## 编写Dockerfile

|  |
| --- |
|  |

## 构造镜像build

如果你的文件不是Dockerfile

Docker build -f /root/mycentos/Dockerfile -t mycentos:1.0 .

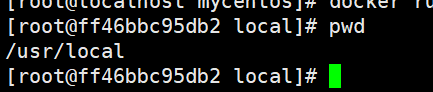
否则

Docker build -t mycentos:1.0 .

## 运行镜像



## 进入镜像



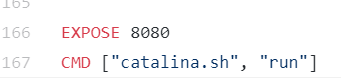
# 14【掌握】Dockerfile详解【3】

## 1，CMD说明

Dockerfile 中可以有多个CMD的命令，但只有最后一个生效，CMD会被docker run之后的参数替换掉

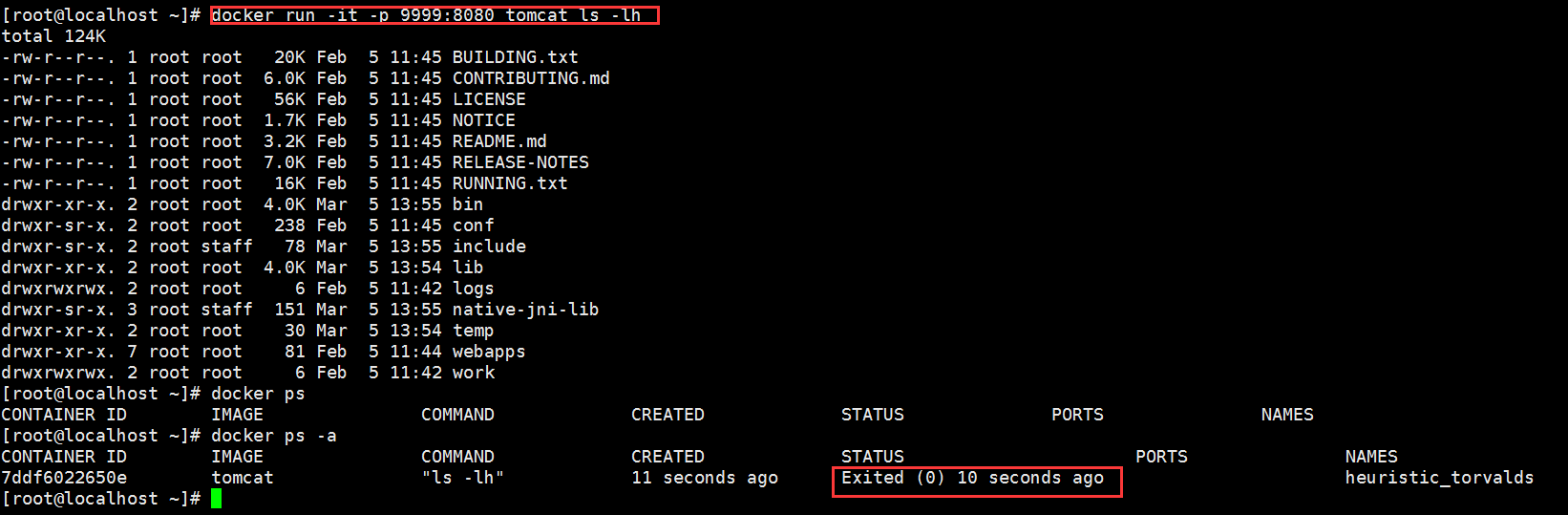
可以使用tomcat的案例演示

2.1查看tomcat的Dockerfile



从上面的可以看出最后执行了一个catalina.sh的命令  相当于./bin/catalina.sh    
运行如下命令，可以看出下图的效果

docker run -it -p 9999:8080 tomcat ls -lh



发现这个tomcat的容器启动之后就退出了

那是因为在启动容器的时候后面加了ls -lh那么会覆盖Dockerfile里面的  CMD ["catalina.sh", "run"]

## 2，ENTRYPOINT说明

使用ENTRYPOINT来运行命令，在run 运行的参数会追加到新的命令后面

## 3，自定义ls镜像来说明CMD和ENTRYPOINT的区别

目地：讲解CMD和ENTRYOINT的区别

编写Dockerfile

FROM centos

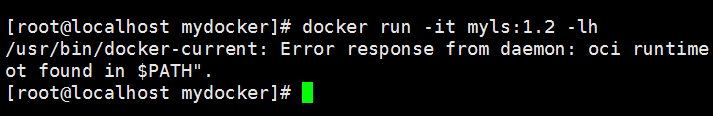
CMD [ "ls", "-a" ]

构建镜像

docker build -t myls:1.2 .

启动镜像

docker run -it myls:1.2 -lh



出现错误，这是因为Dockerfile里面使用的是CMD ["ls", "-a"]运行是传一个-lh过去会覆盖

修改Dockerfile

FROM centos

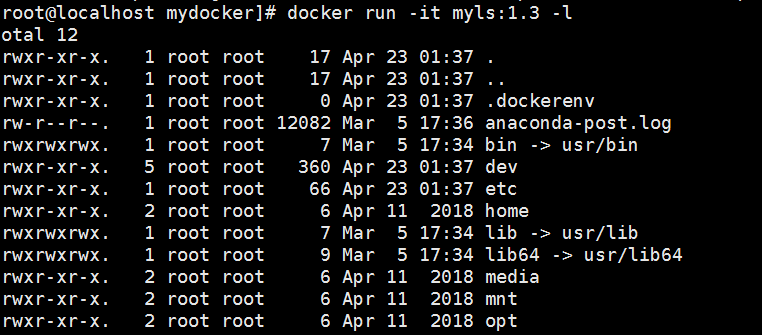
ENTRYPOINT [ "ls", "-a" ]

构建镜像

docker build -t myls:1.3 .

