**加我微信可进群学习交流：**

**微信号：**

luckylucky421302

也可通过扫描下面二维码添加



**课程更新的知识点会通过微信公众号免费分享给大家，可以关注我的公众号**

****

docker安装可参考我的微信公众号文章，大家打开文章，搜索docker可看到安装步骤：

https://mp.weixin.qq.com/s?\_\_biz=MzU0NjEwMTg4Mg==&mid=2247484160&idx=1&sn=894b777d522176291065655afd976178&chksm=fb638c15cc1405033be46c1d81803d7fd49f33751e37a9ef25a028c018b081ac82c9b60ab6c7&token=1373889250&lang=zh\_CN#rd

Docker的基本组成：客户端，守护进程，镜像，容器，仓库

Docker官网：https://docs.docker.com/

Docker的github地址：<https://github.com>/docker/docker

docker hub 官网

<https://registry.hub.docker.com>

如果docker官方registry拉取镜像速度很慢，可以尝试daocloud提供的加速器服务

<https://dashboard.daocloud>.io/mirror

怎么把镜像上传到hub仓库？

docker login

用户名 ：xianchao

密码：\*\*\*\*\*

上述用户名和密码是自己注册的docker hub的用户名和密码

docker push xiancaho/centos

**docker是什么？**

Docker 是一个[开源](https://baike.baidu.com/item/%E5%BC%80%E6%BA%90/246339)的应用容器引擎，让开发者可以打包他们的应用以及依赖包到一个可移植的镜像中，然后发布到任何流行的 [Linux](https://baike.baidu.com/item/Linux)或[Windows](https://baike.baidu.com/item/Windows/165458) 机器上，也可以实现[虚拟化](https://baike.baidu.com/item/%E8%99%9A%E6%8B%9F%E5%8C%96/547949)。容器是完全使用[沙箱](https://baike.baidu.com/item/%E6%B2%99%E7%AE%B1/393318)机制，相互之间不会有任何接口。

