**Ansible自动化运维**

准备6台虚拟机

control 192.168.4.253/24

node1 192.168.4.11/24

node2 192.168.4.12/24

node3 192.168.4.13/24

node4 192.168.4.14/24

node5 192.168.4.15/24

在control主机操作：

#修改主机名解析文件

[root@control ~]# dnf -y install vim

[root@control ~]# vim /etc/hosts

… …

192.168.4.253 control

192.268.4.1 node1

192.168.4.2 node2

192.168.4.3 node3

192.168.4.4 node4

192.168.4.5 node5

#生成ssh密钥，实现无密码连接所有node主机

[root@control ~]# ssh-keygen -f /root/.ssh/id\_rsa -N ''

[root@control ~]# for i in node1 node2 node3 node4 node5

do

ssh-copy-id $i

done

#安装ansible软件包

[root@control ~]# tar -xf ansible\_soft.tar.gz

[root@control ~]# cd ansible\_soft

[root@control ansible\_soft]# dnf -y install \*

#创建ansible的配置文件

#(可以参考/etc/ansible/ansible.cfg)

[root@control ~]# mkdir ~/ansible

[root@control ~]# vim ~/ansible/ansible.cfg

[defaults]

inventory = ~/ansible/hosts #被控制端主机清单

#forks = 5 #ssh并发数量

#ask\_pass = True #使用密钥还是密码远程

#host\_key\_checking = False #是否校验密钥

#创建一个被控制端主机的清单文件

#[参考/etc/ansible/hosts]

[root@control ~]# vim ~/ansible/hosts

[test] #定义主机组（组名称任意）

node1 #定义组中的具体主机

[proxy] #定义一个组名称叫proxy[任意字符]

node2 #这个组里有一台主机node2

[webserver] #定义一个webserver组

node[3:4] #在这个组里有两台主机node3和node4

[database]

node5

[cluster:children] #嵌套组（children为关键字）

webserver

database

#嵌套组就是在组中包含其他组

++++++++++++++++++++++++++++++++++++++++++++

使用ansible管理其他主机

++++++++++++++++++++++++++++++++++++++++++++

使用ansible管理其他主机有两种方式:

1.命令行执行ansible ad-hoc命令

2.把要做的动作行为写入一个文件[playbook剧本]

ansible读取剧本自动完成相应的任务.

Ansible ad-hoc是一种通过命令行批量管理的方式，

命令基本格式如下：

# ansible 主机集合 -m 模块名 -a "参数"

其他参数：-k使用密码远程、-i指定主机列表文件

#-m后面加模块[module]

#-a后面是给模块传递参数[argument]

#### 注意事项 ####

执行ansible命令一定要先cd到ansible目录

#### 注意事项 ####

[root@control ~]# cd ~/ansible

[root@control ansible]# ansible all --list-host

#all是ansible自带的已经定义好的特殊的组，所有主机都在这个组中.

第一个ansible模块【ping模块】

[root@control ansible]# ansible node1 -m ping

#可以使用单独的某一台主机的域名

[root@control ansible]# ansible test -m ping

#也可以使用hosts中定义的组名称

[root@control ansible]# ansible webserver -m ping

[root@control ansible]# ansible all -m ping

#################################

失败检查:

检查/etc/hosts

检查~/ansible/ansible.cfg

检查~/ansible/hosts

#################################

多数脚本都支持参数，默认模块为command。

[root@control ansible]# ansible node1 -m command -a "uptime"

[root@control ansible]# ansible node1 -m command -a "uname -r"

[root@control ansible]# ansible node1 -m command -a "ip a s eth0"

[root@control ansible]# ansible node1 -a "ip a s eth0"

[root@control ansible]# ansible all -a "date"

#注意这里的网卡名称可能不一样!

通过ansible-doc获取帮助。

[root@control ansible]# ansible-doc -l #列出所有模块

[root@control ansible]# ansible-doc -l | wc -l

[root@control ansible]# ansible-doc command #查看具体模块的帮助

command和shell模块的区别，command模块的命令不启动shell,直接通过ssh执行命令，command不支持bash的特性，如管道和重定向等功能，所有需要调用shell的功能都无法使用。不可以使用shell模块执行交互命令，如vim、top等。

[root@control ansible]# ansible test -m command -a "ps | wc -l" #报错

[root@control ansible]# ansible test -m command -a "ls &" #报错

[root@control ansible]# ansible test -m shell -a "ps aux | wc -l" #进程数量

[root@control ansible]# ansible test -m shell -a "who" #登陆信息

[root@control ansible]# ansible test -m shell -a "touch /tmp/txt.txt"

注意：Ansible使用SSH远程连接被管理主机，退出SSH后所有状态失效。

[root@control ansible]# ansible test -m shell -a "cd /tmp"

#切换目录

[root@control ansible]# ansible test -m shell -a "touch my.txt"

#创建文件（注意：没有指定路径，是在家目录创建的文件）

[root@control ansible]# ansible test -m shell -a "ls /tmp/my.txt"

#查看文件失败

[root@control ansible]# ansible test -m shell -a "chdir=/tmp touch b.txt"

#使用chdir参数切换工作目录

shell模块支持判断(creates、removes），格式如下：

creates 文件名：文件存在，不执行shell命令，

removes 文件名：文件不存在，不执行shell命令。

[root@control ansible]# ansible test -m shell -a "touch /tmp/b.txt creates=/tmp/b.txt"

#ansible会远程test主机，判断目标主机有没有/tmp/b.txt文件

#如果有该文件则ansible对应的shell命令不执行

#如果没有该文件则ansible对应的shell命令执行

假设我有一个xx.zip压缩包

#ansible test -m shell -a "unzip /root/xx.zip"

#远程test主机，执行unzip命令解压一个压缩包文件

#ansible test -m shell -a "unzip /root/xx.zip removes=/bin/unzip"

#ansible会远程test主机，判断目标主机有没有/bin/unzip程序

#如果没有这个程序，则不执行shell对应的命令

#如果有这个程序，则执行对应的shell命令

#######################

script模块

#######################

在控制端主机可以任意写脚本,ansible自动远程被管理端主机，拷贝这个脚本，到目标主机执行这个脚本.