运行镜像

docker run -it myls:1.3 -l



发现 【 -l 】被追加到 ls 后面了哦

# 15【掌握】Dockerfile详解【4】

自定义一个tomcat

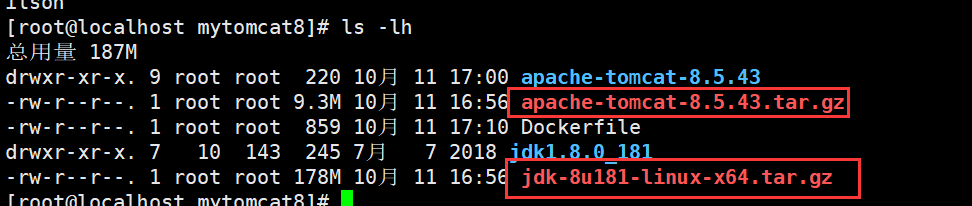
基于centos 去做

|--安装jdk 安装tomcat

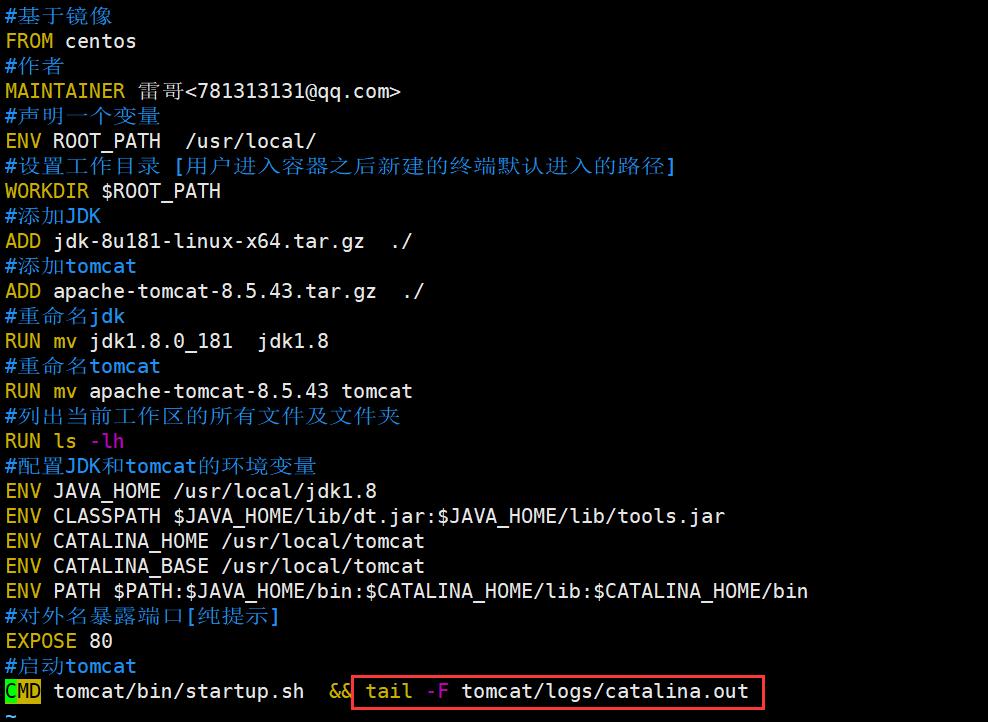
基于openjdk去做

|--安装tomcat

## 下载jdk和tomcat上传到linux并和Dockerfile放一个目录



## 编写Dockerfile

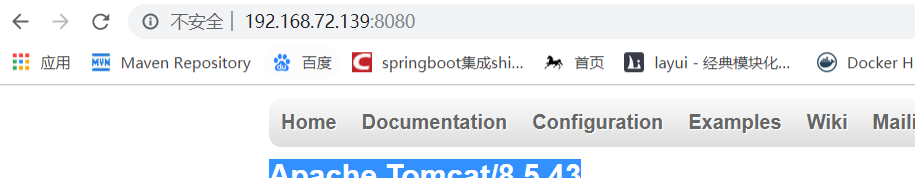


## 使用Dockerfile构造镜像



## 运行测试





# 16【掌握】Dockerfile详解【5】

## 制作jar包的镜像

基于jdk

### 生成jar



### 上传jar

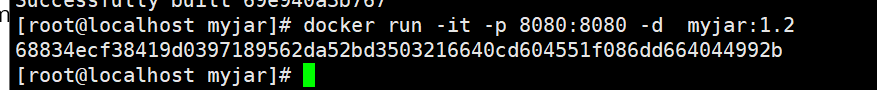
