docker

应用程序封装和交付的核心技术

应用部署在前端盘，多个应用共享同一个操作系统

优点：• 轻量级

• 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，容器不需要安装操作系统

• 容器使用的共享公共库和程序

缺点：• 依赖于底层操作系统

• 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

容器不需要仿真支持

容器技术的核心有以下几个内核技术组成:

– CGroups(Control Groups)-资源隔离管理

多个容器同时工作时会争抢系统资源，要限制每个容器使用的资源

– NameSpace-进程隔离

多个容器运行的进程和端口互不冲突，可以单独运行，通过在底层LINUX系统中创建虚拟交换机，每个容器的网卡与虚拟机交换机相连，实现各容器之前通讯

信号隔离

网络隔离 每个容器的IP都不一样

主机名隔离

– SELinux安全

由于容器是在LINUX系统内部实现的，防火墙无法拦截，所以需要使用SELINUX做安全策略

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

安装docker

64位操作系统

建议RHEL7，内核是3版本才对docker完整支持

关闭防火墙

1. 真机创建2个虚拟机硬盘，4G内存，只需要一块网卡

qemu-img create -b testnode.qcow2 -f qcow2 docker1.img

qemu-img create -b testnode.qcow2 -f qcow2 docker2.img 50G

sed ‘s/demo/docker1’ demo.xml > /etc/libvirt/qemu/docer1.xml

virsh define /etc/libvirt/qemu/docker1.xml

virsh edit docker1

virsh start docker1

docker1 192.168.1.10

cd public

cp docker包 ./

createrepo

#--------------------虚拟机--------------------#

修改主机名和静态 ip 地址

echo docker01 >/etc/hostname

配置静态 ip 地址 /etc/sysconfig/network-scripts/ifcfg-eth0

BOOTPROTO="static"

IPADDR="192.168.1.10"

NETMASK="255.255.255.0"

GATEWAY="192.168.1.254"

二、装包

]# yum -y install docker-engine

xxx.repo 增加该YUM源

1. 启动服务

systemctl restart docker

systemctl enable docker

ifconfig 会多出一块docker0网卡

重启虚拟机

镜像 像虚拟机模板

容器 像虚拟机实例

操作系统有多个镜像后端盘，容器的后端是这些镜像，不能用操作系统做后端，镜像是只读的

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 镜像不是一次性做成功的，一层一层封装起来的，联合加载COW2技术

• 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)

Docker常用命令

docker virsion // 查看版本

docker images // 查看镜像

docker search 关键字 // 搜索镜像,要上网从官网镜像仓库搜索

docker pull 镜像名字 // 下载镜像,从官网镜像仓库下载

docker push 镜像名字 // 上传镜像（上传到自己的私有仓库）

docker help 命令 //查看命令帮助

docker save 镜像名:Tag标签 > xx.tar //导出镜像

docker load < xx.tar //导出镜像

docker run -it 镜像名:Tag标签 镜像里面的命令 //启动容器

i：交互式 t：终端

docker run -it centos:lasted /bin/bash

docker run -it centos（省略，标签默认是lasted，打开默认的命令）

docker ps 查看正在运行中的容器，开启另一个终端(查看容器信息)

docker ps -a 查看所有的容器

docker history 镜像名:Tag标签 //查看镜像封装的层次

每开启一个镜像，会在镜像上多一层读写层，然后把这两层合并成一个新的镜像，以此类推

docker inspect （镜像名:Tag标签／容器／虚拟交换机） //查看详细信息

环境变量 cmd(默认启动命令)

docker rmi 镜像名:Tag标签 //删除本地镜像（要先删除容器）

docker tag 原镜像名:原Tag标签 目标原镜像名:目标原Tag标签 //修改镜像名称和标签，创建一个别名（其实是创建了软链接）

镜像yum配置参考当前所在LINUX系统的YUM源配置

sysctl -w net.ipv4.ip\_forward=1 打开路由转发

vim /etc/sysctl.conf

每次运行docker run 都会重新根据这个镜像重新创建一个新的前端盘，之前的容器已经关闭了

容器常用命令

docker run //运行容器

-i 交互式

-t 终端

-d 后台进程

交互式的脚本一定要分配终端-d

启动一个交互式的终端 /bin/bash /usr/bin/python

docker run -it centos /bin/bash

docker run -it centos /usr/bin/python

启动一个服务（非交互）nginx redis

dock run -d nginx 放在后台运行，返回一个启动的ID

dock run --name web01 -d nginx

启动一个交互式的终端放在后台运行 /bin/bash &

docker run -itd centos 返回一个启动的ID

docker ps //查看容器列表

-q 只显示ID

-a 所有容器

docker stop 容器ID｜容器名称 //关闭容器

docker stop $(docker ps -q)

docker rm $(docker stop $(docker ps -qa))