[root@control ansible]# vim /root/test.sh

#!/bin/bash

dnf -y install httpd

systemctl start httpd

[root@control ansible]# ansible test -m script -a "/root/test.sh"

##################

file模块

##################

file模块可以创建文件、目录、链接；修改权限与属性等

很多ansible模块都具有【幂等性】的特征。

幂等性：任意次执行所产生的影响均与一次执行的影响相同。

[root@control ansible]# ansible test -m file \

-a "path=/tmp/file.txt state=touch" #新建文件

[root@control ansible]# ansible test -m file \

-a "path=/tmp/mydir state=directory"

#创建目录

[root@control ansible]# ansible test -m file \

-a "path=/tmp/file.txt owner=sshd group=adm mode=0777"

#修改文件或目录权限

#owner=sshd是修改文件的所有者【sshd是用户名】

#group=adm是修改文件的所属组【adm是组名】

#mode=0777是修改文件的权限

[root@control ansible]# ansible test -m file \

-a "path=/tmp/mydir state=absent"

#state=absent是删除文件或目录

[root@control ansible]# ansible test -m file \

-a "path=/tmp/file.txt state=absent"

#state=absent是删除文件或目录

[root@control ansible]# ansible test -m file \

-a "src=/etc/hosts path=/tmp/host.txt state=link"

#给/etc/hosts文件创建一个链接文件/tmp/host.txt

#############

copy模块

#############

拷贝文件的模块

[root@control ~]# cd ansible

[root@control ansible]# echo AAA > ~/a3.txt #新建测试文件

[root@control ansible]# ansible test -m copy \

-a "src=~/a3.txt dest=/root/"

#src是源文件，dest是目标文件

[root@control ansible]# echo BBB > ~/a3.txt

#修改a3.txt文件

[root@control ansible]# ansible test -m copy \

-a "src=~/a3.txt dest=/root/"

#因为内容修改了，会重新拷贝一次.

[root@control ansible]# echo CCC > ~/a3.txt

#再次修改文件内容

#copy模块可以将文件拷贝到远程主机，如果目标主机有同名文件时，通过定义backup=yes则先备份再拷贝。

[root@control ansible]# ansible test -m copy \

-a "src=~/a3.txt dest=/root/a3.txt backup=yes"

#################

fetch模块

#################

把被管理端主机的文件拷贝过来到ansible管理主机

fetch模块与copy类似，但是作用相反,可以将其他主机的文件拷贝到本地。

[root@control ansible]# ansible all -m fetch \

-a "src=/etc/hostname dest=/tmp"

#把所有5台被管理端主机的hostname文件拷贝到本机的/tmp目录

###########################

在修改单个文件的单行内容时可以使用lineinfile模块。

###########################

[root@control ansible]# ansible test -m lineinfile \

-a "path=/etc/issue line='hello world'"

#在/etc/issue文件中添加一行内容hello world，默认添加到最后

[root@control ansible]# ansible test -m lineinfile \

-a "path=/etc/issue line='helle world'"

#基于幂等原则，重复执行，不会创建多行内容

[root@control ansible]# ansible test -m lineinfile \

-a "path=/etc/issue line='insert' insertafter='Kernel'"

#将内容插入到Kernel行的后面

[root@control ansible]# ansible test -m lineinfile \

-a "path=/etc/issue regexp='hello' line='ni hao'"

#在/etc/issue文件中正则匹配包含hello的行，把整行内容替换为ni hao

#如果无法匹配到hello，则在文件最后添加一行nihao

#如果有多行内容包含hello，则仅替换最后一行

################################

replace模块可以替换所有关键词

################################

[root@control ansible]# ansible test -m replace \

-a "path=/etc/issue.net regexp=Kernel replace=Ocean"

#将/etc/issue文件全文所有的Kernel替换为Ocean

###########################

user模块

###########################

[root@control ansible]# ansible test -m user \

-a "name=tuser1"

#远程test组中的所有主机并创建系统账户tuser1

[root@control ansible]# ansible test -m user -a \

"name=tuser2 uid=1010 group=adm groups=daemon,root \

home=/home/tuser2"

#创建账户并设置对应的账户属性

#uid设置用户ID

#group设置用户的基本组

#groups设置用户的附加组

#home指定用户的家目录

#在linux中可以用户信息cat /etc/passwd

#在linux中查看组信息cat /etc/group

[root@control ansible]# ansible test -m user -a \

"name=tuser1 \

password={{'abc'| password\_hash('sha512')}}"

#修改账户密码,密码是abc，密码需要经过sha512算法加密

#给tuser1用户修改密码

#password\_hash的意思是密码要加密

#sha512是加密的算法

[root@control ansible]# ansible test -m user \

-a "name=tuser1 groups=root,daemon"

#修改tuser1账户的附加组

[root@control ansible]# ansible test -m user \

-a "name=tuser1 state=absent"

#删除账户tuser1

[root@control ansible]# ansible test -m user \

-a "name=tuser2 state=absent remove=true"

#删除tuser2账户同时删除家目录、邮箱

###########################

yum\_repository模块

###########################

创建yum配置文件的模块(/etc/yum.repos.d/文件)

[root@control ansible]# ansible test -m yum\_repository \

-a "name=myyum description=hello \

baseurl=ftp://192.168.4.254/centos \

gpgcheck=no"

#新建一个yum源配置文件/etc/yum.repos.d/myyum.repo

#yum源文件名为myyum，该文件的内容如下：

[myyum]

baseurl = ftp://192.168.4.254/centos

gpgcheck = 0

name = hello

[root@control ansible]# ansible test \

-m yum\_repository -a "name=myyum state=absent"

#删除上面创建的myyum这个yum配置文件.

######################

yum模块

######################

#管理软件包的模块，安装，卸载，升级... ...

state: present(安装)|absent(卸载)|latest(升级)。

[root@control ansible]# ansible test -m yum \

-a "name=unzip state=present"

#安装unzip软件包

[root@control ansible]# ansible test -m yum \

-a "name=unzip state=latest"

#升级unzip软件包，软件名称可以是\*，代表升级所有软件包

[root@control ansible]# ansible test -m yum \

-a "name=unzip state=absent"

#卸载unzip软件包

#提示：如果name指定的软件名称是以@开始，则代表组包.

##########################

service模块

##########################

service为服务管理模块（启动、关闭、重启服务等），

state:started | stopped | restarted，

enabled:yes设置开机启动。

[root@control ansible]# ansible test -m yum \

-a "name=httpd"

#使用yum模块安装软件包

[root@control ansible]# ansible test -m service \

-a "name=httpd state=started"

#启动httpd服务

[root@control ansible]# ansible test -m service \

-a "name=httpd state=stopped"

#关闭httpd服务

[root@control ansible]# ansible test -m service \

-a "name=httpd state=restarted"

#重启httpd服务

[root@control ansible]# ansible test -m service \

-a "name=httpd enabled=yes"

#设置httpd为开机自启动

#准备实验环境:（在node1主机操作）

#关闭node1这台虚拟机,

#在vmware设置里面给这台虚拟机添加一块磁盘

#重启虚拟机，使用lsblk查看是否有磁盘

[root@node1 ~]# lsblk

[root@node1 ~]# parted /dev/sdb mklabel gpt

#parted 硬盘名称 mklabel 分区表类型(mrb或者gpt)