**docker的优点：**

隔离应用依赖

创建应用镜像并进行复制

创建容易分发的即启即用的应用

允许实例简单、快速地扩展

测试应用并随后销毁它们

自动化测试和持续集成、发布

**快**

运行时的性能可以获取极大提升

管理操作(启动，停止，开始，重启等等) 都是以秒或毫秒为单位的。

**敏捷**

像虚拟机一样敏捷，而且会更便宜，在bare metal(裸机)上布署像点个按钮一样简单。

**灵活**

将应用和系统“容器化”，不添加额外的操作系统，

**轻量**

你会拥有足够的“操作系统”，仅需添加或减小镜像即可。在一台服务器上可以布署100~1000个Containers容器。

**便宜**

开源的，免费的，低成本的。由现代Linux内核支持并驱动。注\* 轻量的Container必定可以在一个物理机上开启更多“容器”，注定比VMs要便宜。

**docker缺点：**

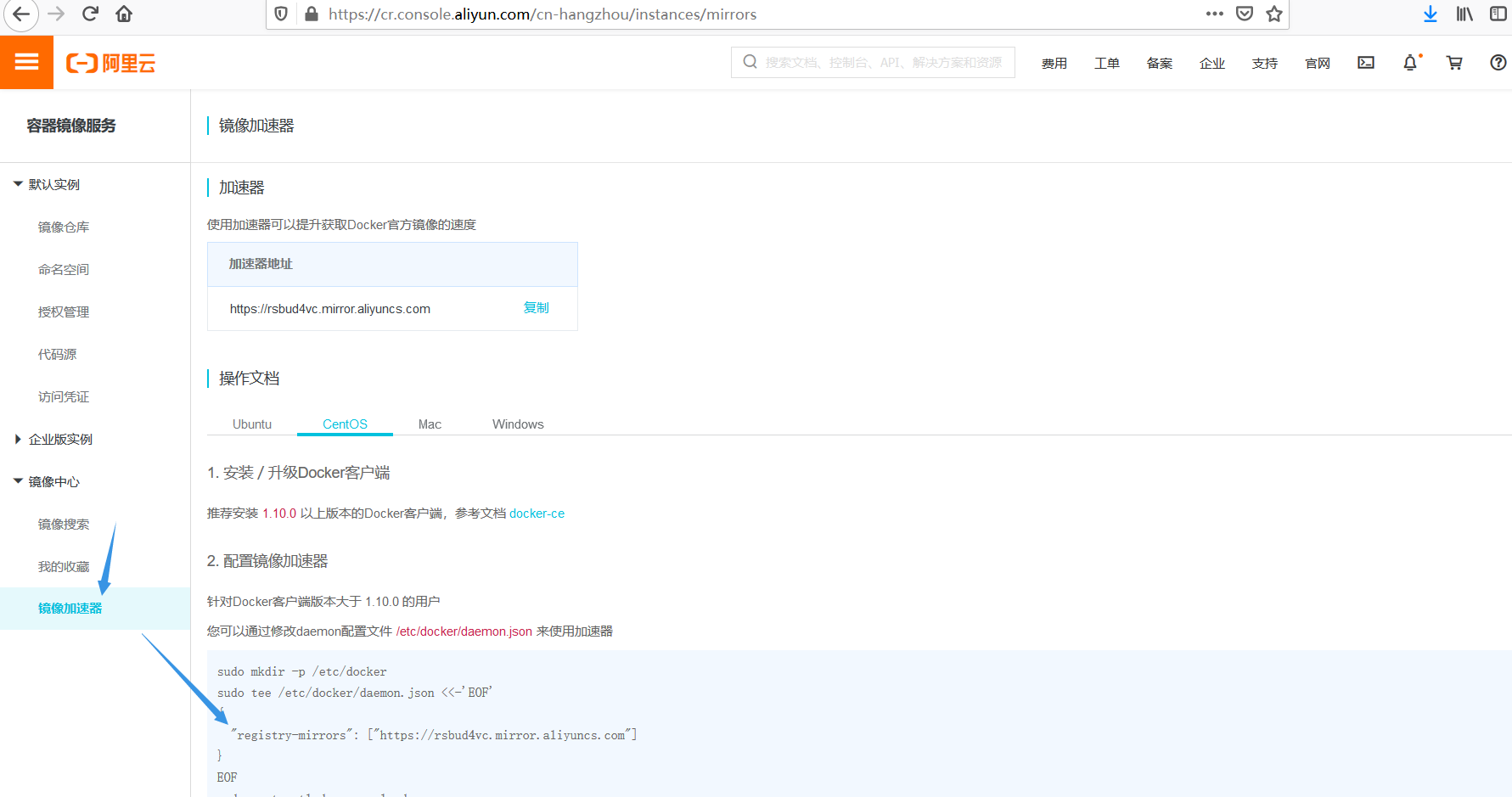
所有容器共用linux kernel资源，资源能否实现最大限度利用，所以在安全上也会存在漏洞。

**配置阿里云的镜像加速器**

**登陆阿里云镜像地址**

<https://cr.console.aliyun.com/cn-hangzhou/instances/mirrors>

如果没有开通，可开通阿里云的镜像服务



找到镜像加速器，然后按照箭头方向操作

修改/etc/docker/daemon.json，变成如下

{

"exec-opts": ["native.cgroupdriver=systemd"],

"log-driver": "json-file",

"log-opts": {

"max-size": "100m"

},

"storage-driver": "overlay2",

"storage-opts": [

"overlay2.override\_kernel\_check=true"

]

