存放路径

[privilege_escalation]                       # 普通用户提权配置
#become=True
#become_method=sudo
#become_user=root
#become_ask_pass=False
```

3.2.3 inventory主机配置清单

`ansible主要功用在于批量主机操作，为了便捷的使用其中部分主机，可以在inventory_file中将其分组命名`

- 默认inventory_file为/etc/ansible/hosts
- inventory_file 可以有多个，也可以通过Dynamic Inventory动态生成
- 建议生产环境中，在每个项目的目录下创建独立的hosts文件

官方文档：`https://docs.ansible.com/ansible/latest/user_guide/intro_inventory.html`

```
# Inventory参数说明
ansible_ssh_host                # 将要连接的远程主机名，与想要设定的主机别名不同时，可以通过此变量设置
ansible_ssh_port                # 远程主机ssh端口，如果不是默认端口号，可以通过此变量设置，如：ip:port(xxx.xxx.xxx.xxx:2222)
ansible_ssh_user                # 默认ssh用户名
ansible_ssh_pass                # ssh密码(不推荐，此方法不安全，可以使用命令行 --ask-pass 参数或 ssh密钥形式)
ansible_sudo_pass               # sudo密码(不推荐，建议使用命令行 --ask-sudo-pass 参数)
ansible_sudo_exe                # sudo命令路径(适用于1.8及以上版本)
ansible_connection              # 与主机连接类型，如：local,ssh 或 paramiko.
ansible_ssh_provate_key_file    # ssh使用的私钥，适用于多个密钥
ansible_shell_type              # 目标系统shell类型，默认使用 sh ，可以设置为 csh 或 fish
ansible_python_interpreter      # 目标主机python路径，适用情况：系统中存在多个python环境，或路径不是/usr/bin/python情况
```

示例：

```
192.168.0.1

[webservers]
192.168.0.10:2222
192.168.0.11

[dbservers]
192.168.0.21
192.168.0.22
192.168.0.23

# [dbservers]
# 192.168.0.[21:23]
```

```
# 组嵌套
[webservers]
192.168.0.10:2222
192.168.0.11

[dbservers]
192.168.0.[21:23]

[appservers]
192.168.0.[100:200]

# 定义testsrvs组中包含两个其他分组，实现组嵌套
[testsrvs:children]
webservers
dbservers
```

```
# 参数定义
[test]
192.168.0.20        ansible_connection=local        # 指定本地连接，无需ssh配置，ansible_connection=ssh 需要StrictHostKeyChecking no
192.168.0.30        ansible_connection=ssh  ansible_ssh_port=2222  ansible_ssh_user=admin ansible_ssh_password=admin123

# 显示别名
[websrvs]
web01 ansible_ssh_host=192.168.0.31 ansible_ssh_password=123456
web02 ansible_ssh_host=192.168.0.32

[websrvs:vars]
ansible_ssh_password=admin123

some_host   ansible_ssh_port=2222   ansible_ssh_user=admin
```

```
aws_host     ansible_ssh_private_key_file=/path/to/file
freebsd_host ansible_python_interpreter=/path/to/file
```

3.3 Ansible相关工具

/usr/bin/ansible	主程序，命令行执行工具
/usr/bin/ansible-doc	查看配置文档，模块功能查看工具，相当于man
/usr/bin/ansible-playbook	定制自动化任务，编排剧本工具，相当于脚本
/usr/bin/ansible-vault	文件加密工具
/usr/bin/ansible-console	基于Console界面与用户交互的执行工具

3.3.1 应用场景

- Ansible Ad-Hoc 即利用ansible命令，主要用于临时命令使用场景
- Ansible playbook 主要用于长期规划好的，大型项目场景，需要有前期的规划过程

3.3.2 使用前准备

ansible相关工具大多数是通过SSH协议，实现对远程主机的配置管理，应用部署，任务执行等功能

推荐在使用此工具前，先配置ansile主控端能基于密钥认证的方式连接各个节点(配置免密登录)

示例1：利用sshpass批量实现基于key验证脚本

```
$ vim /etc/ssh/ssh_config
Host *
        GSSAPIAuthentication yes
        ForwardX11Trusted yes
        SendEnv LANG LC_CTYPE LC_NUMERIC LC_TIME LC_COLLATE
LC_MONETARY LC_MESSAGES
        SendEnv LC_PAPER LC_NAME LC_ADDRESS LC_TELEPHONE
LC_MEASUREMENT
        SendEnv LC_IDENTIFICATION LC_ALL LANGUAGE
        SendEnv XMODIFIERS
        StrictHostKeyChecking no

$ vim hosts.list
192.168.0.21
192.168.0.22

$ vim push_ssh_key.sh
#!/bin/bash
rpm -q sshpass &> /dev/null || yum install -y sshpass
[ -f /root/.ssh/id_rsa ] || ssh-keygen -f /root/.ssh/id_rsa -P ''

export SSHPASS=admin123

while read IP;do
    sshpass -e ssh-copy-id -o StrictHostKeyChecking=no $IP
done < hosts.list
```

示例2：实现基于key验证的脚本

```
$ vim ssh_key.sh
#!/bin/bash
IPLIST="
192.168.0.21
192.168.0.22
192.168.0.23
192.168.0.24
192.168.0.25
192.168.0.26"

rpm -q sshpass &> /dev/null || yum install -y sshpass
[ -f /root/.ssh/id_rsa ] || ssh-keygen -f /root/.ssh/id_rsa -P ''

export SSHPASS=admin123

for IP in $IPLIST;do
    {sshpass -e ssh-copy-id -o StrictHostKeyChecking=no $IP;} &
done
wait
```

3.3.3 ansible-doc

此工具用来显示模块帮助信息，相当于man

格式

```
$ ansible-doc [options] [module...]
-l, --list      # 列出可用模块
-s, --snippet   # 显示指定模块的playbook片段
```

示例：

```
# 列出所有模块
$ ansible-doc -l

# 查看指定模块帮助
$ ansible-doc ping
$ ansible-doc -s ping

# 查看指定插件
$ ansible-doc -t connection -l
$ ansible-doc -t lookup -l
```

3.3.4 ansible

此工具为ansible ad-hoc的主要执行工具

格式:

```
$ ansible <host-pattern> [-m module_name] [-a args]
```

```
# 选项说明
--version           # 显示版本
-m module           # 指定使用的模块名称
--list-hosts        # 显示主机列表，可简写为 --list
-v                  # 显示详细的执行过程， -vv -vvv 可显示的更加详细
-C, --check         # 检查，不具体执行
-T, --timeout=TIMEOUT   # 执行超时时间，默认10s
-k, --ask-pass      # 提示输入ssh密码
-u, --user=REMOTE_USER  # 远程执行的用户，默认为root
-b, --become        # 代替旧版sudo切换
--become-user=USERNAME  # 指定sudo的用户，默认为root
-K, --ask-become-pass   # 提示输入sudo时口令
-f FORKS, --forks FORKS # 指定并发同时执行ansible任务的主机数
```

示例:

```
# 将普通用户提升权限(在被控端需要该用户sudo授权)
$ ansible 192.168.0.21 -m shell -a "whoami" -u admin -k -b --become-
user=root
```

3.3.5 host pattern

用于匹配被控制的主机列表

- ALL: 表示匹配所有主机

  示例:

  ```
  $ ansible all -m ping
  ```

- *: 通配符

  ```
  $ ansible "*" -m ping
  $ ansible 192.168.0.* -m ping
  $ ansible "srvs" -m ping
  $ ansible "192.168.0.21 192.168.0.22" -m ping
  ```

- 或

  ```
  $ ansible "websrvs:appsrvs" -m ping
  $ ansible "192.168.0.21:192.168.0.22" -m ping
  ```

- 与

  ```
  # 既在websrvs组中，又在dbsrvs组中的主机
  $ ansible "websrvs:&dbsrvs" -m ping
  ```

- 非

  ```
  # 在websrvs组中，并且不在dbsrvs组中的主机
  # 注意：此处需要使用单引号
  $ ansible 'websrvs:!dbsrvs' -m ping
  ```

- 综合

  ```
  $ ansible 'websrvs:dbsrvs:&appsrvs:!ftpsrvs' -m ping
  ```

- 正则表达式

  ```
  $ ansible "websrvs:dbsrvs" -m ping
  $ ansible "~(web|app).*" -m ping
  ```

示例:

```
$ ansible 'kube*:etcd:!192.168.0.20' -a reboot&&reboot
```

3.3.6 执行状态

```
$ grep -A 14 '\[colors\]' /etc/ansible/ansible.cfg
[colors]
#highlight = white
#verbose = blue
```

```
#warn = bright purple
#error = red
#debug = dark gray
#deprecate = purple
#skip = cyan
#unreachable = red
#ok = green
#changed = yellow
#diff_add = green
#diff_remove = red
#diff_lines = cyan
```

- 绿色： 执行成功并且不需要做改变的操作
- 黄色：执行成功并且对目标主机进行变更操作
- 红色：执行失败

3.3.7 ansible-playbook

此工具用于执行编写好的playbook任务

示例: hello.yaml

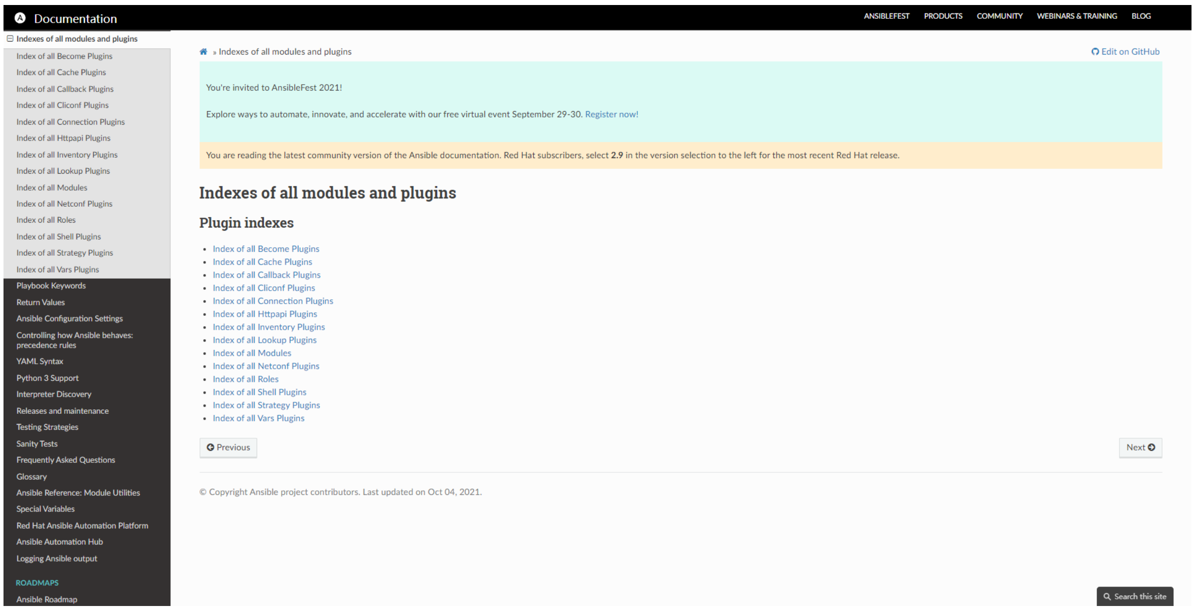
```
---
- hosts: websrvs
  remote_user: root
  gather_facts: no

  tasks:
    - name: hello world
      command: /usr/bin/wall hello world
```

4. Ansible常用模块

2015年底时ansible拥有270多个模块，到2020年3月，拥有3387个模块。虽然ansible拥有众多的模块，但最常用的大概在20-30个模块。

参考文档