[root@node1 ~]# parted /dev/sdb mkpart primary 1 2G

#parted 硬盘名称 mkpart primary 分区开始位置 结束位置

#mkpart是创建分区,primary是主分区

[root@node1 ~]# lsblk

#注意事项：每个同学的硬盘名称可能不同，注意根据情况修改

##########################

lvg模块:

##########################

创建、删除卷组(VG)，修改卷组大小，

state:present(创建)|absent(删除)。

[root@control ansible]# ansible test -m yum -a "name=lvm2"

#安装lvm2软件包

[root@control ansible]# ansible test -m lvg -a "vg=myvg pvs=/dev/sdb1"

#调用lvg模块

#vg=是需要创建的VG名称

#PVS=是指定由那个磁盘或分区创建VG

【在node1主机再给磁盘分一个区】

[root@node1 ~]# parted /dev/sdb mkpart primary 2G 4G

【在control主机通过ansible给node1的myvg卷组扩容】

[root@control ansible]# ansible test -m lvg -a "vg=myvg pvs=/dev/sdb1,/dev/sdb2"

#######################

lvol模块

#######################

创建、删除逻辑卷(LV)，修改逻辑卷大小，

state:present(创建)|absent(删除)。

[root@control ansible]# ansible test -m lvol \

-a "lv=mylv vg=myvg size=1G"

#调用lvol模块创建逻辑卷

#lv=指定逻辑卷的名称

#vg=指定卷组的名称

#size=指定逻辑卷的大小

[root@control ansible]# ansible test -m lvol \

-a "lv=mylv vg=myvg size=2G"

#把前面创建的逻辑卷mylv扩容到2G容量.

[root@control ansible]# ansible test -m lvol \

-a "lv=mylv vg=myvg state=absent force=yes"

#删除前面创建的逻辑卷mylv

[root@control ansible]# ansible test -m lvg -a "vg=myvg state=absent"

#删除前面创建的卷组myvg

**sudo给普通用户授权**

【root可以提前给普通用户授权，授权普通用户执行特定的管理命令】

比如root可以给tom授权，让他可以执行systemctl命令

root可以给jerry授权，让他可以执行parted命令

root systemctl ---->tom

root parted ---->jerry

sudo基本流程如下：

1.管理员需要先授权（修改/etc/sudoers文件）

2.普通用户以sudo的形式执行命令

3.可以通过sudo -l查看授权情况

修改/etc/sudoers的方法如下：

visudo（带语法检查，默认没有颜色提示）

vim /etc/sudoers（不带语法检查，默认有颜色提示）

授权格式如下：

用户或组 主机列表=(提权身份) [NOPASSWD]:命令列表

注意事项：命令需要写绝对路径。

#NOPASSWD的意思是执行sudo不需要输入自己的密码.

下面我们通过ansible批量给所有被管理主机node1-node5

修改/etc/sudoers

[root@control ~]# ansible all -m user -a "name=alice \

password={{'123456' | password\_hash('sha512')}}"

#批量创建系统账户，账户名称为alice，密码为123456。

[root@control ~]# ansible all -m lineinfile \

-a "path=/etc/sudoers line='alice ALL=(ALL) /usr/bin/systemctl'"

#调用lineinfile模块，修改/etc/sudoers文件

#在该文件的末尾添加一行新的内容,内容是line=后面的内容

[验证]

可以任意找一台node主机，su - alice切换用户

执行sudo systemctl restart 服务名

[root@control ~]# ansible all -m lineinfile \

-a "path=/etc/sudoers line='alice ALL=(ALL) NOPASSWD:ALL'"

######################

修改ansible配置文件

######################

默认ansible是通过root远程被管理端主机

可以修改配置文件以普通用户远程被管理端主机执行自动化运维

#可以参考/etc/ansible/ansible.cfg#

[root@control ansible]# vim ~/ansible/ansible.cfg

[defaults]

inventory = ~/ansible/hosts

remote\_user = alice #以什么用户远程被管理主机

#host\_key\_checking = False #是否校验密钥

[privilege\_escalation]

become = true #是否需要切换用户

become\_method = sudo #如何切换用户

become\_user = root #切换成什么用户

become\_ask\_pass = no #sudo是是否需要输入密码

#为了让ansible以后可以通过alice远程被管理端主机

#我们需要给alice用户远程ssh密钥

[root@control ~]# for i in node1 node2 node3 node4 node5

do

ssh-copy-id alice@$i

done

###############################

修改主机清单文件

###############################

修改inventory主机清单配置文件（参考即可，非必须的操作）

如果个别主机账户不同该如何处理呢？如何有些主机需要使用密码呢？有些主机的SSH端口不是22呢？

[root@control ~]# cat ~/ansible/hosts

[test]

node1 ansible\_ssh\_port=220 #自定义远程SSH端口

[proxy]

node2 ansible\_ssh\_user=alice #自定义远程连接的账户名

[webserver]

node[3:4] ansible\_ssh\_pass=密码 #自定义远程连接的密码

[database]

node5 ansible\_ssh\_private\_key\_file=密钥文件 #自定义远程连接的密钥

[cluster:children]

webserver

database

++++++++++++++++++++++++++++++++++++++

**Playbook剧本**

++++++++++++++++++++++++++++++++++++++

剧本的样式，格式参考课程PPT

[root@control ansible]# vim test.yml

---

- hosts: all #定义需要远程管理的主机是谁

tasks: #定义需要执行的任务是什么

- name: ping all #给第一个任务取个名称

ping: #第一个任务调用ping模块

[root@control ansible]# ansible-playbook test.yml

hosts由一个或多个组或主机组成，逗号分隔，tasks由一个或多个任务组成，多个任务按顺序执行，可以使用-f选项自定义并发量。

[root@control ansible]# vim ~/ansible/test.yml

- hosts: test,webserver #远程谁

tasks:

- name: This is my first playbook #定义名字

ping: #模块

- name: Run a shell command #第二个任务

shell: touch ~/shell.txt #模块

[root@control ansible]# ansible-playbook ~/ansible/test.yml -f 5

#############################

多个play的Playbook文件

#############################

[root@control ansible]# vim ~/ansible/test.yml

#第一个play

---

- hosts: test

tasks:

- name: This is first play

ping:

#第二个play

- hosts: webserver

tasks:

- name: This is second play

ping:

为了更好的让vim自动使用两个空格缩进可以做如下操作:

修改VIM配置文件，自动使用空格替换tab键。

[root@control ansible]# vim ~/.vimrc

autocmd FileType yaml setlocal ai ts=2 sw=2 et

#使用2个空格自动替换tab键，tabstop=2、expandtab

#开启自动缩进对齐，缩进宽度为2个空格，shiftwidth=2

[root@control ansible]# vim ~/ansible/test\_john.yml

---

- hosts: webserver

tasks:

- name: Add the user 'johnd'

user:

name: johnd

uid: 1040

group: daemon

password: "{{ '123' | password\_hash('sha512') }}"

#hosts定义需要远程的对象是webserver组

#tasks是需要执行的任务

# -name是第一个任务的描述信息

# user是需要调用的模块，user下面的缩进内容是给user的参数

# name是需要创建的用户名，uid是用户ID号

# group是用户属于哪个基本组

# password是用户的密码，密码是123，密码经过sha512算法加密

用户管理，创建系统账户、账户属性、设置密码。

[root@control ansible]# cat ~/ansible/test\_john.yml

---

hosts: webserver

tasks:

- name: Add the user 'johnd'

user:

name: johnd

uid: 1040

group: daemon

password: password: "{{ '123' | password\_hash('sha512') }}"

#hosts定义需要远程的对象是webserver组

#taksk定义需要执行的任务

#user是需要调用的模块，user下面的缩进内容是给user的参数

#name是需要创建的用户名，uid是用户ID号

#grup是基本组，password是设置用户的密码.

[root@control ansible]# cat ~/ansible/user\_james.yml

---

- hosts: webserver

tasks:

- name: Add 'james' with a bash shell, set 'bin' and 'adm' to the user's groups

user:

name: james

shell: /bin/bash

groups: bin,adm

password: "{{ '123' | password\_hash('sha512') }}"

#与上一个例子类似创建用户

#groups是用户的附加组，shell是指定用户的解释器

[root@control ansible]# cat ~/ansible/user\_johnd.yml

---

- hosts: webserver

tasks:

- name: Remove the user 'johnd'

user:

name: johnd

state: absent

#删除系统账户johnd

#state等于absent用来删除用户

################################################

使用playbook剧本，自动分区，创建VG，创建LV

################################################

【在node1上确认下自己磁盘的情况】

[root@node1 ~]# lsblk

#看自己磁盘名称

#看自己磁盘现在有几个分区

#比如，我的是/dev/sdb，已经有2个分区

【在control主机编写剧本给node1创建新的分区，创建VG和LV】

[root@control ansible]# vim lvm.yml

---

- hosts: node1

tasks:

- name: Create a new primary partition #任务的描述信息

parted: #调用parted模块进行分区

device: /dev/sdb #对/dev/sdb磁盘进行分区

label: gpt #分区表类型为gpt，或msdos

number: 1 #分区编号(创建第几个分区)

state: present #present是创建分区,absent是删除分区

part\_start: 1MiB #分区的开始位置

part\_end: 2GiB #分区的结束位置

- name: Create a volume group #第二个任务的描述信息

lvg: #调用lvg模块,创建VG卷组

vg: myvgtoo #创建的卷组名称

pvs: /dev/sdb1 #使用哪个分区创建PV

- name: Create a logical volume #第三个任务的描述信息

lvol: #调用lvol模块创建LV

vg: myvgtoo #使用哪个VG创建LV

lv: mylvtoo #需要创建的LV名称

size: 512 #要创建的LV大小,默认单位m

[root@control ansible]# ansible-playbook lvm.yml

[root@control ~]# vim /tmp/ansible/package.yml

---

- hosts: webserver #需要远程的主机是谁

tasks: #定义剧本需要执行的任务

- name: Install a list of packages #第一个任务的描述信息

yum: #调用yum模块安装软件

name: #安装软件的名字

- httpd #安装httpd软件

- mariadb #安装mariadb软件

- mariadb-server #安装mariadb-server

- name: install group soft #第二个任务的描述信息

yum: #调用yum模块安装软件组包

name: "@Development tools" #安装哪个组包

- name: update software #第三个任务的描述信息

yum: #调用yum模块升级软件

name: '\*' #需要升级哪些软件

state: latest #latest代表升级软件

#备注:state的值可以是(present|absent|latest)

#present代表安装软件(默认是present)

#absent代表卸载软件

#latest代表升级软件

++++++++++++++++++++++++++

setup模块

++++++++++++++++++++++++++

ansible\_facts用于采集被管理设备的系统信息，所有收集的信息都被保存在变量中，每次执行playbook默认第一个任务就是Gathering Facts，使用setup模块可以查看收集到的facts信息。

[root@control ansible]# ansible test -m setup

#远程test主机获取该主机的facts信息并显示在屏幕上

尝试找出下面这些变量

ansible\_all\_ipv4\_addresses #IP地址

ansible\_bios\_version #主板BIOS版本

ansible\_memtotal\_mb #总内存

ansible\_hostname #主机名

ansible\_devices.sdb.partitions.sdb1.size #某个磁盘的大小

#################################

debug模块可以显示facts变量的信息

#################################

[root@control ansible]# vim debug.yml

---

- hosts: node1

tasks:

- debug:

var: ansible\_all\_ipv4\_addresses

- debug:

msg: "主机名是:{{ ansible\_hostname }}"

- debug:

var: ansible\_devices.sda.partitions.sda1.size

- debug:

msg: "总内存大小:{{ ansible\_memtotal\_mb }}"

#备注调用debug模块显示某些具体的变量值

#debug模块可以显示变量的值，可以辅助排错

#debug模块有两个参数，var和msg（引用变量需要{{}})。

#var后面调用变量不需要使用{{}}

#msg后面调用变量需要使用{{}}

#######################################

**在ansible中定义变量的若干中方法**

#######################################

1.Inventory变量

2.Host Facts变量

3.Register变量

4.Playbook变量

5.Playbook提示变量

6.变量文件

7.命令行变量

1.在inventory主机清单中定义变量

[root@control ansible]# vim ~/ansible/hosts

[test]

node1 myvar1="hello the world" myvar2="content"

[proxy]

node2

[webserver]

node[3:4]

[database]

node5

[cluster:children]

webserver

database

[webserver:vars]

yourname="jacob"

#备注，在node1主机后面给该主机添加两个变量myvar1和myvar2.