}

让配置文件生效

sudo systemctl daemon-reload

sudo systemctl restart docker

1. **docker的基本用法**

**镜像**

查看官网镜像：docker search centos

下载镜像：docker pull centos

查看镜像：docker images

删除镜像：docker rmi centos

**启动容器**

**以交互式方式启动并进入到容器**

docker pull centos

docker run --name=hello -it centos /bin/bash

ctrl+p，ctrl+q退出容器或者输入exit，这样容器可以在后台运行

**以守护进程方式启动容器**

docker run --name=hello1 -itd centos /bin/bash

**查看正在运行的容器**

docker ps

可以看见刚才启动的两个容器hello

**停止容器**

docker stop hello

**启动已经停止的容器**

docker start hello

**进入容器**

docker exec -it hello /bin/bash

1. **进入容器部署简单的nginx**

**创建一个映射了80端口的交互式容器**

docker run --name nginx -p 80 -itd centos /bin/bash （这个容器用的是centos7）

**安装nginx**

docker exec -it nginx /bin/bash

进入容器

yum install wget -y

yum install nginx -y

**安装文本编辑器vim**

yum install vim-enhanced -y

**创建静态页面**

mkdir /var/www/html -p

cd /var/www/html/

cat index.html

<html>

<head>

<title>nginx in docker</title>

</head>

<body>

<h1>hello,docker</h1>

</body>

</html>

**修改nginx配置文件**

修改nginx配置文件中的root路径，如下

vim /etc/nginx/nginx.conf

root /var/www/html/;

**运行nginx**

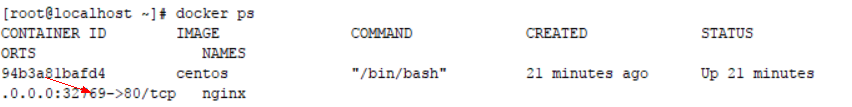
然后启动nginx

/usr/sbin/nginx

**验证网站访问,复制一个终端窗口，执行如下命令**

docker ps能查看到nginx映射的端口如下

docker port nginx 也可以看到端口映射情况



curl <http://192.168.0.6:3278>

可以看到网站内容,如下所示：

<html>

<head>

<title>nginx in docker</title>

</head>

<body>

<h1>hello,docker</h1>

</body>

</html>

也可以直接访问容器的ip:port

curl 172.17.0.4:80

也可以请求到网页

<html>

<head>

<title>nginx in docker</title>

</head>

<body>

<h1>hello,docker</h1>

</body>

</html>

**查看容器端口映射情况**

docker port nginx

**查看容器的ip**

登录到容器

docker exec -it nginx /bin/bash

执行 ip addr就可以看到容器的ip地址了

**流量走向**

访问物理节点ip:port（物理节点端口）--🡪容器ip:port（容器里部署的服务的端口）->就可以访问到容器里部署的应用了

1. **使用dockerfile构建镜像**

**docker部署nginx静态网站**

mkdir dockerfile

cd dockerfile/

**vim dockerfile**

|  |
| --- |
| FROM centos  MAINTAINER xianchao  RUN yum install wget -y  RUN yum install nginx -y  COPY index.html /usr/share/nginx/html/  EXPOSE 80  ENTRYPOINT ["/usr/sbin/nginx","-g","daemon off;"] |

**vim index.html**

|  |
| --- |
| <html>  <head>  <title>page added to dockerfile</title>  </head>  <body>  <h1>i am in df\_test </h1>  </body>  </html> |

**构建镜像：**

docker build -t="dockerfile/nginx" .

**查看镜像是否构建成功：**

docker images | grep nginx

显示如下说明镜像部署成功：

dockerfile/nginx latest baee97a76499 About a minute ago 344MB

**基于刚才的镜像启动容器：**

docker run -d -p 80 --name html2 dockerfile/nginx

**查看容器具体信息可按如下命令:**

**docker ps | grep html  
可显示如下信息，包含容器映射的端口等**

ac4c965ed113 dockerfile/nginx "/usr/sbin/nginx -g …" About a minute ago Up About a minute 0.0.0.0:32769->80/tcp html2

**查看容器里部署的nginx网站的内容：**

curl http://172.16.0.6:32769

1. **dockerfile指令**

**指令格式**

FROM centos

MAINTAINER xianchao [“15011572657@163.com”](mailto:)

RUN yum install wget -y

RUN yum install nginx -y

EXPOSE 80

（1）FROM

基础镜像，必须是可以下载下来的

（2）MAINTAINER

指定镜像的作者信息

（3）RUN：指定在当前镜像构建过程中运行的命令

包含两种模式

1. Shell

RUN <command> (shell模式，这个是最常用的，需要记住)