```
# https://docs.ansible.com/ansible/latest/collections/all_plugins.html
#
https://docs.ansible.com/ansible/latest/modules/list_of_all_modules.ht
ml
#
https://docs.ansible.com/ansible/latest/modules/modules_by_category.ht
ml
```



Documentation

ANSIBLEFEST PRODUCTS COMMUNITY WEBINARS & TRAINING BLOG

Indexes of all modules and plugins
- Index of all Become Plugins
- Index of all Cache Plugins
- Index of all Callback Plugins
- Index of all Cliconf Plugins
- Index of all Connection Plugins
- Index of all Httpapi Plugins
- Index of all Inventory Plugins
- Index of all Lookup Plugins
- Index of all Modules
- Index of all Netconf Plugins
- Index of all Roles
- Index of all Shell Plugins
- Index of all Strategy Plugins
- Index of all Vars Plugins

Playbook Keywords
Return Values
Ansible Configuration Settings
Controlling how Ansible behaves: precedence rules
YAML Syntax
Python 3 Support
Interpreter Discovery
Releases and maintenance
Testing Strategies
Sanity Tests
Frequently Asked Questions
Glossary
Ansible Reference: Module Utilities
Special Variables
Red Hat Ansible Automation Platform
Ansible Automation Hub
Logging Ansible output

ROADMAPS
Ansible Roadmap

» Indexes of all modules and plugins   Edit on GitHub

You're invited to AnsibleFest 2021!

Explore ways to automate, innovate, and accelerate with our free virtual event September 29-30. Register now!

You are reading the latest community version of the Ansible documentation. Red Hat subscribers, select **2.9** in the version selection to the left for the most recent Red Hat release.

Indexes of all modules and plugins

Plugin indexes

- Index of all Become Plugins
- Index of all Cache Plugins
- Index of all Callback Plugins
- Index of all Cliconf Plugins
- Index of all Connection Plugins
- Index of all Httpapi Plugins
- Index of all Inventory Plugins
- Index of all Lookup Plugins
- Index of all Modules
- Index of all Netconf Plugins
- Index of all Roles
- Index of all Shell Plugins
- Index of all Strategy Plugins
- Index of all Vars Plugins

Previous   Next

© Copyright Ansible project contributors. Last updated on Oct 04, 2021.

Search this site

4.1 Command模块

- 功能：在远程主机执行命令，此为默认模块，可忽略 “-m” 选项
- 此命令不支持 $VARNAME<>|;&等，可以用shell模块实现
- 此命令不具有幂等性
- 示例：

```
# 在远程主机上先切换到 “/etc”目录下，再执行 “cat centos-release” 命令
$ ansible websrvs -m command -a 'chdir=/etc cat centos-release'
node01 | CHANGED | rc=0 >>
CentOS Linux release 7.9.2009 (Core)
node02 | CHANGED | rc=0 >>
CentOS Linux release 7.9.2009 (Core)

# 在远程主机上先切换到 “/etc”目录下，再判断是否存在“/tmp/f1.txt”文件，存
在，则不执行“cat centos-release”命令，不存在，则执行
$ ansible websrvs -m command -a 'chdir=/etc creates=/tmp/f1.txt
cat centos-release'
node01 | SUCCESS | rc=0 >>
skipped, since /tmp/f1.txt exists
node02 | CHANGED | rc=0 >>
```

```
CentOS Linux release 7.9.2009 (Core)

# 在远程主机上先切换到 “/etc”目录下，再判断是否存在“/tmp/f1.txt”文件，不存在，则不执行“cat centos-release”命令，存在，则执行
$ ansible websrvs -m command -a 'chdir=/etc removes=/tmp/f1.txt cat centos-release'
node01 | CHANGED | rc=0 >>
CentOS Linux release 7.9.2009 (Core)
node02 | SUCCESS | rc=0 >>
skipped, since /tmp/f1.txt does not exist

$ ansible websrvs -m command -a 'ls'
node02 | CHANGED | rc=0 >>
anaconda-ks.cfg
node01 | CHANGED | rc=0 >>
anaconda-ks.cfg

$ ansible websrvs -m command -a 'chdir=/var du -sh'
node02 | CHANGED | rc=0 >>
236M    .
node01 | CHANGED | rc=0 >>
236M    .
```

4.2 Shell模块

- 功能：和command类似
- 支持各种符号，如 $VARNAME<>|;&等
- 不具有幂等性
- 示例:

```
# 修改服务配置
$ ansible all -m shell -a "sed -i 's/^Listen 80/Listen 8080/' /etc/httpd/conf/httpd.conf"

$ ansible websrvs -m shell -a 'echo $HOSTNAME'
node01 | CHANGED | rc=0 >>
ansible-node01
node02 | CHANGED | rc=0 >>
ansible-node02

$ ansible websrvs -m shell -a 'useradd admin && echo centos | passwd --stdin admin'
node01 | CHANGED | rc=0 >>
```

```
Changing password for user admin.
passwd: all authentication tokens updated successfully.
node02 | CHANGED | rc=0 >>
Changing password for user admin.
passwd: all authentication tokens updated successfully.

$ ansible websrvs -m shell -a 'echo hello > /tmp/hello.html'
node01 | CHANGED | rc=0 >>

node02 | CHANGED | rc=0 >>
```

- 注意事项：调用bash执行复杂命令，shell模块也可能会调用失败
- 解决办法：将复杂命令写入脚本，copy至远程服务器，执行完成后，返回执行结果

4.3 Script模块

- 功能：再远程主机上运行ansible服务器上的脚本（无需执行权限）
- 不具有幂等性
- 示例：

```
$ ansible websrvs -m script -a /tmp/test.sh
node01 | CHANGED => {
    "changed": true,
    "rc": 0,
    "stderr": "Shared connection to node01 closed.\r\n",
    "stderr_lines": [
        "Shared connection to node01 closed."
    ],
    "stdout": "test.sh\r\nansible-node01\r\n",
    "stdout_lines": [
        "test.sh",
        "ansible-node01"
    ]
}
node02 | CHANGED => {
    "changed": true,
    "rc": 0,
    "stderr": "Shared connection to node02 closed.\r\n",
    "stderr_lines": [
        "Shared connection to node02 closed."
    ],
    "stdout": "test.sh\r\nansible-node02\r\n",
    "stdout_lines": [
        "test.sh",
        "ansible-node02"
    ]
}
```

4.4 Copy模块

- 功能：从ansible服务器主控端复制文件到远程主机
- 注意：src=file 如果没有指明路径，则为当前目录或当前目录下的files目录下的file文件
- 示例：

```
# 如果目标存在，默认为覆盖，可以指定为备份
$ ansible websrvs -m copy -a "src=/root/test1.sh
dest=/tmp/test2.sh owner=admin mode=600 backup=yes"

# 指定内容，直接生成目标
$ ansible websrvs -m copy -a "content='test line1\ntest line2\n'
dest=/tmp/test.txt"

# 复制/etc目录自身，注意/etc后面不带/
$ ansible websrvs -m copy -a "src=/etc dest=/backup"

# 复制/etc下文件，不包括/etc自身，注意/etc后面带/
$ ansible websrvs -m copy -a "src=/etc/ dest=/backup"
```

4.5 Fetch模块

- 功能：从远程主机提取文件至ansible主控端，与copy相反，暂不支持目录
- 示例：

```
$ ansible websrvs -m fetch -a "src=/root/test.sh
dest=/data/scripts"

$ ansible websrvs -m fetch -a "src=/etc/redhat-release
dest=/data/os"
node02 | CHANGED => {
    "changed": true,
    "checksum": "0d3186157c40752f89db0e618a5866935b523e7b",
    "dest": "/data/os/node02/etc/redhat-release",
    "md5sum": "902962816d0ec4fbb532949f70a41ae7",
    "remote_checksum":
"0d3186157c40752f89db0e618a5866935b523e7b",
    "remote_md5sum": null
}
node01 | CHANGED => {
    "changed": true,
    "checksum": "0d3186157c40752f89db0e618a5866935b523e7b",
    "dest": "/data/os/node01/etc/redhat-release",
    "md5sum": "902962816d0ec4fbb532949f70a41ae7",
    "remote_checksum":
"0d3186157c40752f89db0e618a5866935b523e7b",
    "remote_md5sum": null
}

# 验证主控端/data/os目录
$ tree /data/os/
/data/os/
├── node01
│   └── etc
│       └── redhat-release
└── node02
    └── etc
        └── redhat-release

4 directories, 2 files
```

4.6 File模块

- 功能：设置文件属性，创建软链接等
- 示例：

```
# 创建空文件
$ ansible all -m file -a 'path=/data/test.txt state=touch'
$ ansible all -m file -a 'path=/data/test.txt state=absent'
$ ansible all -m file -a 'path=/root/test.sh owner=admin
mode=755'

# 创建目录
$ ansible all -m file -a 'path=/data/mysql state=directory'

# 创建软链接
```

```
$ ansible all -m file -a 'src=/data/testfile dest=/data/testfile-link state=link'

# 递归修改目录属性，但不递归至子目录
$ ansible all -m file -a 'path=/data/mysql state=directory owner=mysql group=mysql' == chown

# 递归修改目录及子目录属性
$ ansible all -m file -a 'path=/data/mysql state=directory owner=mysql group=mysql recurse=yes'  == chown -R
```

4.7 Yum和Apt模块

- 功能：yum管理软件包，只支持RHEL，CentOS，fedora，不支持Ubuntu其他版本；apt模块管理Debian相关版本的软件包
- 示例：

```
# 安装软件包
$ ansible websrvs -m yum -a 'name=httpd state=present'