### 编写dockerfiler



### 生成镜像



### 运行镜像测试



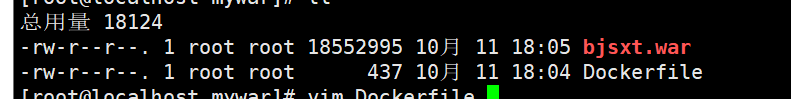
## 制作war包镜像

基于tomcat

### 生成war



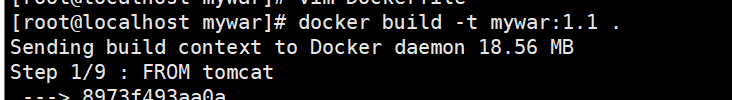
### 上传war



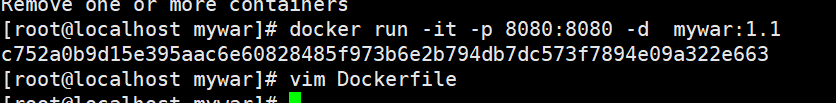
### 编写dockerfiler



### 生成镜像



### 运行镜像测试



# 17【熟悉】本地镜像发布到阿里云

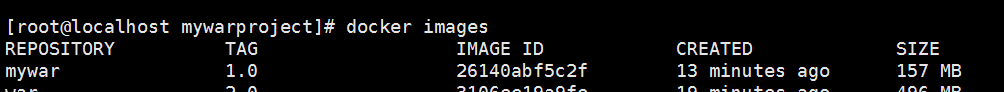
## 1，镜像的生成方法

1，docker commit

2，Dockerfile

## 2，本地镜像推送到阿里云

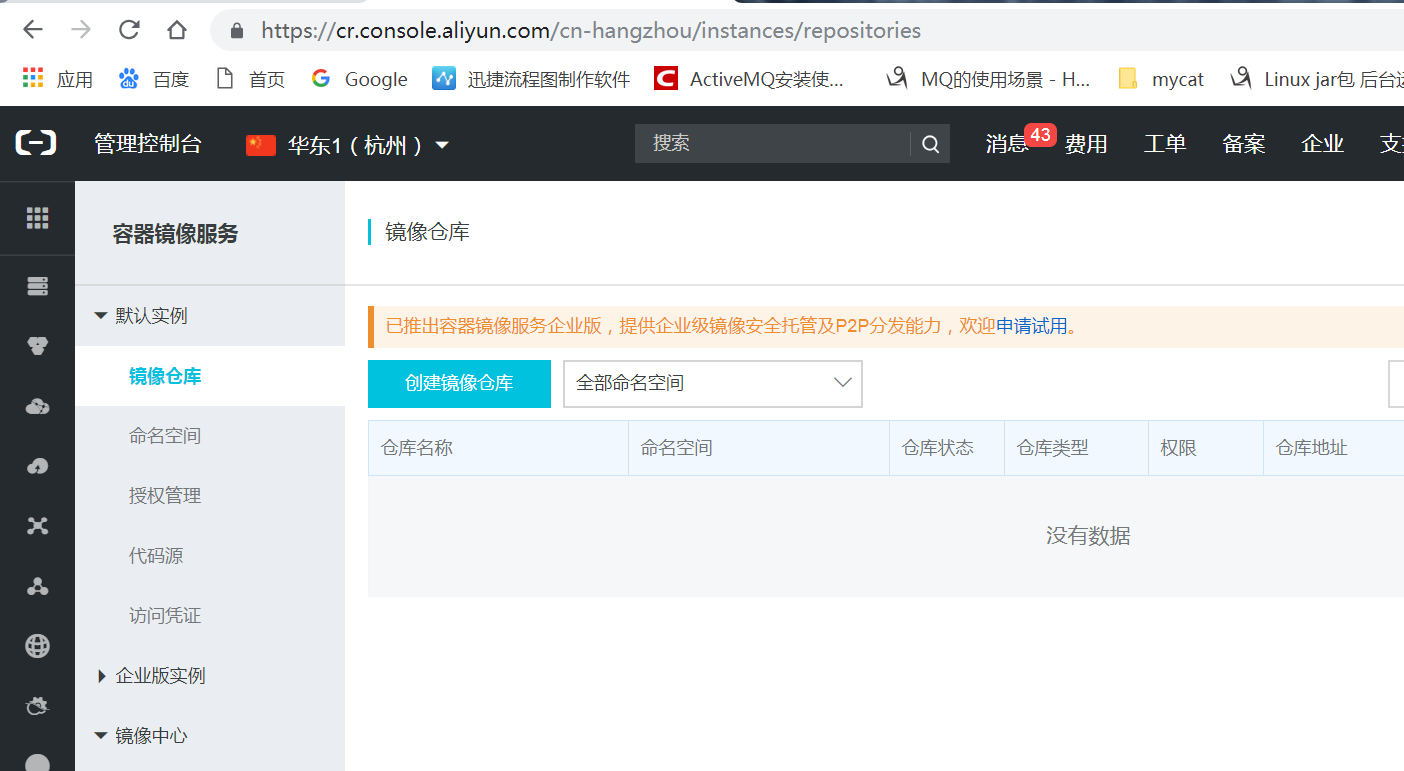
#### 2.1以修改的mywar的image为例来说明



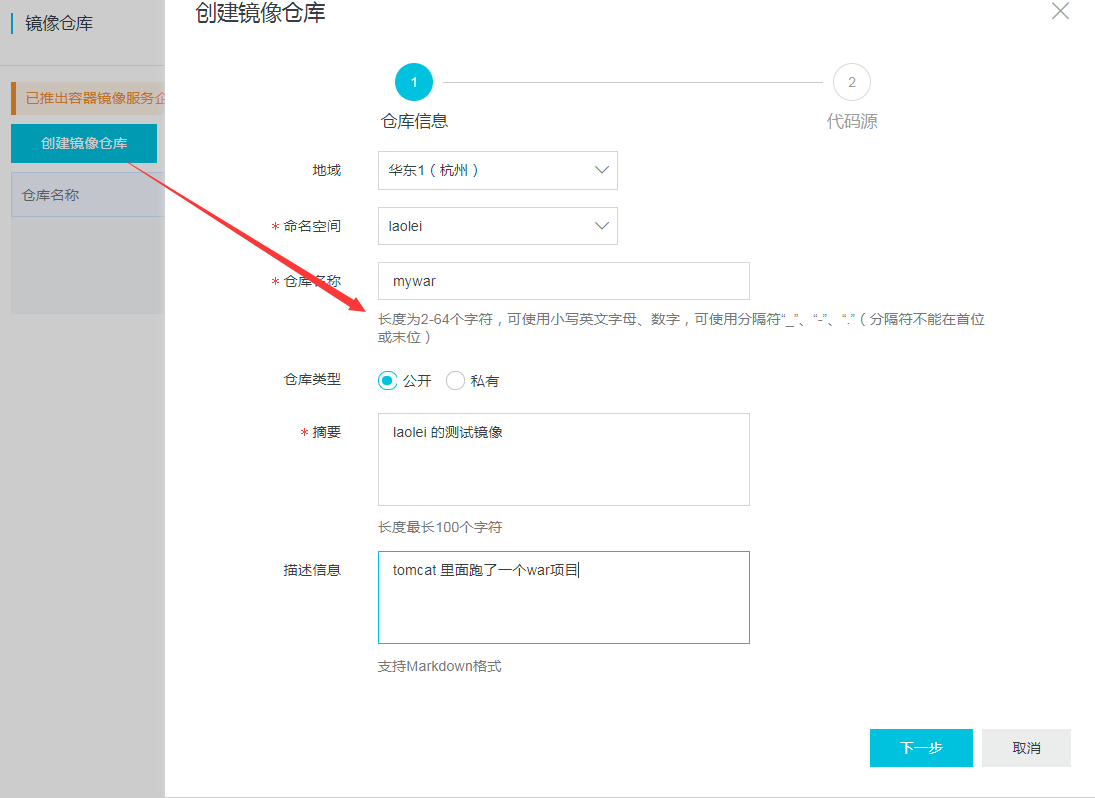