RUN /bin/sh -c command

RUN echo hello

1. RUN [“executable”，“param1”，“param2”](exec模式)

RUN [“/bin/bash”,”-c”,”echo hello”]

（4）EXPOSE指令

指定运行该镜像容器使用的端口，可以是一个或者多个端口，也可以指定多个EXPOSE

1. CMD（该指令是在容器运行时执行的命令，docker run 启动的时候如果指定参数，则会覆盖dockerfile中的CMD后面的内容的）

CMD[“executable”，“param1”，“param2”]（exec模式）

CMD command param1 param2 （shell模式）

CMD [“param1”,”param2”](作为ENTRYPOINT指令的默认参数)

**例：**cd /root/dockerfile/test

**cat dockerfile**

#first dockerfile

FROM centos

MAINTAINER xianchao

RUN yum clean all

RUN yum install nginx -y

EXPOSE 80

CMD ["/usr/sbin/nginx","-g","daemon off;"]

**构建镜像：**

docker build -t="dockerfile/test-cmd:v1" .

**基于上面构建的镜像运行一个容器**

docker run -p 80 --name cmd\_test2 -d dockerfile/test-cmd:v1

（不需要跟nginx -g “daemon off；”了）

docker ps可以看到下面信息

1. b903d5a71279 dockerfile/test-cmd:v1 "/usr/sbin/nginx -g …" 7 seconds ago Up 6 seconds 0.0.0.0:32770->80/tcp cmd\_test2
2. ENTERYPOINT

ENTERYPOINT [“executable”,“param1”,“param2”](exec模式)

ENTERYPOINT command param1 param2 （shell模式）

例：

vim Dockerfile

#first dockerfile

FROM centos

MAINTAINER xianchao

RUN yum clean all

RUN yum install nginx -y

EXPOSE 80

ENTRYPOINT ["/usr/sbin/nginx","-g","daemon off;"]

**ENTRYPOINT和CMD区别**

使用ENTRYPOINT，docker run运行时如果指定参数不会覆盖dockerfile中ENTRYPOINT后面的内容

使用CMD，docker run运行时如果指定参数会覆盖dockerfile中CMD后面的内容

1. ADD

ADD <src>...<dest>

ADD [“<src>”...“<dest>”]

1. COPY

COPY<src>..<dest>

COPY[“<src>”...“<dest>”]

ADD vs COPY

ADD包含类似tar的解压功能

如果单纯复制文件，docker推荐使用COPY

例;替换/usr/share/nginx下的index.html

cd /root/dockerfile/test1

cat dockerfile

FROM centos

MAINTAINER xianchao

RUN yum install wget -y

RUN yum install nginx -y

COPY index.html /usr/share/nginx/html/

EXPOSE 80

ENTRYPOINT ["/usr/sbin/nginx","-g","daemon off;"]

vim index.html

<html>

<head>

<title>page added to dockerfile</title>

</head>

<body>

<h1>i am in df\_test </h1>

</body>

</html>

docker build -t="dockerfile/copy:v1" .

docker run -d -p 80 --name html3 dockerfile/copy:v1

docker ps | grep html3

显示如下：

478868402ac4 dockerfile/copy:v1 "/usr/sbin/nginx -g …" 15 seconds ago Up 12 seconds 0.0.0.0:32771->80/tcp html3

curl 192.168.0.6:32771

显示的就是替换后的页面

<html>

<head>

<title>page added to dockerfile</title>

</head>

<body>

<h1>i am in df\_test </h1>

</body>

</html>

1. VOLUME

VOLUME [“/data”](卷)

对数据有持久化的功能

1. WORKDIR（填写绝对路径）

WORKDIR /path/to/workdir

1. ENV

设置环境变量

ENV <key><value>

ENV <key>=<value>...