# 启用epel源安装软件包
$ ansible websrvs -m yum -a 'name=nginx state=present enablerepo=epel'

# 升级除kernel和foo开头外的所有包
$ ansible websrvs -m yum -a 'name=* state=lastest exclude=kernel*,foo*'

# 删除(卸载)
$ ansible websrvs -m yum -a 'name=httpd state=absent'

# 查看包
$ ansible websrvs -m yum -a 'list=tree'
```

4.8 Service模块

- 功能：管理服务
- 示例：

```
# 启动服务，并设置开机启动
$ ansible all -m service -a 'name=httpd state=started
enabled=yes'

# 停止服务
$ ansible all -m service -a 'name=httpd state=stopped'

# 重载服务
$ ansible all -m service -a 'name=httpd state=reloaded'

# 重启服务
$ ansible all -m service -a 'name=httpd state=restarted'
```

4.9 User模块

- 功能：管理用户
- 示例：

```
# 创建用户
$ ansible all -m user -a 'name=user1 comment="test user" uid=2000
home=/app/user1 group=root'

$ ansible all -m user -a 'name=nginx comment=nginx uid=1000
group=nginx groups="root,daemon" shell=/sbin/nologin system=yes
create_home=no home=/data/nginx non_unique=yes'

# 删除用户及其家目录
$ ansible all -m user -a 'name=nginx state=absent remove=yes'
```

4.10 Group模块

- 功能：管理用户组
- 示例：

```
# 创建组
$ ansible websrvs -m group -a 'name=nginx gid=1000 system=yes'

# 删除组
$ ansible websrvs -m group -a 'name=nginx state=absent'
```

4.11 Setup模块

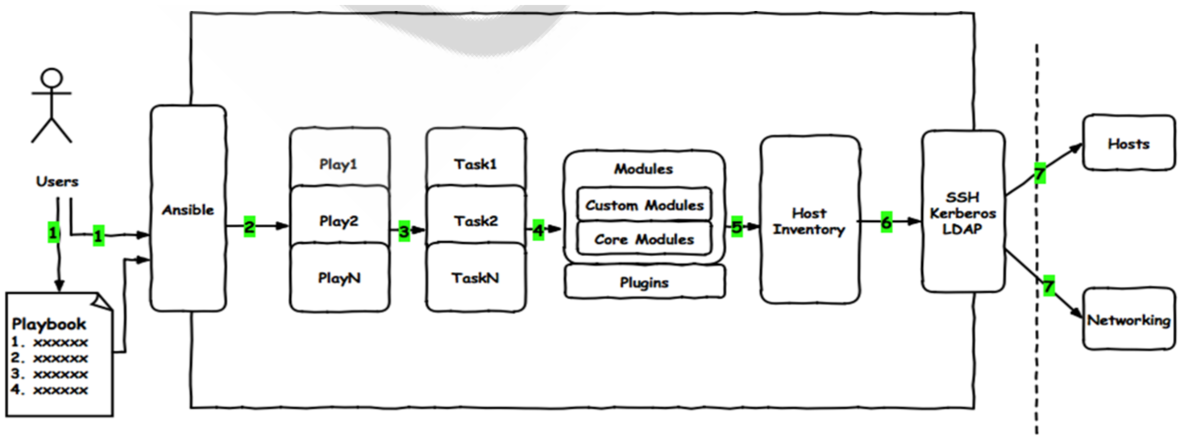
- 功能：setup模块用来收集主机系统信息，这些facts信息可以直接以变量形式使用，但如果主机较多，会影响执行速度，可以使用gather_facts: no禁止ansible收集信息
- 示例：

```
$ ansible all -m setup
$ ansible all -m setup -a "filter=ansible_nodename"
$ ansible all -m setup -a "filter=ansible_hostname"
$ ansible all -m setup -a "filter=ansible_domain"
$ ansible all -m setup -a "filter=ansible_memtotal_mb"
$ ansible all -m setup -a "filter=ansible_memory_mb"
$ ansible all -m setup -a "filter=ansible_memfree_mb"
$ ansible all -m setup -a "filter=ansible_os_family"
$ ansible all -m setup -a
"filter=ansible_distribution_major_version"
$ ansible all -m setup -a "filter=ansible_distribution_version"
$ ansible all -m setup -a "filter=ansible_processor_vcpus"
# 获取所有IP地址
$ ansible all -m setup -a "filter=ansible_all_ipv4_addtrsses"
$ ansible all -m setup -a "filter=ansible_architecture"
$ ansible all -m setup -a "filter=ansible_uptime_seconds"
$ ansible all -m setup -a "filter=ansible_processor*"
$ ansible all -m setup -a "filter=ansible_env"
```

5. Playbook

阿良教育： www.aliangedu.cn

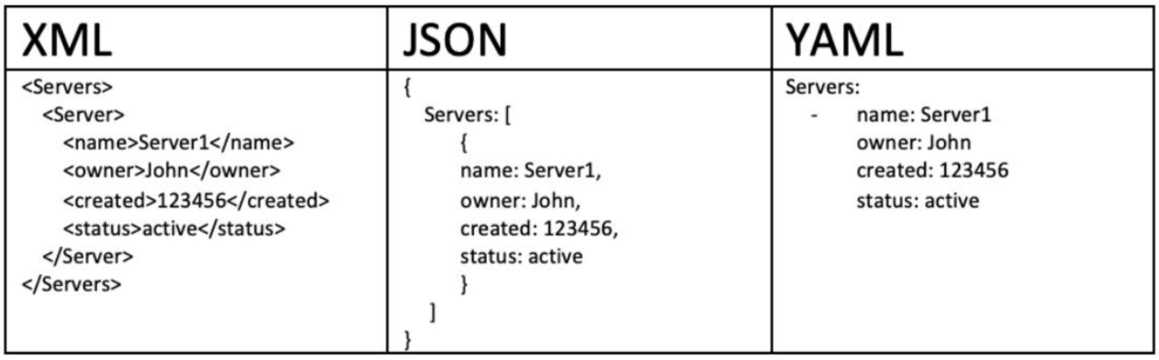
5.1 playbook介绍



- playbook剧本是由一个或多个“play”组成的列表
- play的主要功能在于将预定义的一组主机，装扮成事先通过ansible中的task定义好的角色
- task是调用ansible的一个模块，将多个play组织再一个playbook中，即可以联合起来，按事先编排好的机制执行预定义的动作
- playbook文件采用YAML语言编写

5.2 三种常见的数据格式

- XML：Extensible Markup Language，可扩展标记语言，可用于数据交换和配置
- JSON：JavaScript Object Notation，JavaScript对象标记法，主要用来数据交换或配置，不支持注释
- YAML：YAML Ain't Markup Language，YAML不是一种标记语言，主要用来配置，大小写敏感，不支持tab



XML	JSON	YAML

<Servers>
  <Server>
    <name>Server1</name>
    <owner>John</owner>
    <created>123456</created>
    <status>active</status>
  </Server>
</Servers>
	{
	  Servers: [
	    {
	    name: Server1,
	    owner: John,
	    created: 123456,
	    status: active
	    }
	  ]
	}
		Servers:
		  - name: Server1
		       owner: John
		       created: 123456
		       status: active

- 可以用工具相互转换，参考网站：

```
https://www.json2yaml.com/
https://www.bejson.com/json/json2yaml/
```

5.3 Playbook核心组件

一个playbook中由多个组件组成，其中所用到的常见组件类型如下:

- Hosts：执行的远程主机列表
- Tasks：任务集，由多个task远程组成的列表，每个task都是一个字典，一个完整的代码块功能需最少元素包括name和task，一个name只能有一个task
- Variables：内置变量或自定义变量在playbook中调用
- Templates：模板，可替换模板文件中的变量并实现一些简单逻辑文件
- Hnadlers和notify结合使用，由特定条件触发的操作，满足条件方可执行
- Tags：标签，指定某条任务执行，用于选择运行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，即便如此，有些代码为测试器确实没有发生变化的时候，依然会非常的长，此时，如果确信其没有变化，就可以通过tags跳过这些代码片段

5.3.1 Hosts组件

- playbook中的每个play的目的都是为了让特定主机以某个指定用户身份执行任务
- hosts主要用于指定要执行的任务主机，须事先定义在主机清单中
- 示例：

```
- hosts: websrvs
- hosts: 192.168.0.21
- hosts: websrvs:appsrvs  # 两个组的并集
- hosts: websrvs:&appsrvs # 两个组的交集
- hosts: websrvs:!appsrvs # 在websrvs组中，但不在appsrvs中
```

5.3.2 remote_user组件

- 可用于Host和Task中，也可以通过指定其通过sudo方式在远程主机上执行任务，其可用于play全局或某个任务中
- 可以在sudo时使用sudo_user指定sudo时切换的用户，默认为root
- 示例：

```yaml
- hosts: websrvs
  remote_user: root

  tasks:
    - name: test connection
      ping:
      remote_user: admin
      sudo: yes
      sudo_user: yuan
      # become: yes
      # become_user: yuan
```

- 注意点：2.9版本 sudo已经替换为become

5.3.3 task列表和action组件

- play主体部分是task list，task list有一个或多个task，每个task按次序逐个在hosts中指定的所有主机上执行，即在所有主机上执行完一个task后，再执行下一个task
- task的目的是使用指定的参数执行模块，再模块参数中可以使用变量。模块执行为幂等的，意味着多次执行是安全的，其结果一致
- 每个task都应该有其name，用于playbook执行结果输出，建议其内容能够清晰描述任务执行步骤，如果没有提供name，则action的结果将用于输出
- task两种格式：

```yaml
action: module arguments  #示例: action: shell wall hello
module: arguments   # 推荐使用此格式  # 示例: shell: wall hello
```

- 示例1：

```yaml
---
- hosts: websrvs
  remote_user: root
  gather_facts: no      # 不收集系统信息，提高执行效率

  tasks:
    - name: test network connection
      ping:
    - name: excute command
      command: wall "hello world!"
```

- 示例2：

```
---
- hosts: websrvs
  remote_user: root
  gather_facts: no

  tasks:
    - name: install httpd
      yum: name=httpd
    - name: start httpd
      service: name=httpd state=started enabled=yes
```

5.3.4 其他组件说明

- 某任务状态再运行后为changed时，可通过notify通知给相应的handlers任务
- 可以通过tags给task打标签，再ansible-playbook命令上使用-t指定调用
- 示例1：

```
---
- hosts: httpd
  remote_user: root
  gather_facts: no

  tasks:
    - name: install httpd
      yum: name=httpd
    - name: copy config file to httpd server
      copy: src=/etc/httpd/conf/httpd.conf
dest=/etc/httpd/conf/httpd.conf
      notify: restart service
      tags:
        - change config
    - name: start httpd
      service: name=httpd state=started enabled=yes
      tags:
        - start service
  handlers:
    - name: restart service
      service: name=httpd state=restarted
```

5.3.5 ShellScripts VS Playbook

- shell脚本：