#### 2.2登陆阿里开发平台

<https://dev.aliyun.com/search.html>   完成登陆

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



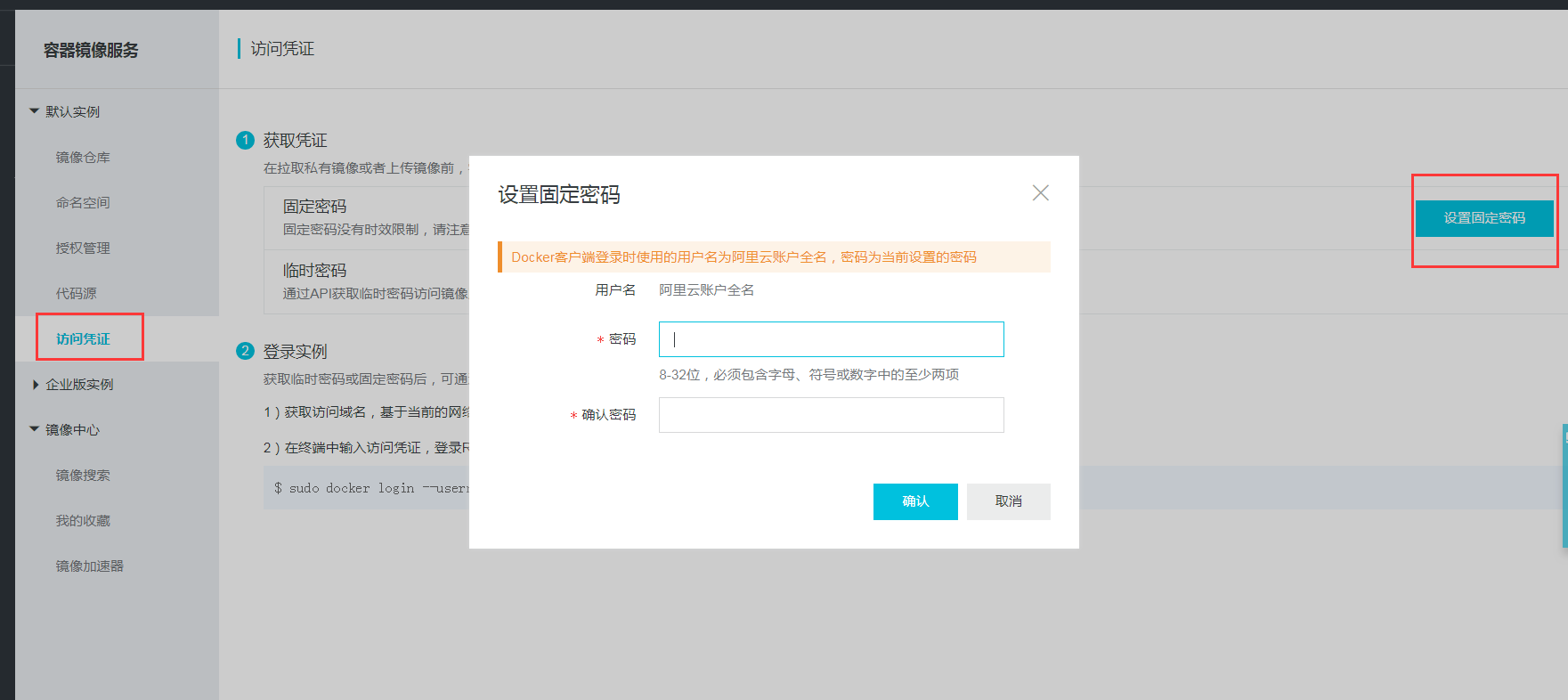
#### 2.3创建镜像仓库



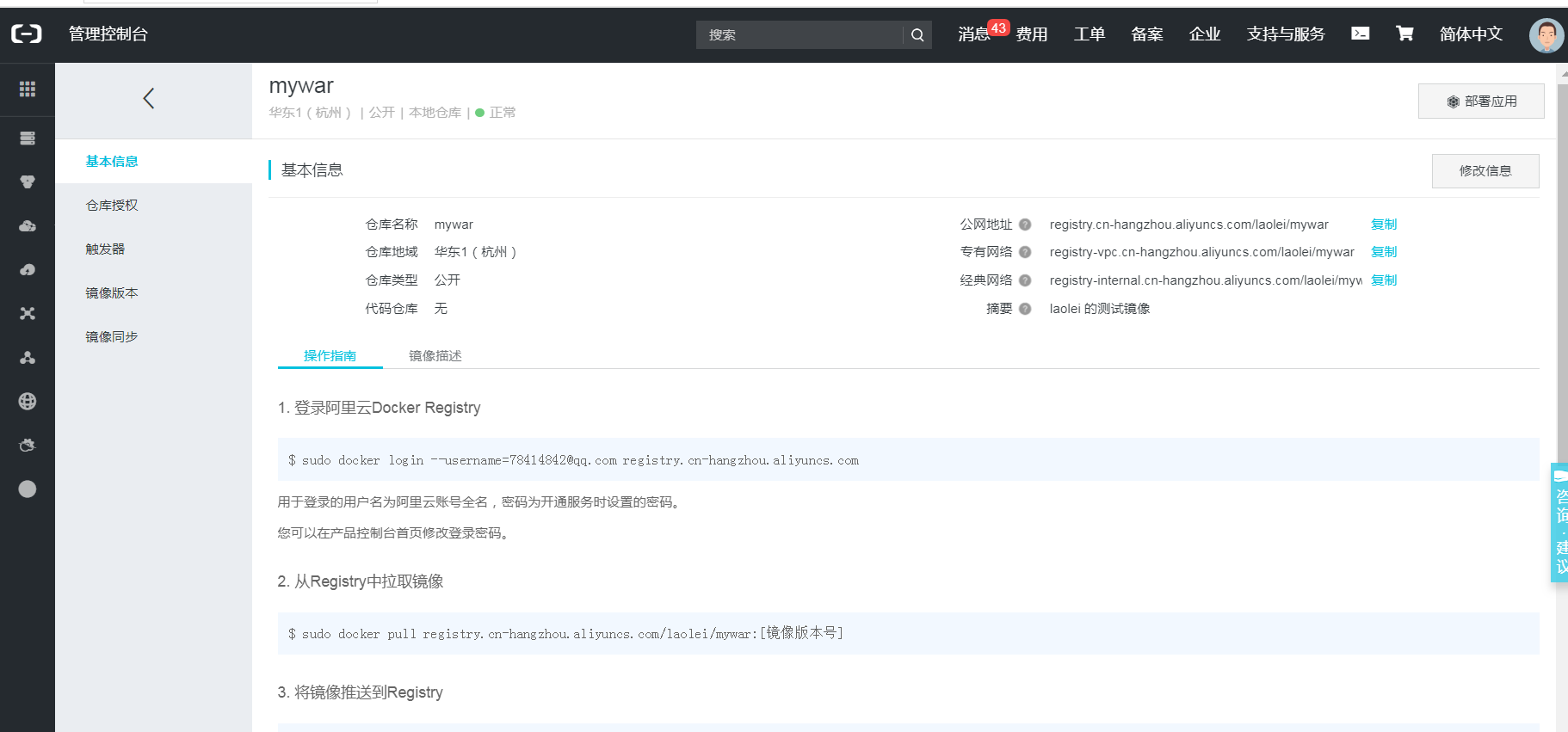




#### 2.4设置镜像仓库的访问密码



#### 2.5 打开阿像页面

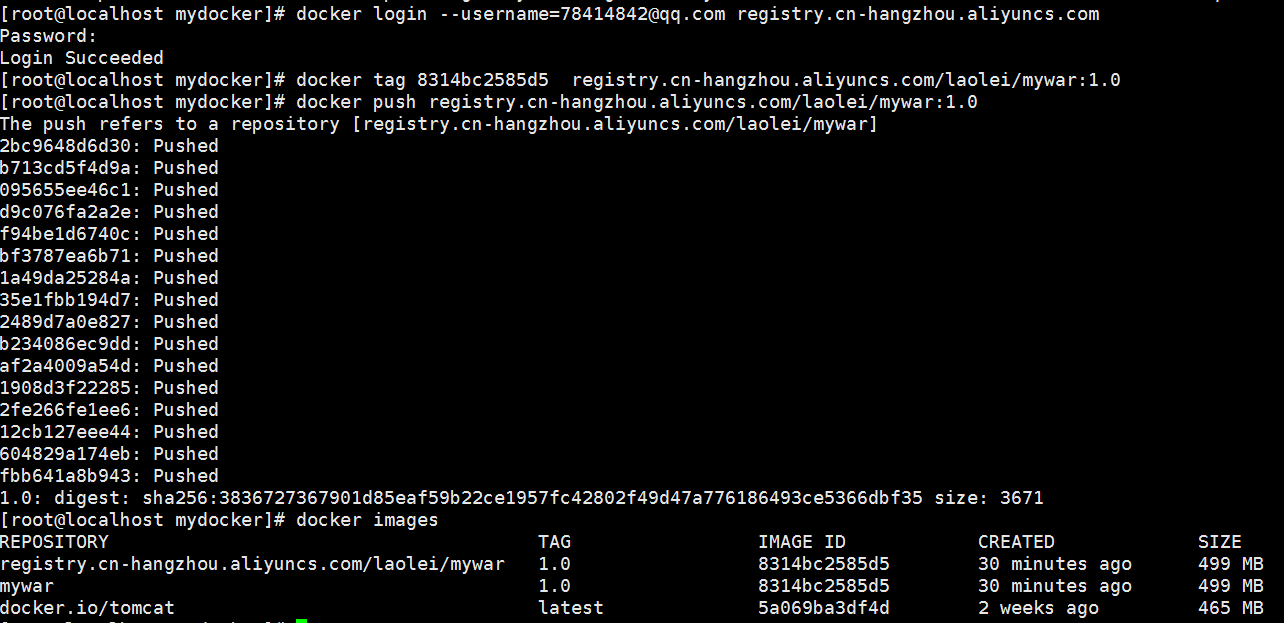


按里面的操作来就行了

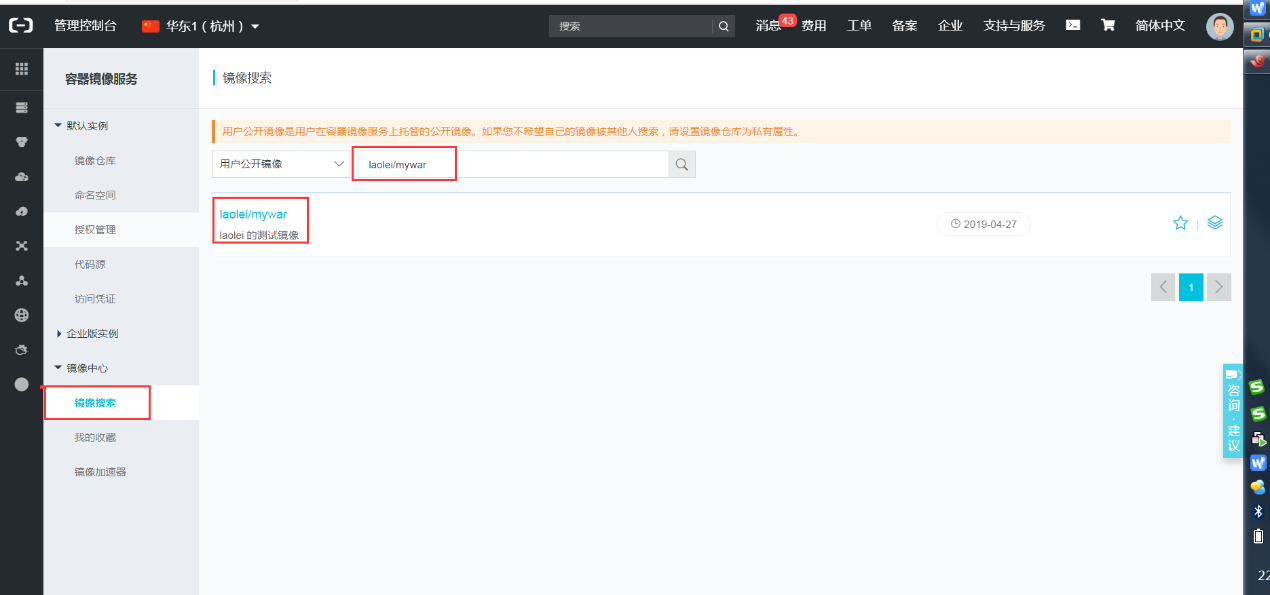
$ sudo docker login --username=78414842@qq.com registry.cn-hangzhou.aliyuncs.com

