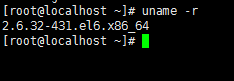
# 一、Linux6.5 离线安装docker-17.12.1

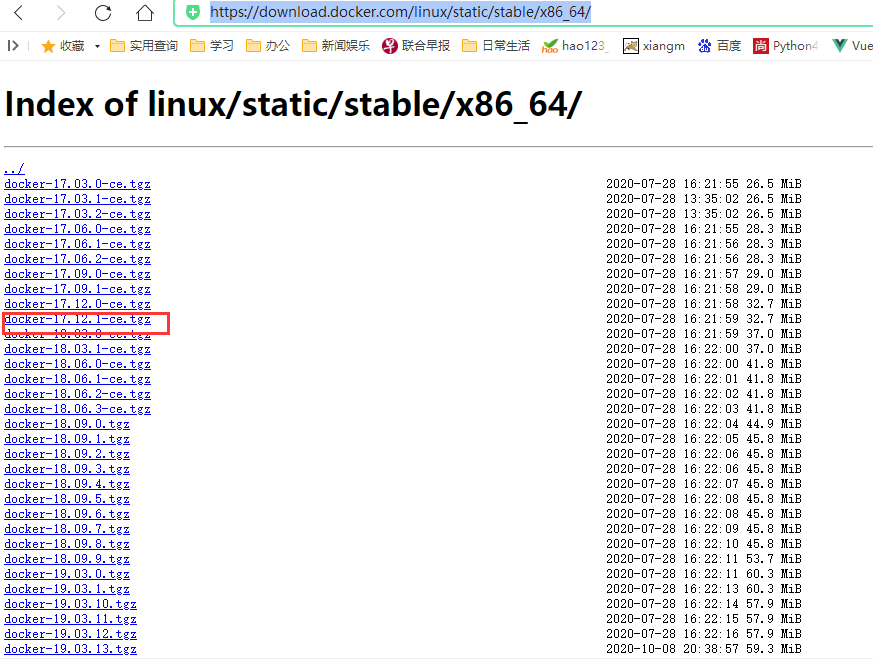
## 下载Docker

下载地址是<https://download.docker.com/linux/static/stable/x86_64/，由于docker1.8>起 要求内核3.10起，参看内核

uname –r

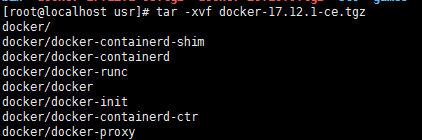


发现我们的版本低于3.10，所以下载docker1.7版本



## 解压安装

tar -xvf docker-17.12.1-ce.tgz



## 将解压出来的docker文件内容移动到 /usr/bin/ 目录下

cp docker/\* /usr/bin/

dockerd &

## 将docker注册为service服务

如果目录不存在，需新建对应目录，否则不能创建文件

vim /etc/systemd/system/docker.service

## 启动docker服务

启动前先授权

chmod +x /etc/systemd/system/docker.service

# 二、CentOS7 离线安装docker-19.03.9

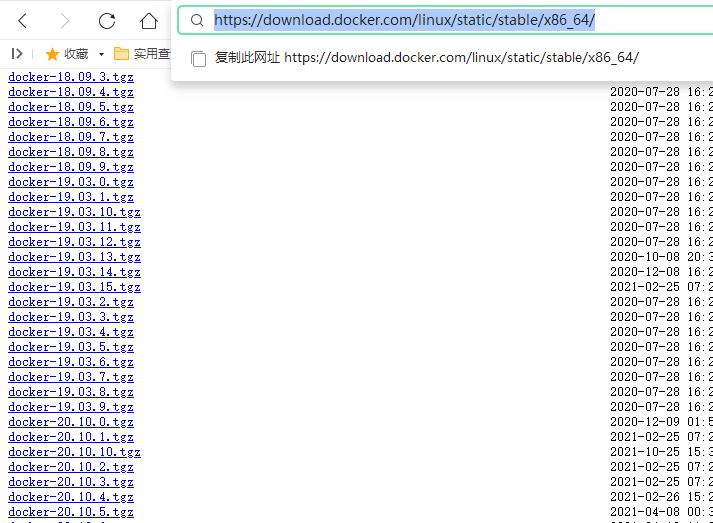
## 下载Docker

安装之前，要求操作系统为64位，内核版本为3.10及以上。  
确认本机已经安装了满足要求的Linux内核。使用命令uname -r来检查内核版本信息。



下载docker-19.03.9.tgz，将二进制包下载并上传服务器，

下载地址为<https://download.docker.com/linux/static/stable/x86_64/>



## 删除已有的Docker

yum remove docker \

docker-client \

docker-client-latest \

docker-common \

docker-latest \

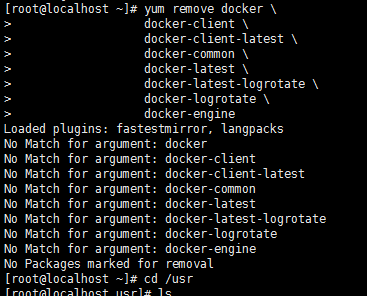
docker-latest-logrotate \

docker-logrotate \

docker-engine

或者

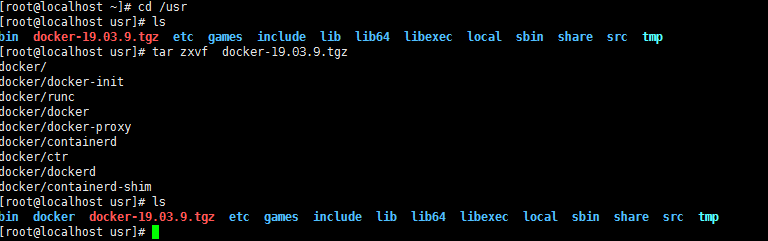
**sudo** **yum remove** docker \  
                  docker-client \  
                  docker-client-latest \  
                  docker-common \  
                  docker-latest \  
                  docker-latest-logrotate \  
                  docker-logrotate \  
                  docker-engine



## 解压

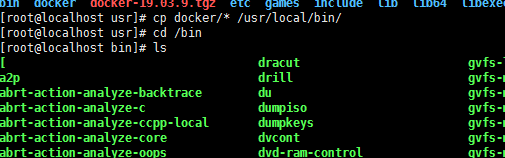
在服务器上解压安装包，tar zxvf  [docker-19.03.7.tgz](https://download.docker.com/mac/static/stable/x86_64/docker-19.03.7.tgz)  ，解压后出现docker目录

tar zxvf docker-19.03.9.tgz



## 将docker目录下的可执行文件拷贝到系统的bin目录下

cp docker/\* /usr/local/bin/



## 查看docker版本，查看是否安装成功。

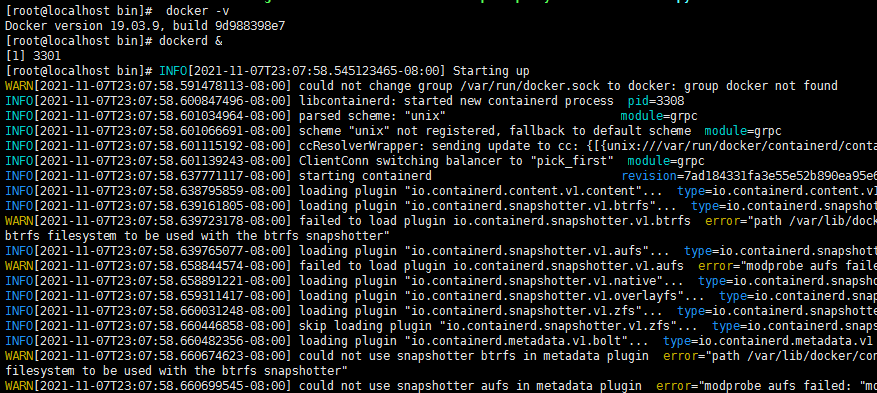
docker -v



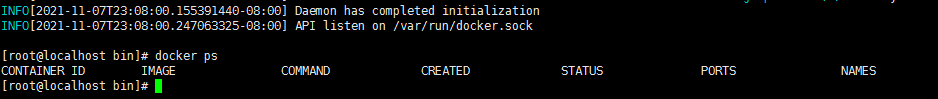
## 启动

启动docker。命令：dockerd &。执行完命令以后，终端会出现一堆日志，直接执行回车，按下电脑上的enter键，日志内容消失，docker已在后台执行，使用docker ps 查看，已出现相关信息，如图所示：

dockerd &



docker ps



docker安装成功！

## 自动启动

将docker添加到系统的systemctl服务中，使之能够开机自启

1. 将docker注册为service

[root@localhost bin]# vim /etc/systemd/system/docker.service

内容为：

[Unit]

Description=Docker Application Container Engine

Documentation=https://docs.docker.com

After=network-online.target firewalld.service

Wants=network-online.target

[Service]

Type=notify

# the default is not to use systemd for cgroups because the delegate issues still

# exists and systemd currently does not support the cgroup feature set required

# for containers run by docker

ExecStart=/usr/local/bin/dockerd

ExecReload=/bin/kill -s HUP $MAINPID

# Having non-zero Limit\*s causes performance problems due to accounting overhead

# in the kernel. We recommend using cgroups to do container-local accounting.

LimitNOFILE=infinity

LimitNPROC=infinity

LimitCORE=infinity

# Uncomment TasksMax if your systemd version supports it.

# Only systemd 226 and above support this version.

#TasksMax=infinity

TimeoutStartSec=0

# set delegate yes so that systemd does not reset the cgroups of docker containers

Delegate=yes

# kill only the docker process, not all processes in the cgroup

KillMode=process

# restart the docker process if it exits prematurely

Restart=on-failure

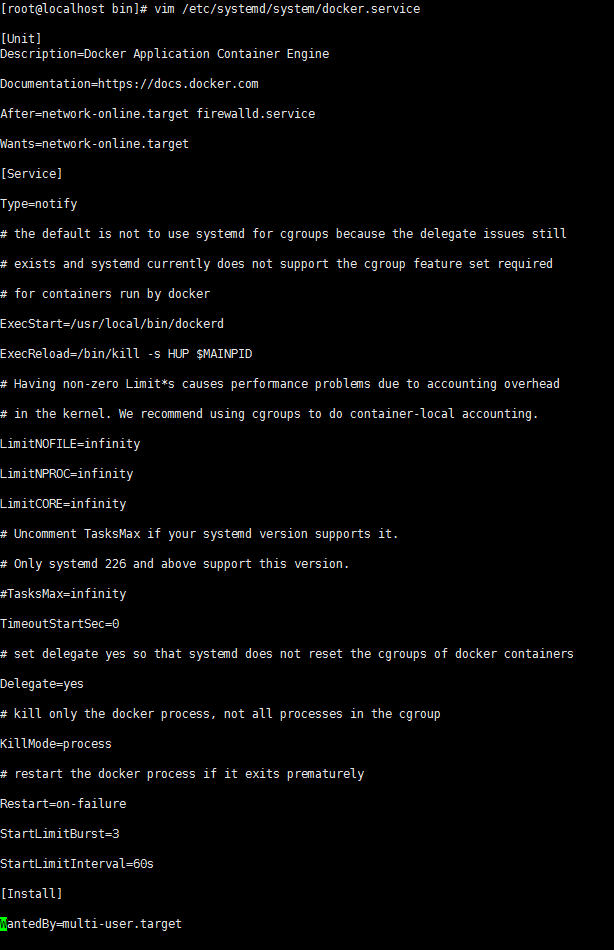
StartLimitBurst=3

StartLimitInterval=60s

[Install]

WantedBy=multi-user.target

如下图：



1. 授权

[root@localhost bin]# chmod +x /etc/systemd/system/docker.service

1. 启动

#重载unit配置文件

systemctl daemon-reload

[root@localhost bin]# systemctl daemon-reload

#启动Docker

systemctl start docker

[root@localhost bin]# systemctl start docker

#设置开机自启

systemctl enable docker.service

[root@localhost bin]# systemctl enable docker.service

1. 验证

systemctl status docker #查看Docker状态

docker -v #查看Docker版本

docker info

## 容器自动启动

设置自启动参数restart

docker run -d -p 8080:8080 --restart=always myweb

--restart具体参数值详细信息：

       no -  容器退出时，不重启容器；

       on-failure - 只有在非0状态退出时才从新启动容器；

       always - 无论退出状态是如何，都重启容器；

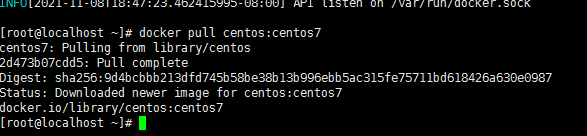
如果创建时未指定 --restart=always ,可通过update 命令

docker update --restart=always myweb

# 三、Docker 安装jdk1.8

## 下载centos镜像

docker pull centos:centos7



这步操作需要联网

## 下载jdk1.8并上传

下载jdk1.8并上传到/usr/local/src目录，进行解压

cd /usr/local/src/

tar -zxvf jdk-8u311-linux-x64.tar.gz



拷贝字体文件夹font

## 创建Dockerfile

在/usr/local目录下创建jdk目录，把/usr/local/src下的jdk-8u311-linux-x64.tar.gz复制到/usr/local/jdk目录下，然后创建Dockerfile

mkdir /usr/local/jdk

cd /usr/local/jdk

cp ../src/jdk-8u311-linux-x64.tar.gz ./

vi Dockerfile

Dockerfile的内容如下：

from centos:centos7

maintainer duhj

run mkdir -p /usr/local/jdk

workdir /usr/local/jdk

add jdk-8u311-linux-x64.tar.gz /usr/local/jdk

env JAVA\_HOME /usr/local/jdk/jdk1.8.0\_311

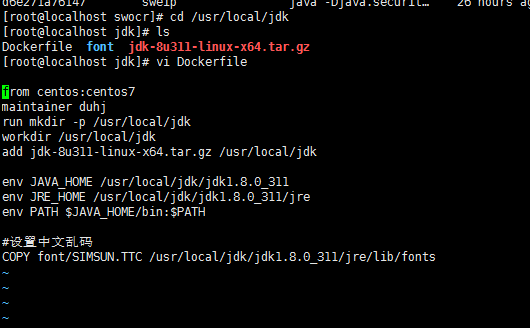
env JRE\_HOME /usr/local/jdk/jdk1.8.0\_311/jre

env PATH $JAVA\_HOME/bin:$PATH

#设置中文乱码

COPY font/SIMSUN.TTC /usr/local/jdk/jdk1.8.0\_311/jre/lib/fonts

如下图



## 使用Dockerfile构建jdk1.8镜像

必须在Dockerfil文件同一目录，如：/usr/local/jdk

docker build -t jdk1.8 .