```
#!/bin/bash
yum install -y httpd
cp /tmp/httpd.conf /etc/httpd/conf/httpd.conf
cp /tmp/vhosts.conf /etc/httpd/conf.d/
systemctl enable --now httpd
```

- playbook：

```
---
- hosts: httpd
  remote_user: root
  gather_facts: no

    tasks:
    - name: install httpd
      yum: name=httpd
    - name: copy config file to httpd server
      copy: src=/tmp/httpd.conf dest=/etc/httpd/conf/
    - name: copy config file to httpd server
      copy: src=/tmp/vhosts.conf dest=/etc/httpd/conf.d/
    - name: start httpd
      service: name=httpd state=started enabled=yes
```

5.4 Playbook命令

`ansible-playbook <filename.yaml> ...[options]`

常见选项：

- --syntax-check　　语法检查，可缩写为--syntax，相当于bash -n
- -C, --check　　模拟执行，只检测可能会发生的变化，但不真正执行操作，dry run
- --list-hosts　　列出运行任务的主机
- --list-tags　　列出tag
- --list-tasks　　列出task
- --limit 主机列表　　只针对主机列表中的特定主机执行
- -i INVENTORY　　指定主机清单文件，通常一个项对应一个主机清单文件
- --start-at-task START_AT_TASK　　从指定的task开始执行任务，而不是从开头开始，START_AT_TASK 为任务名称
- -v, -vv, -vvv　　显示具体执行过程，详细程度根据-v数量变化

5.5 Playbook综合示例

5.5.1 创建mysql用户

```
---
- hosts: dbsrvs
  remote_user: root
  gather_facts: no

  tasks:
    - {name: create group, group: name=mysql system=yes gid=1000}
    - name: create user
      user: name=mysql system=yes uid=1000 shell=/sbin/nologin
group=mysql home=/data/mysql create_home=no
```

5.5.2 安装nginx

```
---
- hosts: nginx
  remote_user: root
  gather_facts: no

  tasks:
    - name: add group nginx
      group: name=nginx state=present
    - name: add user nginx
      user: name=nginx state=present group=nginx
    - name: Install Nginx
      yum: name=nginx state=present
    - name: web page
      copy: src=files/index.html dest=/usr/share/nginx/html/index.html
    - name: Start Nginx
      service: name=nginx state=started enabled=yes
    - name: wait nginx started
      wait_for: port=80 state=started delay=10
    - name: curl http
      uri: url="http://localhost" return_content=yes
      register: curl
    - name: echo http content
      debug: msg={{ curl.content }}
```

5.5.3 安装和卸载httpd

- 安装httpd，并修改配置启动

```yaml
---
- hosts: httpd
  remote_user: root
  gather_facts: no

  tasks:
    - name: Install Httpd
      yum: name=httpd
    - name: Modify config list port
      lineinfile:
        path: /etc/httpd/conf/httpd.conf
        regexp: '^Listen'
        line: 'Listen 8080'
    - name: Modify config data1
      lineinfile:
        path: /etc/httpd/conf/httpd.conf
        regexp: '^DocumentRoot "/var/www/html"'
        line: 'DocumentRoot "/data/html"'
    - name: Modify config data2
      lineinfile:
        path: /etc/httpd/conf/httpd.conf
        regexp: '^<Directory "/var/www/html">'
        line: '<Directory "/data/html">'
    - name: Mkdir website dir
      file: path=/data/html state=directory
    - name: web html
      copy: src=files/index.html dest=/data/html
    - name: Start Service
      service: name=httpd state=started enabled=yes
```

- 卸载httpd

```yaml
---
- hosts: httpd
  remote_user: root
  gather_facts: no

  tasks:
    - name: remove httpd package
      yum: name=httpd state=absent
    - name: remove apache user
      user: name=apache state=absent
    - name: remove config file
      file: name=/etc/httpd state=absent
    - name: remove web html
      file: name=/data/html state=absent
```

5.5.4 二进制安装mysql5.6

- 准备二进制安装包、配置文件及初始化脚本

```
$ cd files
# 下载mysql二进制安装包(安装包下载地址:
https://downloads.mysql.com/archives/community/
https://cdn.mysql.com/archives/mysql-5.6/mysql-5.6.46-linux-glibc2.12-
x86_64.tar.gz)
$ ll
total 393732
-rw-r--r-- 1 root root 403177622 Oct 10 09:31 mysql-5.6.46-linux-
glibc2.12-x86_64.tar.gz

# 修改mysql配置文件
$ cat my.cnf
[mysqld]
socket=/tmp/mysql.sock
user=mysql
symbolic-links=0
datadir=/data/mysql
innodb_file_per_table=1
log-bin
pid-file=/data/mysql/mysqld.pid

[client]
port=3306
socket=/tmp/mysql.sock

[mysqld_safe]
log-error=/var/log/mysqld.log

# mysql初始化脚本
$ cat soure_mysql.sh
#!/bin/bash
/usr/local/mysql/bin/mysql_secure_installation << EOF
```

```
y
admin123
admin123
y
y
y
y
EOF
```

- playbook脚本

```
---
- hosts: dbsrvs
  remote_user: root
  gather_facts: no

  tasks:
    - name: install packages
      yum: name=libaio,perl-Data-Dumper,perl-Getopt-Long
    - name: create mysql group
      group: name=mysql gid=306
    - name: create mysql user
      user: name=mysql uid=306 group=mysql shell=/sbin/nologin
system=yes create_home=no home=/data/mysql
    - name: create data dir
      file: path=/data/mysql state=directory owner=mysql group=mysql
    - name: copy tar to remote host and file mode
      unarchive: src=files/mysql-5.6.46-linux-glibc2.12-x86_64.tar.gz
dest=/usr/local owner=root group=root
    - name: create linkfile to /usr/local/mysql
      file: src=/usr/local/mysql-5.6.46-linux-glibc2.12-x86_64
dest=/usr/local/mysql state=link
    - name: data dir
      shell: chdir=/usr/local/mysql/ ./scripts/mysql_install_db --
datadir=/data/mysql --user=mysql
      tags: data
    - name: config my.cnf
      copy: src=files/my.cnf dest=/etc/my.cnf
    - name: service script
      shell: /bin/cp /usr/local/mysql/support-files/mysql.server
/etc/init.d/mysqld
    - name: enable service
      shell: /etc/init.d/mysqld start;chkconfig --add mysqld;chkconfig
mysqld on
      tags: service
    - name: PATH variable
      copy: content='PATH=/usr/local/mysql/bin:$PATH'
dest=/etc/profile.d/mysql.sh
    - name: secure script
      script: files/soure_mysql.sh
      tags: script
```

5.6 忽略错误

- 如果一个task出错，默认将不会继续执行后续其他task
- 利用ignore_errors: yes 可以忽略此task错误，继续向下执行其他task
- 示例:

```
---
- hosts: websrvs
  tasks:
    - name: error
      command: /bin/false
      ignore_errors: yes
    - name: continue
      command: wall continue
```

5.7 使用变量

- 变量名：仅能由字母，数字和下划线组成，且只能以字母开头
- 变量定义：`variable=value  variable: value`
- 示例：

```
http_port=80
http_Port: 80
```

- 变量调用：

  通过{{ variable_name }}调用变量，且变量名前后建议加空格，有时用"{{ variable_name }}"才生效
- 变量来源：
  - ansible的setup facts 远程主机的所有变量都可以直接调用
  - 通过命令行指定变量，优先级最高

    ```
    $ ansible-playbook -e varname=value test.yaml
    ```
  - 再playbook文件中定义

```
vars:
  var1: value1
  var2: value2
```

  - 再独立的变量YAML文件中定义

```
- hosts: all
  vars_files:
    - vars.yml
```

  - 再主机清单中定义

    主机(普通)变量：主机组中主机单独定义，优先级高于公共变量

    组(公共)变量：针对主机组中所有主机定义统一变量

  - 在项目中针对主机和主机组定义

    在项目目录中创建host_vars和group_vars目录

  - 在role中定义

- 变量优先级从高到底如下:

```
-e 选项定义变量 --> playbook中vars_files --> playbook中vars变量定义 --> host_vars/主机名文件 --> 主机清单中主机变量 --> group_vars/all文件 --> 主机清单组变量
```

5.7.1 使用**setup**模块中变量

- 本模块自定在playbook中调用，不要用ansible命令调用，生成的系统状态信息，存放在facts变量中
- facts包含的信息很多，有主机名，IP，CPU，内存，网卡等
- facts变量使用场景:
  - 通过facrs变量获取被控端cpu个数信息，从而生成不同nginx配置文件
  - 通过facrs变量获取被控端内存信息，生成不同memcached配置文件
  - 通过facts变量获取被控端主机名称信息，生成不同zabbix配置文件
  - ...
- 示例1：

```
$ ansible node01 -m setup -a 'filter="ansible_default_ipv4"'
node01 | SUCCESS => {
    "ansible_facts": {
        "ansible_default_ipv4": {
            "address": "10.10.10.51",
            "alias": "ens33",
            "broadcast": "10.10.10.255",
            "gateway": "10.10.10.2",
            "interface": "ens33",
            "macaddress": "00:0c:29:83:e4:6f",
            "mtu": 1500,
            "netmask": "255.255.255.0",
            "network": "10.10.10.0",
            "type": "ether"
        },
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false
}
```

- 示例2:

```
---
# var1.yml
- hosts: all
  remote_user: root
  gather_facts: yes

  tasks:
    - name: create log file
      file: name=/data/{{ ansible_nodename }}.log state=touch
owner=admin mode=600
```

- 示例3:

```
---
# show_ip.yaml
- hosts: all

  tasks:
    - name: show ens33 ip address {{ ansible_facts["ens33"]
["ipv4"]["address"] }}
      debug:
        msg: IP Address {{ ansible_ens33.ipv4.address }}
        # msg: IP Address {{ ansible_facts["ens33"]["ipv4"]
["address"] }}
        # msg: IP Address {{ ansible_facts.ens33.ipv4.address }}
        # msg: IP Address {{ ansible_default_ipv4.address }}
        # msg: IP Address {{
ansible_ens33.ipv4.address.split('.')[-1] }}  # 取IP中的最后一个数字
```

5.7.2 在命令行中定义变量

- 示例1：

```
$ vim var2.yaml
---
- hosts: all
  remote_user: root

  tasks:
    - name: install package
      yum: name={{ pkname }} state=present

$ ansible-playbook -e pkname=nginx var2.yaml
```

- 示例2：

```
# 将多个变量放在一个文件中
$ cat vars
pkname1: memcached
pkname2: vsftpd

$ vim var2.yml
---
- hosts: all
  remote_user: root

  tasks:
    - name: install package
      yum: name={{ pkname1 }} state=present
    - name: install package
      yum: name={{ pkname2 }} state=present

