# Docker概述

## 概述

### 什么是容器

容器技术已经成为应用程序封装和交付的核心技术

容器技术的核心有以下几个内核技术组成:

- Cgroups(Control Groups) -资源管理

- NameSpace-进程隔离

- SELinux安全

由于是在物理机上实施隔离,启动一个容器,可以像启动一个进程一样快速

### 什么是Docker

Docker是完整的一套容器管理系统

Docker提供了一组命令,让用户更加方便直接地使用容器技术,而不需要过多关心底层内核技术

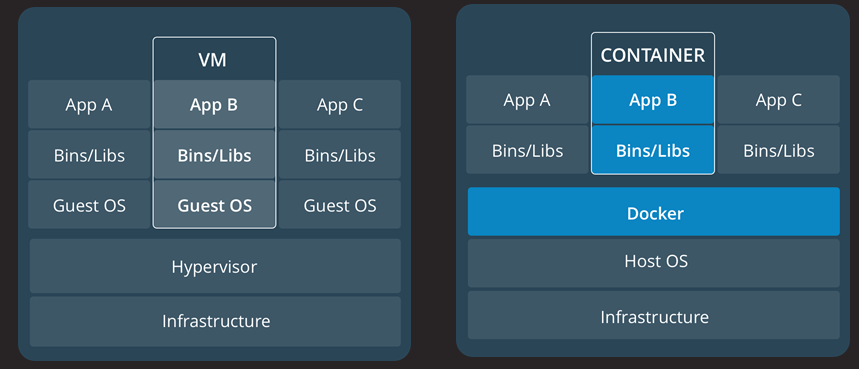
## Docker特性

### Docker优点

相比于传统的虚拟化技术,容器更加简洁高效

传统虚拟机需要给每个VM安装操作系统

容器使用的共享公共库和程序



### Docker缺点

容器的隔离性没有虚拟化强

共用Linux内核,安全性有先天缺陷

SELinux难以驾驭

监控容器和容器排错是挑战

# 部署Docker

## 安装软件

### 安装前准备

需要64位操作系统

至少RHEL7以上版本(6系列最好升级内核版本到3以上)

关闭防火墙

### 安装Docker

软件包列表:

- docker-engine

- docker-engine-selinux

启动2台虚拟机(2cpu,1.5G以上内存),20G以上硬盘

自定义yum源

[root@docker1 ~]# yum -y install docker-engine

[root@docker1 ~]# yum -y install docker-engine-selinux

[root@docker1 ~]# systemctl start docker

[root@docker1 ~]# ifconfig

[root@docker1 ~]# systemctl enable docker

[root@docker1 ~]# vim /etc/hosts

192.168.1.21 docker1

192.168.1.22 docker2

# Docker镜像

## 基本概念

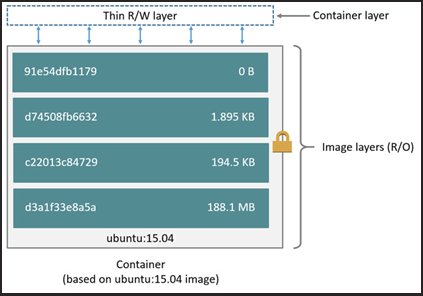
### 什么是镜像

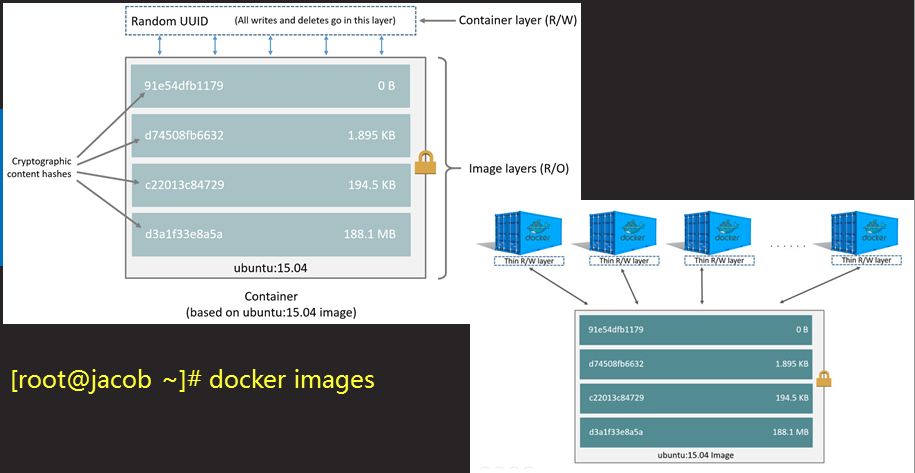
在Docker中容器是基于镜像启动的

镜像是启动容器的核心

镜像采用分层设计

使用快照的COW技术,确保底层数据不丢失





### Docker hub镜像仓库

https://hub.docker.com

Docker官方提供公共镜像的仓库(Registry)

[root@docker1 ~]# docker search nginx

[root@docker1 ~]# docker search centos

[root@docker1 ~]# docker search busybox

[root@docker1 ~]# docker pull busybox

查看帮助:man docker-search或docker help search

## 镜像操作

### 下载、上传镜像

下载镜像(从镜像仓库中下载镜像)

[root@docker1 ~]# docker pull busybox

上传镜像(上传镜像到仓库)

### 导入、导出镜像

导入镜像(通过tar包文件导入镜像)

[root@docker1 ~]# docker load < busybox.tar

[root@zqy84 /]# for i in docker{1,2} ; do ssh $i "cd /root/docker\_images/ ; for n in \* ; do docker load < \${n} ; done" ; done

导出镜像(将本地镜像导出为tar文件)

[root@docker1 ~]# docker save busybox:latest > busybox.tar

### 启动镜像

启动centos镜像生成一个容器

[root@docker1 ~]# docker images

[root@docker1 ~]# docker run -it centos:latest /bin/bash

it为run的参数

开启另一个终端(查看容器信息)

[root@docker1 ~]# docker ps

# Docker基本命令

## 镜像常用命令

### 命令列表

- docker images //查看镜像列表

- docker history //查看镜像制作历史

- docker inspect //查看镜像底层信息

- docker pull //下载镜像

- docker push //上传镜像

- docker rmi //删除本地镜像

- docker save //镜像另存为tar包

- docker load //使用tar包导入镜像

- docker search //搜索镜像

- docker tag //修改镜像名称和标签

### docker images

查看镜像列表

- 镜像仓库名称

- 镜像标签

- 镜像ID

- 创建时间

- 大小

[root@docker1 ~]# docker images

### docker history

查看镜像历史

- 了解镜像制作过程

- 详细参考后面的dockerfile内容

[root@docker1 ~]# docker history centos

CMD默认启动命令

### docker inspect

查看镜像底层信息

- 了解镜像环境变量、存储卷、标签等信息

[root@docker1 ~]# docker inspect centos

### docker rmi

删除本地镜像

- 注意:启动容器时删除镜像会提示错误

[root@docker1 ~]# docker rmi centos

Error response from daemon: conflict: unable to remove repository reference "centos" (must force) - container 74682e9cd173 is using its referenced image e934aafc2206