关闭$()里面所有的容器，可以把()里面命令的执行结果做为这个函数的参数

docker start 容器ID｜容器名称 //启动容器

docker restart 容器ID｜容器名称 //重启容器

docker attach | exec 容器ID（可以省略，只要前面唯一匹配即可） //进入容器，对于已经运行的容器

exec 会在上帝进程下启动一个新的bash再开子进程

docker exec -it 7e4e bash

attach 会直接连接到PID为1的上帝进程 exit 退出以后会直接关闭这个容器！！！！如果这个容器开启的是不可交互的进程，退出以后这个容器会直接删除。（按ctrl+pq 将进程放入后台）

docker inspect 容器ID //查看容器底层信息，查看IP

docker inspect -f “{{.NetworkSettings.IPAddress}}”容器ID //过滤IP地址，查找到顶级对象，对象.属性

docker top 容器ID //查看容器进程列表，可以查询容器的PID

docker rm 容器ID//删除容器docker

创建自定义镜像

创建镜像模板

先创建一个容器，在容器中配置YUM源再安装常用软件，再将该容器重新打包成一个新的镜像模板

docker commit 容器ID name:label

docker run -it 新的镜像模板

Dockerfile语法格式 （使用脚本进行封装镜像）

– FROM:基础镜像

– MAINTAINER:镜像创建者信息

– EXPOSE:开放的端口

– 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 //相当于cd到该工作目录

– CMD:容器启动时执行的命令,仅可以有一条CMD

使用Dockerfile工作流程

mkdir xx; cd xx

vim **Dockerfile**

docker build -t name:label Dockerfile(dockerfile所在文件夹)

Dockerfile文件案例

vim Dockerfile

FROM centos:latest //在哪个镜像的基本上进行封装

RUN rm -f /etc/yum.repos.d/\* //RUN表示在容器里面执行该命令

ADD local.repo /etc/yum.repos.d/local.repo //ADD 表示把一个文件放进容器里，前面路径不能写绝对路径，要写相对路径

RUN yum -y install net-tools psmisc vim

CMD ["/usr/bin/python"]//容器启动时默认执行的命令，仅可以有一条

容器启动时执行的命令只能用Dockerfile做，commit做不了

docker build -t name:label .

ctrl+d 退出容器

FROM testos:latest

RUN yum -y install httpd

ADD httpd.conf /etc/httpd/conf/httpd.conf

WORKDIR /var/www/html

RUN echo "test html" > index.html

ENV EnvironmentFile=/etc/sysconfig/httpd

EXPOSE 80

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

自定义私有镜像仓库

要想外界访问容器的服务，需要把容器的服务映射成物理机的服务

服务端：

创建私有仓库配置文件

vim /etc/docker/daemon.json

{

"insecure-registries" : ["192.168.1.10:5000"]

}

完成配置以后重启 docker 服务

systemctl restart docker

启动私有仓库服务

docker run -d -p 5000:5000 registry

上传镜像

1、给镜像打标签，待上传

docker tag busybox:latest 192.168.1.10:5000/busybox:latest

docker tag testos:python 192.168.1.10:5000/testos:python

docker tag testos:httpd 192.168.1.10:5000/testos:httpd

2、上传

docker push 192.168.1.10:5000/busybox:latest

docker push 192.168.1.10:5000/testos:python

docker push 192.168.1.10:5000/testos:httpd

客户端：

将服务端/etc/docker/daemon.json拷贝到客户端上

重启 docker 服务

systemctl restart docker

启动容器

docker run -it 192.168.1.10:5000/busybox

docker run -it 192.168.1.10:5000/tetsos

docker run -d 192.168.1.10:5000/testos:httpd

查看私有仓库有哪些镜像

<http://192.168.1.10:5000/v2/_catalog>

查看私有仓库的镜像有哪些标签

<http://192.168.1.10:5000/v2/testos/tags/list>

频繁更新的数据不要放在容器中，容器中只放软件，不保存任何数据

如果：apache日志、页面不适合放在容器中，使用卷

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

存储卷映射

docker run -d -v 物理机文件夹：容器类文件夹 镜像:标签

docker run -d -v

/var/webroot:/var/www/html 192.168.1.10:5000/testos:httpd

/var/webroot:/var/www/html：用物理机的/var/webroot目录替换容器的/var/www/html，替换的该目录就相当于是httpd的网页文件的根目录，进到容器中的/var/www/html其实和物理机的目录是同一个文件夹，和mount很像

物理机文件夹由另一台NFS服务器提供

Docker网络架构

创建虚拟网卡

cp /etc/sysconfig/network-scripts/ifcfg-eth0 ifcfg-br0

TYPE=Bridge

NAME=br0

DEVICE=br0

ONBOOT=yes

IPADDR=

查看默认Docker创建的网络模型

docker network list

创建虚拟交换机

docker network create --driver bridge docker1//不指定网段自动分配

docker network create --driver bridge --subnet=192.168.100.0/24 docker1

ifconfig br-xxxx..

创建一个新的容器，使用新的交换机

docker run -it --network docker1 testos

--network docker1 选择容器和哪个网络相连，不指定是默认的docker0

多个虚拟交换机不能通信，是因为容器里的防火墙阻止了

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

物理机的80端口转发给容器的80端口

创建容器，使用宿主机的端口

docker run -d -p 80:80 -v /var/webroot:/var/www/html

docker run -d -p 81:80 -v /var/webroot:/var/www/html

使用物理机的81端口转发给容器的80端口

数据和服务能很好的解耦用Docker，因为Docker不保存数据，需要一个很大的后端存储

#------------------------------------------#

redhat debina

#------------------------------------------#

RHEL,CentOS debina

fedora ubuntu

安装 yum apt-get

管理 rpm dpkg

#------------------------------------------#