$ ansible-playbook -e '@vars' var2.yml
```

5.7.3 在playbook文件中定义

- 示例1：

```
# var3.yml
---
- hosts: all
  remote_user: root
  vars:
    username: admin
    groupname: admin

  tasks:
    - name: create group
      group: name={{ groupname }} state=present
    - name: create user
      user: name={{ username }} group={{ groupname }}
state=present
```

- 示例2：变量相互调用

```
---
- hosts: all
  vars:
    suffix: "txt"
    file: "{{ ansible_nodename }}.{{ subffix }}"

  tasks:
    - name: test var
      file: path="/data/{{ file }}" state=touch
```

- 示例3：安装多个包

```
# install1.yaml
- hosts: websrvs
  vars:
    web: httpd
    db: mariadb-server

  tasks:
    - name: install {{ web }} {{ db }}
      yum:
        name:
          - "{{ web }}"
          - "{{ db }}"
        state: latest

# install2.yml
- hosts: websrvs
  tasks:
    - name: install packages
      yum: name={{ pack }}
      vars:
        pack:
          - httpd
          - memcached
```

- 作业：安装指定版本的MySQL

5.7.4 使用变量文件

- 可以在一个独立的playbook文件中定义变量，在另一个playbook文件中引用变量，比playbook中定义的优先级更高
- 示例1:

```
# vars.yaml
---
package_name: mariadb-server
service_name: mariadb


# var.yml
---
- hosts: dbsrvs
  remote_user: root
  vars_file:
    - vars.yaml

  tasks:
    - name: install package
      yum: name={{ package_name }} state=present
      tags: install
    - name: start service
      service: name={{ service_name }} state=started enabled=yes
```

5.7.5 针对主机和主机组的变量

- 项目的主机变量
  在主机清单文件中为指定的主机定义变量，以便于在playbook中使用

```
[websrvs]
node01 http_port=80 maxRequestsPerChild=808
node02 http_port=8080 maxRequestsPerChild=900
```

- 项目的组变量

```
# 赋予组内所有指定主机可用变量，如遇变量名相同，优先级低于主机变量
[websrvs:vars]
http_port=80
ntp_server=ntp.aliyun.com
nfs_server=nfs.aliyun.com
```

5.7.6 register注册变量

- 在playbook中可以使用register将捕获命令的输出保存在临时变量中，然后使用debug模块进行显示输出
- 示例1：

```
# 利用debug模块输出变量
- hosts: dnsrvs
  tasks:
    - name: get variable
      shell: hostname
      register: name
    - name: "print variable"
      debug:
        msg: "{{ name }}"      # 输出register注册的name变量全部信息
        # msg: "{{ name.cmd }}"  # 显示命令
        # msg: "{{ name.rc }}"   # 显示命令成功与否
        # msg: "{{ name.stdout }}"  # 显示命令的输出结果为字符串形式
        # msg: "{{ name.stdout_lines }}"  # 显示命令的输出结果为列表形式
        # msg: "{{ name.stdout_lines[0] }}"  # 显示命令的输出结果的列表中第一个元素
        # msg: "{{ name[stdout_lines] }}"  # 显示命令的输出结果为列表形式

# 解释:
# 1. 第一个task中，使用register注册变量名为name，当shell模块执行完毕后，会将数据放到该变量中
# 2. 第二个task中，使用debug模块输出，从变量name中获取数据
```

- 示例2：

```
# 使用注册变量创建文件
- hosts: dbsrvs
  tasks:
    - name: get variable
      shell: hostname
      register: name
    - name: create file
      file: dest=/tmp/{{ name.stdout }}.log state=touch
```

- 示例3：

```
# 启动服务并检查
- hosts: websrvs
  vars:
    package_name: nginx
    service_name: nginx

  tasks:
    - name: install {{ package_name }}
      yum: name={{ package_name }}
    - name: start {{ service_name }}
      service: name={{ service_name }} state=started enabled=yes
    - name: check
      shell: ps aux | grep {{ service_name }} | grep -v grep
      register: check_service
    - name: debug
      debug:
        msg: "{{ check_service.stdout_lines }}"
```

5.8 实战案例: 批量部署mysql

5.8.1 利用register注册变量批量部署

```
$ mkdir -p files
# 下载mysql二进制安装包
$ tree ./
./
├── files
│   ├── my.cnf
│   ├── mysql-5.7.33-linux-glibc2.12-x86_64.tar.gz
│   └── mysql-8.0.23-linux-glibc2.12-x86_64.tar.xz
└── install_mysql.yaml
```

```
1 directory, 4 files

$ cat files/my.cnf
[mysqld]
server-id=1
log-bin
datadir=/data/mysql
socket=/data/mysql/mysql.sock

log-error=/data/mysql/mysql.log
pid-file=/data/mysql/mysql.pid

[client]
socket=/data/mysql/mysql.sock
```

```
# install_mysql.yaml
- hosts: dbsrvs
  remote_user: root
  gather_facts: no

  vars:
    mysql_version: 5.7.33
    mysql_file: mysql-{{ mysql_version }}-linux-glibc2.12-
x86_64.tar.gz
    mysql_root_password: Admin#123

  tasks:
    - name: install packages
      yum:
        name:
          - libaio
          - numactl-libs
          - MySQL-python
        state: latest
    - name: create mysql group
      group: name=mysql gid=306
    - name: create mysql user
      user: name=mysql uid=306 group=mysql shell=/sbin/nologin
create_home=no home=/data/mysql system=yes
    - name: copy tar to remote host and file mode
      unarchive: src=files/{{ mysql_file }} dest=/usr/local/
owner=root group=root
    - name: create linkfile /usr/local/mysql
      file: src=/usr/local/mysql-{{ mysql_version }}-linux-glibc2.12-
x86_64 dest=/usr/local/mysql state=link
```

```
    - name: data dir
      shell: /usr/local/mysql/bin/mysqld --initialize --user=mysql --
datadir=/data/mysql
      tags: data
    - name: config my.cnf
      copy: src=files/my.cnf dest=/etc/my.cnf
    - name: service script
      shell: /bin/cp /usr/local/mysql/support-files/mysql.server
/etc/init.d/mysqld
    - name: PATH variable
      copy: content='PATH=/usr/local/mysql/bin:$PATH'
dest=/etc/profile.d/mysql.sh
    - name: enable service
      shell: chkconfig --add mysqld;/etc/init.d/mysqld start
      tags: service
    - name: get password
      shell: awk '/A temporary password/{print $NF}'
/data/mysql/mysql.log
      register: password
    - name: "print password"
      debug:
        msg: "{{ password.stdout }}"
    - name: change password
      shell: /usr/local/mysql/bin/mysqladmin -uroot -p'{{
password.stdout }}' password {{ mysql_root_password }}
```

```
$ ansible-playbook install_mysql.yaml
# 此处注意：第一次执行脚本时无法修改密码，需在主控端执行以下操作后再次执行playbook

$ ansible dbsrvs -m shell -a "service mysqld stop; rm -rf
/data/mysql/*"

$ ansible-playbook install_mysql.yaml
```

5.8.2 网络下载并部署mysql

```
$ tree ./
./
├── files
│   └── my.cnf
└── install_mysql.yaml

1 directory, 2 files
```

```yaml
# install_mysql.yaml
- hosts: dbsrvs
  vars:
    password: 123456
  tasks:
    - name: download mysql-package
      get_url:
        url:
https://mirrors.tuna.tsinghua.edu.cn/mysql/downloads/MySQL-5.7/mysql-
5.7.33-linux-glibc2.12-x86_64.tar.gz
        dest: /usr/local/mysql-5.7.33-linux-glibc2.12-x86_64.tar.gz
        force: yes
    - name: tar mysql-package
      unarchive:
        src: /usr/local/mysql-5.7.33-linux-glibc2.12-x86_64.tar.gz
        dest: /usr/local
        owner: root
        group: root
        mode: 0755
        copy: no
    - name: create linkfile /usr/local/mysql
      file:
        src: /usr/local/mysql-5.7.33-linux-glibc2.12-x86_64
        dest: /usr/local/mysql
        state: link
    - name: create bin link
      shell: "ln -s /usr/local/mysql/bin/* /usr/bin/"
      ignore_errors: yes
    - name: copy my.cnf
      copy:
        src: files/my.cnf
        dest: /etc/my.cnf
    - name: install packages
      yum:
        name: libaio,perl-Data-Dumper,perl-Getopt-Long
        state: present
    - name: create mysql group
      group:
        name: mysql
        gid: 306
    - name: create mysql user
      user:
        name: mysql
        uid: 306
        group: mysql
        shell: /sbin/nologin
        system: yes
    - name: create work dir
      file:
        path: /data/mysql
        state: directory
        mode: 0755
        owner: mysql
        group: mysql
    - name: Initialization mysql
      shell: "mysqld --initialize --user=mysql --datadir=/data/mysql"
      ignore_errors: yes
    - name: service script
      shell: "/bin/cp /usr/local/mysql/support-files/mysql.server
/etc/init.d/mysqld;chkconfig --add mysqld;chkconfig mysqld on"
    - name: start service
      service:
        name: mysqld
        state: started
        enabled: yes
    - name: set root password
      shell: "mysqladmin -uroot -p\"`awk '/A temporary password/{print
$NF}' /data/mysql/mysql.log`\" password {{ password }}"
      register: error
      ignore_errors: yes
    - name: retry set new password
      shell: "mysqladmin -uroot -p'{{ password }}'" password {{
password }}
```

5.8.3 利用脚本部署mysql

```
$ tree ./
./
├── files
│?? ├── my.cnf
│?? ├── mysql-5.7.33-linux-glibc2.12-x86_64.tar.gz
│?? └── set_pass.sh
├── install_mysql.yaml
└── vars.yml

1 directory, 5 files

$ cat files/set_pass.sh
#!/bin/bash
```

```
MYSQL_ROOT_PASSWORD=123456
MYSQL_OLDPASSWORD=`awk '/A temporary password/{print $NF}'
/data/mysql/mysql.log`

mysqladmin -uroot -p$MYSQL_OLDPASSWORD password $MYSQL_ROOT_PASSWORD
&> /dev/null