#给webserver组定义变量,vars是关键词不可以改变,webserver是上面定义的组

#给这个组定义变量yourname="jacob"

下面编写剧本调用刚才的变量：(在剧本中需要调用变量是要使用{{}})

[root@control ansible]# vim ~/ansible/inventory\_var.yml

---

- hosts: test #定义需要远程管理的主机是谁

tasks: #定义剧本需要执行的任务

- name: create a file #第一个任务的描述信息

shell: echo {{myvar1}} > /tmp/{{myvar2}}

#调用shell模块使用echo命令在/tmp目录下创建一个文件

#echo的内容是在inventory主机清单中提前定义好的变量myvar1

#创建的文件名是inventory主机清单中提前定义好的变量myvar2

- hosts: webserver #剧本中的第二个剧目,需要连接的主机

tasks: #第二个剧目要完成哪些任务

- name: create a user #第二个剧目中的第一个任务描述信息

user: #调用user模块创建用户

name: "{{yourname}}" #需要创建的用户名是yourname这个变量

#注意事项：

#在ansible剧本中当调用变量时，

#开始位置就调用变量,就需要在{{}}外面加双引号

#如果是在后面或者中间位置调用变量{{}}外面可以不加双引号

#如:

# "{{ myvar1 }}"

# nihao {{ myvar1 }}

2.在剧本中直接调用系统自带的ansible\_facts变量.

[root@control ansible]# vim ~/ansible/facts\_var.yml

---

- hosts: all

tasks:

- name: Use facts info.

copy:

content: "{{ansible\_hostname}}:{{ansible\_bios\_version}}"

dest: /tmp/facts.txt

#定义剧本，远程所有被管理主机

#调用copy模块，拷贝文件

#拷贝文件时没有源文件，直接通过content参数指定需要拷贝的文件内容是什么即可

#需要拷贝的内容调用了ansible\_facts变量

#拷贝这些内容到所有主机/tmp/facts.txt文件

#{{}}的作用是调用前面的变量,括号中是变量名.

[root@control ansible]# ansible-playbook facts\_var.yml

###########################

register变量

###########################

register可以将某个命令的执行结果保存到变量中。

[root@control ansible]# vim ~/ansible/register.yml

---

- hosts: test

tasks:

- name: save shell result to a variable.

shell: hostname

register: myvar

- name: print the variable's value through debug

debug:

msg: "{{ myvar }}"

#编写剧本，远程test主机

#第一个任务调用shell模块执行hostname命令

#通过register关键词，把上一条命令的输出结果保存到myvar变量

#第二个任务调用debug模块，显示myvar变量的值

#{{}}的作用是调用前面的变量,括号中是变量名.

[root@control ansible]# vim ~/ansible/register.yml

---

- hosts: test

tasks:

- name: save shell result to a variable.

shell: hostname

register: myvar

- name: print the variable's value through debug

debug:

msg: "{{ myvar.stdout }}"

#通过"."我们还可以仅提取部分数据.

#{{}}的作用是调用前面的变量,括号中是变量名.

########################################################

Playbook变量(使用vars关键词可以在playbook内定义变量)

########################################################

[root@control ansible]# vim ~/ansible/playbook\_var.yml

---

- hosts: test

vars: #vars是关键词，用来定义变量用的

iname: heal #具体变量名是iname，值是heal

ipass: '123456' #再定义一个变量名是ipass，值是123456

#注意密码必须是字符串，需要引号

tasks: #tasks定义需要执行的任务

- name: Use variables create user. #给任务写个描述信息

user: #调用user模块创建用户

name: "{{ iname }}" #用户名的是前面定义的变量

password: "{{ ipass | password\_hash('sha512') }}"

#密码是前面定义好的ipass,管道给password\_hash把密码加密下.

#{{}}的作用是调用前面的变量,括号中是变量名.

##############################

vars\_prompt提示变量

##############################

[root@control ansible]# vim prompt\_var.yml

---

- hosts: test

vars\_prompt: #vars\_prompt是关键词,定义提示变量

- name: iname #name定义变量的名称为iname

prompt: "请输入用户名" #执行剧本时的提示信息[任意字符]

private: no #private/隐私/,回显用户名

- name: ipasswd #name定义变量的名称为iname

prompt: "请输入密码" #执行剧本时的提示信息[任意字符]

private: yes #private/隐私/不显示密码

tasks:

- name: Create a user.

user:

name: "{{ iname }}"

password: "{{ ipasswd | password\_hash('sha512') }}"

#调用user模块创建用户和组,用户名是前面定义好的变量,密码也是变量.

#{{}}的作用是调用前面的变量,括号中是变量名.

#################################

专门定义个变量文件，给剧本调用

#################################

[root@control ansible]# vim ~/ansible/variables.yml

---

iname: cloud

ipass: '123456'

#创建文件，定义两个变量(定义变量的文件名是variables.yml,文件名可以任意,路径任意)

[root@control ansible]# vim ~/ansible/file\_var.yml

---

- hosts: test

vars\_files: variables.yml #调用前面的变量文件(注意路径和文件名要对)

tasks: #tasks定义需要执行哪些任务

- name: create user. #第一个任务的描述信息

user: #第一个任务调用user模块创建用户

name: "{{ iname }}" #用户名是变量

password: "{{ ipass | password\_hash('sha512') }}"

#密码是变量

###################################

执行ansible-playbook命令时定义变量

###################################

[root@control ansible]# vim command\_var.yml

---

- hosts: test

tasks:

- name: create user

user:

name: "{{ iname }}"

password: "{{ ipass | password\_hash('sha512') }}"

#这个剧本，不定义变量，直接调用user模块创建用户.

#创建的用户名是iname这个变量,密码是ipass这个变量[变量暂时未定义]

[root@control ansible]# ansible-playbook command\_var.yml \

-e iname="beth" -e ipass="123456"

#通过ansible-playbook命令执行剧本时才定义变量并赋值.

#

#

[root@control ansible]# vim firewall.yml

---

- hosts: test #hosts定义需要远程的主机

tasks: #tasks定义需要执行哪些剧本

- name: install firewalld. #name第一个任务定义描述信息

yum: #第一个任务调用yum模块安装软件

name: firewalld #需要安装的软件名称未firewalld

state: present #state等于present代表安装软件

- name: run firewalld. #定义第二个任务的描述信息

service: #第二个任务调用service模块启动服务

name: firewalld #启动的服务名称未firewalld

state: started #state等于started代表启动服务

enabled: yes #enabled等于yes是设置服务为开机自启动

- name: set firewalld rule. #第三个任务的描述信息

firewalld: #第三个任务调用firewalld模块设置防火墙规则

port: 80/tcp #在防火墙规则中添加一个tcp，80端口的规则

permanent: yes #permaenent 是设置永久规则

immediate: yes #immediate 是让规则立刻生效

state: enabled #state等于enabled是添加防火墙规则

#在默认zone中添加一条放行80端口的规则

######################################

template模块

######################################

template模块可以拷贝文件，类似copy模块

template模块拷贝的文件内容中可以包含变量,但是copy模块不可以

如果希望每个拷贝的文件内容都不一样呢？

如何给所有web主机拷贝index.html内容是各自的IP地址？

Ansible可以利用Jinja2模板引擎读取变量，之前在playbook中调用变量，

也是Jinja2的功能，Jinja2模块的表达式包含在分隔符"{{ }}"内。

[root@control ansible]# mkdir ~/ansible/template

[root@control ansible]# vim ~/ansible/template/index.html

Welcom to {{ansible\_hostname}} on {{ ansible\_ens33.ipv4.address }}.