### docker save|load

保存本地镜像另存为tar文件

- 方便其他人使用tar包导入镜像

[root@docker1 ~]# docker save centos > centos.tar

使用tar包文件导入镜像

[root@docker1 ~]# docker load < centos.tar

### docker tag

重命名镜像名称(复制)

[root@docker1 ~]# docker images

[root@docker1 ~]# docker tag centos:latest xxhh:ooxx

[root@docker1 ~]# docker images

## 容器常用命令

### 命令列表

- docker run //运行容器

- docker ps //查看容器列表

- docker stop //关闭容器

- docker start //启动容器

- docker restart //重启容器

- docker attach|exec //进入容器

- docker inspect //查看容器底层信息

- docker top //查看容器进程列表

- docker rm //删除容器

### docker run

使用镜像启动容器

[root@docker1 ~]# docker run -itd centos bash

-i 交互式

-t 终端

-d 后台运行

### docker ps

列出容器列表

- docker ps //查看正在运行的容器

- docker ps -a //查看所有容器列表

- docker ps -aq //仅显示容器id

### docker stop|start|restart

管理容器

- docker stop //关闭容器

- docker start //开启容器

- docker restart //重启容器

### docker attach|exec

进入容器

- docker attach //进入容器,exit会导致容器关闭

- docker exec //进入容器,退出时不会关闭容器

ctrl+(p+q)可以退出容器不关闭

### docker inspect

查看容器底层信息

### docker top

查看容器进程列表

### docker rm

删除容器

- 注意,删除正在运行的容器时会提示错误

[root@docker1 ~]# docker rm $(docker ps -aq)

# 总结和答疑

## 删除容器

### 问题现象

删除容器,提示错误:

- docker rm 0cc

Error response from daemon:You cannot remove a running container

0cc385eb6f63373397e678347890914347dc8efa3cefd809c1ed8c157651a261. Stop the container before attempting removal or use -f

### 故障分析及排除

原因分析

- 容器没有关闭

解决办法

- 先关闭容器,才能删除容器

官方的nginx镜像用的是debian操作系统

root@e7b3b302049a:/# cat /etc/debian\_version

8.6

debian的软件包的后缀为.dpb

debian的包管理命令为dpkg

debian的解决依赖关系的装包命令为apt-get

dpkg -l | grep nginx 查看软件是否安装

dpkg -L nginx 查看软件安装位置

dpkg -S nginx 查看拥有该软件文件的包

# 自定义镜像与仓库

## 自定义镜像

### docker commit

使用镜像启动容器,在该容器基础上修改

另存为一个新镜像

示例

[root@docker1 ~]# docker rm $(docker ps -aq)

[root@docker1 ~]# docker run -it centos

[root@b7b2722484a4 /]# rm -rf /etc/yum.repos.d/\*

[root@b7b2722484a4 /]# vi /etc/yum.repos.d/local.repo

[root@b7b2722484a4 /]# yum repolist

[root@b7b2722484a4 /]# yum -y install vim net-tools psmisc iproute tree bash-completion

### Dockerfile

Dockerfile语法格式

- FROM:基础镜像

- MAINTAINER:镜像创建者信息

- EXPOSE:开放端口

- ENV:设置变量

- ADD:复制文件到镜像

- RUN:制作镜像时执行的命令,可以有多个

- WORKDIR:定义容器默认工作目录

- CMD:容器启动时执行的命令,仅可以有一条CMD

单独获取容器的IP地址,

docker inspect -f “{{.NetworkSettings.IPAddress}}” 容器ID

使用Dockerfile工作流程

- mkdir build;cd build

- vim Dockerfile

- docker build -t imagename Dockerfile所在目录

Docker文件案例

[root@docker1 ~]# mkdir rq

[root@docker1 ~]# cd rq

[root@docker1 rq]# cp /etc/yum.repos.d/CentOS7.repo .

[root@docker1 rq]# vim Dockerfile

FROM centos:latest

RUN rm -rf /etc/yum.repos.d/\*

ADD CentOS7.repo /etc/yum.repos.d/local.repo

RUN yum -y install vim net-tools psmisc iproute tree bash-completion

[root@docker1 rq]# docker build -t myos /root/rq/

[root@docker1 ~]# mkdir rq2

[root@docker1 ~]# cd rq2

[root@docker1 rq2]# vim Dockerfile

FROM myos

RUN yum -y install openssh-server initscripts

RUN sshd-keygen

RUN echo "123456" | passwd --stdin root

ENV EnvironmentFile=/etc/sysconfig/sshd

EXPOSE 22

CMD ["/usr/sbin/sshd","-D"]

[root@docker1 rq2]# docker build -t myos:sshd /root/rq2

[root@docker1 ~]# mkdir rq3

[root@docker1 ~]# cd rq3

[root@docker1 rq3]# vim Dockerfile

FROM myos:latest

RUN yum -y install httpd

ENV EnvironmentFile=/etc/sysconfig/httpd

WORKDIR /var/www/html

ADD index.html index.html

EXPOSE 80

EXPOSE 443

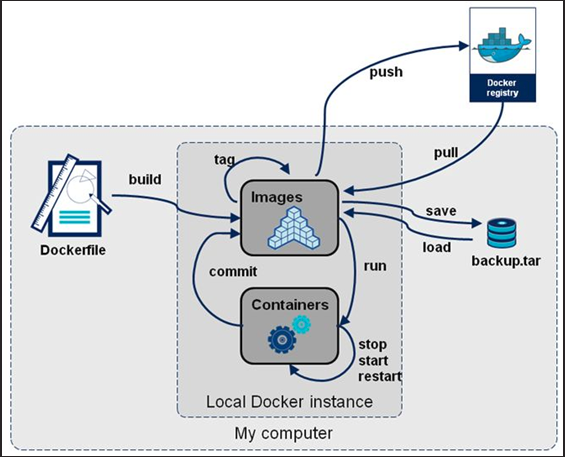
CMD ["/usr/sbin/httpd","-DFOREGROUND"]

[root@docker1 rq3]# docker build -t myos:httpd /root/rq3

## 自定义镜像仓库

### registry基本概念

共享镜像的一台服务器(镜像化的一台服务器)



### 自定义私有仓库

流程:

- docker pull registry

- vim /etc/docker/daemon.json

{

"insecure-registries" : ["192.168.1.21:5000"]

}

- docker run -d -p 5000:5000 registry

- docker tag 镜像 IP:5000/镜像:label

- docker push IP:5000/镜像:label

进入registry容器查看/etc/docker/registry/config.yml

示例

服务端:

vim /etc/docker/daemon.json

{

"insecure-registries" : ["192.168.1.21:5000"]