$ cat vars.yml
---
mysql_version: 5.7.33
```

```
- hosts: dbsrvs
  remote_user: root
  gather_facts: yes

  var_files:
    - vars.yml

  tasks:
    - name: install packages centos7
      yum: name=libaio,perl-Data-Dumper
      when: ansible_facts['distribution_major_version'] == "7"
    - name: install packages centos8
      yum: name=libaio,perl-Data-Dumper,ncurses-compat-libs
      when: ansible_facts['distribution_major_version'] == "8"
    - name: create mysql group
      group: name=mysql gid=306
    - name: create mysql user
      user: name=mysql uid=306 group=mysql shell=/sbin/nologin
system=yes create_home=no home=/data/mysql
    - name: create mysql data dir
      file: path=/data/mysql state=directory owner=mysql group=mysql
mode=0755
    - name: copy tar to remote host and file mode
      unarchive: src=files/mysql-{{ mysql_version }}-linux-glibc2.12-
x86_64.tar.gz dest=/usr/local/ owner=root group=root
    - name: create linkfile /usr/local/mysql
      file: src=/usr/local/mysql-{{ mysql_version }}-linux-glibc2.12-
x86_64 dest=/usr/local/mysql state=link
    - name: PATH variable
      copy: content='PATH=/usr/local/mysql/bin:$PATH'
dest=/etc/profile.d/mysql.sh
    - name: PATH variable entry
      shell: . /etc/profile.d/mysql.sh
    - name: config my.cnf
```

```
      copy: src=files/my.cnf dest=/etc/my.cnf
    - name: init data dir
      shell: chdir=/usr/local/mysql ./bin/mysqld --initialize --
user=mysql --datadir=/data/mysql
    - name: service script
      shell: /bin/cp /usr/local/mysql/support-files/mysql.server
/etc/init.d/mysqld
    - name: enable service
      shell: /etc/init.d/mysqld start;chkconfig --add mysqld;
chkconfig mysqld on
      tags: service
    - name: set mysql password
      script: files/set_pass.sh
      tags: script
```

6. Template 模板

模板是一个文本文件，可以作为生成文件的模板，并且模板文件中还可以嵌套jinja2语法

6.1 jinja2语法

jinja2是一个现代的，设计者友好的，仿照django模板的python模板语言。速度快，被广泛使用，并且提供了可选的沙箱模板执行环境以保证安全。

特性：

- 沙箱中执行
- 强大的HTML自动转义系统保护系统免受XSS
- 模板继承
- 及时编译最优的python代码
- 可选提前编译模板的时间
- 易于调试，异常的行数直接指向模板中的对应行
- 可配置的语法

官方地址:

```
https://jinja.palletsprojects.com/en/3.0.x/
```

中文文档:

```
https://www.w3cschool.cn/yshfid/
```

jinja2语言支持多种数据类型和操作：

- 字面量：如字符串，使用单引号或双引号；数字，整数，浮点数
- 列表：[item1, item2, ...]
- 元组：(item1, item2, ...)
- 字典：{key1:value1, key2:value2, ...}
- 布尔型：true/false
- 算术运算：+、-、*、/、//、%、**
- 比较操作符：==、!=、>、<、>=、<=
- 逻辑运算：and、or、not
- 流表达式：for、if、when

6.2 Template

- 功能：可以根据和参考模块文件，动态生成相类似的配置文件
- template文件必须存放于templates目录下，且命名为.j2结尾
- yaml/yml文件需和templates目录平级，目录结构如下:
  ```
  $ tree ./
  ./
  ├── temnginx.yml
  └── templates
      └── nginx.conf.j2

  1 directory, 2 files
  ```
- 示例：

```yaml
# temnginx.yml
---
- hosts: websrvs
  remote_user: root

  tasks:
    - name: template config to remote hosts
      template: src=nginx.conf.j2 dest=/etc/nginx/nginx.conf
```

- template替换: nginx.conf.j2

```
......
worker_processes {{ ansible_processor_vcpus }}
......
```

```yaml
---
- hosts: websrvs
  remote_user: root

  tasks:
    - name: install nginx
      yum: name=nginx
    - name: template config to remote hosts
      template: src=nginx.conf.j2 dest=/etc/nginx/nginx.conf
    - name: start service
      service: name=nginx state=started enabled=yes
```

- template算数运算

  示例：

```
# nginx.conf.j2
worker_processes {{ ansible_processor_vcpus ** 2 }};
worker_processes {{ ansible_processor_vcpus + 2 }};
```

```yaml
- hosts: websrvs
  remote_user: root

  tasks:
    - name: install nginx
      yum: name=nginx state=present
    - name: template config to remote hosts
      template: src=nginx.conf.j2 dest=/etc/nginx/nginx.conf
      notify: restart nginx
    - name: start service
      service: name=nginx state=started enabled=yes
  handlers:
    - name: restart nginx
      service: name=nginx state=restarted
```

6.3 使用流程控制if和for

template中也可以使用流程控制for循环和if条件判断，实现动态生成文件功能

6.3.1 for循环

- 格式:

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

  示例1:

  ```
  {% for i in range(1,10) %}
      server_name web-{{ i }};
  {% endfor %}
  ```

  示例2:

  ```
  # tempnginx.yml
  - hosts: websrvs
    remote_user: root
    vars:
      nginx_vhosts:
        - 80
        - 81
        - 82
    tasks:
      - name: template config
        template: src=nginx.conf2.j2 dest=/data/nginx.conf

  ```

  ```
  # nginx.conf2.j2
  {% for vhost in nginx_vhosts %}
      server {
          listen {{ vhost }}
      }
  {% endfor %}
  ```

  ```
  $ ansible-playbook -C tempnginx.yml --limit=192.168.0.21
  ```

  示例3：

```
# tempnginx2.yml
- hosts: websrvs
  remote_user: root
  vars:
    nginx_vhosts:
      - listen: 8080

  tasks:
    - name: template config
      template: src=nginx.conf3.j2 dest=/data/nginx.conf

```

```
# nginx.conf3.j2
{% for vhost in nginx_vhosts %}
    server {
        listen {{ vhost.listen }}
    }
{% endfor %}
```

示例4：

```
# tempnginx3.yml
- hosts: websrvs
  remote_user: root
  vars:
    nginx_vhosts:
      - listen: 8080
        server_name: "www.baidu.com"
        root: "/var/www/html"
      - listen: 8081
        server_name: "www.example.com"
        root: "/var/nginx/web1"

  tasks:
    - name: template config
      template: src=nginx.conf4.j2 dest=/data/nginx.conf

```

```
# nginx.conf4.j2
{% for vhost in nginx_vhosts %}
    server {
        listen {{ vhost.listen }}
        server_name {{ vhost.server_name }}
        root {{ vhost.root }}
    }
{% endfor %}
```

示例5：

```
$ cat hosts
[websrvs]
192.168.0.21
192.168.0.22
```

```
# nginx.conf5.j2
upstream webservers {
{% for i in groups['websrvs'] %}
    server {{i}}:{{http_port}}
{% endfor %}
```

6.3.2 IF判断

在模板文件中还可以使用if条件判断，决定是否生成相关的配置信息

- 示例1:

```
# tempnginx.yml
- hosts: websrvs
  remote_user: root
  vars:
    nginx_vhosts:
      - web1:
        listen: 8080
        root: "/var/www/nginx/web1/"
      - web2:
        listen: 8081
        root: "/var/www/nginx/web2/"
      - web3:
        listen: 8082
        root: "/var/www/nginx/web3/"
  tasks:
    - name: template config
      template: src=nginx.conf1.j2 dest=/data/nginx.conf
```

```
# templates/nginx.conf1.j2
{% for vhost in nginx_vhosts %}
server {
    listen {{ vhost.listen }}
    {% if vhost.server_name is defined %}
server_name {{ vhost.server_name }}   # 注意缩进
    {% endif %}
root {{ vhost.root }}
}
{% endfor %}
```

- 示例2：生成keepalived配置文件

```
vrrp_instrance VI_1 {
{% if ansible_fqdn == "ka1" %}
    state MASTER
    priority 100
{% elif ansible_fqdn == "ka2" %}
    state SLAVE
    priority 80
{% endif %}
......
}
```

6.4 循环迭代 with_items(loop)

- 迭代：当有需要重复性执行的任务时，可以使用迭代机制
- 对迭代项的引用，固定内置变量名为"item"
- 在task中使用with_items给定要迭代的元素列表
- 注意：ansible 2.5版本后，可以使用loop代替with_items

6.4.1 列表元素格式

- 字符串
- 字典
- 示例1：

```
- hosts: websrvs
  remote_user: root

  tasks:
    - name: add several users
```

```
      user: name={{ item }} state=present groups=wheel
      with_items:
        - testuser1
        - testuser2
        - testuser3
        - testuser4
        - testuser5

# user: name=testuser1 state=present groups=wheel
# user: name=testuser2 state=present groups=wheel
```

- 示例2：卸载mariadb

```
- hosts: websrvs
  remote_user: root

  tasks:
    - name: stop service
      shell: /etc/init.d/mysqld stop
    - name: delete files and dir
      file: path={{ item }} state=absent
      with_items:
        - /usr/local/mysql
        - /usr/local/mariadb-10.2.27-linux-x86_64
        - /etc/init.d/mysqld
        - /etc/profile.d/mysql.sh
        - /etc/my.cnf
        - /data/mysql
    - name: delete user
      user: name=mysql state=absent remove=yes
```

6.4.2 迭代嵌套子变量

- 示例1：

```
- hosts: websrvs
  remote_user: root

  tasks:
    - name: add some groups
      group: name={{ item }} state=present
      with_items:
        - nginx
        - mysql
        - apache
    - name: add some user
```

```
      user: name={{ item.user }} group={{ item.group }}
state=present
      with_items:
        - { user: "nginx", group: "nginx" }
        - { user: "mysql", group: "mysql" }
        - { user: "apache", group: "apache" }
```

- 示例2(loop):

```
- hosts: websrvs
  remote_user: root

  tasks:
    - name: create to hard links
      file:
        src: '/tmp/{{ item.src }}'
        dest: '{{ item.dest }}'
        state: hard
      loop:
        - { src: x, dest: y }
        - { src: z, dest: k }
```

- 示例3(until):

```
- hosts: websrvs
  remote_user: root
  gather_facts: false

  tasks:
    - debug: msg="until"
      until: false
      retries: 3
      delay: 1
```

- 示例4(with_lines逐行处理):

```
- hosts: localhost

  tasks:
    - debug: msg={{ item }}
      with_lines: ps aux
```

6.5 使用when

- when语句可以实现条件测试判断
- 如果需要根据变量、facts或此前任务执行结果来作为某个task执行与否的前提时，可以用到条件判断

- 通过task后添加when子句即可使用条件判断
- 示例1(jinja2语法格式):

```yaml
- hosts: websrvs
  remote_user: root
  tasks:
    - name: "shutdown RedHat flavored systems"
      command: /sbin/shutdown -h now
      when: ansible_os_family == "RedHat"
```

- 示例2(判断主机名):

```yaml
- hosts: websrvs
  remote_user: root
  tasks:
    - name: install packages
      yum: name=nginx
      when: ansible_fqdn is match("web*")
```

- 示例3(判断服务状态是否需要重启):