1. USER

USER daemon

USER nginx

USER user

USER uid

USER user:group

USER uid:gid

USER user:gid

USER uid:group

1. ONBUILD

ONBUILD [INSTRUCTION]

为镜像添加触发器

当一个镜像被其他镜像作为基础镜像时需要写上OBNBUILD

会在构建时插入触发器指令

**例：演示ONBUILD指令**

vim dockerfile

FROM centos

MAINTAINER xianchao

RUN yum install wget -y

RUN yum install nginx -y

ONBUILD COPY index.html /usr/share/nginx/html/

EXPOSE 80

ENTRYPOINT ["/usr/sbin/nginx","-g","daemon off;"]

docker build -t="onbuild-nginx:v1" .

docker run -d --name html4 -p 80 onbuild-nginx:v1

docker ps | grep html4

显示如下：

65f4a5be9355 onbuild-nginx:v1 "/usr/sbin/nginx -g …" 14 seconds ago Up 11 seconds 0.0.0.0:32772->80/tcp html4

curl 192.168.0.6:32772

显示还是以前nginx默认的内容，没有被替换，表示ONBUILD这个指令后面的COPY没有生效

还是在刚在路径下构建新的镜像

vim dockerfile

FROM onbuild-nginx:v1

MAINTAINER xianchao

ENTRYPOINT ["/usr/sbin/nginx","-g","daemon off;"]

EXPOSE 80

docker build -t="onbuild-nginx1" .

docker run -d --name htm5 -p 80 onbuild-nginx1

docker ps | grep htm5

显示如下：

e56542310692 onbuild-nginx1 "/usr/sbin/nginx -g …" 12 seconds ago Up 8 seconds 0.0.0.0:32773->80/tcp htm5

curl 192.168.0.6:32773

显示如下：

<html>

<head>

<title>page added to dockerfile</title>

</head>

<body>

<h1>i am in df\_test </h1>

</body>

</html>

显示的就是已经重新构建的镜像，页面就是替换之后的了，说明我们基于ONBUILD指令的镜像作为基础镜像，在构建镜像，会触发ONBUILD后面的COPY命令运行

**5.dockerfile构建过程**

从基础镜像运行一个容器

执行一条指令，对容器做出修改

执行类似docker commit 的操作，提交一个新的镜像层

再基于刚提交的镜像运行一个新的容器

执行dockerfile中的下一条指令，直至所有指令执行完毕

**6.Docker容器的数据管理**

**Docker容器的数据卷**

**什么是数据卷？**

数据卷是经过特殊设计的目录，可以绕过联合文件系统（UFS），为一个或者多个容器提供访问，数据卷设计的目的，在于数据的永久话，它完全独立与容器的生存周期，因此，docker不会在容器删除时删除其挂载的数据卷，也不会存在类似的垃圾收集机制，对容器引用的数据卷进行处理，同一个数据卷可以只支持多个容器的访问。

**数据卷的特点：**

1.数据卷在容器启动时初始化，如果容器使用的镜像在挂载点包含了数据，这些数据会被拷贝到新初始化的数据卷中

2.数据卷可以在容器之间共享和重用

3.可以对数据卷里的内容直接进行修改

4.数据卷的变化不会影像镜像的更新

5.卷会一直存在，即使挂载数据卷的容器已经被删除

**数据卷的使用：**

**1.为容器添加数据卷**

docker run -v /datavolume:/data -it centos /bin/bash

**如：**

mkdir /datavolume

docker run --name volume -v ~/datavolume:/data -itd centos /bin/bash

注：datavolume为宿主机目录，data为docker启动的volume容器的目录

这样在宿主机的/datavolume目录下创建的数据就会同步到容器的/data目录下

**（1）为数据卷添加访问权限**

docker run --name volume1 -v ~/datavolume1:/data:ro -itd centos /bin/bash

添加只读权限之后在docker容器的/data目录下就不能在创建文件了，为只读权限；在宿主机下的/datavolume1下可以创建东西

**2.使用dockerfile构建包含数据卷的镜像**

dockerfile指令：

volume[“/data”]

cat dockerfile

FROM centos

VOLUME ["/datavolume3","/datavolume6"]

CMD /bin/bash

**使用如下构建镜像**

docker build -t="volume" .