}

重启服务

[root@docker1 ~]# systemctl restart docker

启动私有仓库

[root@docker1 ~]# docker run -itd -p 5000:5000 registry

验证

[root@docker1 ~]# curl http://192.168.1.21:5000/v2

上传镜像到私有仓库

[root@docker1 ~]# docker tag busybox:latest \

192.168.1.21:5000/busybox:latest

[root@docker1 ~]# docker push 192.168.1.21:5000/busybox:latest

[root@docker1 ~]# for i in latest python sshd httpd;do

> docker tag myos:${i} 192.168.1.21:5000/myos:${i}

> docker push 192.168.1.21:5000/myos:${i}

> docker rmi 192.168.1.21:5000/myos:${i}

> done

客户端:

vim /etc/docker/daemon.json

{

"insecure-registries" : ["192.168.1.21:5000"]

}

重启服务

[root@docker2 ~]# systemctl restart docker

查看仓库中的镜像

http://192.168.1.21:5000/v2/\_catalog

查看某一镜像的所有标签

http://192.168.1.21:5000/v2/<repo>/tags/list

示例

[root@docker2 ~]# curl http://192.168.1.21:5000/v2/myos/tags/list

# 持久化存储

## 存储卷

### 卷的概念

docker容器不保持任何数据

重要数据请使用外部卷存储(数据持久化)

容器可以挂载真实机目录或共享存储为卷

### 主机卷的映射

将真实机目录挂载到容器中提供持久化存储

[root@docker1 ~]# docker run -v /nfsmount:/nfsmount -it centos bash

## 共享存储

### 共享存储基本概念

一台共享存储服务器可以提供给所有Docker主机使用

共享存储服务器(NAS、SAN、DAS等)

如:

- 使用NFS创建共享存储服务器

- 客户端挂载NFS共享,并最终映射到容器中

### 使用共享存储的案例

服务器

- yum -y install nfs-utils

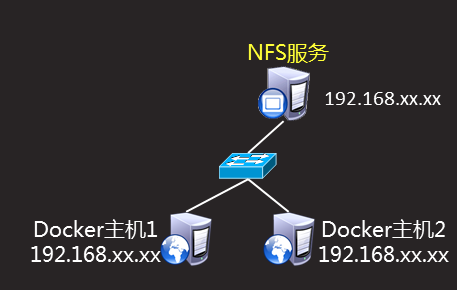
- vim /etc/exports

- systemctl start nfs

Docker主机

- mount挂载共享

- 运行容器时,使用-v选项映射磁盘到容器中



创建一个NFS服务器

docker1 mount nfs 服务器共享出来的目录

在docker1上启动nginx容器

把mount的共享目录映射给容器

docker2 mount nfs 服务器共享出来的目录

在docker2上启动apache容器

把mount的共享目录映射给容器

实现apache和nginx容器共享同一个web目录

docker1,nginx(nfs)

NFS

docker2,apache(nfs)

示例

[root@nfs ~]# yum -y install nfs-utils

[root@nfs ~]# mkdir nfsmount

[root@nfs ~]# vim /etc/exports

/nfsmount 192.168.1.0/24(rw,no\_root\_squash)

[root@nfs ~]# systemctl start nfs

[root@nfs ~]# systemctl enable nfs

随后在两台docker上也安装nfs-utils并挂载

[root@docker1 ~]# yum -y install nfs-utils

[root@docker1 ~]# vim /etc/fstab

192.168.1.23:/nfsmount /nfsmount nfs \_netdev 0 0

[root@docker1 ~]# mount -a

[root@docker1 ~]# echo "123456" > /nfsmount/index.html

启动docker

[root@docker1 ~]# docker run -v /nfsmount:/usr/share/nginx/html -itd nginx

[root@docker1 ~]# docker exec -it 01bd030794c9 /bin/bash

# Docker网络架构

## Linux网桥

### 创建虚拟网卡

真实网卡配置文件

- cat /etc/sysconfig/network-scripts/ifcfg-eth0

虚拟网卡配置文件

- cat /etc/sysconfig/network-scripts/ifcfg-eth0:0

### 创建虚拟网桥

## Docker网络拓扑

### 查看Docker默认网络模型

查看默认Docker创建的网络模型

[root@docker1 ~]# docker network list

NETWORK ID NAME DRIVER SCOPE

6566ded7e62b bridge bridge local 桥接模型

8026a7097c21 host host local 主机模型

3209025770de none null local 无网络

### 使用Docker创建网桥

新建Docker网络模型

查看默认Docker创建的网络模型

- 自定义网段

示例

创建docker1虚拟交换机,并设置ip网络为10.10.10.0/24

[root@docker1 ~]# docker network create --subnet=10.10.10.0/24 docker1

连接docker0的容器

[root@docker1 ~]# docker run -itd busybox

连接docker1的容器

[root@docker1 ~]# docker run -itd --network=docker1 busybox

[root@docker1 ~]# for i in $(docker ps | awk '$2~/busybox/{print $1}')

> do

> docker inspect $i | grep IPAddress

> done

(也可以使用-f选项过滤,但需要分成两批)

### 使用自定义网桥

启动容器,使用刚刚创建的自定义网桥

### 客户端访问容器内的资源

默认容器通过SNAT可以访问外网

但外部网络的主机不可以访问容器内的资源

端口映射

- 使用端口映射可以实现外部网络访问容器内的资源

-p 宿主机端口:容器端口

# 总结和答疑

## 提交镜像

### 问题现象

推送镜像到registry,提示错误:

### 故障分析及排除

原因分析

- 问题1:提示The push refers to a repository

[docker.io/library/centos]

解决办法

- 问题1:先要修改镜像tag,才可以继续push镜像到registry

# ansible基础

## ansible简介

### 什么是ansible

ansible是2013年推出的一款IT自动化和DevOps软件,2015年被REDHAT收购.是基于Python研发,糅合很多老运维工具的优点,实现了批量操作系统配置,批量程序部署,批量运行命令等功能.

ansible可以实现:

- 自动化部署APP

- 自动化管理配置项

- 自动化持续交付

- 自动化(AWS)云服务管理

### 为什么选择ansible

选择一款配置管理软件,无外乎从以下几点来权衡利弊

- 活跃度(社区)