```yaml
- hosts: websrvs
  remote_user: root
  tasks:
    - name: check nginx service
      command: systemctl is-active nginx
      ignore_errors: yes
      register: check_nginx
    - name: Httpd Restart  # 如果check nginx执行成功 (check_nginx.rc == 0) , 则执行重启nginx, 否则跳过
      service: name=nginx state=restarted
      when: check_nginx.rc == 0
```

- 分组判断

```yaml
tasks:
  - name: "shutdown CentOS6 and Debin 7"
    command: /sbin/shutdown -h now
    when: (ansible_facts['distribution'] == "CentOS" and absible_facts['distribution_major_version'] == "6") or (ansible_facts['distribution'] == "Debian" and ansible_facts['distribution_major_version'] == "7")
```

- 列表形式表示and关系

```yaml
tasks:
  - name: shutdown centos7
    command: /sbin/shutdown -h now
    when:
      - ansible_facts['distribution'] == "CentOS"
      - absible_facts['distribution_major_version'] == "7"
```

- 判断是否定义

  ```yaml
  tasks:
    - debug: msg="undefined"
      when: bar is undefind
      # when: foo is undefind
  ```

- 和循环一起使用

  ```yaml
  tasks:
    - debug: msg="item > 3"
      with_items: [1,2,3,4,5]
      when: item > 3
  ```

- 判断执行状态

  ```yaml
  tasks:
    - command: /bin/true
      register: result
      ignore_errors: True
    - debug: msg="failed"
      when: result is failed
    - debug: msg="succeeded"
      when: result is succeeded
    - debug: msg='skipped'
      when: result is skipped
  ```

- failed_when 满足条件时，使任务失败

  ```yaml
  tasks:
    - command: echo faild
      register: result
      failed_when: "'faild' in result.stdout"
      # failed_when: false  不满足条件，任务正常执行
      # failed_when: true  满足条件，使任务失败
    - debug: msg="echo failed_when"
  ```

6.6 分组block

当想满足一个条件下，执行多个任务时，需要分组，不再是每个任务都设置when

- 示例:

  ```yaml
  tasks:
    - block:
        - debug: msg="first"
        - debug: msg="second"
      when:
        - ansible_facts['distribution'] == "CentOS"
        - absible_facts['distribution_major_version'] == "7"
  ```

6.7 changed_when

6.7.1 关闭changed状态

当确定某个task不会对被控制端做修改时，执行结果却显示黄色的changed状态，可以通过chenged_when: false 关闭changed状态

```
tasks:
  - name: check sshd service
    shell: ps aux | grep sshd
    changed_when: false
```

6.7.2 利用chenged_when检查task返回结果

根据chenged_when检查task返回结果，决定是否继续向下执行

```
tasks:
  - name: install nginx
    yum: name=nginx
  - name: config file
    template: src="nginx.conf.j2" dest="/etc/nginx/nginx.conf"
    notify: restart nginx
  - name: check config
    shell: /use/sbin/nginx -t
    register: check_nginx_config
    changed_when:
      - (check_nginx_config.stdout.find('successful'))  # 执行结果中如果
有successful则继续执行，如果没有则停止向下执行
      - false   # nginx -t 每次执行成功是changed状态，关闭此状态
  - name: start nginx
    service: name=nginx state=restarted enabled=yes
handlers:
  - name: restart nginx
    service: name=nginx state=restarted
```

6.8 滚动执行

- 当管理节点过多时，会导致超时问题
- 默认情况下，ansible将尝试并行管理playbook中所有机器

- 对于滚动跟新用例，可以使用 `serial` 关键字定义ansible一次应管理多少主机，可以将 `serial` 关键字指定为百分比，表示每次并行执行的主机数占总数比例
- 示例1:

```
- hosts: all
  serial: 2 # 每次只同时处理2个主机，将所有task执行完成后，再选下2个主机执行所有task，直至所有主机执行完成
```

- 示例2：

```
- hosts: all
  serial: "20%"   # 每次只同时处理20%的主机
```

7. 实战案例

利用**playbook**实现批量编译安装部署**httpd-2.4**

```
# install_httpd.yml
- hosts: websrvs
  remote_user: root

  vars:
    download_dir: /usr/local/src
    install_dir: /apps/httpd
    httpd_version: httpd-2.4.46
    apr_version: apr-1.7.0
    apr_util_version: apr-util-1.6.1
    httpd_url: https://mirrors.cloud.tencent.com/apache/httpd
    apr_url: https://mirrors.cloud.tencent.com/apache/apr

  tasks:
    - name: install packages
      yum: name=gcc,make,pcre-devel,openssl-devel,expat-devel,bzip2 state=installed
    - name: download httpd file
      unarchive: src="{{ httpd_url }}/{{ httpd_version }}.tar.bz2" dest={{ download_dir }} owner=root remote_src=yes
    - name: download apr file
      unarchive: src="{{ apr_url }}/{{ apr_version }}.tar.bz2" dest={{ download_dir }} owner=root remote_src=yes
    - name: download apr-util file
      unarchive: src="{{ apr_url }}/{{ apr_util_version }}.tar.bz2" dest={{ download_dir }} owner=root remote_src=yes
    - name: prepare apr dir
      shell: chdir={{ download_dir }} mv {{ apr_version }} {{
download_dir }}/{{ httpd_version }}/srclib/apr
    - name: prepare apr-util dir
      shell: chdir={{ download_dir }} mv {{ apr_util_version }} {{
download_dir }}/{{ httpd_version }}/srclib/apr-util
    - name: create install dir
      file: name={{ install_dir }} state=directory
      ignore_errors: yes
    - name: build httpd
      shell: chdir={{ download_dir }}/{{ httpd_version }} ./configure
--prefix={{ install_dir }} --enable-so --enable-ssl --enable-cgi --
enable-rewrite --with-zlib --with-pcre --with-included-apr --enable-
modules=most --enable-mpms-shared=all --with-mpm=prefork && make -j {{
ansible_processor_vcpus }} && make install
    - name: create group
      group: name=apache gid=80 system=yes
    - name: create user
      user: name=apache uid=80 group=apache system=yes
shell=/sbin/nologin create_home=no home={{ install_dir }}/conf/httpd
    - name: set httpd user
      lineinfile: path={{ install_dir }}/conf/httpd.conf
regexp='^User' line='User apache'
    - name: set httpd group
      lineinfile: path={{ install_dir }}/conf/httpd.conf
regexp='^Group' line='Group apache'
    - name: set variable PATH
      shell: echo PATH={{ install_dir }}/bin:$PATH >>
/etc/profile.d/httpd.sh
    - name: prepare service file
      template: src=httpd.service.j2
dest=/usr/lib/systemd/system/httpd.service
    - name: start service
      service: name=httpd state=started enabled=yes
```

```
{# httpd.service.j2 #}
[Unit]
Description=The Apache HTTP Server
After=network.target remote-fs.target nss-lookup.target
Documentation=man:httpd(8)
Documentation=man:apachectl(8)

[Service]
Type=forking
# EnvironmentFile=/etc/sysconfig/httpd
```

```
ExecStart={{ install_dir }}/bin/apachectl start
# ExecStart={{ install_dir }}/bin/httpd $OPTIONS -k start
ExecReload={{ install_dir }}/bin/apachectl graceful
# ExecReload={{ install_dir }}/bin/httpd $OPTIONS -k graceful
ExecStop={{ install_dir }}/bin/apachectl stop
killSignal=SIGCONT
PrivateTmp=true

[Install]
WantedBy=multi-user.target
```

8. Role角色

- roles是在ansible中，playbooks的目录组织结构。而模块化之后，成为roles的组织结构，易读，代码可重用，层次清晰。
- roles能够根据层次型结构自动装载变量文件、tasks以及handlers等。

8.1 案例：

假设有2台服务器，一台需要部署nginx，一台需要部署mysql，该如何定义playbook呢？

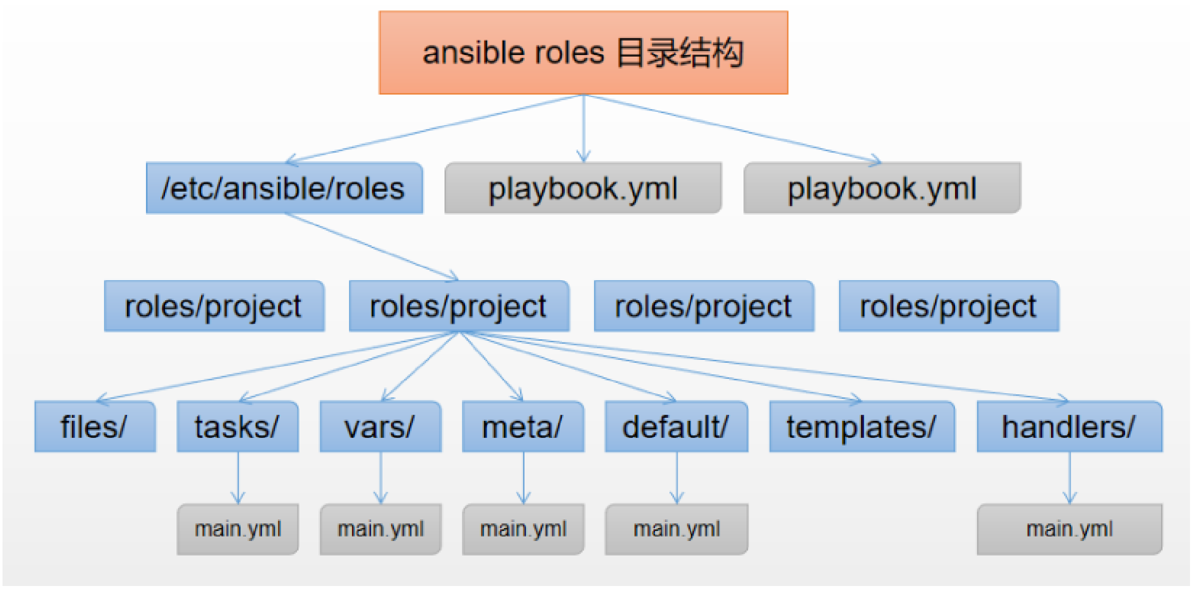
可能会想到分别写2个playbook文件应用到对应的服务器上，但是这些playbook不利于维护，也不利于模块化调用，比如，后来又增加一台服务器，这台服务器既是nginx服务器，又是mysql服务器，我们只能再写一个playbook文件来完成需求，这样playbook中的代码就重复了。

为了避免代码重复，roles能够实现代码重复被调用。定义一个角色叫websrvs，第二个角色叫dbsrvs。那么调用时如下来调用：

```
- hosts: host1
  role:
    - websrvs

- hosts: host2
  role:
    - dbsrvs

- hosts: host3
  role:
    - websrvs
    - dbsrvs
```

这样代码就可以重复利用了,每个角色可以被独立重复调用。下面举例说明使用方式。

8.2 Roles目录结构



在 任意目录下创建一个playbook.yml入口文件，定义如果使用role，同级有个roles的目录，里面的每个目录代表一个role，role目录中包含以下目录，这些目录必须包含一个main.yaml 文件。