**启动容器**

docker run --name volume-dubble -it volume

**会看到这个容器下有两个目录，datavolume3和datavolume6**

**7.Docker的数据卷容器**

**什么是数据卷容器：**

命名的容器挂载数据卷，其他容器通过挂载这个容器实现数据共享，挂载数据卷的容器，就叫做数据卷容器

**挂载数据卷容器的方法**

docker run --volumes-from [container name]

**例：**

docker run --name data-volume -itd volume（volume这个镜像是上面创建的带两个数据卷的镜像，datavolume3，ddatavolume6）

docker exec -it data-volume /bin/bash（进入到容器中）

touch datavolume6/lucky.txt

退出容器exit

**创建一个新容器挂载刚才data-volume这个容器创建的数据卷**

docker run --name data-volume2 --volumes-from data-volume -itd centos /bin/bash

进入到新创建的容器

docker exec -it data-volume2 /bin/bash

查看容器的/datavolume6目录下是否新创建了lucky.txt文件

cd /datavolume6

可以看见有刚才在上一个容器创建的文件lucky.txt

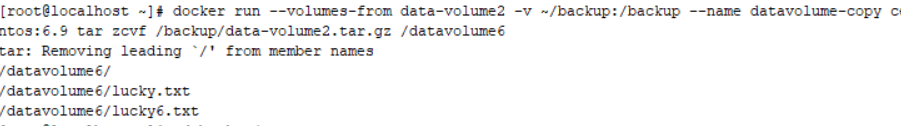
**8.docker数据卷的备份和还原**

**数据备份方法：**

docker run --volumes-from [container name] -v $(pwd):/backup centos tar czvf /backup/backup.tar [container data volume]

**例子：**

docker run --volumes-from data-volume2 -v /root/backup:/backup --name datavolume-copy centos tar zcvf /backup/data-volume2.tar.gz /datavolume6



可以看见已经备份成功了

**数据还原方法：**

docker run --volumes-from [container name] -v $(pwd):/backup centos tar xzvf /backup/backup.tar.gz [container data volume]

例：

docker exec -it data-volume2 /bin/bash

cd /datavolume6

rm -rf lucky.txt

docker run --volumes-from data-volume2 -v /root/backup/:/backup centos tar zxvf /backup/data-volume2.tar.gz -C /datavolume6

docker exec -it data-volum2 /bin/bash

cd /datavolum6

可以看到还原后的数据

**9.docker部署tomcat**

下面需要的压缩包在百度网盘，链接如下：

链接：https://pan.baidu.com/s/1bjnswP8K8wfKlYyPXyIN8Q

提取码：uzp5

mkdir tomcat8

cd tomcat8

把apache-tomcat-8.0.26.tar.gz 和jdk-8u45-linux-x64.rpm传到这个目录下

ls可以查看到tomcat8下面有如下几个文件



**cat dockerfile**

FROM centos

MAINTAINER xianchao

RUN yum install wget -y

ADD jdk-8u45-linux-x64.rpm /usr/local/

ADD apache-tomcat-8.0.26.tar.gz /usr/local/

RUN cd /usr/local && rpm -ivh jdk-8u45-linux-x64.rpm

RUN mv /usr/local/apache-tomcat-8.0.26 /usr/local/tomcat8

EXPOSE 8080

**开始构建镜像**

docker build -t="tomcat8:v1" .

**运行一个容器**

mkdir /usr/local/tomcat8/webapps -p

docker run --name tomcat8 -v /usr/local/tomcat8/webapps:/usr/local/tomcat8/webapps -itd -p 8080 tomcat8:v1 /bin/bash

**进入到容器**

docker exec -it tomcat8 /bin/bash

**启动tomcat**

/usr/local/tomcat8/bin/startup.sh

**查看进程，看看是否启动成功**

ps -ef | grep tomcat

**打开新的终端窗口，查看刚才创建的tomcat8这个容器的详细信息：**

docker ps | grep tomcat

显示如下信息

4d4c91cff4b5 tomcat8:v1 "/bin/bash" About a minute ago Up About a minute 0.0.0.0:32776->8080/tcp tomcat8