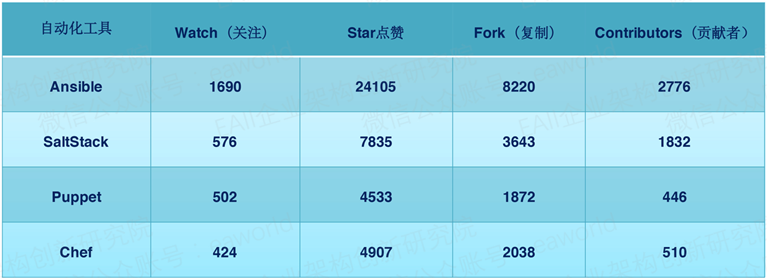
- 学习成本

- 使用成本

- 编码语言

- 性能

- 使用是否广泛



ansible优点

- 只需要SSH和Python即可使用

- 无客户端

- ansible功能强大,模块丰富

- 上手容易,门槛低

- 基于Python开发,做二次开发更容易

- 使用公司比较多,社区活跃

### ansible特性

模块化设计,调用特定的模块完成特定任务

基于Python语言实现

- paramiko

- PYYAML(半结构化语言)

- Jinja2

其模块支持JSON等标准输出格式,可以采用任何编程语言重写

部署简单

主从模式工作

支持自定义模块

支持playbook

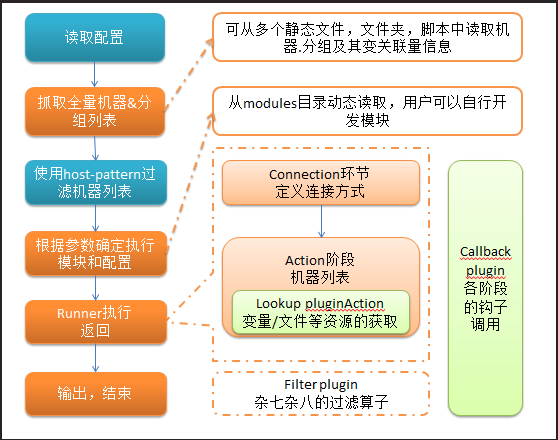
易于使用

支持多层部署

支持异构IT环境

### 工作流程

ansible大体执行过程



## ansible安装

### 软件依赖关系

对管理主机

- 要求Python 2.6或者Python 2.7

ansible使用以下模块,都需要安装

- paramiko

- PYYAML

- Jinja2

- httplib2

- six

对于被托管主机

- ansible默认通过SSH协议管理机器

- 被管理主机要开启ssh服务,允许ansible主机登录

- 在托管节点上也需要安装Python2.5或以上的版本

- 如果托管节点上开启了SElinux,需要安装libselinux-python

### 安装ansible

ansible可以基于源码运行

源码安装

- pip,需要配置扩展软件包源extras

- git

yum install epel-release

yum install git python2-pip

- pip安装依赖模块

pip install paramiko PYYAML Jinja2 httplib2 six

ansible源码下载

- git clone git://github.com/ansible/ansible.git

- yum install python-setuptools python-devel

- python setup.py build

- python setup.py install

php方式安装

- pip install ansible

**yum扩展源安装简单,自动解决依赖关系(推荐)**

- http://mirror.centos.org/.../.../extras/

- yum install ansible

安装完成以后验证

- ansible --version

示例

安装6台虚拟机,cpu、mem默认,硬盘10G以上

/etc/hosts

192.168.1.30 ansible

192.168.1.31 web1

192.168.1.32 web2

192.168.1.33 db1

192.168.1.34 db2

192.168.1.35 cache

在192.168.1.30主机上安装ansible

安装完成后执行ansible --version验证

[root@zqy84 ~]# for i in {31..36}; do

rclone <<EOF

$i

EOF

expect <<EOF

sleep 30

spawn virsh console node$i

expect "换码符" { send "\r" }

expect "login:" { send "root\r" }

expect "Password" { send "123456\r" }

expect "#" { send "eip.sh $i\r" }

expect "#" { send "systemctl restart network\r" }

expect "#" { send "exit\r"}

expect "#" { send "exit\r"}

EOF

done

[root@ansible ~]# yum install ansible

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

[root@ansible ~]# for i in 192.168.1.{31..35}

> do

> scp /etc/hosts $i:/etc

> done

# ad-hoc

## 主机管理

### 主机定义与分组

安装ansible之后可以做一些简单的任务

ansible配置文件查找顺序

- 首先检测ANSIBLE\_CONFIG变量定义的配置文件

- 其次检查当前目录下的 ./ansible.cfg文件

- 再次检查当前用户家目录下 ~/ansible.cfg文件

- 最后检查/etc/ansible/ansible.cfg文件

/etc/ansible/ansible.cfg是ansible的默认配置文件路径

ansible.cfg配置文件

- inventory定义托管主机地址配置文件

- 先编辑/etc/ansible/hosts文件,写入远程主机的地址.

格式

- # 表示注释

[组名称]

主机名称或ip地址,登录用户名,密码、端口等信息

测试

- ansible [组名称] --list-hosts

inventory参数说明

- ansible\_ssh\_host

- 将要连接的远程主机名与你想要设定的主机别名不同,可以通过此变量设置

- ansible\_ssh\_port

- ssh端口号:如果不是默认的端口号,通过此变量设置

- ansible\_ssh\_user

- 默认的ssh用户名

inventory参数说明

- ansible\_ssh\_pass

- ssh密码(这种方式并不安全,我们强烈建议使用--ask-pass或SSH密钥)

- ansible\_sudo\_pass

- sudo密码(建议使用 --ask-sudo-pass)

- ansible\_sudo\_exe(new in version 1.8)

- sudo命令路径(适用于1.8及以上版本)

- ansible\_connection

- 与主机的连接类型,如:local,ssh或paramiko,1.2以前默认使用paramiko,1.2以后默认使用’smart’,它会根据是否支持ControlPersist来判断’ssh’方式是否可行

- ansible\_ssh\_private\_key\_file

- ssh使用的私钥文件,适用于有多个密钥,而你不想使用SSH代理的情况

- ansible\_shell\_type

- 目标系统的shell类型,默认情况下,命令的执行使用’sh’语法,可设置为’csh’或’fish’

- ansible\_python\_interpreter

- 目标主机的python路径,适用情况:系统中有多个Python,或者命令路径不是”/usr/bin/python”

- 分组定义、范围定义样例

[web]

web1

web2

[db]

db[1:2]

[cache]

192.168.1.16

[app1:children] //指定子分组(app可改:children不改)，web，db是提前分好的组

web

db

[web]

web[1:2]

[web:vars] //web组:变量(vars不改)，web组的多台机器共用一个用户名和密码

ansible\_ssh\_user=”root”

ansible\_ssh\_pass=”pwd”

ansible\_ssh\_port=”22”

[cache]

c01 ansible\_ssh\_user=”root” ansible\_ssh\_pass=”pwd”

自定义配置文件

- 创建文件夹myansible

- 创建配置文件ansible.cfg

[defaults]

inventory = myhost

- 配置主机文件

[nginx]

192.168.1.11

192.168.1.12

192.168.1.13

- ansible nginx --list-hosts

示例

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

inventory = /etc/ansible/hosts

host\_key\_checking = False //是否检测主机的key(输入yes/no)

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

[web]

web1

web2

[db]

db1

db2

[other]

cache

[root@ansible ~]# ansible web --list-hosts

示例

[root@ansible ~]# cd /var/

[root@ansible var]# mkdir ooxx

[root@ansible var]# cd ooxx/

[root@ansible ooxx]# vim ansible.cfg

[defaults]

inventory = myhosts

host\_key\_checking = False

[root@ansible ooxx]# vim myhosts

[app1]

web1

db1

[app2]

web2

db2

cache

[root@ansible ooxx]# ansible app1 --list-hosts

当连接过主机后会在用户家目录的.ansible/cp/下生成缓存,再次连接可通过缓存无需密码.