或者

docker build -t jdk8:base .



如果第三步报错，错误大概为

OCI runtime create failed: container\_linux.go:349: starting container process caused "process\_linux.go:449: container init caused \"write /proc/self/attr/keycreate: permission denied\"": unknown

需要修改如下配置：（谨慎操作）

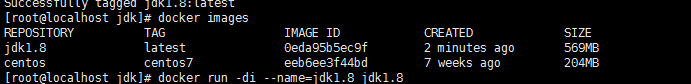
1. 编辑config文件：vi /etc/selinux/config
2. 将SELINUX=enforcing改为SELINUX=disable
3. 重启服务器

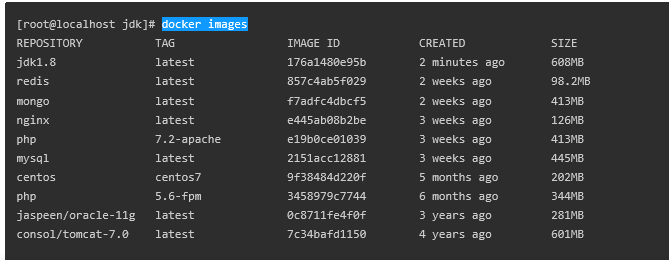
原因是：

* linux环境下使用离线方式（tgz安装包形式）安装docker
* 使用了docker load < xx.tar导入镜像
* 输入docker images和docker ps -a显示该镜像已正常导入，且已经通过docker run生成了容器

## 查看构建的镜像

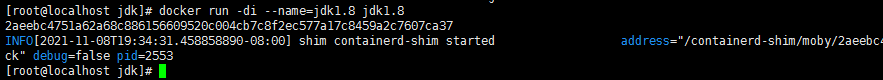
docker images





## 启动jdk容器

docker run -di --name=jdk1.8 jdk1.8



或者

进入镜像并查看JDK是否安装成功:

docker run -di jdk8:base

## 进入jdk容器，查看版本

#运行容器

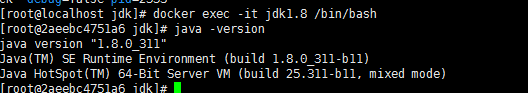
docker start jdk1.8

进入容器

[root@localhost jdk]# docker exec -it jdk1.8 /bin/bash

查看jdk版本

[root@1ab4fca37eac jdk]# java –version



# 四、Docker 安装tomcat（docker pull tomcat）

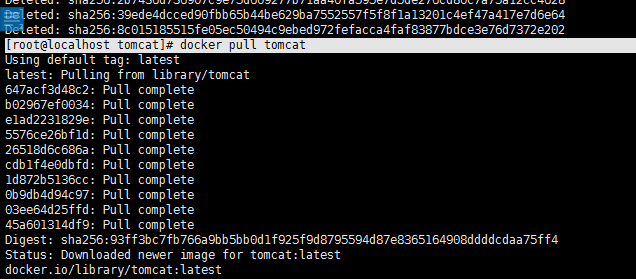
## 查找 Docker Hub 上的 tomcat 镜像：

docker search tomcat

## 拉取官方最新版本的 tomcat 镜像：

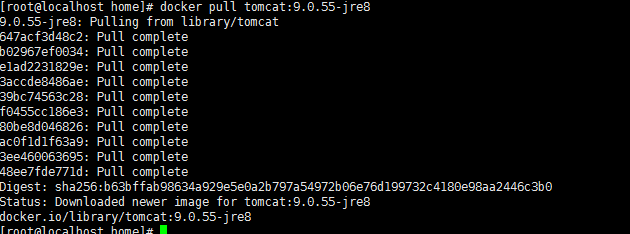
docker pull tomcat

[root@localhost tomcat]# docker pull tomcat



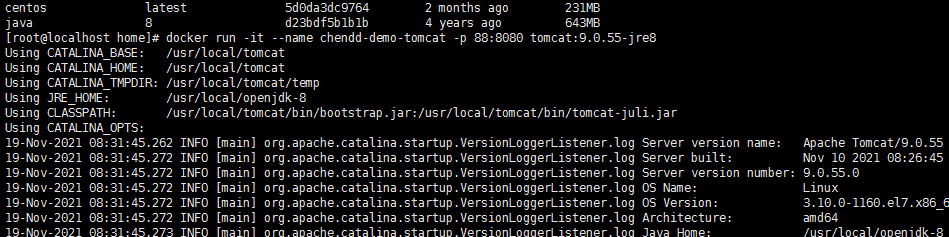
等待下载完成后，我们就可以在本地镜像列表里查到 REPOSITORY 为 tomcat 的镜像：

或者按照版本下载

[root@localhost home]# docker pull tomcat:9.0.55-jre8

按照版本下载启动

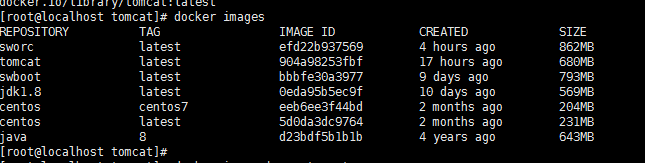
[root@localhost home]# docker run -it --name chendd-demo-tomcat -p 88:8080 tomcat:9.0.55-jre8



docker images|grep tomcat



或者docker images



## 启动

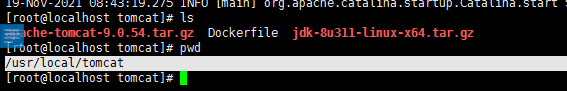
docker run -d -p 88:8080 tomcat

# 五、Docker 安装tomcat（文件安装）

## 上传tomcat到服务器

上传apache-tomcat-9.0.54.tar.gz和 jdk-8u311-linux-x64.tar.gz到/ usr/local/tomcat

安装的时候好像必须挂载jdk，虽然我已经单独加了jdk



## 编写Dockerfile文件

[root@localhost tomcat]# vi Dockerfile

文件内容如下：

#该镜像基于centos

FROM centos

#把java和tomcat添加到镜像中

ADD jdk-8u311-linux-x64.tar.gz /usr/local/jdk

ADD apache-tomcat-9.0.54.tar.gz /usr/local

#在基于centos的镜像中安装vim编辑器

#RUN yum -y install vim

ENV MYPATH /usr/local

WORKDIR $MYPATH

#配置java与tomcat环境变量

ENV JAVA\_HOME /usr/local/jdk/jdk1.8.0\_311

ENV CLASSPATH $JAVA\_HOME/lib/dt.jar:$JAVA\_HOME/lib/tools.jar

ENV CATALINA\_HOME /usr/local/apache-tomcat-9.0.54

ENV CATALINA\_BASE /usr/local/apache-tomcat-9.0.54

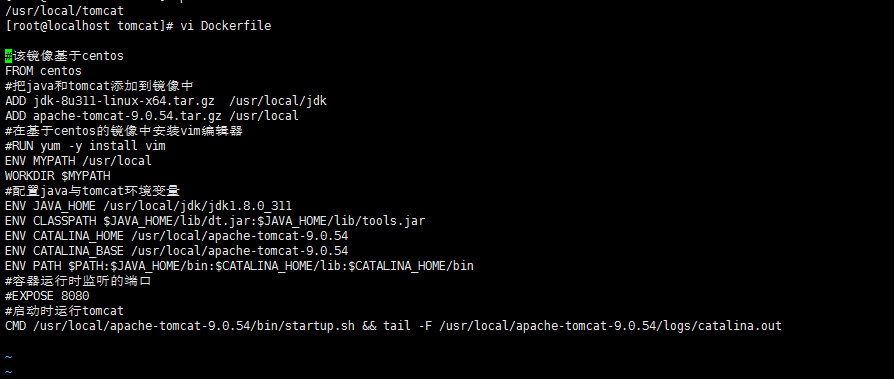
ENV PATH $PATH:$JAVA\_HOME/bin:$CATALINA\_HOME/lib:$CATALINA\_HOME/bin

#容器运行时监听的端口

#EXPOSE 8080

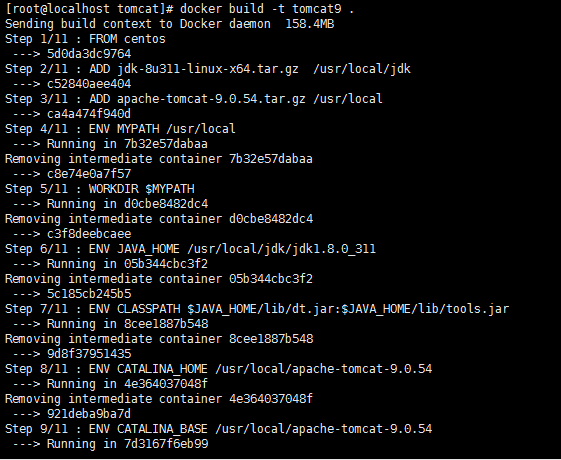
#启动时运行tomcat

CMD /usr/local/apache-tomcat-9.0.54/bin/startup.sh && tail -F /usr/local/apache-tomcat-9.0.54/logs/catalina.out

~ 

## 构建镜像

[root@localhost tomcat]# docker build -t tomcat9 .



## 启动运行

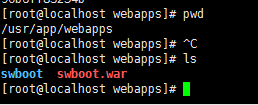
[root@localhost tomcat]# docker run -d -p 88:8080 tomcat9



注意，要开通linux 88 端口，88端口映射tomcat8080端口

## 部署应用

1. 上传部署文件到服务器，比如上传swboot.war 到/usr/app/webapps下，



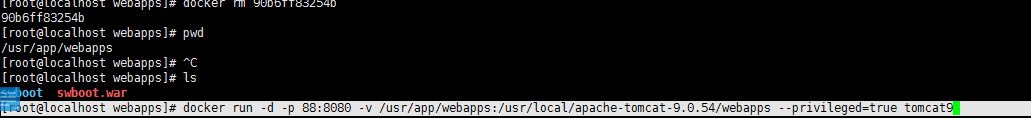
1. 部署应用，部署方式有三种，分别是挂载的方式，直接拷贝的方式和创建Dockfile文件的方式

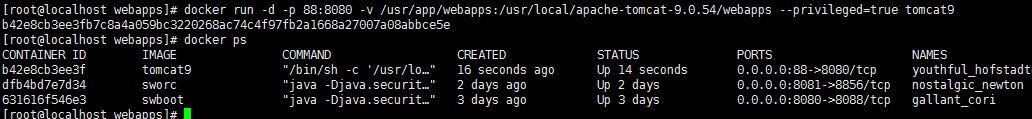
**方式1 挂载方式：**

启动的时候，把本地的war包挂载到容器里tomcat的部署目录，映射本地端口 88到容器的8080端口

docker run -d -p 88:8080 -v /usr/app/webapps:/usr/local/apache-tomcat-9.0.54/webapps --privileged=true tomcat9

[root@localhost webapps]# docker run -d -p 88:8080 -v /usr/app/webapps:/usr/local/apache-tomcat-9.0.54/webapps --privileged=true tomcat9