#文件内容里面的{{ansible\_hostname}}是ansible\_facts变量:主机名

#文件内容里面的{{ ansible\_ens33.ipv4.address }}是ansible\_facts变量：网卡IP

#注意：网卡名称不要照抄

[root@control ansible]# vim template.yml

---

- hosts: webserver

tasks:

- name: use template copy index.html to webserver.

template:

src: ~/ansible/template/index.html

dest: /tmp/index.html

#hosts定义需要远程的目标主机是谁

#tasks定义需要执行的任务是什么

#- name定义任务的描述信息

#任务需要调用的模块是template模块

#template模块需要两个参数，src指定需要拷贝的源文件，dest指定需要拷贝的目标位置

#src: ~/ansible/template/index.html可以上面创建的文件,文件中包含变量

#dest: /tmp/index.html拷贝到目标主机放在/tmp目录下

####################################

剧本的中断问题

####################################

默认ansible在遇到error会立刻停止playbook剧本

如果一个剧本里面有20个任务，执行到第3个时失败，则不再往下执行

[root@control ansible]# vim error.yml

---

- hosts: test

tasks:

- name: start a service that does not exist.

service:

name: hehe #没有这个服务

state: started

- name: touch a file.

file:

path: /tmp/service.txt

state: touch

如果有些任务就是可能会失败，

失败了我也需要剧本继续执行，

则可以使用ignore\_errors忽略错误。

[root@control ansible]# vim error.yml

---

- hosts: test

# ignore\_errors: true #针对playbook全局忽略错误

tasks:

- name: start a service that does not exist.

service:

name: hehe #没有这个服务

state: started

ignore\_errors: true #针对某一个任务忽略错误

- name: touch a file.

file:

path: /tmp/service.txt

state: touch

#在某一个任务后面或者在全局添加ignore\_errors都可以，根据自己的需求选择。

当某个任务需要依赖其他任务怎么办？

可以通过handlers定义一组任务，

仅当某个任务触发(notify)handlers时才执行相应的任务，

如果有多个notify触发执行handlers任务，也仅执行一次。

[root@control ansible]# vim handlers.yml

---

- hosts: test

tasks:

- name: create directory. #多次执行playbook该任务状态不再是changed

file:

path: /tmp/parents/subdir/ #调用file模块创建目录

state: directory #state等于directory代表创建目录

notify: touch file #notify后面名称必须和handlers中的任务名称一致

handlers: #通过handlers再定义一组任务

- name: touch file #给任务写描述信息（任务的名字）

file: #调用file模块创建文件

path: /tmp/parents/subdir/new.txt #需要创建的文件名

state: touch #state等于touch代表创建文件

#备注：仅当file模块执行成功，

#并且状态为changed时才会通过notify触发执行handlers下面的任务，

#notity后面的名称必须和handlers下面定义的任务名称一致（名称可以任意）。

#所以多次执行该剧本时，handlers任务不会被重复执行.

########################################

when条件判断

########################################

when可以定义判断条件，条件为真时才执行某个任务。

常见条件操作符有：==、!=、>、>=、<、<=。

多个条件可以使用and或or分割，when表达式中调用变量不要使用{{ }}。

[root@control ansible]# vim when\_1.yml

---

- hosts: test

tasks:

- name: check memory size.

service:

name: NetworkManager

state: stopped

when: ansible\_memfree\_mb < 700

#远程主机剩余内存不足700M则关闭NetworkManager服务

#ansible\_memfree\_mb这个是ansible自带的facts变量,代表剩余内存的容量.

[root@control ansible]# vim when\_2.yml

---

- hosts: test

tasks:

- name: touch a file

file:

path: /tmp/when.txt

state: touch

when: >

ansible\_distribution == "CentOS"

and

ansible\_distribution\_major\_version == "7"

#判断操作系统是CentOS 7则创建测试文件[我们用的不是centos7，是RedHat8]。

#YAML的语法格式中>支持多行输入，但不保留换行符（计算机会认为实际是一行内容）。

#ansible\_distribution和ansible\_distribution\_major\_version都是自带的facts变量.

#可以使用setup模块查看这些变量.

#如果我们需要当条件满足时执行N个任务.

#我们可以给N个任务后面都加when判断(但是很麻烦)

#此时可以使用block定义一个任务块,当条件满足时执行整个任务块.

#任务块就是把一坨任何合并一下.

#使用block语句可以将多个任务合并为一个任务组.

[root@control ansible]# vim block\_1.yml

---

- hosts: test

tasks:

- name: define a group of tasks.

block: #block是关键词，定义任务组

- name: install httpd #任务组中的第一个任务

yum: #调用yum模块安装httpd软件包

name: httpd

state: present

- name: start httpd #任务组中的第二个任务

service: #调用service模块启动httpd服务

name: httpd

state: started

when: ansible\_distribution == "RedHat" #仅当条件满足再执行任务组.

#注意:when和block是对齐的,他们在一个级别,当条件满足要执行的是任务组[不是某一个任务]

#判断条件是看远程的目标主机使用的Linux发行版本是否是RedHat.

#对于block任务块，我们可以使用rescue语句定义在block任务执行失败时要执行的其他任务，

#还可以使用always语句定义无论block任务是否成功，都要执行的任务。

#当block任务块执行失败时，rescue就是一个弥补措施.

#always是不管block任务执行成功或失败，都要执行的任务.

[root@control ansible]# vim block\_2.yml

---

- hosts: test

tasks:

- block:

- name: touch a file test1.txt

file:

name: /tmp/test1.txt #如果改为/tmp/xyz/test1.txt就无法创建成功

state: touch

rescue:

- name: touch a file test2.txt

file:

name: /tmp/test2.txt

state: touch

always:

- name: touch a file test3.txt

file:

name: /tmp/test3.txt

state: touch

#默认在/tmp/目录下创建test1.txt会成功，所以不执行rescue(创建test2.txt)

#如果我们把block中的任务改为创建/tmp/xyz/test1.txt（因为没有xyz目录所以会失败)

#当block失败时就执行rescue任务(创建test2.txt)

#但是不管是否成功都执行always任务(创建test3.txt)

##################################

loop循环

##################################

相同模块需要反复被执行怎么处理？使用loop循环可以避免重复。

[root@control ansible]# vim simple\_loop.yml

---

- hosts: test

tasks:

- name: mkdir multi directory.

file:

path: /tmp/{{item}} #注意,item是关键字，调用loop循环的值

state: directory

loop: #loop是关键词,定义循环的值,下面是具体的值

- School

- Legend

- Life

#最终在/tmp目录下创建三个子目录.file模块被反复执行了三次.