### 动态主机

无限可能

- ansible Inventory包含静态和动态的Inventory,静态Inventory指在文件/etc/ansible/hosts中指定的主机和组,动态Inventory指通过外部脚本获取主机列表,按照其要求格式返回给ansible命令

Json

- JSON(JavaScript Object Notation,JavaScript对象表示法),一种基于文本独立于语言的轻量级数据交换格式

注意事项:

- 主机部分必须是列表格式

- Hostdata行,其中的”hosts”部分可以省略,但使用时,必须是”hosts”

脚本输出主机列表

#!/usr/bin/python

import json

hostlist = {}

hostlist[“bb”] = [“192.168.1.15”,”192.168.1.16”]

hostlist[“192.168.1.13”]={

“ansible\_ssh\_user”:”root”,”ansible\_ssh\_pass”:”pwd”

}

hostlist[“aa”]={

“hosts” :[“192.168.1.11”,”192.168.1.12”],

“vars” :{

“ansible\_ssh\_user”:”root”,”ansible\_ssh\_pass”:”pwd”

}

}

print(json.dumps(hostlist))

脚本输出样例

{

“aa” :{

“hosts” :[“192.168.1.11”,”192.168.1.12”],

“vars” :{

“ansible\_ssh\_user” : “root”,

“ansible\_ssh\_pass” : “pwd”

}

},

“bb” :[“192.168.1.15”,”192.168.1.16”],

“192.168.1.13”:{“ansible\_ssh\_user” : “root”,

“ansible\_ssh\_pass” : “pwd”}

}

## 批量执行

### ansible命令基础

ansible <host-pattern> [options]

- host-pattern 主机或定义的分组

-M 指定模块路径

-m 使用模块,默认command模块

-a or --args模块参数

-i inventory文件路径,或可执行脚本

-k 使用交互式登录密码

-e 定义变量

-v 详细信息,-vvvv开启debug模式

列出要执行的主机

- ansible all --list-hosts

批量检测主机

- ansible all -m ping

批量执行命令

- ansible all -m command -a ‘id’ -k

ansible 主机的集合 -m 模块的名称 -a 模块的参数

示例

ansible all -m command -a 'uptime'

ansible db -m command -a 'df -h'

ansible all -m command -a 'free -m'

### 批量部署证书文件

每次交互输入密码太麻烦

密码写入配置文件安全性很差

不同主机不同密码

使用key方式认证

给所有主机部署公钥

[root@ansible ~]# ansible all -m authorized\_key -a "user=root exclusive=true manage\_dir=true key='$(< /root/.ssh/id\_rsa.pub)'" -k

# 批量配置管理

## 模块

### ansible-doc和ping模块

ansible-doc

- 模块的手册相当于shell的man,很重要

- ansible-doc -l 列出所有模块

- ansible-doc modulename 查看帮助

ping模块

- 测试网络连通性,ping模块没有参数

- 注:测试ssh的连通性

ansible host-pattern -m ping

### command模块

command模块

- 默认模块,远程执行命令

- 用法

ansible host-pattern -m command -a ‘[args]’

- 查看所有机器负载

ansible all -m command -a ‘uptime’

- 查看日期和时间

ansible all -m command -a ‘date +%F\_%T’

command模块注意事项:

- 该模块通过-a跟上要执行的命令可以直接执行,若命令里有如下字符则执行不成功

- “<”,”>”,”l”,”&”

- 该模块不启动shell直接在ssh进程中执行,所有使用到shell的命令执行都会失败

- 下列命令执行会失败

ansible all -m command -a ‘ps aux | grep ssh’

ansible all -m command -a ‘set’

### shell|raw模块

shell | raw模块

- shell模块用法基本和command一样,区别是shell模块是通过/bin/sh进行执行命令,可以执行任意命令

- raw模块,用法和shell模块一样,可以执行任意命令

- 区别是raw没有chdir、creates、removes参数

- 执行以下命令查看结果

ansible t1 -m command -a ‘chdir=/tmp touch f1’

ansible t1 -m shell -a ‘chdir=/tmp touch f2’

ansible t1 -m raw -a ‘chdir=/tmp touch f3’

案例

在web1,db2创建用户z3

修改z3的密码为123456

[root@ansible ~]# ansible web1,db2 -m shell -a 'useradd z3;echo 123456 | passwd --stdin z3'

思考:

在所有web主机上创建用户li4

修改li4的密码为123123

要求:li4与z3不能同时出现在同一台主机上

[root@ansible ~]# ansible web -m shell -a 'id z3 || (useradd li4 ; echo 123123 | passwd --stdin li4)'

[root@ansible ~]# vim uu.sh

#!/bin/bash

id z3

if [ $? != 0 ];then

useradd li4

echo 123123 | passwd --stdin li4

fi

### script模块

script模块

- 命令太复杂?

- 在本地写脚本,然后使用script模块批量执行

ansible t1 -m script -a ‘urscript’

- 注意:该脚本包含但不限于shell脚本,只要指定Sha-bang解释器的脚本都可运行

### copy模块

copy模块

- 复制文件到远程主机

- src:将本地路径复制到远程服务器,绝对路径和相对路径都可,路径为目录时会递归复制.若路径以”/”结尾,只复制目录里的内容,若不以”/”结尾,则复制包含目录在内的整个内容,类似于rsync

- dest:必选项.远程主机的绝对路径,如果源文件是一个目录,那该路径必须是目录

- backup:覆盖前先备份原文件,备份文件包含时间信息.有两个选项:yes|no

- force:若目标主机包含该文件,但内容不同,如果设置为yes,则强制覆盖,设为no,则只有当目标主机的目标位置不存在该文件时才复制.默认为yes

- 复制文件

ansible t1 -m copy -a ‘src=/root/alog dest=/root/a.log’

- 复制目录

ansible t1 -m copy -a ‘src=urdir dest=/root/’

示例

[root@ansible ~]# ansible all -m copy -a 'src=/etc/resolv.conf \

dest=/etc/resolv.conf'