或者直接挂载war

docker run -d -v /usr/app/webapps/swboot.war:/usr/local/apache-tomcat-9.0.54/webapps/swboot.war -p 88:8080 tomcat9

查看启动日志

[root@localhost webapps]# docker logs -f -t --tail=800 b42e8cb3ee3f

**方式2 文件拷贝：**

把war从本地拷贝至容器对应的tomcat部署文件目录，启动tomcat后会自动解压war包

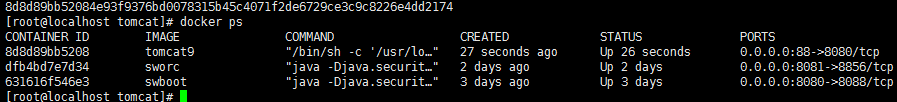
操作步骤如下：

启动tomcat 容器

[root@localhost tomcat]# docker run -d -p 88:8080 tomcat9

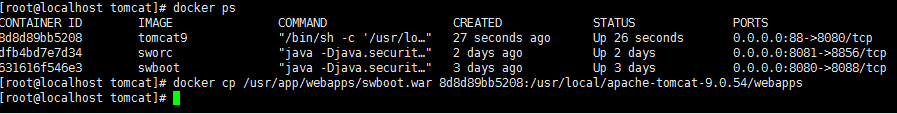


查看tomcat容器ID



拷贝文件

[root@localhost tomcat]# docker cp /usr/app/webapps/swboot.war 8d8d89bb5208:/usr/local/apache-tomcat-9.0.54/webapps



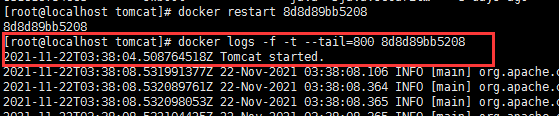
重启tomcat 或者重启 docker restart器

root@localhost tomcat]# docker restart 8d8d89bb5208



查看日志

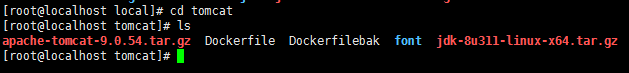
[root@localhost tomcat]# docker logs -f -t --tail=800 8d8d89bb5208



**方式3 Dockerfile文件：（建议）**

1）先创建tomcat镜像，tomcat镜像以jdk为基础镜像

在/usr/local/jdk中上传文件 font 和apache-tomcat-9.0.54.tar.gz，以jdk为基础，不用再add jdk



2）创建tomcat镜像

vi Dockerfile

#FROM centos

FROM jdk1.8

# 以jdk为基础镜像，不用安装java

#ADD jdk-8u311-linux-x64.tar.gz /usr/local/jdk

# 把jdk加入镜像并解压

ADD apache-tomcat-9.0.54.tar.gz /usr/local

#在基于centos的镜像中安装vim编辑器

#RUN yum -y install vim

ENV MYPATH /usr/local

WORKDIR $MYPATH

#配置java与tomcat环境变量

#ENV JAVA\_HOME /usr/local/jdk/jdk1.8.0\_311

#ENV CLASSPATH $JAVA\_HOME/lib/dt.jar:$JAVA\_HOME/lib/tools.jar

ENV CATALINA\_HOME /usr/local/apache-tomcat-9.0.54

ENV CATALINA\_BASE /usr/local/apache-tomcat-9.0.54

#ENV PATH $PATH:$JAVA\_HOME/bin:$CATALINA\_HOME/lib:$CATALINA\_HOME/bin

ENV PATH $PATH:$CATALINA\_HOME/lib:$CATALINA\_HOME/bin

COPY font/SIMSUN.TTC /usr/share/fonts/SIMSUN.TTC

#容器运行时监听的端口

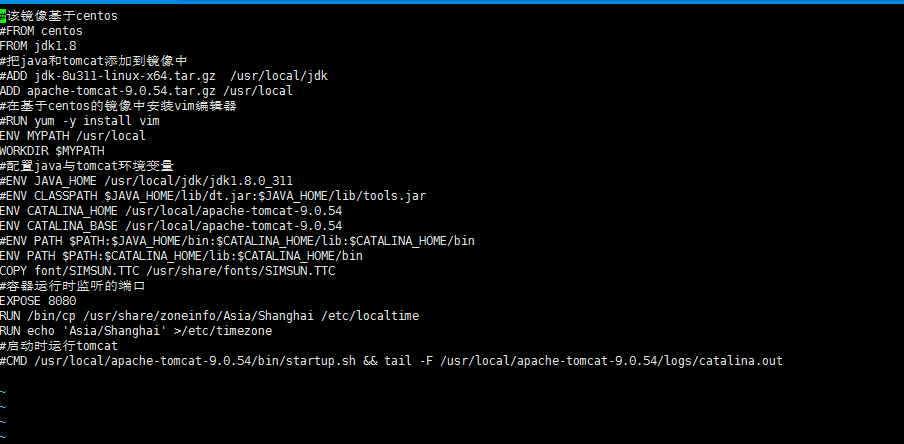
EXPOSE 8080

RUN /bin/cp /usr/share/zoneinfo/Asia/Shanghai /etc/localtime

RUN echo 'Asia/Shanghai' >/etc/timezone

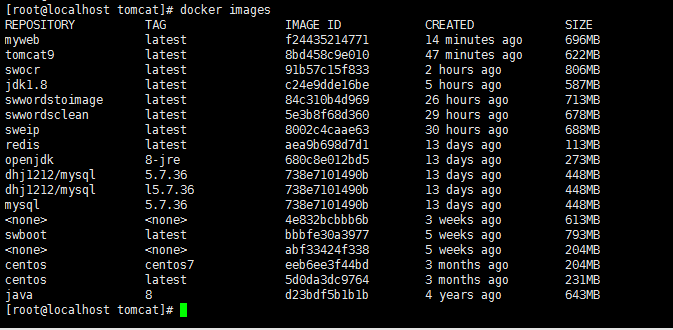
#启动时运行tomcat，暂时不启动，在项目中启动

#CMD /usr/local/apache-tomcat-9.0.54/bin/startup.sh && tail -F /usr/local/apache-tomcat-9.0.54/logs/catalina.out



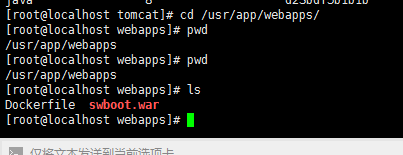
3）构建tomcat镜像

[root@localhost tomcat]# docker build -t tomcat9 .



1. 把项目上传到服务器

例如我们把项目上传到/usr/app/webapps/



**上边都是构建tomcat基础镜像，如果已经有tomcat基础镜像，则从以下步骤开始**

1. 构建项目镜像，以tomcat为基础

Vi Dockerfile

#基础镜像

FROM tomcat9

#维护者

MAINTAINER duhj

#将war拷贝到/usr/local/tomcat/webapps/下

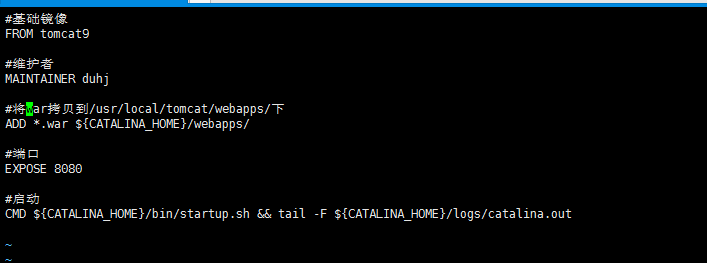
ADD \*.war ${CATALINA\_HOME}/webapps/

#端口

EXPOSE 8080

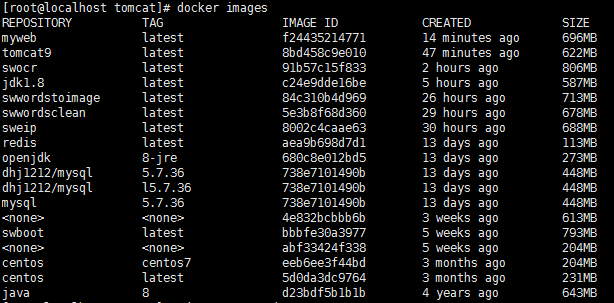
#启动

CMD ${CATALINA\_HOME}/bin/startup.sh && tail -F ${CATALINA\_HOME}/logs/catalina.out



1. 构建镜像

docker build -t myweb .



1. 启动容器

**docker run -d -p 8080:8080 myweb**

# 六、Docker 部署java jar

## 上传jar

上传swboot.jar 到/usr/app/swboot/下

## 创建Dockerfile

在/usr/app/swboot下创建Dockerfile文件

vi Dockerfile

内容如下：

FROM java:8

MAINTAINER duhj

# VOLUME 指定了临时文件目录为/tmp。

# 其效果是在主机 /var/lib/docker 目录下创建了一个临时文件，并链接到容器的/tmp

VOLUME /tmp

# 暴漏端口

#EXPOSE 8088

# 将jar包添加到容器中并命名为swboot.jar

ADD swboot.jar swboot.jar

# 运行命名jar包

RUN bash -c 'touch /swboot.jar'

#ENTRYPOINT ["java","-jar","/swboot.jar"]

ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom","-jar","/swboot.jar"]



文件说明如下：

FROM java:8 基于jdk8环境

EXPOSE 8080 对外暴露的端口是8080

VOLUME /tmp 表示挂载到/tmp目录

ADD cumd-SNAPSHOT.jar /app.jar 把jar包复制到镜像服务里面的根目录，并改成名称为app.jar

RUN bash -c 'touch /app.jar' 执行创建app.jar

ENTRYPOINT ["java","-jar","/app.jar"] 执行启动命令 java -jar

ENTRYPOINT ["java","-jar","/swboot.jar"]

## 构建镜像

进入到前端文件的根目录下打镜像 docker build -t 前端镜像名

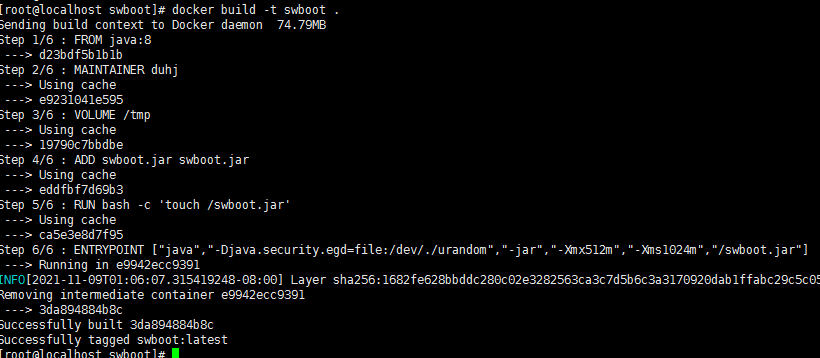
本例子在/usr/app/swboot下执行

docker build -t swboot .

[root@localhost swboot]# docker build -t swboot .

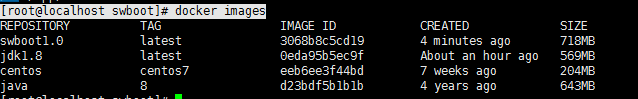
docker build -t swboot .

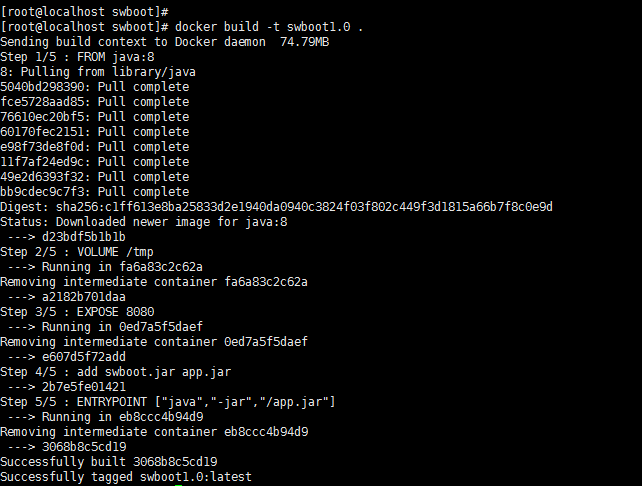
[[root@localhost swocr]# docker build -t sworc .



