**Docker2**

两大系统

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

Redhat debian

Rhel/centos debian

Fedora ubantu

Yum包(rpm) apt-get(dpkg)

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

包管理上有区别，其他但没有区别 nginx可以用apt-get安装 dpkg -l 寻找nginx 包dpkg -s nginx

Docker创建镜像：

1. 创建新doker ：

# docker run -it --name centos1 centos

1. 配置内部环境：

# 配置yum与相关软件包

1. 查看相关镜像：

# docker ps

4.创建新的镜像：

# docker commit 0b0349329c0d(容器ID) myos:latest

Dockerfile

创建一个文件夹-------Cd 进入-----------vim编写文件

Dockerfile但语法格式：

– 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 （将文件放入容器）

– RUN:制作镜像时执行的命令,可以有多个（在容器中执行但命令）

– 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

FROM centos:latest

RUN rm -f /etc/yum.repo.d/\*

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

RUN yum install -y net-tools

创建步骤：

#Mkdir docker-maker文件名

#Cd docker-maker

#Vim Dockerfile(必须要是这个名字)：

----------------------------------------------------

FROM centos:latest

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

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

RUN yum install -y net-tools vim psmisc

-----------------------------------------------------

#ls docker-maker

Dockerfile local.repo

#docker build -t centos:latest .

CMD [定默认的启动命令]

在docker中搭建apache

查找启动文件位置

cat /usr/lib/systemd/system/httpd.service

ExecStart=/usr/sbin/httpd $OPTIONS -DFOREGROUND

将http.conf文件拷贝到文件夹下面

Cp /etc/httpd/conf/httpd.conf .

修改其中但servername 为 localhost

修改dockerfile文件

FROM myos

RUN yum install -y httpd

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

WORKDIR /var/www/html

RUN echo "hello nsd1803" >index.html

EXPOSE 80

MAINTAINER lixin

ENV EnvironmentFile=/etc/sysconfig/httpd

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

Docker镜像仓库：

创建私有仓库

配置文件 /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

打标记

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

docker tag myos:latest 192.168.1.10:5000/myos:latest

docker tag myos:python 192.168.1.10:5000/myos:python

docker tag myos:httpd 192.168.1.10:5000/myos:httpd

上传镜像

docker push 192.168.1.10:5000/busybox:latest

docker push 192.168.1.10:5000/myos:latest

docker push 192.168.1.10:5000/myos:python

docker push 192.168.1.10:5000/myos:httpd

客户机使用私有镜像源

配置 daemon.json

重启服务 systemctl restart docker

启动容器

docker run -it 192.168.1.10:5000/busybox

docker run -it 192.168.1.10:5000/myos

docker run -d 192.168.1.10:5000/myos:httpd

查看仓库中的镜像

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

查看私有仓库有什么样的镜像

curl http://192.168.1.10:5000/v2/\_catalog

查看私有仓库的镜像有什么样的标签

curl http://192.168.1.10:5000/v2/myos/tags/list

**存储卷：**

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

#docker run -v /data:/data -it centos bash

-v 用前面的路径，替换掉后面的文件路径

docker run -d -v /var/webroot:/var/www/html/ 192.168.1.10:5000/centos

试验：可以尝试nfs+docker实现多个docker的同步更新（先用nfs挂载文件index.html到/var/webroot）

Docker虚拟网桥

设置网桥需要安装包bridge-utils才能重启服务

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

– vim /etc/sysconfig/network-scripts/ifcfg-br0

# brctl show

查看默认Docker创建的网络模型

[root@jacob ~]# docker network list

**查看：**

[root@jacob ~]# ip a s docker0

[root@jacob ~]# brctl show docker0 //启动容器会绑定该网桥

**新建Docker网络模型**

~]# docker network create --driver bridge test01

~]# docker network list

~]# ip a s

~]# docker network inspect test01

**自定义网段**

docker network create --subnet=172.30.0.0/16 test01

**启动容器,使用刚刚创建的自定义网桥**

[root@jacob ~]# docker run --network=bridge|host|none ... ...

[root@jacob ~]# docker run --network=test01 -id nginx

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

**客户端如何访问容器：**

启动一个http服务

docker run -d -v /var/webroot:/var/www/html/ 192.168.1.10:5000/centos：http -p 80:80（物理机端口：docker服务端口）

用户可以直接访问物理机上。

容器通过封装服务，并与物理机但端口对接，使用户访问物理机端口时，直接访问到封装好的运行环境里

**下载百度页面：**

在可以物理机上下载页面文件到index中： curl <http://www.baidu>.com -o index.html

由于之前已经做过nfs共享，直接将页面文件共享到docker中，可以实现对http://192.168.1.10的网页访问

/etc/sysconf.d/70-

/usr/share/doc/systemd/

**rp\_filter**参数用于控制系统是否开启对数据包源地址的校验。

即rp\_filter参数有三个值，0、1、2，具体含义：

0：不开启源地址校验。

1：开启严格的反向路径校验。对每个进来的数据包，校验其反向路径是否是最佳路径。如果反向路径不是最佳路径，则直接丢弃该数据包。

2：开启松散的反向路径校验。对每个进来的数据包，校验其源地址是否可达，即反向路径是否能通（通过任意网口），如果反向路径不同，则直接丢弃该数据包。

系统rp\_filter参数的配置为：

[root@localhost ~]# sysctl -a | grep rp\_filter

**数据包发到了eth1网卡，如果这时候开启了rp\_filter参数，并配置为1，则系统会严格校验数据包的反向路径**

**进的网卡和响应数据包出的网卡不是同一个网卡，这时候系统会判断该反向路径不是最佳路径，而直接丢弃该请求数据包。**

解决办法：

1.修改路由表，使响应数据包从eth1出，即保证请求数据包进的网卡和响应数据包出的网卡为同一个网卡。

2.关闭rp\_filter参数。（注意all和default的参数都要改）

1)修改/etc/sysctl.conf文件，然后sysctl -p刷新到内存。

2)使用sysctl -w直接写入内存：sysctl -w net.ipv4.conf.all.rp\_filter=0

3)修改/proc文件系统： echo "0">/proc/sys/net/ipv4/conf/all/rp\_filter