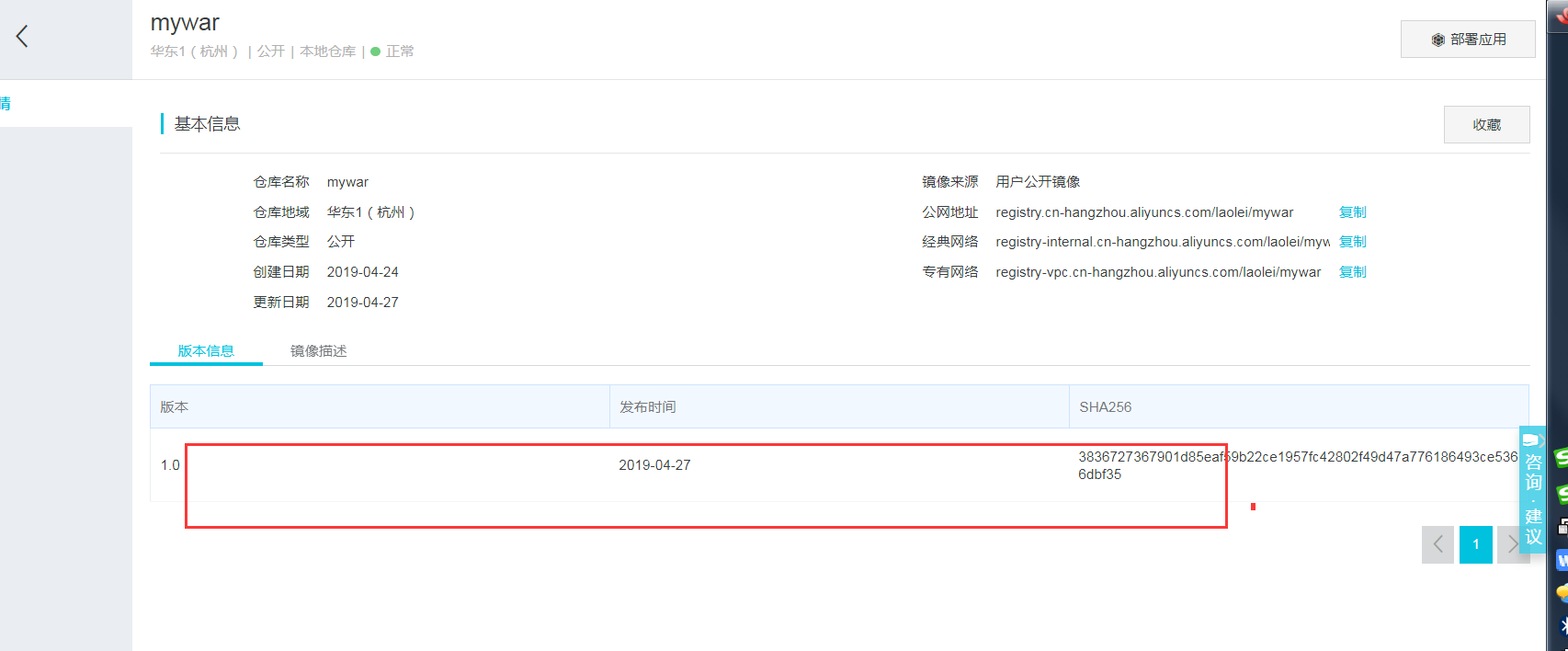
$ sudo docker tag [ImageId] registry.cn-hangzhou.aliyuncs.com/laolei/mywar:[镜像版本号]

$ sudo docker push registry.cn-hangzhou.aliyuncs.com/laolei/mywar:[镜像版本号]



#### 2.6  的阿里的公有仓库里面查询试试



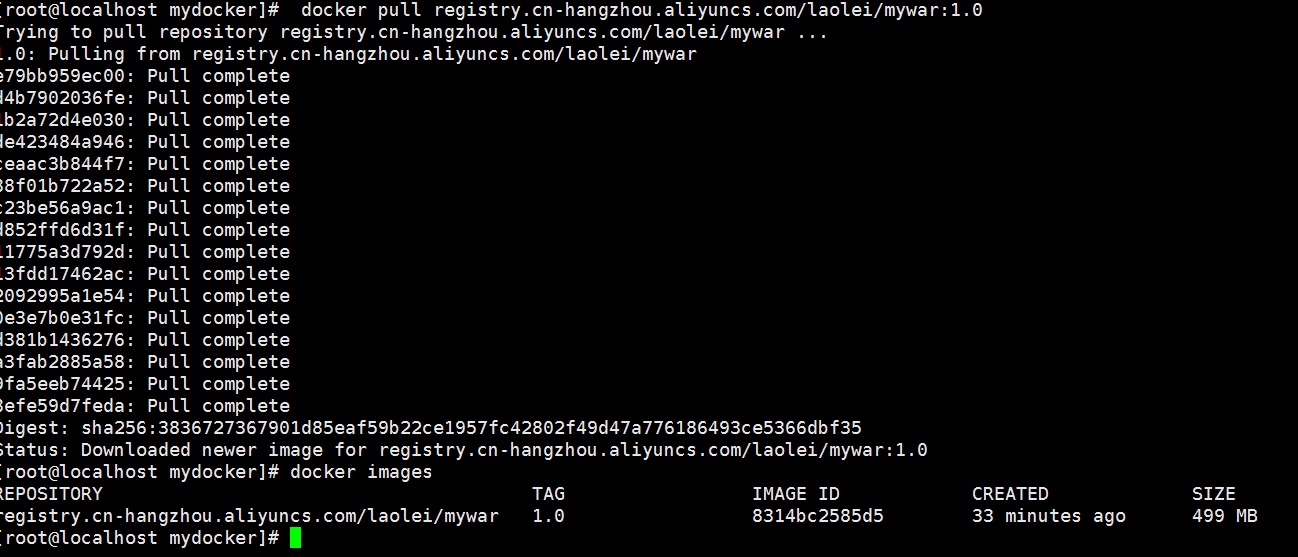


**4，把阿里云的镜像下载到本地**

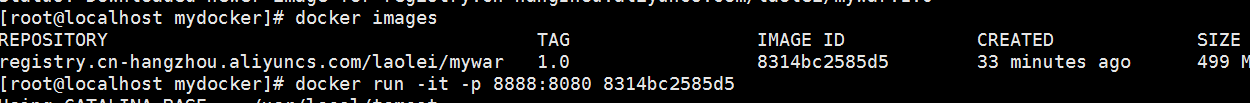
4.1语法

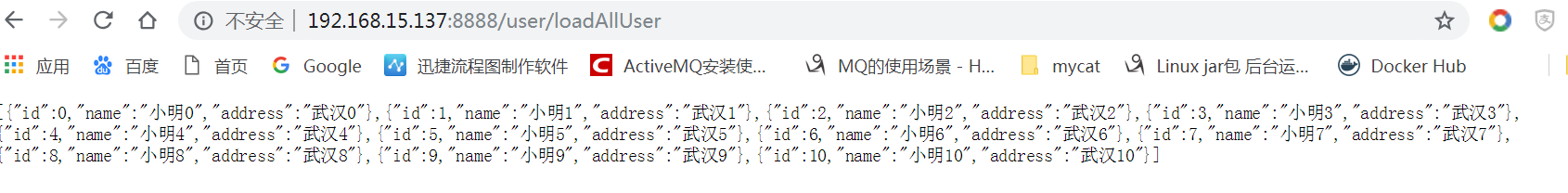
docker pull registry.cn-hangzhou.aliyuncs.com/laolei/mywar:[镜像版本号]