构建成功后，查看构建结果

[root@localhost swboot]# docker images





## 启动运行

运行 docker run -p 防火墙端口2:前端端口 -d --name=容器名 前端镜像名

[root@localhost swboot]# docker run -d --name=swboot swboot -p 8080:8088

或者

[root@localhost swboot]# docker run -d -p 8080:8088 swboot

[root@localhost swocr]# docker run -d -p 8081:8856 sworc

p 是做端口映射，此时将服务器中的8080端口映射到容器中的8088(项目中端口配置的是8088)端口

-d参数是让容器后台运行



-p 将容器8080映射到宿主机端口8088

## 启动运行带jvm参数

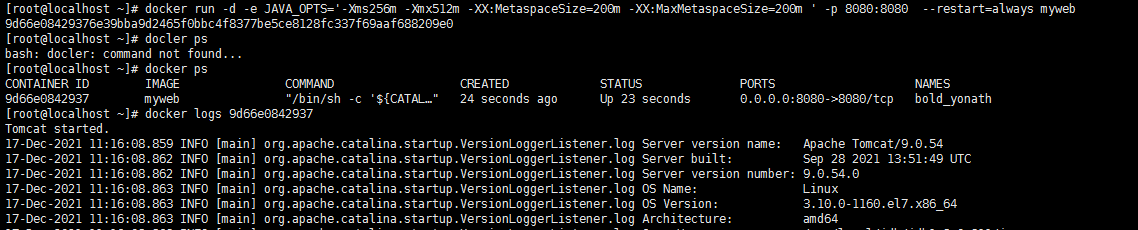
docker run -d -e JAVA\_OPTS='-Xms256m -Xmx512m -XX:MetaspaceSize=200m -XX:MaxMetaspaceSize=200m ' -p 8084:8856 swocr

例如：

[root@localhost swocr]# docker run -d -e JAVA\_OPTS='-Xms1024m -Xmx512m -XX:MetaspaceSize=200m -XX:MaxMetaspaceSize=200m ' -p 8084:8856 swocr

docker run -d -e JAVA\_OPTS='-Xms256m -Xmx512m -XX:MetaspaceSize=200m -XX:MaxMetaspaceSize=200m ' -p 8080:8080 --restart=always myweb

9d66e08429376e39bba9d2465f0bbc4f8377be5ce8128fc337f69aaf688209e0



## 查看打印日志

docker logs [OPTIONS] CONTAINER ID

OPTIONS说明：

-f : 跟踪日志输出

--since :显示某个开始时间的所有日志

-t : 显示时间戳

--tail :仅列出最新N条容器日志

例如：

查看指定时间后的日志，只显示最后100行：

docker logs -f -t --since="2020-10-01" --tail=100 CONTAINER ID

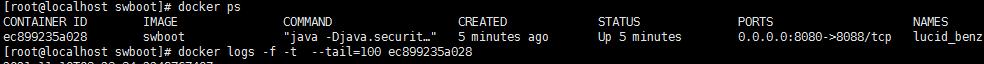
查个指定时间区段的日志

docker logs -t --since="2020-10-01T19:00:00" --until "2020-10-01T19:00:00" CONTAINER ID

操作举例：

docker logs -f -t --tail=行数 容器名ID

[root@localhost swboot]# docker logs -f -t --tail=500 ec899235a028



或者

docker logs -f -t --tail=行数 容器名

[root@localhost swboot]# docker logs -f swboot

## 访问

例如：http://192.168.3.111:8080/cr/access/findMyAllCraccess

## CentOS开通防火墙

CentOS升级到7之后，无法使用iptables控制Linuxs的端口，原因是Centos 7使用firewalld代 替了原来的iptables

**进行8080端口开放配置**：

[root@localhost sysconfig]# firewall-cmd --zone=public --add-port=8080/tcp --permanent

[root@localhost swocr]# firewall-cmd --zone=public --add-port=8081/tcp --permanent

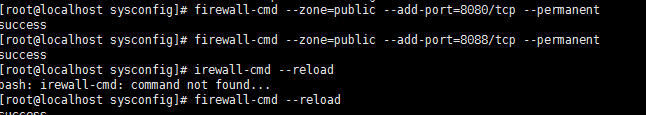
firewall-cmd --zone=public --add-port=6379/tcp --permanent

**命令解析：**

--zone #作用域

--add-port=8080/tcp #添加端口，格式为：端口/通讯协议

--permanent #永久生效，没有此参数设置，重启后会失效。



**重启防火墙，输入命令：**

firewall-cmd --reload

[root@localhost sysconfig]# firewall-cmd --reload



**查看开放的端口：**

firewall-cmd --list-ports

[root@localhost swboot]# firewall-cmd --list-ports



## 删除镜像和容器

**查看当前正在运行的容器：**

docker ps

[root@localhost swboot]# docker ps



查看所有容器

docker ps –a

**删除正在运行的容器：**

docker rm 容器id

先查看**容器**docker ps，获取ID，在根据ID删除

先停止

[root@localhost swboot]# docker stop 3fa30a373cb8

或者

[root@localhost swboot]# ps -ef |grep java

[root@localhost swboot]# kill -9 3737

再删除

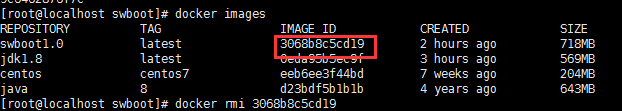
[root@localhost swboot]# docker rm 9c6462876f7c



**查看镜像：**

docker images

[root@localhost swboot]# docker images

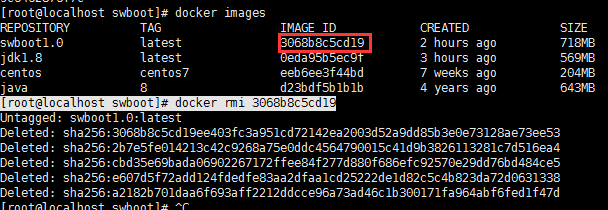


**删除镜像：**

docker rmi 镜像id

删除之前docker images命令查看镜像，找到ID，在根据ID删除

[root@localhost swboot]# docker rmi 3068b8c5cd19



**重新启动镜像：**

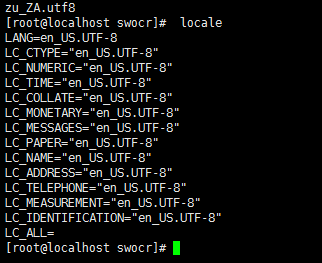
[root@localhost swboot]# docker build -t swboot .

**重新启动容器：**

## 解决中文乱码问题

**查看当前编码：**

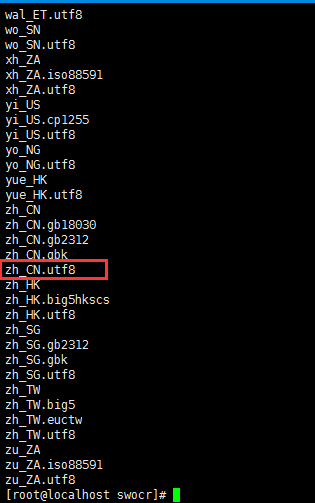
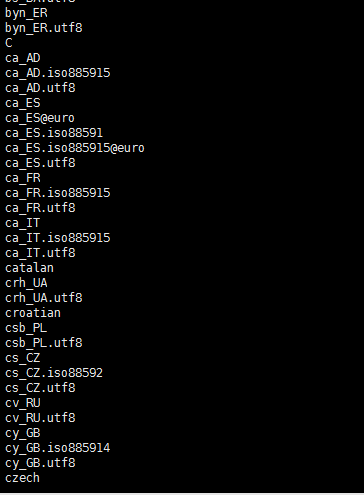
[root@localhost swocr]# locale



**查看已经存在的编码：**locale –a

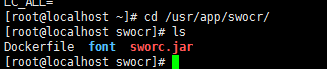
[root@localhost swocr]# locale -a

注意有的机器是C.UTF-8，有的机器是zh\_CN.utf8，都支持中文

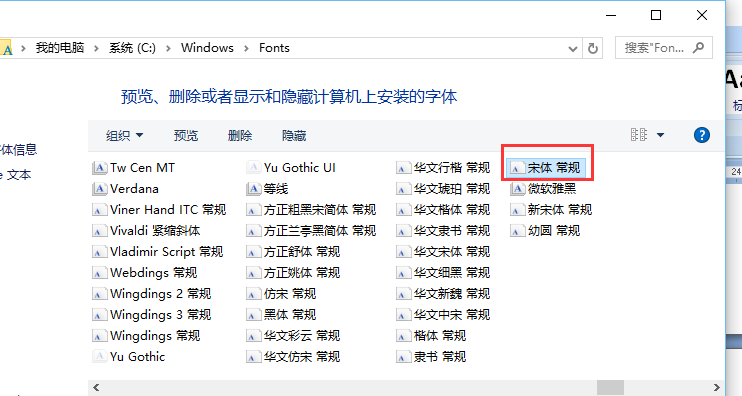
 

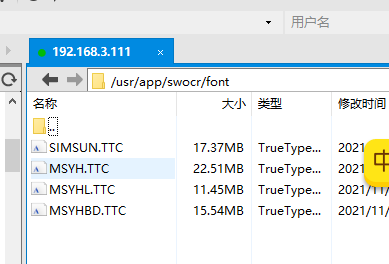
**在项目Dockerfile中设置：**

在dockerfile 同级目录创建font目录，



在font中放入想要的字体，我们需要安装中文宋体，所以放入中文宋体字体，可以在Windows系统目录 C:\Windows\Fonts 复制需要的字体到项目里, 然后在Dockerfile 引入.，因为Windows和Linux字体是通用的。





然后Dockerfile引入，在Dockerfile加入如下代码

COPY font/SIMSUN.TTC /usr/share/fonts/SIMSUN.TTC

如果是JDK1.8，同样需要把字体拷贝到jre中，

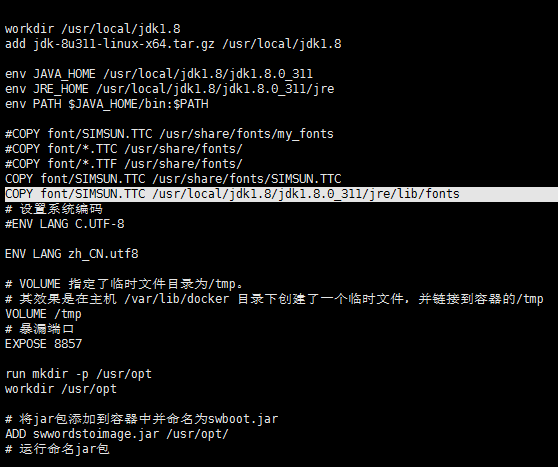
COPY font/SIMSUN.TTC /usr/local/jdk1.8/jdk1.8.0\_311/jre/lib/fonts

# 设置系统编码

#ENV LANG C.UTF-8

ENV LANG zh\_CN.utf8

这个代码作用是把中文字体拷贝到linux 中的/usr/share/fonts/目录，然后设置系统 环境为中文

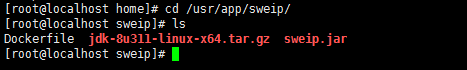


设置完成后，重新启动镜像

## Jdk和jar

**第一种方法：**

1. 上传jdk-8u311-linux-x64.tar.gz 和sweip.jar到项目目录



1. 创建Dockerfile

vi Dockerfile

内容如下：

from centos:centos7

maintainer duhj

run mkdir -p /usr/local/jdk1.8

workdir /usr/local/jdk1.8

add jdk-8u311-linux-x64.tar.gz /usr/local/jdk1.8

env JAVA\_HOME /usr/local/jdk1.8/jdk1.8.0\_311

env JRE\_HOME /usr/local/jdk1.8/jdk1.8.0\_311/jre

env PATH $JAVA\_HOME/bin:$PATH

ENV LANG zh\_CN.utf8

# VOLUME 指定了临时文件目录为/tmp。

# 其效果是在主机 /var/lib/docker 目录下创建了一个临时文件，并链接到容器的/tmp

VOLUME /tmp

# 暴漏端口

EXPOSE 8856

run mkdir -p /usr/opt

# 设置工作目录

workdir /usr/opt

# 将jar包添加到容器中

ADD sweip.jar /usr/opt/

# 设置时区

RUN /bin/cp /usr/share/zoneinfo/Asia/Shanghai /etc/localtime

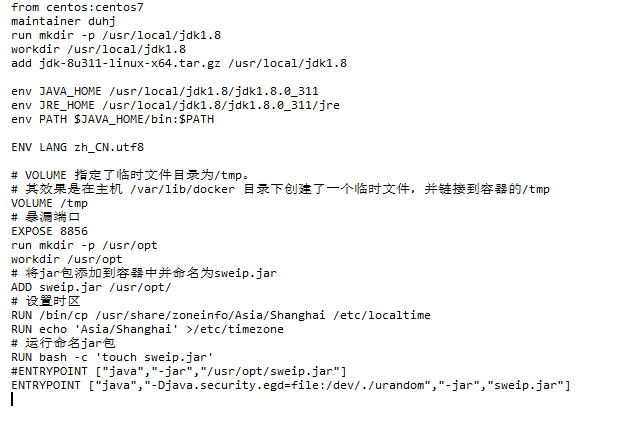
RUN echo 'Asia/Shanghai' >/etc/timezone

# 运行命名jar包

RUN bash -c 'touch sweip.jar'

#ENTRYPOINT ["java","-jar","/usr/opt/sweip.jar"]