### lineinfile|replace模块

lineinfile模块

- 类似sed的一种行编辑替换模块

- path目的文件,指定要修改的文件

- regexp正则表达式,匹配要修改的行

- line替换后的结果,最终修改后的样子

ansible t1 -m lineinfile -a ‘path=”/etc/selinux/config”

regexp=”^SELINUX=” line=”SELINUX=disabled”’

replace模块

- path目的文件,指定要修改的文件

- regexp正则表达式,匹配要修改的字符串

- line替换后的结果,最终修改后的样子

- 替换指定字符

ansible t1 -m replace -a ‘path=”/etc/selinux/config”

regexp=”^(SELINUX=).\*” replace=”\1disabled”’

示例

[root@ansible ~]# ansible cache -m lineinfile -a 'path="/etc/resolf.conf" \

regexp="^nameserver" line="nameserver 192.168.1.1"'

[root@ansible ~]# ansible cache -m replace -a 'path="/etc/resolf.conf" \

regexp="^(nameserver).\*" replace="\1 192.168.1.254"'

### yum模块

yum模块

- 使用yum包管理器来管理软件包

- config\_file:yum的配置文件

- disable\_gpg\_check:关闭gpg\_check

- disablerepo:不启用某个源

- name:要进行操作的软件包名字,也可传递一个url或一个本地的rpm包的路径

- state:状态(present,absent,latest,removed,installed)

- 删除软件包

ansible t1 -m yum -a ‘name=”lrzsz” state=absent’

- 删除多个软件包

ansible t1 -m yum -a ‘name=”lrzsz,lftp” state=absent’

- 安装软件包

ansible t1 -m yum -a ‘name=”lrzsz”’

- 安装多个软件包

ansible t1 -m yum -a ‘name=”lrzsz,lftp”’

### service模块

service模块

- name:必选项,服务名称

- enabled:是否开机启动 yes|no

- sleep:执行restarted,会在stop和start之间沉睡几秒钟

- state:对当前服务执行启动,停止、重启、重新加载等操作(started,stopped,restarted,reloaded)

ansible t1 -m service -a ‘name=”sshd” enabled=”yes” state=”started”’

### setup模块

setup模块

- 主要用于获取主机信息,playbooks里经常会用的另一个参数gather\_facts与该模块相关,setup模块下经常用的是filter参数

- filter过滤所需信息

ansible t1 -m setup -a ‘filter=ansible\_distribution’

# 总结和答疑

## 批量部署证书文件

### 问题现象

部署证书文件时报错

“msg”:”Using a SSH password instead of a key is not possible because Host Key checking is enabled and sshpass does not support this. Please add this host’s fingerprint to your known\_hosts file to manage this host.”

### 故障分析及排除

原因分析

- ansible.cfg文件的host\_key\_checking字段没有修改或者修改错误

解决方案

- 修改ansible.cfg文件

host\_key\_checking = False

# playbook基础

## 基础知识

### ansible七种武器

第一种武器

- ansible命令,用于执行临时性的工作,必须掌握

第二种武器

- ansible-doc是ansible模块的文档说明,针对每个模块都有详细的说明及应用案例介绍,功能和Linux系统man命令类似,必须掌握

第三种武器

- ansible-console是ansible为用户提供的交互式工具,用户可以在ansible-console虚拟出来的终端上像Shell一样使用ansible内置的各种命令,这为习惯使用Shell交互方式的用户提供了良好的使用体验

第四种武器

- ansible-galaxy从github上下载管理Roles的一款工具,与python的pip类似

第五种武器

- ansible-playbook是日常应用中使用频率最高的命令,工作机制:通过读取先编写好的playbook文件实现批量管理,可以理解为按一定条件组成的ansible任务集,必须掌握

第六种武器

- ansible-vault主要用于配置文件加密,如编写的playbook文件中包含敏感信息,不想其他人随意查看,可用它加密/解密这个文件

第七种武器

- ansible-pull

- ansible有两种工作模式pull/push,默认使用push模式工作,pull和push工作模式机制刚好相反

- 适用场景:有大批量机器需要配置,即便使用高并发线程依旧要花费很多时间(多于2000台)

- 通常在配置大批量机器的场景下使用,灵活性稍有欠缺,但效率几乎可以无限提升,对运维人员的技术水平和前瞻性规划有较高要求

### JSON简介

JSON是什么

- JSON是JavaScript对象表示法,它是一种基于文本(字符串)独立于语言(与语言无关)的轻量级数据交换格式

- JSON中的分隔符限于单引号”’”、小括号”()”、中括号”[]”、大括号”{}”、冒号”:”和逗号”,”

JSON特性

- JSON是纯文本

- JSON具有”自我描述性”(人类可读)

- JSON具有层级结构(值中存在值)

- JSON可通过JavaScript进行解析

JSON语法规则

- 数据在名称/值对中

- 数据由逗号分隔

- 大括号保存对象

- 中括号保存数组

JSON数据的书写格式是:名称/值对

- 名称/值对包括字段名称(在双引号中),后面写一个冒号,然后是值,例如:

“诗仙” : ”李白”

JSON语法规则之数组

{ “诗人”:

[“李白”,”杜甫”,”白居易”,”李贺”]

}

复合复杂类型

{ “诗人”:

[{“李白”:”诗仙”,”年代”:”唐”},

{“杜甫”:”诗圣”,”年代”:”唐”},

{“白居易”:”诗魔”,”年代”:”唐”},

{“李贺”:”诗鬼”,”年代”:”唐”}

]

}

### YAML简介

YAML是什么

- 是一个可读性高,用来表达数据序列的格式

- YAML(YAML Ain’t Markup Language)

- YAML参考了多种语言,如:C语言、Python、Perl等,并从XML、电子邮件的数据格式中获得灵感,Clark Evans在2001年首次发表了这种语言,目前已有数种编程语言或脚本语言支持这种语言

YAML基础语法

- YAML的结构通过空格来展示

- 数组使用”- ”来表示

- 键值对使用”: ”来表示

- YAML使用一个固定的缩进风格表示数据层级结构关系

- 一般每个缩进级别由两个以上空格组成

- #表示注释

注意:

- 不要使用tab,缩进是初学者容易出错的地方之一

- 同一层级缩进必须对齐

YAML的键值表示方法

- 采用冒号分隔

- 后面必须有一个空格

- YAML键值对例子

“诗仙”: “李白”

- 或

“李白”:

“诗仙”

- 复杂YAML的键值对嵌套

“诗人”:

“李白”: “诗仙”

或

“诗人”:

“李白“

“诗仙”

数组

[“李白”,”杜甫”,”白居易”,”李贺”]

vim中的末行模式中%!xxd -r将16进制转化为10进制,%!xxd将10进制转化为16进制

Linux命令行中xxd命令可将文件或标准输入转换为16进制

### Jinja2模板简介

Jinja2是什么

- Jinja2是基于Python的模板引擎,包含变量和表达式两部分,两者在模板求值时会被替换为值,模板中还有标签,控制模板的逻辑

为什么要学习Jinja2模板

- 因为playbook的模板使用Python的Jinja2模块来处理

JInja2模板基本语法

- 模板的表达式都是包含在分隔符”{{ }}”内的

- 控制语句都是包含在分隔符”{% %}”内的

- 模板支持注释,都是包含在分隔符”{# #}”内,支持块注释

- 调用变量

{{varname}}

- 计算

{{2+3}}

- 判断

{{1 in [1,2,3]}}

Jinja2模板控制语句

{% if name == ‘诗仙’ %}

李白

{% elif name == ‘诗圣’ %}

杜甫

{% elif name == ‘诗魔’ %}

白居易

{% else %}

李贺

{% endif %}

Jinja2模板控制语句

{% if name == ... ... %}

... ...