#mkdir /tmp/School; mkdir /tmp/Legend; mkdir /tmp/Life

[root@control ansible]# vim complex\_loop.yml

---

- hosts: test

tasks:

- name: create multi user.

user:

name: "{{item.iname}}"

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

loop:

- { iname: 'term', ipass: '123456' }

- { iname: 'amy' , ipass: '654321' }

#loop循环第一次调用user模块创建用户,user模块创建用户会读取loop里面的第一个值.

#loop第一个值里面有两个子值,iname和ipass

#创建用户item.iname就是loop第一个值里面的iname=term

#修改密码item.ipass就是loop第一个值里面的ipass=123456

#loop循环第二次调用user模块创建用户,user模块创建用户会读取loop里面的第二个值.

#loop第二个值里面有两个子值,iname和ipass

#创建用户item.iname就是loop第二个值里面的iname=amy

#修改密码item.ipass就是loop第二个值里面的ipass=654321

######################################

使用ansible-vault管理敏感数据

######################################

使用ansible-vault对敏感数据进行加密处理

encrypt（加密）、decrypt（解密）、view（查看）、rekey（修改密码）

[root@control ansible]# echo 123456 > data.txt #新建测试文件

[root@control ansible]# ansible-vault encrypt data.txt #加密文件

[root@control ansible]# cat data.txt

[root@control ansible]# ansible-vault view data.txt #查看加密文件

[root@control ansible]# ansible-vault decrypt data.txt #解密文件

[root@control ansible]# cat data.txt

[root@control ansible]# ansible-vault encrypt data.txt #加密文件，密码111

New Vault password: 111

Confirm New Vault password:111

[root@control ansible]# ansible-vault rekey data.txt #修改密码

Vault password: <旧密码>

New Vault password: <新密码>

Confirm New Vault password:<确认新密码>

[root@control ansible]# echo "I'm secret data" > data.txt #需要加密的敏感数据

[root@control ansible]# echo 123456 > pass.txt #加密的密码

[root@control ansible]# ansible-vault encrypt --vault-id=pass.txt data.txt

#使用pass.txt文件中的密码把data.txt文件加密

[root@control ansible]# cat data.txt

[root@control ansible]# ansible-vault decrypt --vault-id=pass.txt data.txt

[root@control ansible]# cat data.txt

加密敏感数据-例1

[root@control ansible]# echo "I'm secret data" > data.txt #需要加密的敏感数据

[root@control ansible]# echo 123456 > pass.txt #加密的密码

[root@control ansible]# ansible-vault encrypt --vault-id=pass.txt data.txt

[root@control ansible]# ansible test -m copy --vault-id=pass.txt \

-a "src=data.txt dest=/tmp/ mode=0600"

#把自己的/data.txt文件拷贝给目标主机，放到/tmp目录下,权限是0600.

加密敏感数据-例2

[root@control ansible]# ansible-vault encrypt variables.yml #加密文件

New Vault password:

Confirm New Vault password:

[root@control ansible]# vim ~/ansible/vault.yml

---

- hosts: test

vars\_files: variables.yml #提前把这个文件加密

tasks:

- name: include vault data, create user.

user:

name: "{{iname}}"

password: "{{ipass|password\_hash('sha512')}}"

[root@control ansible]# ansible-playbook --ask-vault-pass vault.yml

#--ask-vault-pass代表执行剧本时提示输入密码(因为有文件被提前加密了)

###############################

Ansible Role

###############################

在实际生产环境中，为了实现不同的功能，我们会编写大量的playbook文件。

而且，每个playbook还可能会调用其他文件（如变量文件），对于海量的、无规律的文件，管理起来非常痛苦！

Ansible从1.2版本开始支持Roles，Roles是管理ansible文件的一种规范（目录结构）

Roles会按照标准的规范，自动到特定的目录和文件中读取数据。

如果我们创建了一个名称为user.example的Role。

defualts/main.yml：定义变量的缺省值，优先级较低

files目录：存储静态文件的目录

handlers/main.yml:定义handlers

meta/main.yml:写作者、版本等描述信息

README.md:整个角色(role)的描述信息

tasks/main.yml:定义任务的地方

templates目录：存放动态数据文件的地方（模板文件）

vars/main.yml:定义变量，优先级高

[root@control ansible]# mkdir ~/ansible/roles

[root@control ansible]# ansible-galaxy init ~/ansible/roles/issue

#创建一个Role，名称为issue

[root@control ansible]# tree ~/ansible/roles/issue/

#查看目录结构，如果没有tree命令则需要使用yum安装该软件

root@control ansible]# vim ~/ansible/roles/issue/templates/issue.j2

This is the system {{ansible\_hostname}}

Today's date is:{{ansible\_date\_time.date}}

Contact to {{ admin }}

#在templates子目录下创建一个文件，文件名为issue.j2(文件名可以任意)

#文件的内容包含常量和变量,ansible\_hostname和ansible\_date\_time.date是自带的Facts变量

#admin是自定义的变量.

[root@control ansible]# vim ~/ansible/roles/issue/vars/main.yml

---

# vars file for /root/ansible/roles/issue

admin: yoyo@tedu.cn

[root@control ansible]# vim ~/ansible/roles/issue/tasks/main.yml

---

# tasks file for /root/ansible/roles/issue

- name: delever issue file

template:

src: issue.j2

dest: /etc/issue

[root@control ansible]# vim ~/ansible/issue.yml

---

- hosts: test

roles:

- issue

# - role2 #支持加载多个role

##################################

在主配置文件中指定role角色的路径

##################################

[root@control ansible]# vim ~/ansible/ansible.cfg

[defaults]

remote\_user = root

inventory = ./inventory

roles\_path = ./roles #指定存放role的路径(相对或者绝对路径)

[privilege\_escalation]

become=True

become\_method=sudo

become\_user=root

become\_ask\_pass=False

公共Roles仓库(https://galaxy.ansible.com)管理。

[root@control ansible]# ansible-galaxy search 'httpd'

#联网搜索roles

[root@control ansible]# ansible-galaxy info acandid.httpd

#查看roles基本信息

[root@control ansible]# ansible-galaxy install acandid.httpd -p ~/ansible/roles/

#下载roles到特定的目录

[root@control ansible]# ansible-galaxy list -p roles/

#列出本地有哪些roles

使用ansible-galaxy install可以直接下载Role，也可以编写requirements.yml文件下载Role。

[root@control ansible]# cat ~/ansible/roles/requirements.yml

#格式一：直接从Ansible Galaxy官网下载

- src: acandid.httpd

#格式二：从某个git服务器下载

- src: http://gitlab.com/xxx/xxx.git

scm: git

version: 56e00a54

name: nginx-acme

#格式三：下载tar包，支持http、https、file

- src: http://example.com/myrole.tar

name: myrole

[root@control ansible]# ansible-galaxy install -r roles/requirements.yml -p roles