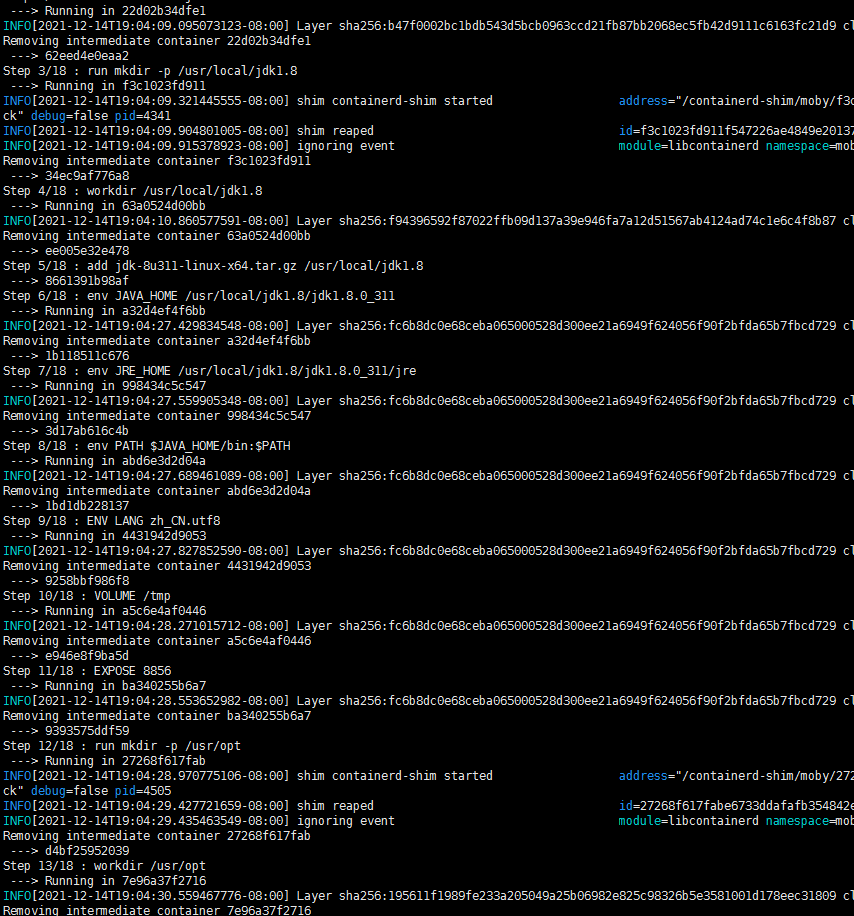
ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom","-jar","sweip.jar"]



1. 构建镜像

docker build -t sweip .

[root@localhost sweip]# docker build -t sweip .



1. 启动容器

[root@localhost sweip]# docker run -d -p 8082:8856 sweip

1. 测试

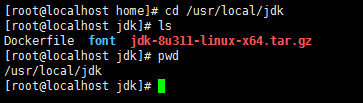
<http://192.168.3.111:8082/>

**第二种方法：（建议）**

* 1. 先构建jdk基础镜像

构建步骤和第三节 docker安装jdk1.8相同

下图为我构建好的jdk1.8镜像



Dockerfile文件如下：

from centos:centos7

maintainer duhj

run mkdir -p /usr/local/jdk

workdir /usr/local/jdk

add jdk-8u311-linux-x64.tar.gz /usr/local/jdk

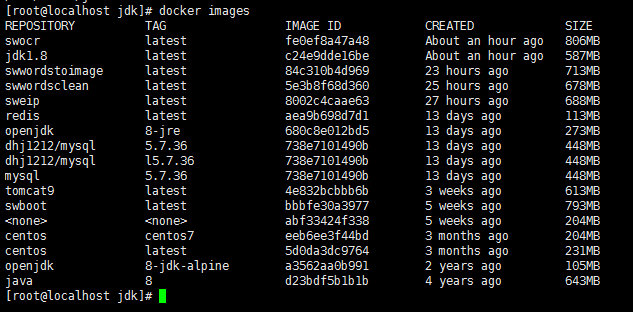
env JAVA\_HOME /usr/local/jdk/jdk1.8.0\_311

env JRE\_HOME /usr/local/jdk/jdk1.8.0\_311/jre

env PATH $JAVA\_HOME/bin:$PATH

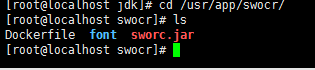
#设置中文乱码

COPY font/SIMSUN.TTC /usr/local/jdk/jdk1.8.0\_311/jre/lib/fonts



* 1. 构建应用jar的镜像，创建Dockerfile

把jar拷贝到目录下，同时需要把font字体也要加入到和jar同样目录，如下：



创建dockerfile

#FROM java:8

#jdk1.8 作为基础镜像

FROM jdk1.8

#FROM jdk8:base

#openjdk:8作为基础镜像

#FROM openjdk:8-jre

#jdk-alpine问题比较多,不建议

#FROM openjdk:8-jdk-alpine

MAINTAINER duhj

#COPY font/SIMSUN.TTC /usr/share/fonts/my\_fonts

COPY font/SIMSUN.TTC /usr/share/fonts/SIMSUN.TTC

# 设置系统编码

#ENV LANG C.UTF-8

ENV LANG zh\_CN.utf8

# VOLUME 指定了临时文件目录为/tmp。

# 其效果是在主机 /var/lib/docker 目录下创建了一个临时文件，并链接到容器的/tmp

VOLUME /tmp

# 暴漏端口

EXPOSE 8856

run mkdir -p /usr/opt

#设置当前工作目录

workdir /usr/opt

# 将jar包添加到容器中并命名为swboot.jar

ADD sworc.jar /usr/opt/

RUN /bin/cp /usr/share/zoneinfo/Asia/Shanghai /etc/localtime

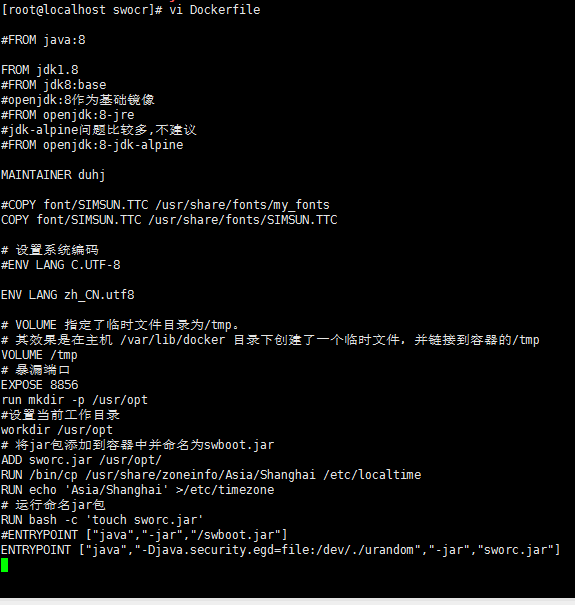
RUN echo 'Asia/Shanghai' >/etc/timezone

# 运行命名jar包

RUN bash -c 'touch sworc.jar'

#ENTRYPOINT ["java","-jar","/swboot.jar"]

ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom","-jar","sworc.jar"]



* 1. 构建镜像

docker build -t swocr .

* 1. 启动容器

[root@localhost swocr]# docker run -d -p 8084:8856 swocr

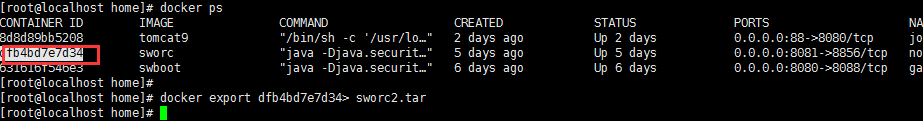
# 七、Docker 导入导出

## 使用export 和 import

**导出：**

首先查找要导出的容器

[root@localhost home]# docker ps



导出到文件夹

[root@localhost home]# docker export dfb4bd7e7d34> sworc2.tar

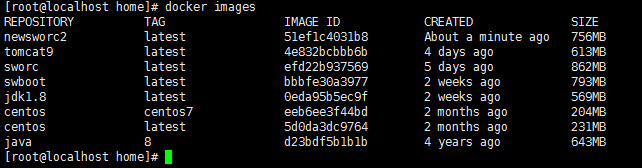


**导入：**

[root@localhost home]# docker import - newsworc2 < sworc2.tar



导出成功后，会有一个镜像如下

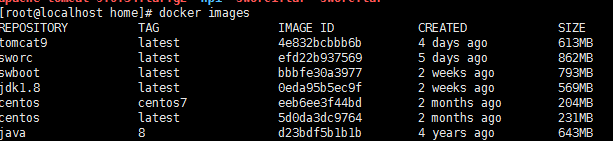


## 使用 save 和 load（建议）

**导出：**

首先查找要导出的镜像

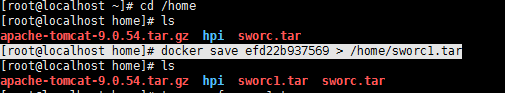
[root@localhost home]# docker images



保存

[root@localhost home]# docker save efd22b937569 > /home/sworc1.tar

有点慢，稍微等待一下，没有任何warn信息就表示保存OK。efd22b937569是镜像ID



****

**导入：**

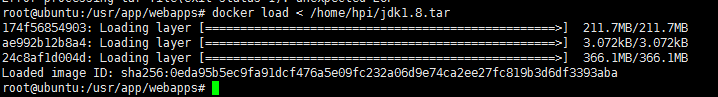
现在就可以在任何装 docker 的地方加载 刚保存的镜像了

docker load < /home/sworc1.tar

[root@localhost home]# docker load < /home/sworc1.tar

例如：

root@ubuntu:/usr/app/webapps# docker load < /home/hpi/jdk1.8.tar



**以上两种方案都可以完成，特别注意：两种方法不可混用。**

**如果使用 import 导入 save 产生的文件，虽然导入不提示错误，但是启动容器时会提示失败，会出现类似"docker: Error response from daemon: Container command not found or does not exist"的错误。**

**两种方案的差别在于：**

* export 导出的镜像文件体积小于 save 保存的镜像
* docker import 可以为镜像指定新名称 docker load 不能对载入的镜像重命名
* docker export 不支持同时将多个镜像打包到一个文件中，docker save 支持同时将多个镜像打包到一个文件中
* export 导出（import 导入）是根据容器拿到的镜像，再导入时会丢失镜像所有的历史记录和元数据信息（即仅保存容器当时的快照状态），所以无法进行回滚操作，而 save 保存（load 加载）的镜像，没有丢失镜像的历史，可以回滚到之前的层（layer）。

## 修改导入的镜像标签

加载成功后这两个地方会是none，需要我们修改标签

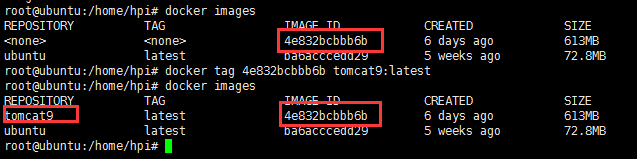


docker tag 999c20aee5da artipub:latest

docker tag [镜像id] [新镜像名称]:[新镜像标签]

例如：

root@ubuntu:/home/hpi# docker tag 4e832bcbbb6b tomcat9:latest



## 导入tomcat

1. 执行导入命令

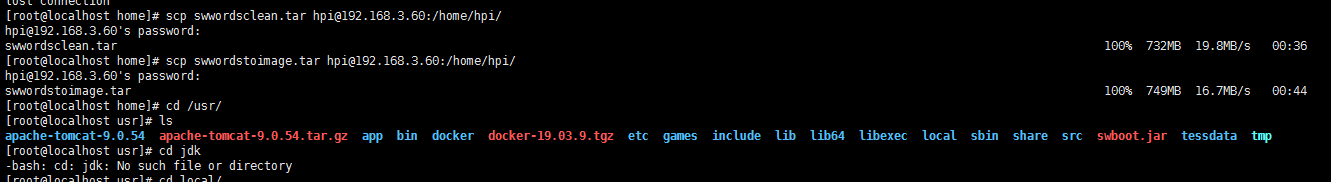
docker load < /home/hpi/tomcat9.tar

1. 挂载工程 参看tomcat部署应用挂载部分

## 服务器对拷连接

把 当前swwordsclean.tar文件拷贝到192.168.3.60服务器/home/hpi目录，60服务器用户是hpi

[root@localhost home]# scp swwordsclean.tar [hpi@192.168.3.60:/home/hpi/](mailto:hpi@192.168.3.60:/home/hpi/)