通过上面可以看到，tomcat在宿主机上映射的端口是32776

这样我们请求k8s的master1节点的ip:32776，就可以访问到tomcat的内容了

**通过这些步骤可以实现在docker中部署tomcat了**

**10.docker容器的网络连接**

**课程概要：**

**docker run创建Docker容器时，可以用–net选项指定容器的网络模式，Docker有以下4种网络模式：**

**** bridge模式：使--net =bridge指定，默认设置；

 host模式：使--net =host指定；

 none模式：使--net =none指定；

 container模式：使用--net =container:NAME orID指定。

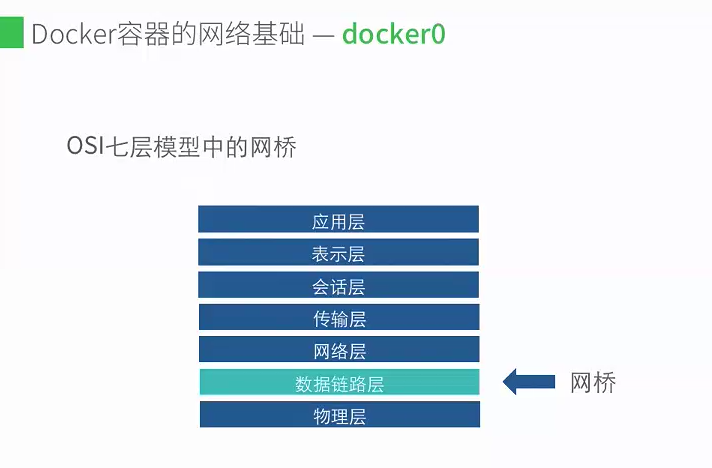
参考：<http://www.docker.org.cn/dockerppt/111.html>

**一：docker 容器的网络基础**

1.docker0：

linux的虚拟网桥

在OSI七层模型中的数据链路层



2.Linux虚拟网桥的特点：

可以设置ip地址

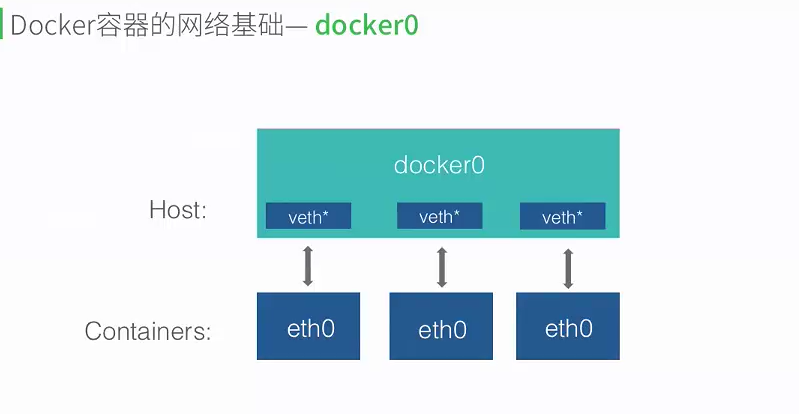
相当于拥有一个隐藏的虚拟网卡

docker0的地址划分：

IP：172.17.42.1 子网掩码：255.255.0.0

MAC：02:42:ac:11:00:00到02:42:ac:11:ff:ff

总共提供了65534个地址

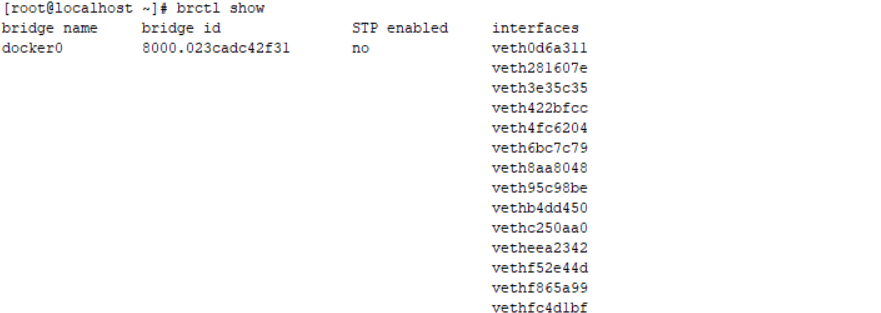


通过veth\*与容器通信

3．安装网桥管理工具：

yum install bridge-utils -y

brctl show 可以查看到有一个docker0的网桥设备，下面有很多接口，每个接口都表示一个启动的docker容器，因为我在docker上启动了很多容器，所以interfaces较多，如下所示：