【综合项目】参考tmooc案例素材。

主机名称 IP地址 角色

control 192.168.4.253 控制节点（manager）

node1 192.168.4.11 被控制节点（test）

node2 192.168.4.12 被控制节点（proxy）

node3 192.168.4.13 被控制节点（web1）

node4 192.168.4.14 被控制节点（web2）

node5 192.168.4.15 被控制节点（database）

步骤一：部署两台后端LNP服务器（没有mariadb）

1）创建role角色

[root@control ansible]# ansible-galaxy init ~/ansible/roles/lnmp

2）准备2台LNP动态网站的素材

拷贝Nginx源码包，编写一个源码编译安装nginx的shell脚本。

[root@control ansible]# cp lnmp\_soft/nginx-1.17.6.tar.gz \

~/ansible/roles/lnmp/files/

[root@control ansible]# vim ~/ansible/roles/lnmp/files/nginx\_install.sh

#!/bin/bash

yum -y install gcc pcre-devel openssl-devel make

cd /tmp

tar -xf /tmp/nginx-1.17.6.tar.gz

cd nginx-1.17.6

./configure --with-http\_ssl\_module

make

make install

新建一个Nginx配置文件模板。

[root@control ansible]# vim ~/ansible/roles/lnmp/files/nginx.conf

#user nobody;

worker\_processes 2;

#error\_log logs/error.log;

events {

worker\_connections 65535;

}

http {

include 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"';

sendfile on;

tcp\_nopush on;

keepalive\_timeout 65;

#gzip on;

server {

listen 80;

server\_name localhost;

location / {

root html;

index index.html index.htm;

}

error\_page 404 /404.html;

error\_page 500 502 503 504 /50x.html;

location = /50x.html {

root html;

}

# pass the PHP scripts to FastCGI server listening on 127.0.0.1:9000

location ~ \.php$ {

root html;

fastcgi\_pass 127.0.0.1:9000;

fastcgi\_index index.php;

include fastcgi.conf;

}

}

}

准备测试首页文件模板。

[root@control ansible]# vim ~/ansible/roles/lnmp/templates/index.html

Welcom to {{ansible\_hostname}} on {{ ansible\_eth0.ipv4.address }}.

#注意网卡名称根据实际情况填写。

2）修改role配置文件。

[root@control ansible]# vim roles/lnmp/tasks/main.yml

---

- name: copy nginx-1.17.6.tar.gz to webserver.

copy:

src: nginx-1.17.6.tar.gz

dest: /tmp/

#拷贝源码包软件

- name: install nginx through shell script.

script: nginx\_install.sh

args:

creates: /usr/local/nginx/sbin/nginx

#执行源码编译安装脚本，如果已经安装nginx，则不再执行安装脚本.

- name: copy nginx.conf to destination host.

copy:

src: nginx.conf

dest: /usr/local/nginx/conf/nginx.conf

- name: copy index.html to destination host.

template:

src: index.html

dest: /usr/local/nginx/html/index.html

- name: install php

yum:

name:

- php

- php-fpm

- php-mysqlnd #rhel8中使用php-mysqlnd替代了php-mysql

- mariadb-devel

- name: run all service.

block:

- service:

name: php-fpm

state: started

- shell: /usr/local/nginx/sbin/nginx

args:

creates: /usr/local/nginx/logs/nginx.pid

#nginx.pid存在，说明nginx已经启动。如果该文件存在，则不再启动nginx。

3）编写Playbook调用role，并执行Playbook。

[root@control ansible]# vim lnmp.yml

---

- hosts: webserver

roles:

- lnmp

[root@control ansible]# ansible-playbook lnmp.yml

步骤二：部署nginx代理服务器

1）创建role角色

[root@control ansible]# ansible-galaxy init ~/ansible/roles/proxy

2）准备代理服务器需要的素材

拷贝Nginx源码包，编写一个源码编译安装nginx的shell脚本。

[root@control ansible]# cp lnmp\_soft/nginx-1.17.6.tar.gz \

~/ansible/roles/proxy/files/

[root@control ansible]# vim ~/ansible/roles/proxy/files/nginx\_install.sh

#!/bin/bash

yum -y install gcc pcre-devel openssl-devel make

cd /tmp

tar -xf /tmp/nginx-1.17.6.tar.gz

cd nginx-1.17.6

./configure --with-http\_ssl\_module

make

make install

新建一个Nginx代理服务器的配置文件模板。

[root@control ansible]# vim ~/ansible/roles/proxy/files/nginx.conf

#user nobody;

worker\_processes 2;

#error\_log logs/error.log;

events {

worker\_connections 65535;

}

http {

include 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"';

sendfile on;

tcp\_nopush on;

keepalive\_timeout 65;

#gzip on;

upstream webs {

server 192.168.4.13;

server 192.168.4.14;

}

server {

listen 80;

server\_name localhost;

location / {

proxy\_pass http://webs;

root html;

index index.html index.htm;

}

error\_page 404 /404.html;

error\_page 500 502 503 504 /50x.html;

location = /50x.html {

root html;

}

}

}

3）修改role配置文件。

[root@control ansible]# vim roles/proxy/tasks/main.yml

---

- name: copy nginx-1.17.6.tar.gz to proxy.

copy:

src: nginx-1.17.6.tar.gz

dest: /tmp/

#拷贝源码包软件

- name: install nginx through shell script.

script: nginx\_install.sh

args:

creates: /usr/local/nginx/sbin/nginx

#执行源码编译安装脚本，如果已经安装nginx，则不再执行安装脚本.

- name: copy nginx.conf to destination host.

copy:

src: nginx.conf

dest: /usr/local/nginx/conf/nginx.conf

- name: run nginx service.

shell: /usr/local/nginx/sbin/nginx

args:

creates: /usr/local/nginx/logs/nginx.pid

#nginx.pid存在，说明nginx已经启动。如果该文件存在，则不再启动nginx。

4）编写Playbook调用role,并执行Playbook。

[root@control ansible]# vim proxy.yml

---

- hosts: proxy

roles:

- proxy

[root@control ansible]# ansible-playbook proxy.yml

步骤三：部署数据库

[root@control ansible]# vim mariadb.yml

---

- hosts: database

tasks:

- name: install mariadb server

yum:

name:

- mariadb

- mariadb-server

- mariadb-devel

- name: run mariadb service

service:

name: mariadb

state: started

[root@control ansible]# ansible-playbook mariadb.yml