# 八、Ubuntu 安装Docker

## 安装openssh

一般Ubuntu都会默认安装openssh-client,但是没有安装openssh-server。

**安装ssh服务器**

hpi@ubuntu:/usr$ sudo apt install openssh-server

按照步骤安装即可

**安装ssh客户端**

hpi@ubuntu:/usr$ sudo apt install openssh-client

**启动**

hpi@ubuntu:/usr$ sudo systemctl start ssh.service

**查看服务**

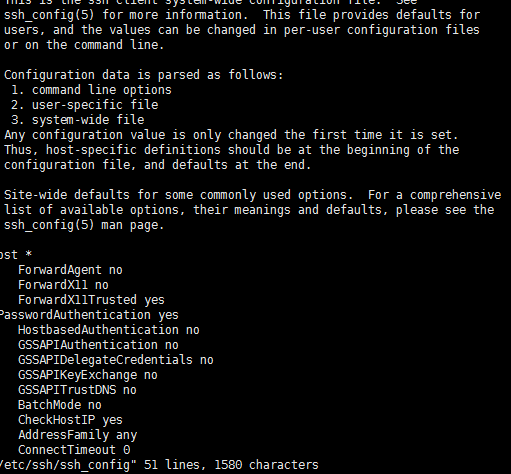
hpi@ubuntu:/usr$ ps -e |grep ssh

然后SSH就可以正常访问了

**配置root登陆权限**

sudo vi /etc/ssh/ssh\_config

去掉PasswordAuthentication yes 前面的#号，然后保存

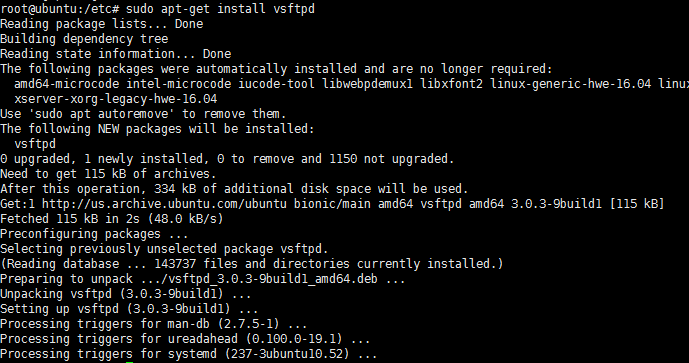


配置ssh服务器，把PermitRootLogin prohibit-password改成PermitRootLogin yes，保存退出。

## 安装vsftp

1)安装

sudo apt-get install vsftpd



可能出现网络连接失败的问题，需要修改DNS网关

最好用root账号

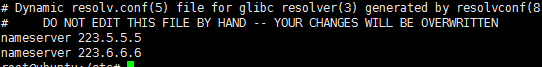
root@ubuntu:/etc# vi resolv.conf

在resolv.conf里面添加阿里DNS

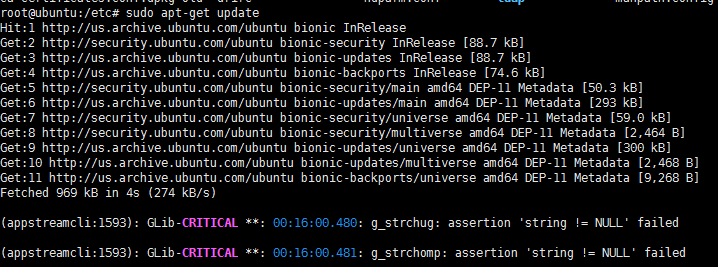
nameserver 223.5.5.5

nameserver 223.6.6.6

保存后，联网更新一下



sudo apt-get update



再去下载依赖包，发现问题解决。

2) 启动、停止、重启vsftp

sudo service vsftpd start

root@ubuntu:/etc# sudo service vsftpd start

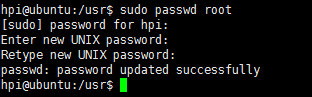
ystemctl status vsftpd.service

root@ubuntu:/etc# systemctl status vsftpd.service

## 设置ROOT账号

打开Ubuntu,输入命令:su root,回车提示输入密码,怎么输入都不对，给root用户设置密码: 命令:sudo passwd root 输入密码,并确认密码。

hpi@ubuntu:/usr$ sudo passwd root



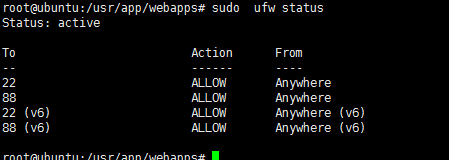
可以正常切换到ROOT账号了



## Ubuntu开放防火墙和端口

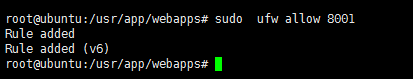
1）查看本地端口开启情况

root@ubuntu:/usr/app/webapps# sudo ufw status



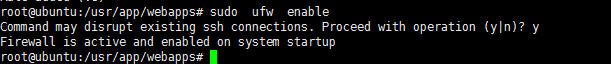
2）打开8001端口(ubuntu下执行)

sudo ufw allow 8001



1. 防火墙开启(ubuntu下执行)

sudo ufw enable



1. 防火墙重启(ubuntu下执行)

sudo ufw reload

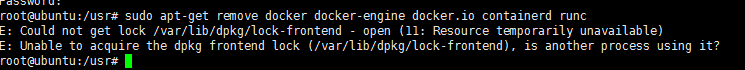


## 遇到的错误

1）如下异常

E: Could not get lock /var/lib/dpkg/lock-frontend - open (11: Resource temporarily unavailable)

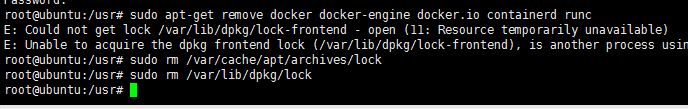
E: Unable to acquire the dpkg frontend lock (/var/lib/dpkg/lock-frontend), is another process using it?



强制解锁

sudo rm /var/cache/apt/archives/lock

sudo rm /var/lib/dpkg/lock

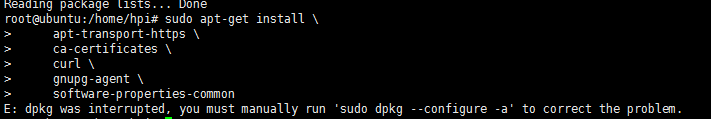


如果以上没有解决，只有重启服务了

sudo systemctl --force --force reboot

1. 遇到如下问题

E: dpkg was interrupted, you must manually run 'sudo dpkg --configure -a' to correct the problem.



解决代码如下：

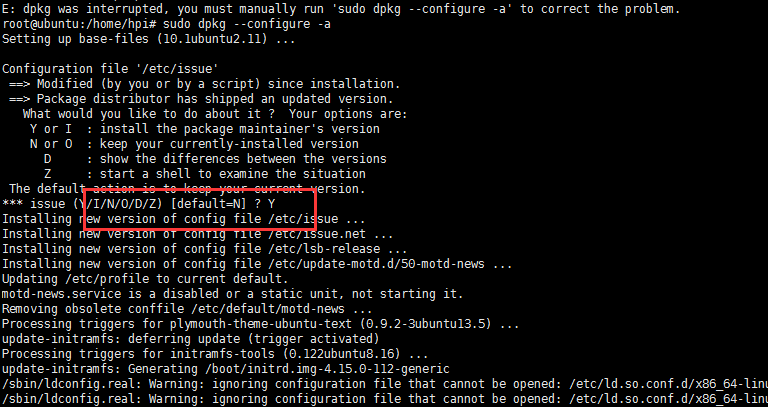
root@ubuntu:/home/hpi# sudo dpkg --configure –a

执行过程中，选择Y

执行完毕，问题解决。

如果又在终端出现了这样的一个问题：、

libc6-dev:依赖:libc6(=2.9-4ubuntu6.1) 但是2.9-Ubuntu6已经安装了  E：不能满足依赖关系。不妨试一下-f选项     于是我在终端输入sudo apt-get -f install



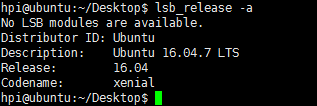
## 删除原docker

1. 查看ubuntu系统是32位的还是64位的：getconf LONG\_BIT

hpi@ubuntu:~/Desktop$ getconf LONG\_BIT



1. 查看系统信息



1. 查看操作系统架构 uname -a

hpi@ubuntu:~/Desktop$ uname -a



查看内核 uname -r

hpi@ubuntu:~/Desktop$ uname –r



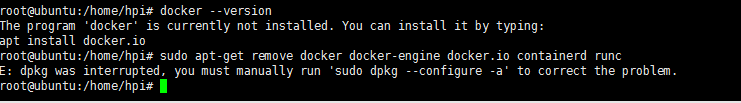
1. 卸载旧版本

检查版本，

root@ubuntu:/home/hpi# docker –version

如果有，卸载当前版本

sudo apt-get remove docker docker-engine docker.io containerd runc



## 使用 Docker 仓库进行安装

在新主机上首次安装 Docker Engine-Community 之前，需要设置 Docker 仓库。之后，您可以从仓库安装和更新 Docker 。

参考：

<https://www.cnblogs.com/walker-lin/p/11214127.html>

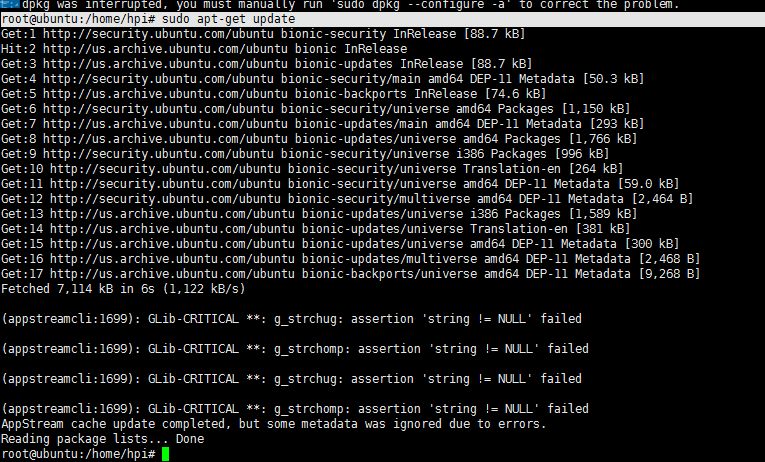
https://www.runoob.com/docker/ubuntu-docker-install.html

1. 设置仓库

更新 apt 包索引。

$ sudo apt-get update

root@ubuntu:/home/hpi# sudo apt-get update



1. 允许apt通过https使用repository安装软件包

sudo apt-get install \

apt-transport-https \

ca-certificates \

curl \

gnupg-agent \

software-properties-common

例如：

root@ubuntu:/home/hpi# sudo apt-get install \

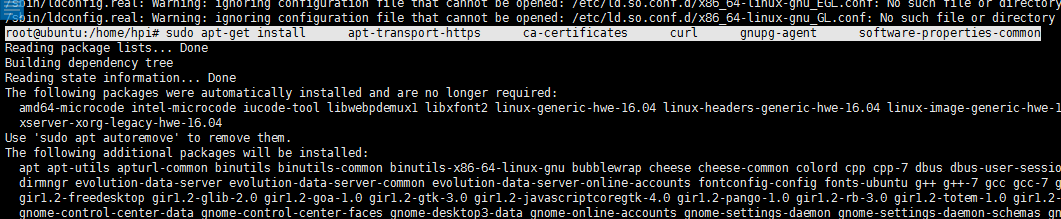
apt-transport-https \

ca-certificates \

curl \

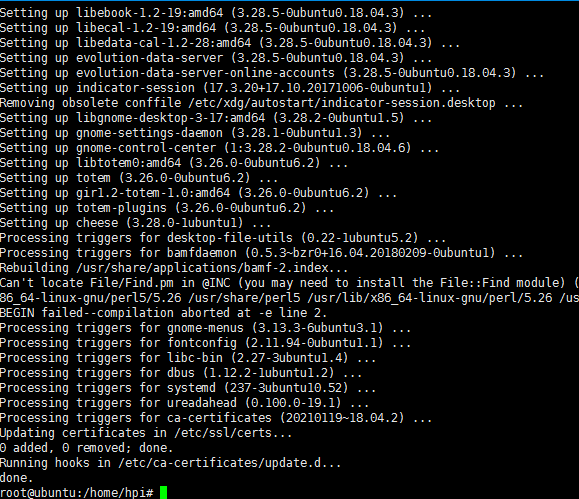
gnupg-agent \

software-properties-common



中间选择Y

一直到安装结束，大概需要10分钟左右。

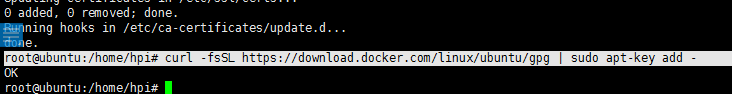


1. 添加Docker官方GPG key

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add –

root@ubuntu:/home/hpi# curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add –

回车后下一行显示OK,表示添加成功。



如出现：gpg: can't connect to the agent: IPC connect call failed 这种问题，可以尝试：

sudo apt remove gpg

udo apt install gnupg1

然后再执行。

Docker官方GPG 有以下几个服务器

curl -fsSL https://mirrors.ustc.edu.cn/docker-ce/linux/ubuntu/gpg | sudo apt-key add –