4.2案例



4.3运行测试





# 18【掌握】Docker的四种网络模式

## 1，概述

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：

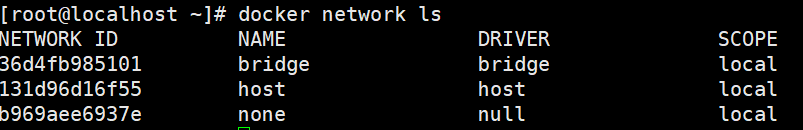
bridge模式：使--net =bridge指定，默认设置；

host模式：使--net =host指定；

none模式：使--net =none指定；

container模式：使--net =container:NAMEorID指定。

可以使用docker network ls来查看



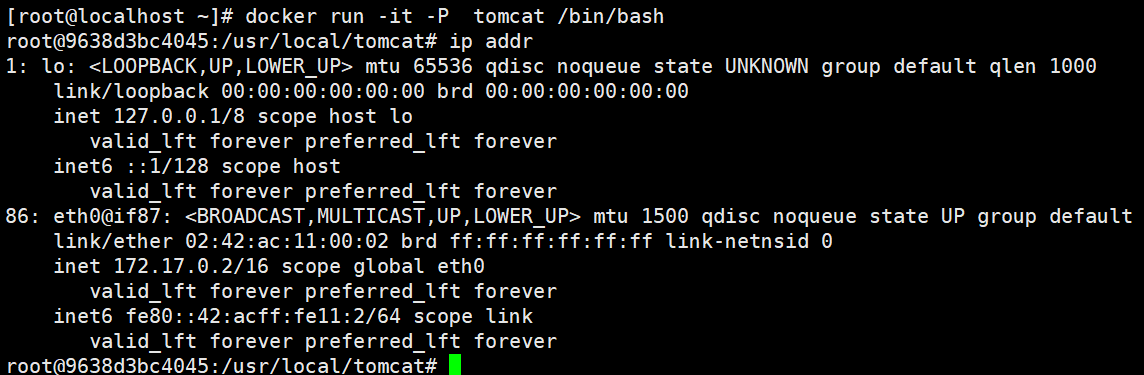
下面我们分别来讲解docker的四种网络模式：

## 2，bridge模式

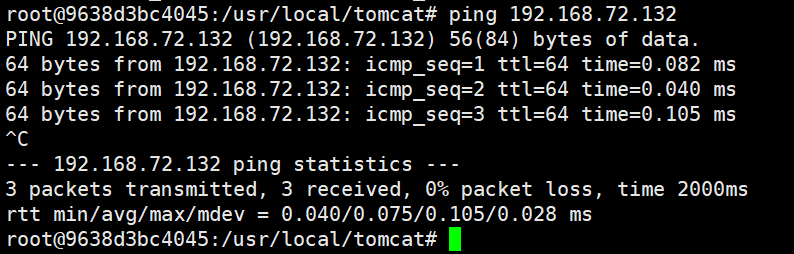
bridge模式是Docker默认的网络设置，此模式会为每一个容器分配Network Namespace、设置IP等，并将并将一个主机上的Docker容器连接到一个虚拟网桥上。当Docker server启动时，会在主机上创建一个名为docker0的虚拟网桥，此主机上启动的Docker容器会连接到这个虚拟网桥上。虚拟网桥的工作方式和物理交换机类似，这样主机上的所有容器就通过交换机连在了一个二层网络中。接下来就要为容器分配IP了，Docker会从RFC1918所定义的私有IP网段中，选择一个和宿主机不同的IP地址和子网分配给docker0，连接到docker0的容器就从这个子网中选择一个未占用的IP使用。如一般Docker会使用172.17.0.0/16这个网段，并将172.17.42.1/16分配给docker0网桥（在主机上使用ifconfig命令是可以看到docker0的，可以认为它是网桥的管理端口，在宿主机上作为一块虚拟网卡使用）。

具体操作：

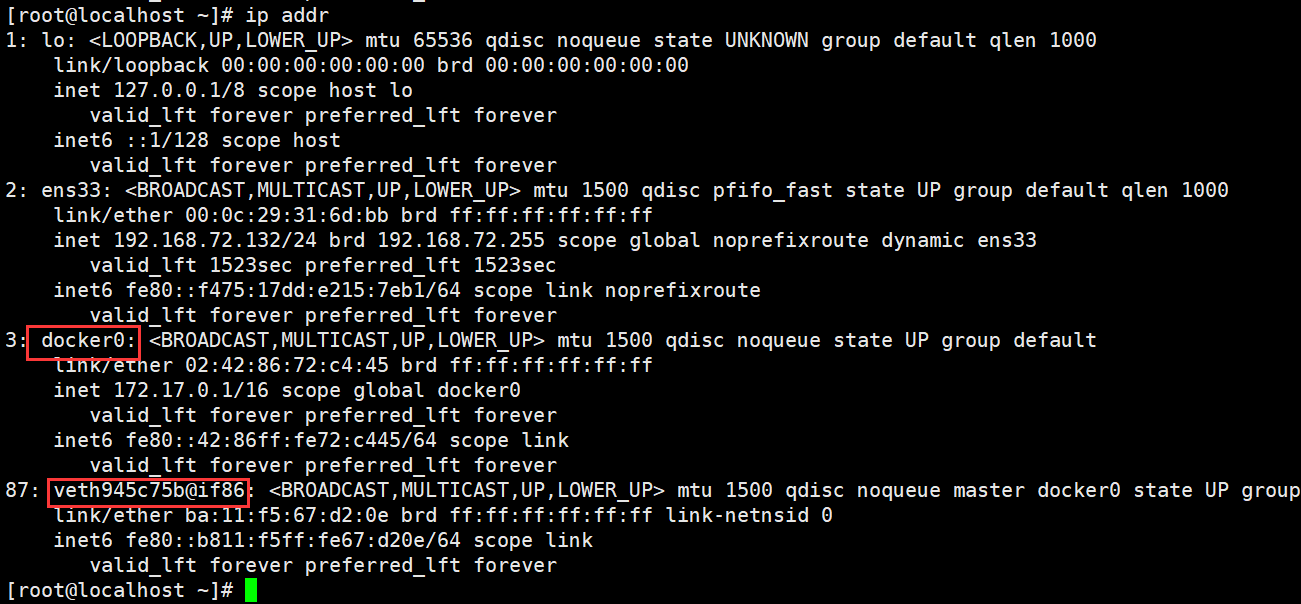
启动容器：（由于是默认设置，这里没指定网络–net =bridge，可以看到容器内创建了eth0）