```
./
├── playbook.yaml
└── roles
    ├── dbsrvs
    │   ├── files
    │   ├── handlers
    │   ├── meta
    │   ├── tasks
    │   ├── templates
    │   └── vars
    └── websrvs
        ├── files
        ├── handlers
        ├── meta
        ├── tasks
        ├── templates
        └── vars

15 directories, 1 file
```

- tasks：存储任务的目录,此目录中至少应该有一个名为main.yml的文件，用于定义各task；其它的文件需要由main.yml进行“包含”调用；
- handlers：此目录中至少应该有一个名为main.yml的文件，用于定义各handler；其它的文件需要由（与notify:名字相同，方便notify通知执行下一条命令）通过main.yml进行“包含”调用；
- vars：此目录中至少应该有一个名为main.yml的文件，用于定义各variable；其它的文件需要由main.yml进行“包含”调用；
- files：存储一些可以用copy调用的静态文件。
- templates：存储由template模块调用的模板文本；（也可以调用变量）
- meta：如果你想要赋予这个角色一些元数据，则可以将元数据写入到meta/main.yml文件中，这些元数据用于描述角色的相关属性，比如 作者信息、角色主要作用等等(非必须)

8.3 基本使用

1. 编写tasks任务列表

```
# roles/websrvs/tasks/main.yaml
- name: install package
  debug: msg="install nginx"
- name: copy nginx.conf to /tmp/nginx.conf
  copy: src=nginx.conf dest=/tmp/nginx.conf
- name: start server
  debug: msg="start server"
```

2. 上传nginx.conf 文件到files目录
3. 编写playbook.yaml文件

```yaml
- hosts: websrvs
  remote_user: root
  roles:
    - websrvs
```

4. 执行测试:

```
$ ansible-playbook playbook.yaml
PLAY [websrvs]
*****************************************************************
*****************************************************************
*****************************************************************

TASK [Gathering Facts]
*****************************************************************
*****************************************************************
*********************************************************
ok: [node01]

TASK [websrvs : install package]
*****************************************************************
*****************************************************************
***********************************************
ok: [node01] => {
    "msg": "install nginx"
}

TASK [websrvs : copy nginx.conf to /tmp/nginx.conf]
*****************************************************************
*****************************************************************
****************************
changed: [node01]

TASK [websrvs : start server]
*****************************************************************
*****************************************************************
**************************************************
ok: [node01] => {
    "msg": "start server"
}

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

```
node01                     : ok=4    changed=1    unreachable=0
 failed=0    skipped=0    rescued=0    ignored=0
```

- 注意： 此处可以使用 `ansible-galaxy role init test` 命令生成roles目录结构

8.4 实战案例

利用Role实现批量编译部署http-2.4

```
$ tree ./
./
├── roles
│   └── httpd
│       ├── defaults
│       │   └── main.yml
│       ├── files
│       │   └── httpd.conf
│       ├── handlers
│       │   └── main.yml
│       ├── meta
│       │   └── main.yml
│       ├── README.md
│       ├── tasks
│       │   └── main.yml
│       ├── templates
│       │   └── httpd.service.j2
│       ├── tests
│       │   ├── inventory
│       │   └── test.yml
│       └── vars
│           └── main.yml
└── site.yaml

10 directories, 11 files
```

```
# site.yaml
- hosts: websrvs
  remote_user: root
  roles:
    - httpd
```

```
# roles/httpd/tasks/main.yml
```

```yaml
---
# tasks file for httpd
- name: install packages
  yum: name=gcc,make,pcre-devel,openssl-devel,expat-devel,bzip2
state=installed
- name: download httpd file
  unarchive: src="{{ httpd_url }}/{{ httpd_version }}.tar.bz2" dest={{
download_dir }} owner=root remote_src=yes
- name: download apr file
  unarchive: src="{{ apr_url }}/{{ apr_version }}.tar.bz2" dest={{
download_dir }} owner=root remote_src=yes
- name: download apr-util file
  unarchive: src="{{ apr_url }}/{{ apr_util_version }}.tar.bz2" dest=
{{ download_dir }} owner=root remote_src=yes
- name: prepare apr dir
  shell: chdir={{ download_dir }} mv {{ apr_version }} {{ download_dir
}}/{{ httpd_version }}/srclib/apr
- name: prepare apr-util dir
  shell: chdir={{ download_dir }} mv {{ apr_util_version }} {{
download_dir }}/{{ httpd_version }}/srclib/apr-util
- name: create install dir
  file: name={{ install_dir }} state=directory
  ignore_errors: yes
- name: build httpd
  shell: chdir={{ download_dir }}/{{ httpd_version }} ./configure --
prefix={{ install_dir }} --enable-so --enable-ssl --enable-cgi --
enable-rewrite --with-zlib --with-pcre --with-included-apr --enable-
modules=most --enable-mpms-shared=all --with-mpm=prefork && make -j {{
ansible_processor_vcpus }} && make install
- name: create group
  group: name=apache gid=80 system=yes
- name: create user
  user: name=apache uid=80 group=apache system=yes shell=/sbin/nologin
create_home=no home={{ install_dir }}/conf/httpd
- name: set httpd user
  lineinfile: path={{ install_dir }}/conf/httpd.conf regexp='^User'
line='User apache'
- name: set httpd group
  lineinfile: path={{ install_dir }}/conf/httpd.conf regexp='^Group'
line='Group apache'
- name: set variable PATH
  shell: echo PATH={{ install_dir }}/bin:$PATH >>
/etc/profile.d/httpd.sh
- name: prepare service file
```

```
  template: src=httpd.service.j2
dest=/usr/lib/systemd/system/httpd.service
- name: copy config file
  copy: src=httpd.conf dest={{ install_dir }}/conf/httpd.conf
  notify: restart httpd
  tags: config
- name: start service
  service: name=httpd state=started enabled=yes
```

```
# roles/httpd/vars/main.yml
---
# vars file for httpd
download_dir: /usr/local/src
install_dir: /apps/httpd
httpd_version: httpd-2.4.46
apr_version: apr-1.7.0
apr_util_version: apr-util-1.6.1
httpd_url: https://mirrors.cloud.tencent.com/apache/httpd
apr_url: https://mirrors.cloud.tencent.com/apache/apr
```

```
# roles/httpd/handlers/main.yml
---
# handlers file for httpd
- name: restart httpd
  service: name=httpd state=restarted
```

```
{# roles/httpd/templates/httpd.service.j2 #}
[Unit]
Description=The Apache HTTP Server
After=network.target remote-fs.target nss-lookup.target
Documentation=man:httpd(8)
Documentation=man:apachectl(8)

[Service]
Type=forking
# EnvironmentFile=/etc/sysconfig/httpd
ExecStart={{ install_dir }}/bin/apachectl start
# ExecStart={{ install_dir }}/bin/httpd $OPTIONS -k start
ExecReload={{ install_dir }}/bin/apachectl graceful
# ExecReload={{ install_dir }}/bin/httpd $OPTIONS -k graceful
ExecStop={{ install_dir }}/bin/apachectl stop
killSignal=SIGCONT
PrivateTmp=true

[Install]
WantedBy=multi-user.target
```

```
# 执行ansible命令运行role对象完成http部署
$ ansible-playbook site.yaml
```

9. 作业

阿良教育： www.aliangedu.cn

- 改造shell脚本 -- 一键部署LNMP

```
#!/bin/bash
NGINX_V=1.15.6
PHP_V=5.6.36
TMP_DIR=/tmp

INSTALL_DIR=/usr/local

PWD_C=$PWD

echo
echo -e "\tMenu\n"
echo -e "1. Install Nginx"
echo -e "2. Install PHP"
echo -e "3. Install MySQL"
echo -e "4. Deploy LNMP"
echo -e "9. Quit"

function command_status_check() {
    if [ $? -ne 0 ]; then
        echo $1
        exit
    fi
}

function install_nginx() {
    cd $TMP_DIR
    yum install -y gcc gcc-c++ make openssl-devel pcre-devel wget
    wget http://nginx.org/download/nginx-${NGINX_V}.tar.gz
    tar zxf nginx-${NGINX_V}.tar.gz
    cd nginx-${NGINX_V}
    ./configure --prefix=$INSTALL_DIR/nginx \
    --with-http_ssl_module \
    --with-http_stub_status_module \
    --with-stream
    command_status_check "Nginx - 平台环境检查失败！"
    make -j 4
    command_status_check "Nginx - 编译失败！"
    make install
    command_status_check "Nginx - 安装失败！"
    mkdir -p $INSTALL_DIR/nginx/conf/vhost
    alias cp=cp ; cp -rf $PWD_C/nginx.conf $INSTALL_DIR/nginx/conf
    rm -rf $INSTALL_DIR/nginx/html/*
    echo "ok" > $INSTALL_DIR/nginx/html/status.html
    echo '<?php echo "ok"?>' > $INSTALL_DIR/nginx/html/status.php
    $INSTALL_DIR/nginx/sbin/nginx
    command_status_check "Nginx - 启动失败！"
}

function install_php() {
    cd $TMP_DIR
    yum install -y gcc gcc-c++ make gd-devel libxml2-devel \
        libcurl-devel libjpeg-devel libpng-devel openssl-devel \
        libmcrypt-devel libxslt-devel libtidy-devel
    wget http://docs.php.net/distributions/php-${PHP_V}.tar.gz
    tar zxf php-${PHP_V}.tar.gz
    cd php-${PHP_V}
    ./configure --prefix=$INSTALL_DIR/php \
    --with-config-file-path=$INSTALL_DIR/php/etc \
    --enable-fpm --enable-opcache \
    --with-mysql --with-mysqli --with-pdo-mysql \
    --with-openssl --with-zlib --with-curl --with-gd \
    --with-jpeg-dir --with-png-dir --with-freetype-dir \
    --enable-mbstring --enable-hash
    command_status_check "PHP - 平台环境检查失败！"
    make -j 4
    command_status_check "PHP - 编译失败！"
    make install
    command_status_check "PHP - 安装失败！"
    cp php.ini-production $INSTALL_DIR/php/etc/php.ini
    cp sapi/fpm/php-fpm.conf $INSTALL_DIR/php/etc/php-fpm.conf
    cp sapi/fpm/init.d.php-fpm /etc/init.d/php-fpm
    chmod +x /etc/init.d/php-fpm
    /etc/init.d/php-fpm start
    command_status_check "PHP - 启动失败！"
}

read -p "请输入编号：" number
```

```
case $number in
    1)
        install_nginx;;
    2)
        install_php;;
    3)
        install_mysql;;
    4)
        install_nginx
        install_php
        ;;
    9)
        exit;;
esac
```