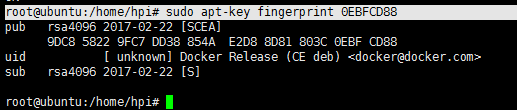
阿里的

sudo curl -fsSL https://mirrors.aliyun.com/docker-ce/linux/ubuntu/gpg | apt-key add -

我们用的是官方的

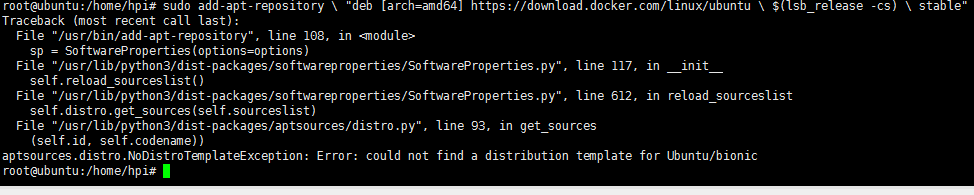
1. 验证key的指纹

root@ubuntu:/home/hpi# sudo apt-key fingerprint 0EBFCD88



1. 添加稳定版repository

root@ubuntu:/home/hpi# sudo add-apt-repository \ "deb [arch=amd64] https://download.docker.com/linux/ubuntu \ $(lsb\_release -cs) \ stable"



国内阿里云版：

sudo add-apt-repository \

"deb [arch=amd64] https://mirrors.aliyun.com/docker-ce/linux/ubuntu \

$(lsb\_release -cs) \

stable"

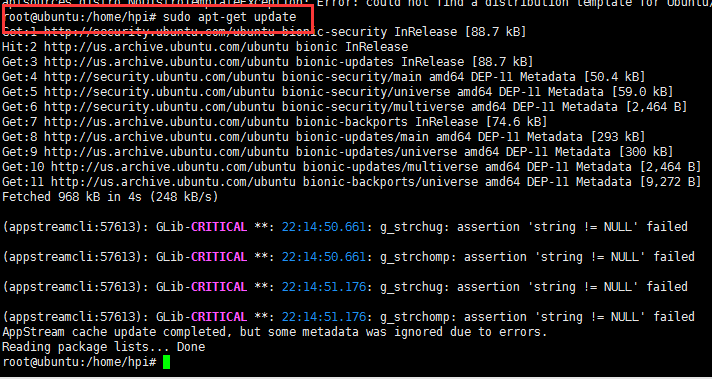
另一个版本

$ **sudo** add-apt-repository \  
   "deb [arch=amd64] https://mirrors.ustc.edu.cn/docker-ce/linux/ubuntu/ **\**  $(lsb\_release -cs) **\**  stable"

1. 再次更新系统软件

sudo apt-get update

root@ubuntu:/home/hpi# sudo apt-get update



1. 安装最新版本的docker ce和containerd

sudo apt-get install docker-ce docker-ce-cli containerd.io

1. 安装指定版本的

查看可获取的版本 apt-cache madison docker-ce

sudo apt-get install docker-ce=<VERSION\_STRING> docker-ce-cli=<VERSION\_STRING> containerd.io

1. 验证：docker –version
2. 重启服务

sudo service docker restart

1. 设置开机自启动并启动 Docker-ce（安装成功后默认已设置并启动，可忽略）

sudo systemctl enable docker

sudo systemctl start docker

1. 升级版本

sudo apt-get update

## 使用 离线文件安装

和centos离线安装一致，参考CentOS7 离线安装docker-19.03.9

## 获取ubuntu镜像

拉去最新的镜像

root@ubuntu:/usr# docker pull ubuntu



# 九、Docker内部文件访问

## 使用docker exec进入Docker容器

**适用于docker1.3.X以上版本**

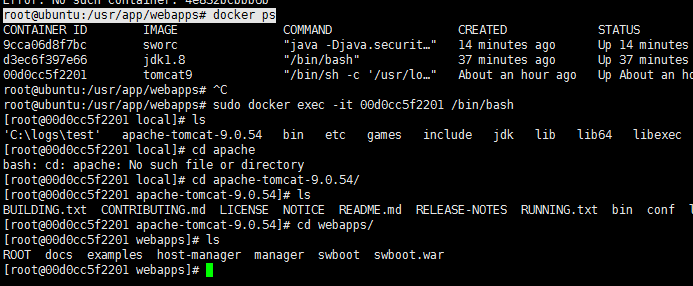
1. 查看要进入的容器信息

root@ubuntu:/usr/app/webapps# docker ps

1. 进入容器

sudo docker exec -it 00d0cc5f2201 /bin/bash

进入后就可以看到该容器的所有目录了



3）退出

[root@00d0cc5f2201 webapps]# exit



## 使用docker attach进入Docker容器

sudo docker attach 00d0cc5f2201

使用该命令有一个问题。当多个窗口同时使用该命令进入该容器时，所有的窗口都会同步显示。如果有一个窗口阻塞了，那么其他窗口也无法再进行操作。

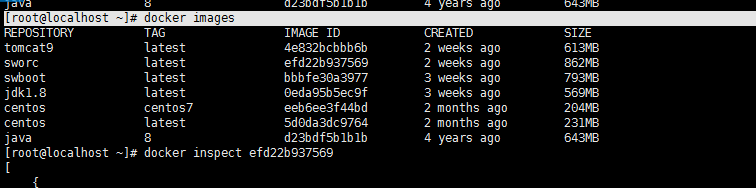
因为这个原因，所以docker attach命令不太适合于生产环境，平时自己开发应用时可以使用该命令。

## 查看镜像底层信息

**先查找镜像**

docker images

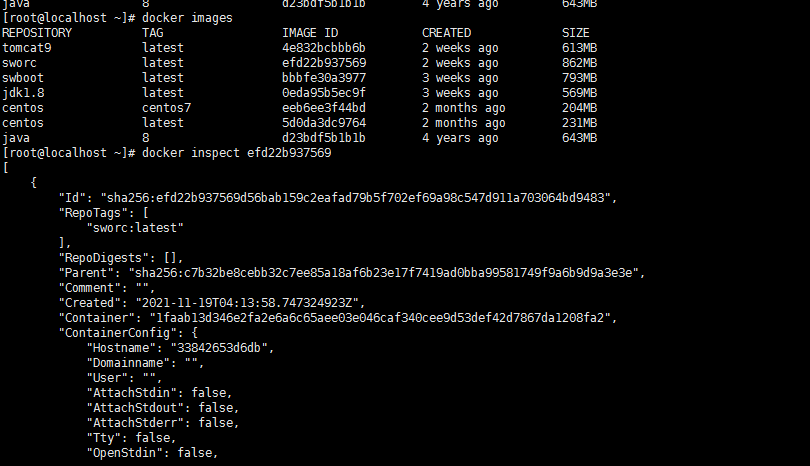
[root@localhost ~]# docker images



**查看镜像底层信息**

docker inspect [image ID/image name]

[root@localhost ~]# docker inspect efd22b937569



# 十、Docker安装openssh-server

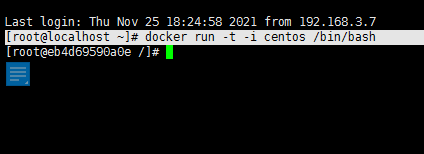
首先，需要从docker官网获得centos或Ubuntu镜像

1. 当本地已有Ubuntu或者centos镜像后（大概200M左右大小），使用如下命令

[root@localhost ~]# docker run -t -i centos /bin/bash

如果是ubuntu，则如下命令

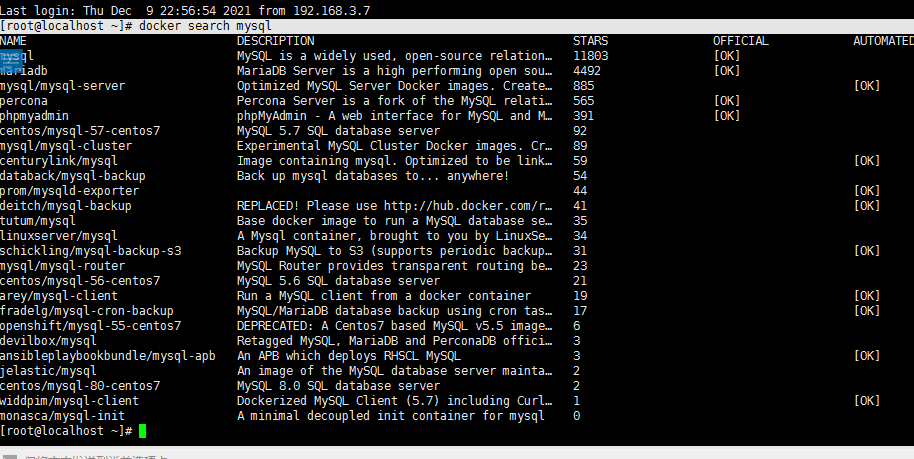
docker run -t -i ubuntu /bin/bash



# 十一、Docker安装MYSQl 5.7.36

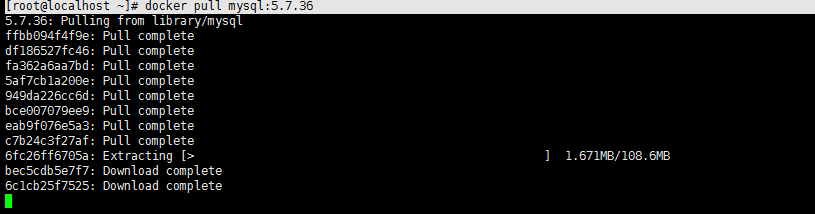
## 查看搜索mysql镜像

[root@localhost ~]# docker search mysql



## 拉取镜像

[root@localhost ~]# docker pull mysql:5.7.36



## 运行镜像

docker run -d -p 3306:3306 -v $PWD/data:/var/lib/mysql -e MYSQL\_ROOT\_PASSWORD={password} --name mysql mysql:5.7.36

**命令解析：**

•$PWD：当前路径，可根据实际情况更换

•MYSQL\_ROOT\_PASSWORD：数据库默认root密码

\*如果不设置，默认的密码为空

•容器内配置文件目录：/etc/mysql/conf.d/

--name：容器名

--p：映射宿主主机端口

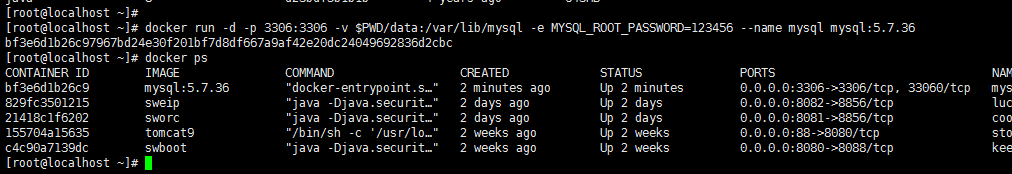
-v：挂载宿主目录到容器目录

-e：设置环境变量，此处指定root密码

-d：后台运行容器

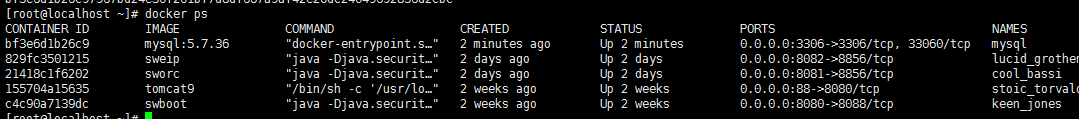
例如：

[root@localhost ~]# docker run -d -p 3306:3306 -v $PWD/data:/var/lib/mysql -e MYSQL\_ROOT\_PASSWORD=123456 --name mysql mysql:5.7.36



## 查看MYSQL容器

$ docker ps

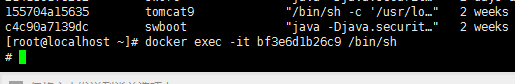


容器ID：bf3e6d1b26c9

## 进入MYSQL容器内部

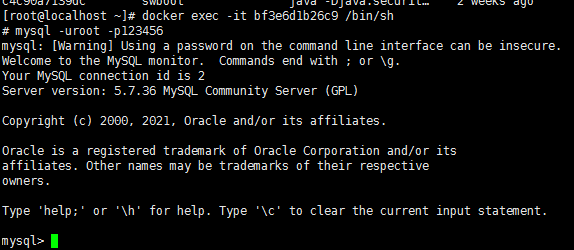
$ docker exec -it bf3e6d1b26c9 /bin/sh

[root@localhost ~]# docker exec -it bf3e6d1b26c9 /bin/sh



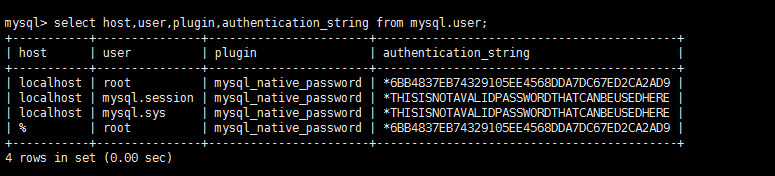
## 连接MYSQL

# mysql -uroot -p123456



查询测试

select host,user,plugin,authentication\_string from mysql.user;