使用ping命令连接Host网络发现，容器与Host网络是连通的：



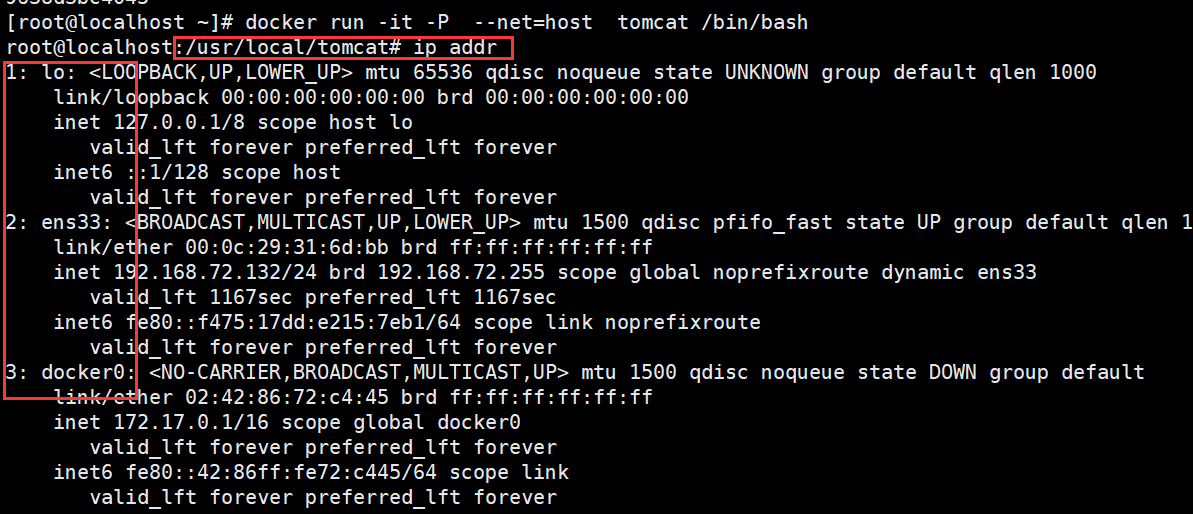
eth0实际上是veth pair的一端，另一端（veth945c）连接在docker0网桥上：



## 3，host模式

如果启动容器的时候使用host模式，那么这个容器将不会获得一个独立的Network Namespace，而是和宿主机共用一个Network Namespace。容器将不会虚拟出自己的网卡，配置自己的IP等，而是使用宿主机的IP和端口。

使用host模式启动容器：



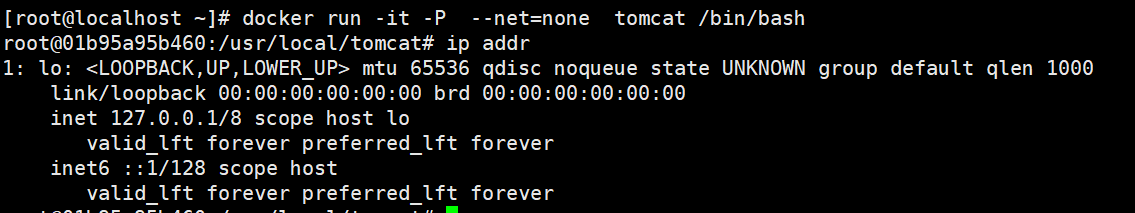
使用host模式启动容器后可以发现，使用ip addr查看网络环境时，看到的都是宿主机上的信息。这种方式创建出来的容器，可以看到host上的所有网络设备。

容器中，对这些设备有全部的访问权限。因此docker提示我们，这种方式是不安全的。如果在隔离良好的环境中（比如租户的虚拟机中）使用这种方式，问题不大。

## 4，none模式

在none模式下，Docker容器拥有自己的Network Namespace，但是，并不为Docker容器进行任何网络配置。也就是说，这个Docker容器没有网卡、IP、路由等信息。需要我们自己为Docker容器添加网卡、配置IP等。

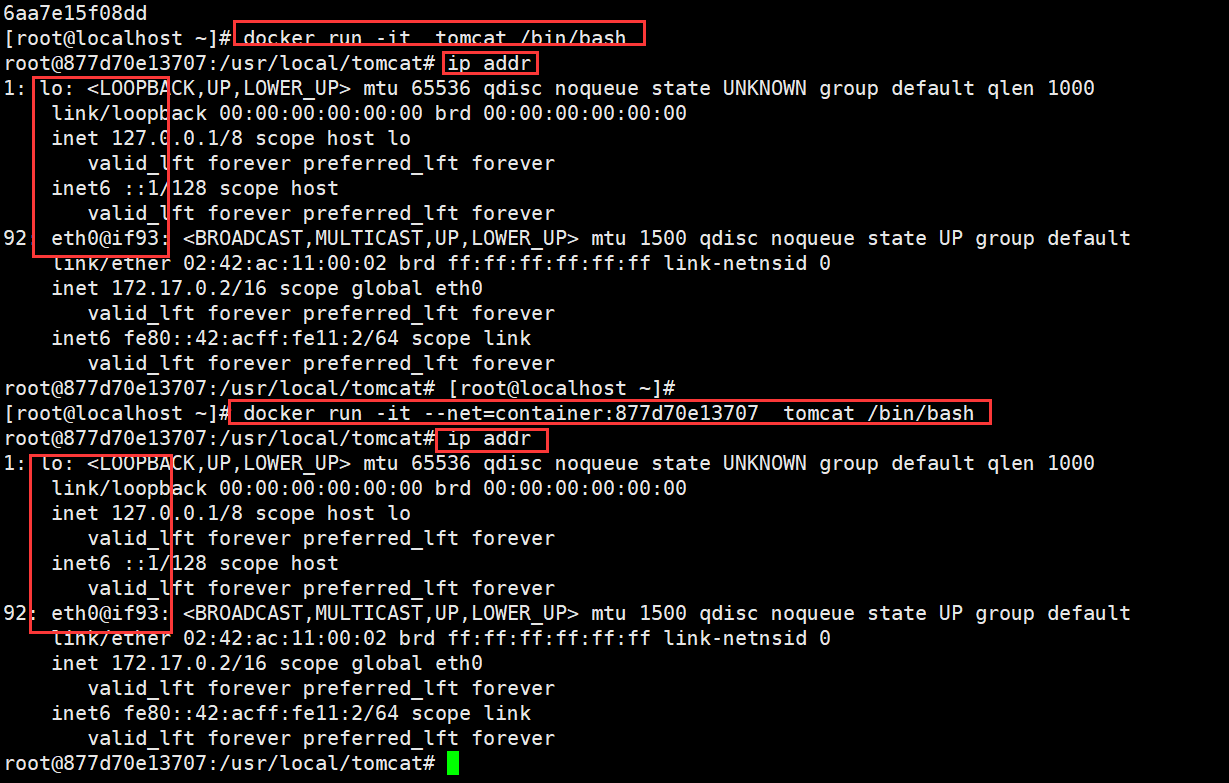
使用--net =none模式启动容器：



## 5，container模式

这个模式指定新创建的容器和已经存在的一个容器共享一个Network Namespace，而不是和宿主机共享。新创建的容器不会创建自己的网卡，配置自己的IP，而是和一个指定的容器共享IP、端口范围等。同样，两个容器除了网络方面，其他的如文件系统、进程列表等还是隔离的。两个容器的进程可以通过lo网卡设备通信。