4.修改docker0地址：

ifconfig docker0 192.168.200.1 netmask 255.255.255.0

然后重启docker：

service docker restart

创建一个容器，进去之后ifconfig可以看见docker0的ip地址是我们刚才设置的了

5.添加虚拟网桥

brctl addbr br0

ifconfig br0 172.16.1.1 netmask 255.255.0.0（我的宿主机docker0地址是172.17.42.1）

更改docker守护进程的启动配置（docker1.7按如下方式改）：

/etc/sysconfig/docker 中添加

other\_args="-b=br0"

重启docker：service docker restart

ps -ef | grep docker 可以看见br0这个参数

docker run -it centos：6.9 /bin/bash

ifconfig可以看见ip地址是172.16.0.1

也就是说们启动docker时，在/etc/sysconfig/docker里修改other\_args="-b=br0"，在创建docker容器的时候，容器的ip就是172.16.\*.\*,也就是我们新建立的网桥

**二：docker 容器的互联（相互之间的访问）**

下面用到的镜像的dockerfile文件如下：

cd dockerfile/inter-image

vim dockerfile

FROM centos

RUN yum install wget -y

RUN yum install nginx -y

RUN sed -i "7s/^/#/g" /etc/nginx/conf.d/default.conf

EXPOSE 80

CMD /bin/bash

docker build -t=”inter-image” .

**允许所有容器间互联（也就是访问）**

**第一种方法：**

**例：**

（1）基于上面的inter-image镜像启动第一个容器test1

docker run --name test1 -it inter-image

进入到容器里面启动nginx：

/usr/sbin/ngnx

Ctrl +p，ctrl+q 退出

（2）基于上面的inter-image镜像启动第二个容器test2

docker run --name test2 -it inter-image

ctrl+p和ctrl+q退出容器

（3）进入到test1容器和test2容器，可以看两个容器的ip，分别是

172.17.0.20和172.17.0.21

docker exec -it test2 /bin/bash

ping 172.17.0.20 可以看见能ping同test1容器的ip

curl http://172.17.0.20

可以访问到test1容器的内容

上述方法假如test1容器重启，那么在启动就会重新分配ip地址，所以为了使ip地址变了也可以访问可以采用下面的方法

**第二种方法：**

可以给容器起一个代号，这样可以直接以代号访问，避免了容器重启ip变化带来的问题

--link

docker run --link=[CONTAINER\_NAME]:[ALIAS] [IMAGE][COMMAND]

**例：**

1.启动一个test3容器

docker run --name test3 -it inter-image /bin/bash

**启动nginx：**

/usr/sbin/nginx

Ctrl+p和ctrl+q退出或者exit退出

2.启动一个test5容器，--link做链接，那么当我们重新启动test3容器时，就算ip变了，也没关系，我们可以在test5上ping别名webtest

docker run --name test5 -it --link=test3:webtest inter-image /bin/bash

ctrl+p和ctrl+q退出

3.test3和test5的ip分别是172.17.0.22和172.17.0.24

4.重启test3容器

docker restart test3

发现ip变成了172.17.0.25

5.进入到test5容器

docker exec -it test5 /bin/bash

ping test3容器的ip别名webtest可以ping通，尽管test3容器的ip变了也可以通

**拒绝容器互联（拒绝容器相互访问）**

docker守护进程的启动项

--icc=false

修改启动项的配置文件

vim /etc/sysconfig/docker

在最后一行添加如下参数：

other\_args="-icc=false"

重启docker容器配置生效，然后分别运行test3，test5，在test5上ping test3的ip（webtest），会发现ping不通