{% elif name == ‘于谦’ %}

{% for method in [抽烟,喝酒,烫头] %}

{{do method}}

{% endfor %}

... ...

{% endif %}

Jinja2过滤器

- 变量可以通过过滤器修改.过滤器与变量用管道符号(|)分割,也可以用圆括号传递可选参数,多个过滤器可以链式调用,前一个过滤器的输出会被作为后一个过滤器的输入

例如

- 把一个列表用逗号连接起来: {{ list|join(‘,’) }}

- 过滤器这里不再一一列举,需要的可以查询在线文档

http://docs.jinkan.org/docs/jinja2/templates.html#builtin-filters

## playbook

### playbook是什么

- playbook是ansible用于配置,部署和管理托管主机剧本,通过playbook的详细描述,执行其中的一系列tasks,可以让远端主机达到预期状态

- 也可以说,playbook字面意思即剧本,现实中由演员按剧本表演,在ansible中由计算机进行安装,部署应用,提供对外服务,以及组织计算机处理各种各样的事情

为什么要使用playbook

- 执行一些简单的任务,使用ad-hoc命令可以方便的解决问题,但有时一个设施过于复杂时,执行ad-hoc命令是不合适的,最好使用playbook

- playbook可以反复使用编写的代码,可以放到不同的机器上面,像函数一样,最大化的利用代码,在使用ansible的过程中,处理的大部分操作都是在编写playbook

### playbook语法基础

playbook语法格式

- playbook由YAML语言编写,遵循YAML标准

- 在同一行中,#之后的内容表示注释

- 同一个列表中的元素应该保持相同的缩进

- playbook由一个或多个play组成

- play中hosts,variables,roles,tasks等对象的表示方法都是键值中间以”:”分隔表示

- YAML还有一个小的怪癖,它的文件开始行都是应该是 ---,这是YAML格式的一部分,表明一个文件的开始

playbook构成

- Target:定义将要执行playbook的远程主机组

- Variable:定义playbook运行时需要使用的变量

- Tasks:定义将要在远程主机上执行的任务列表

- Handler:定义task执行完成以后需要调用的任务

playbook执行结果

使用ansible-playbook运行playbook文件,输出内容为JSON格式,由不通颜色组成便于识别

- 绿色代表执行成功

- 黄色代表系统状态发生改变

- 红色代表执行失败

第一个playbook

--- #第一行,表示开始

- hosts: all

remote: root

tasks:

- ping:

ansible-playbook myping.yml -f 5

- -f并发进程数量,默认是5

- hosts行 内容是一个(多个)组或主机的patterns,以逗号为分隔符

- remote\_user 账户名

tasks

- 每一个play包含了一个task列表(任务列表)

- 一个task在其所对应的所有主机上(通过host pattern匹配的所有主机)执行完毕之后,下一个task才会执行

- 有一点很重要,在一个play之中,所有hosts会获取相同的任务指令,这是play的一个目的所在,即将一组选出的hosts映射到task,执行相同的操作

playbook执行命令

- 给所有主机添加用户plj,设置默认密码123456

- 要求第一次登录修改密码

---

- hosts: all

remote\_user: root

tasks:

- name: create user plj

user: group=wheel uid=1000 name=plj

- shell: echo 123456 | passwd --stdin plj

- shell: chage -d 0 plj

案例:playbook练习

1. 安装Apache并修改监听端口为8080
2. 修改ServerName配置,执行apachectl -t命令不报错
3. 设置默认主页hello world
4. 启动服务并设开机自启

[root@ansible ~]# vim httpd.yml

---

- hosts: web

remote\_user: root

tasks:

- yum:

name: httpd

state: installed

- replace:

path: /etc/httpd/conf/httpd.conf

regexp: '^(Listen).\*'

replace: '\1 8080'

- replace:

path: /etc/httpd/conf/httpd.conf

regexp: '#ServerName.\*'

replace: 'ServerName localhost'

- copy:

src: index.html

dest: /var/www/html/index.html

owner: apache

group: apache

mode: 0644

- service:

name: httpd

state: started

enabled: yes

# playbook进阶

## 语法进阶

### 变量

添加用户

- 给所有主机添加用户plj,设置默认密码123456

- 要求第一次登录修改密码(使用变量)

---

- hosts: 192.168.1.16

remote\_user: root

vars:

username: plj

tasks:

- name: create user “{{username}}”

user: group=wheel uid=1000 name={{username}}

- shell: echo 123456 | passwd --stdin plj

- shell: chage -d 0 {{username}}

设密码

- 解决密码明文问题

- user模块的password为什么不能设置密码呢

- 经过测试发现,password是把字符串直接写入shadow,并没有改变,而Linux的shadow密码是经过加密的,所以不能使用

解决方案

- 变量过滤器password\_hash

{{‘urpassword’ | password\_hash(‘sha512’)}}

变量过滤器

- 给所有主机添加用户plj,设置默认密码123456

- 要求第一次登录修改密码(使用变量)

---

- hosts: 192.168.1.16

remote\_user: root

vars:

username: plj

tasks:

- name: create user “{{username}}”

user: group=wheel uid=1000 password={{‘123456’ | password\_hash(‘sha512’)}} name={{username}}

- shell: chage -d 0 {{username}}

### error

ansible-playbook对错误的处理

- 默认情况判断$?,如果值不为0就停止执行

- 但某些情况我们需要忽略错误继续执行

---

- hosts: 192.168.1.16

remote\_user: root

vars:

username: plj

tasks:

- name: create user “{{username}}”

user: group=wheel uid=1000 password={{‘123456’|password\_hash(‘sha512’)}} name={{username}}

- shell: setenforce 0

- shell: chage -d 0 {{username}}

错误处理方法

- 关闭selinux,如果selinux已经关闭,返回1,若之前已经关闭则不算错误,可以忽略错误继续运行,忽略错误有两种方法

- 第一种方式:

shell: /usr/bin/somecommand || /bin/true

- 第二种方式:

- name: run some command

shell: /usr/bin/somecommand

ignore\_errors: True

在bash中,true可使$?变为0,false可使$?变为1

完整的playbook

---

- hosts:192.168.1.16

remote\_user: root

vars:

username: plj

tasks:

- name: create user “{{username}}”

user: group=wheel uid=1000 password={{‘123456’ | password\_hash(‘sha512’)}} name={{username}}

- shell: setenforce 0

ignore\_errors: true

- shell: chage -d 0 {{username}}

### handlers

当关注的资源发生变化时采取的操作

notify这个action可用于在每个play的最后被触发,这样可以避免有多次改变发生时每次都执行指定的操作,取而代之仅在所有的变化发生完成后一次性地执行指定操作

在notify中列出的操作称为handler,即notify调用handler中定义的操作

前面安装了Apache,修改httpd的配置文件,重新载入配置文件让服务生效

使用handlers来实现

handlers:

- name: restart apache

service: name=apache state=restarted

结合之前实验,完整的playbook

---

- hosts: 192.168.1.16

remote\_user: root

tasks:

- name: config httpd.conf

copy: src=/root/playboot/httpd.conf

dest=/etc/httpd/conf/httpd.conf

notify:

- restart httpd

handlers:

- name: restart httpd

service: name=httpd state=restarted

注意事项:

- notify调用的是handler段的name定义的串,必须一致,否则达不到触发的效果

- 多个task触发同一个notify的时候,同一个服务只会触发一次

- notify可以触发多个条件,在生产环境中往往涉及到某一个配置文件的改变要重启若干服务的场景,handler用到这里非常适合

- 结合vars可以写出非常普遍的服务管理脚本

### when

有些时候需要在满足特定的条件后再触发某一项操作,或在特定的条件下终止某个行为,这个时候需要进行条件判断,when正是解决这个问题的最佳选择,远程中的系统变量facts作为when的条件,可以通过setup模块查看

- when的样例

tashs:

- name: somecommand

command: somecommand

when: expr

一个使用when的例子

---

- name: Install VIM

hosts: all

tasks:

- name: Install VIM via yum

yum: name=vim-enhanced state=installed

when: ansible\_os\_family == “RedHat”

- name: Install VIM via apt

apt: name=vim state=installed

when: ansible\_os\_family == “Debian”

### register

register

- 有时候我们还需要更复杂的例子,如判断前一个命令的执行结果去处理后面的操作,这时候就需要register模块来保存前一个命令的返回状态,在后面进行调用

- command: test command

register: result

- command: run command

when: result

变量注册

- 例如需要判断plj这个用户是否存在

- 如果存在就修改密码,如果不存在就跳过

tasks:

- shell: id {{username}}

register: result

- name: change “{{username}}” password user: password={{‘12345678’|password\_hash(‘sha512’)}} name={{username}}

when: result

变量注册进阶

- 针对运行命令结果的返回值做判定

- 当系统负载超过一定值的时候做特殊处理

---

- hosts: 192.168.1.16

remote\_user: root

tasks:

- shell: uptime | awk ‘{printf(“%.2f\n”,$(NF-2))}’

register: result

- shell: touch /tmp/isreboot

when: result.stdout|float > 0.5

示例

[root@ansible ~]# vim load.yml

---

- hosts: web

remote\_user: root

tasks:

- shell: uptime |awk '{printf("%.2f",$(NF-2))}'

register: result

- service:

name: httpd

state: stopped

when: result.stdout|float > 0.7

### with\_items

with\_items是playbook标准循环,可以用于迭代一个列表或字典,通过{{ item }}获取每次迭代的值

- 例如创建多个用户

---

- hosts: 192.168.1.16

remote\_user: root

tasks:

- name: add users

user: group=wheel password={{‘123456’ | password\_hash(‘sha512’)}} name={{item}} with\_items: [“nb”,”dd”,”plj”,”lx”]

with\_items进阶

- 为不同用户定义不同组

---

- hosts: 192.168.1.16

remote\_user:root

tasks:

- name: add users

user: group={{item.group}} password={{‘123456’ | password\_hash(‘sha512’)}} name={{item.name}} with\_items:

- {name: ‘nb’, group: ‘root’}

- {name: ‘dd’, group: ‘root’}

- {name: ‘plj’, group: ’wheel’}

- {name: ’lx’, group: ‘wheel’}

### with\_nested

嵌套循环:

---

- hosts: 192.168.1.16

remote\_user: root

vars:

un: [a,b,c]

id: [1,2,3]

tasks:

- name: add users

shell: echo {{item}}

with\_nested:

- “{{un}}”

- “{{id}}”

### tags

tags:给指定的任务定义一个调用标识

使用格式:

- name: NAME

- module: arguments

- tags: TAG\_ID

playbook调用方式

- -t TAGS,--tags=TAGS

- --skip-tags=SKIP\_TAGS

- --start-at-task+START\_AT

tags样例:

vars:

soft: httpd

tasks:

- name: install {{soft}}

yum: name={{soft}}

- name: config httpd.conf

copy: src=/root/playbook/httpd.conf dest=/etc/httpd/conf/httpd.conf

- name: config services

service: enabled=yes state=restarted name={{soft}}

tags: restartweb

调用方式

ansible-playbook i.yml --tags=restartweb

### Include and roles

在编写playbook的时候随着项目越来越大,playbook越来越复杂,修改也很麻烦.这时可以把一些play、task或handler放到其他文件中,通过Include指令包含进来是一个不错的选择

tasks:

- include: tasks/setup.yml

- include: tasks/users.yml user=plj #users.yml中可以通过{{ user }}来使用这些变量

handlers:

- include: handlers/handlers.yml

roles像是加强版的Include,它可以引入一个项目的文件和目录

一般所需的目录层级有

- vars:变量层

- tasks:任务层

- handlers:触发条件

- files:文件

- template:模板

- default:默认,优先级最低

假如有一个play包含了一个叫”x”的role,则

---

- hosts: host\_group

roles:

- x

- x/tasks/main.yml

- x/vars/main.yml

- x/handler/main.yml

- x/... .../main.yml

- 都会自动添加进这个play

## 调试

### debug

对于Python语法不熟悉的同学,playbook书写起来容易出错,且排错困难,这里介绍集中简单的排错调试方法

- 检测语法

ansible-playbook --syntax-check playbook.yaml

- 测试运行

ansible-playbook -C playbook.yaml

- 显示受到影响的主机 --list-hosts

- 显示工作的task --list-tasks

- 显示将要运行的tag --list-tags

debug模块可以在运行时输出更为详细的信息,帮助我们排错

debug使用样例

---

- hosts: 192.168.1.16

remote\_user: root

tasks:

- shell: uptime | awk ‘{printf(“%f\n”,$(NF-2))}’

register:result

- shell: touch /tmp/isreboot

when: result.stdout|float > 0.5

- name: Show debug info

debug: var=result

# 总结和答疑