使用--net =container模式启动容器：



通过该例子可以看出来，两者的网络完全相同。

# 19【掌握】docker常用安装

**1，安装步骤梳理**

1. 搜索镜像
2. 拉取镜像
3. 查看镜像
4. 启动镜像
5. 停止容器
6. 移除容器

**2，安装tomcat**

docker hub好查找tomcat镜像  docker search tomcat

从docker hub上拉取tomcat镜像到本地  docker pull tomcat

使用docker images查看是否有拉取到tomcat

使用tomcat镜像运行镜像成为容器

    docker run -it -p 8080:8080 tomcat

                -p 主机端口:容器端口

                -P 随机分配端口

                -i  交互

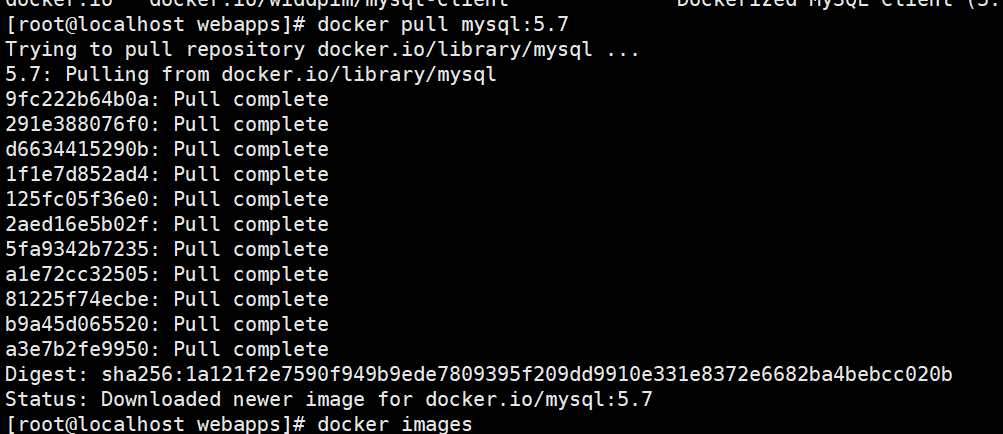
                -t 终端

**3，安装mysql**

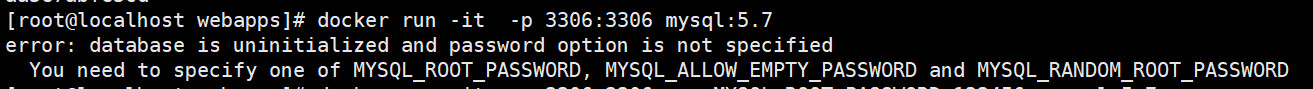
docker hub上查找mysql镜像  docker search mysql



docker hub上拉取mysql镜像到本地  docker pull mysql:5.7



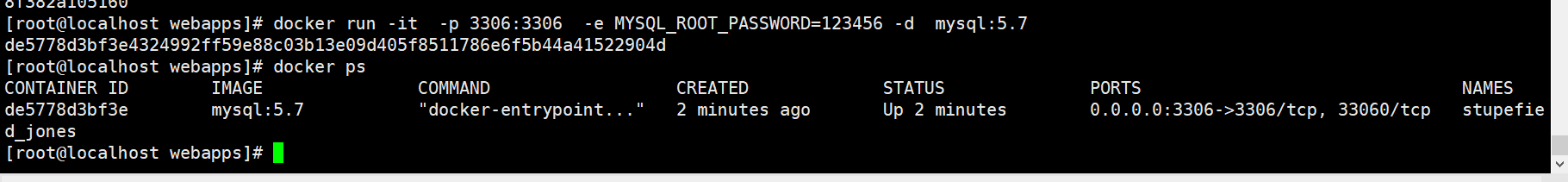
使用mysql5.7镜像创建容器运行存在问题



这是因为mysql镜像一启动中在加设置用户名和密码

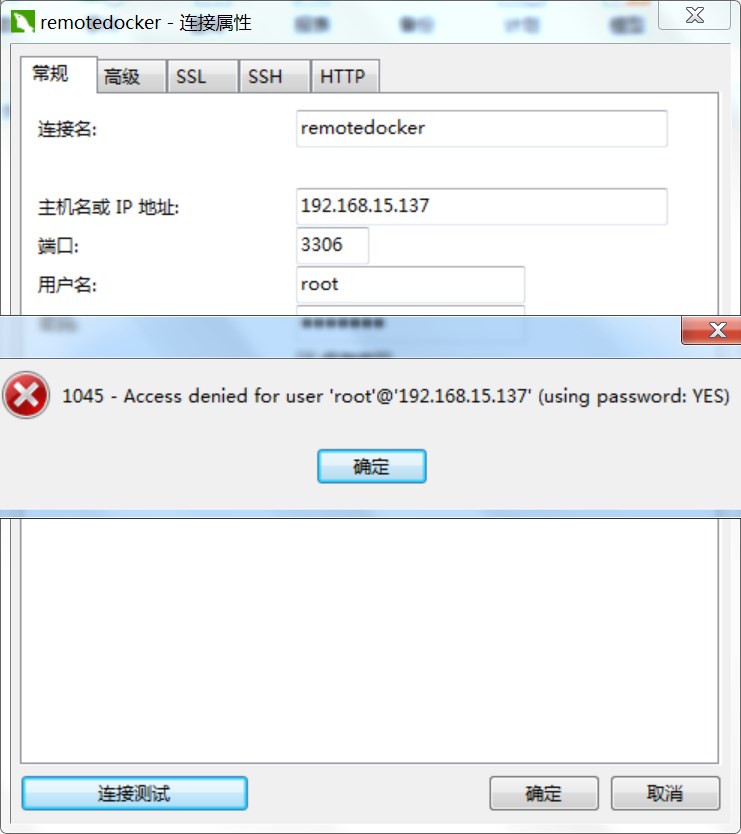
修改成下面的命令

docker run --name mysql -p 3306:3306 -e MYSQL\_ROOT\_PASSWORD=123456 -d mysql:5.7



发现启动起来了

**4，使用navcat连接一下**



如果是5.7的数据库不会报错

发现报错，这是因为docker 的mysql里面的localhost  -root 的密码是启动时设置的是123456现%-root的没有设置

下面是解决方法

1, 进入mysql的docker 容器

docker exec -it mysql /bin/bash

2，登陆mysql

mysql -u root -p

输入密码

3,使用mysql数据库

use mysql

4,执行修改密码的命令

update user set password=password(123456) where host='%'

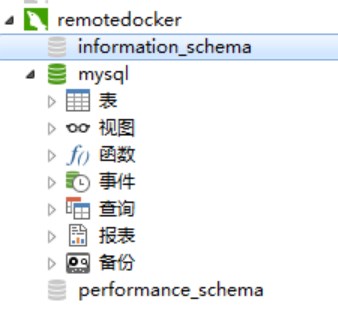
5,重启mysql的docker 容器

exit退出

exit退出mysql容器

docker restart mysql

再就可以进行远程连接了哦



如果出现使用宿主机IP无法访问的情况 在宿主机里面执行如下命令

请顺序运行以下命令：

nmcli connection modify docker0 connection.zone trusted

systemctl stop NetworkManager.service

firewall-cmd --permanent --zone=trusted --change-interface=docker0

systemctl start NetworkManager.service

nmcli connection modify docker0 connection.zone trusted

systemctl restart docker.service

再重启mysql和tomcat的容器

**5，相关文件地址配置**

可以在运行容器时设置 ：

docker run -p 3306:3306 --name mysql -v $PWD/conf/my.cnf:/etc/mysql/my.cnf -v $PWD/logs:/logs -v $PWD/data:/mysql\_data -e MYSQL\_ROOT\_PASSWORD=123456 -d mysql

但要求对应路径对应文件已存在，才能成功挂载相内容到对应位置。

命令说明：

-p 3306:3306：将容器的3306端口映射到主机的3306端口

-v $PWD/conf/my.cnf:/etc/mysql/my.cnf：将主机当前目录下的conf/my.cnf挂载到容器的/etc/mysql/my.cnf

-v $PWD/logs:/logs：将主机当前目录下的logs目录挂载到容器的/logs

-v $PWD/data:/mysql\_data：将主机当前目录下的data目录挂载到容器的/mysql\_data

-e MYSQL\_ROOT\_PASSWORD=123456：初始化root用户的密码