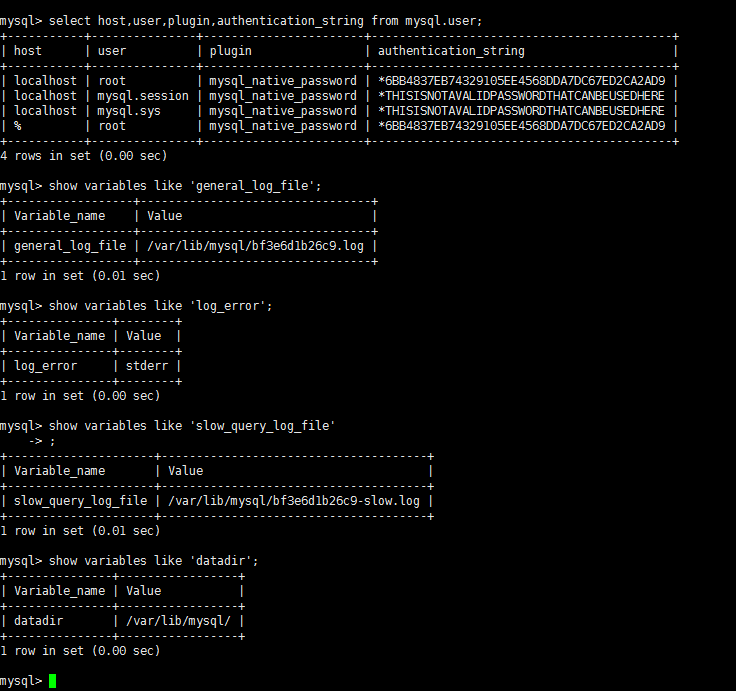
查看mysql配置中数据文件与日志文件路径：

show variables like 'general\_log\_file'; # 日志文件路径

show variables like 'log\_error'; # 错误日志文件路径

show variables like 'slow\_query\_log\_file'; # 慢查询日志文件路径

show variables like 'datadir'; # 查看数据库数据文件路径



查看结果发现日志、慢查询日志和数据文件都位于容器/var/lib/mysql，错误日志权限未配置

既然如此，我们就挂载个空目录成容器的/var/lib/mysql并修改配置文件将错误日志文件也一起放到此路径

## 修改MYSQL 配置文件，防止权限不足的问题

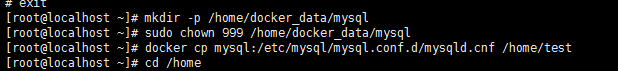
<https://blog.csdn.net/weixin_43507959/article/details/103646691>

1. 宿主机下新建挂载目录和日志文件

mkdir -p /home/docker\_data/mysql

1. 宿主机修改挂载文件/home/docker\_data/mysql的所有者权限为999

sudo chown 999 /home/docker\_data/mysql

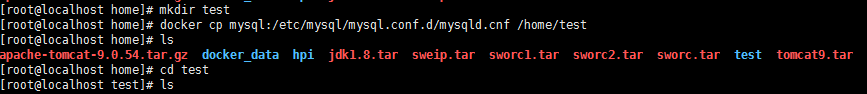


1. 宿主机将容器mysql的配置文件/etc/mysql/mysql.conf.d/mysqld.cnf复制到宿主机目录/home/test

mkdir /home/test

拷贝mysqlid.cnf

docker cp mysql:/etc/mysql/mysql.conf.d/mysqld.cnf /home/test



注意：mysql为容器名称

1. 修改配置文件错误日志路径为/var/lib/mysql （可以不做，我没有做，如果没有做，第7步跳过）

vim /home/test/mysqld.cnf # 编辑配置文件mysqld.cnf

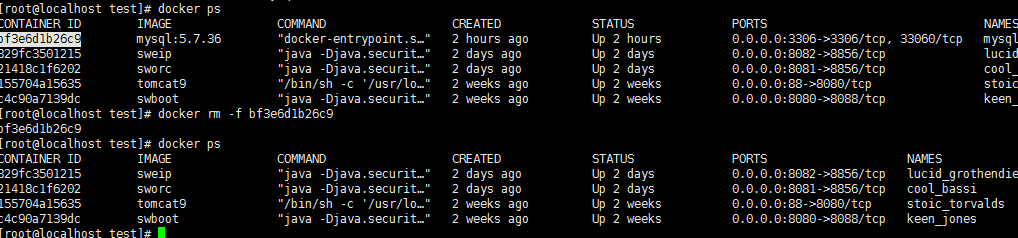
#log-error = /var/log/mysql/error.log # 修改前

log-error = /var/lib/mysql/error.log #修改后

没有的话先不修改

1. 强制删除mysql容器

docker rm -f bf3e6d1b26c9



1. 重新运行容器，映射端口，并挂载配置文件、数据文件，设置账号密码，并设置编码方式为UTF-8

docker run --name mysql5.7.36 -d -e MYSQL\_ROOT\_PASSWORD=123456 -v /home/docker\_data/mysql:/var/lib/mysql -p 3307:3306 mysql:5.7.36 --character-set-server=utf8mb4 --collation-server=utf8mb4\_unicode\_ci

说明：

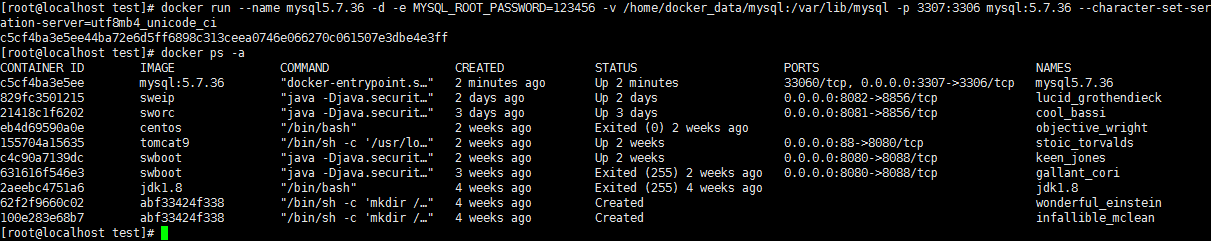
-v: 将宿主机目录挂载到容器(-v /home/docker\_mysql/datadir:/var/lib/mysql 将宿主机的/home/docker\_mysql/datadir挂载到容器的/var/lib/mysql目录，-v 参数可重复多次, 每次挂载一个目录到容器)

例如：

[root@localhost test]# docker run --name mysql5.7.36 -d -e MYSQL\_ROOT\_PASSWORD=123456 -v /home/docker\_data/mysql:/var/lib/mysql -p 3307:3306 mysql:5.7.36 --character-set-server=utf8mb4 --collation-server=utf8mb4\_unicode\_ci



1. 将宿主机的配置文件/home/test/mysqld.cnf复制到容器mysql\_5.7.28的/etc/mysql/mysql.conf.d下。做了第4步做这一步，如果没做就不要做
2. 查看

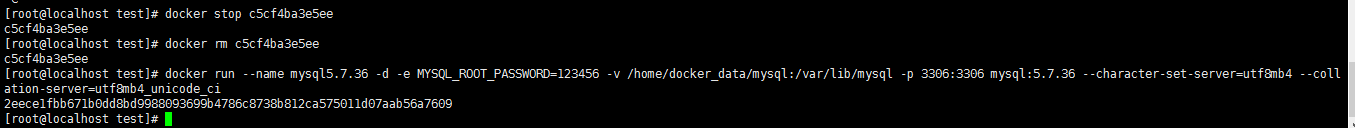


## 启动容器

必须建好/home/docker\_data/mysql并授权

docker run --name mysql5.7.36 -d -e MYSQL\_ROOT\_PASSWORD=123456 -v /home/docker\_data/mysql:/var/lib/mysql -p 3306:3306 mysql:5.7.36 --character-set-server=utf8mb4 --collation-server=utf8mb4\_unicode\_ci

[root@localhost test]# docker run --name mysql5.7.36 -d -e MYSQL\_ROOT\_PASSWORD=123456 -v /home/docker\_data/mysql:/var/lib/mysql -p 3306:3306 mysql:5.7.36 --character-set-server=utf8mb4 --collation-server=utf8mb4\_unicode\_ci



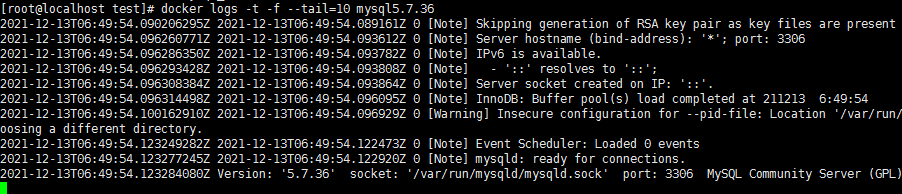
设置为开机自动重启

docker container update --restart=always mysql5.7.36



## 查看运行日志

[root@localhost test]# docker logs -t -f --tail=10 mysql5.7.36



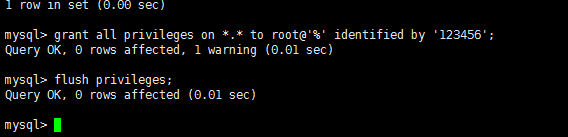
## 赋予远程连接权限

在mysql终端输入：

grant all privileges on \*.\* to root@'%' identified by '123456';

刷新生效：

flush privileges;

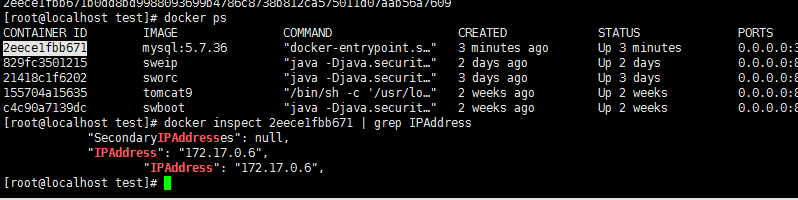


## 开通3306端口

根据不同操作系统操作

## 查看容器IP

docker inspect 容器的id或者容器名 | grep IPAddress

docker inspect 2eece1fbb671 | grep IPAddress  


# 十二、Docker登录仓库

## 登录

首先要在<https://hub.docker.com/> 注册，注册后下可以登录

[root@localhost test]# docker login

Username: dhj1212

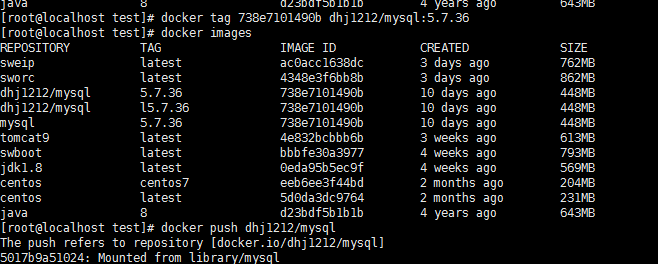
Password：7\*\*\*\*\*\*2@Hpi

上传镜像到仓库

## 指定镜像版本（tag）

docker tag [ImageId] registry.cn-hangzhou.aliyuncs.com/zhangzhiwen/mysql:[镜像版本号]

[root@localhost test]# docker tag 738e7101490b dhj1212/mysql:5.7.36

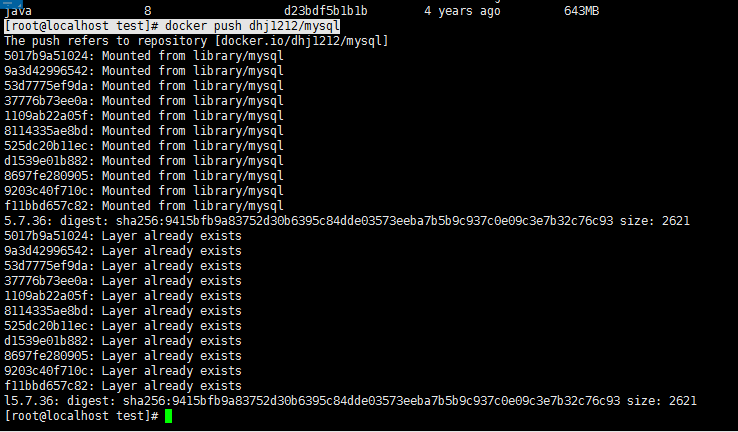


如果不指定版本，上传的时候回报错误

denied: requested access to the resource is denied

## 上传到镜像库

[root@localhost test]# docker push dhj1212/mysql



## 拉入私人镜像

docker pull registry.cn-hangzhou.aliyuncs.com/zhangzhiwen/sell-eureka:[镜像版本号]

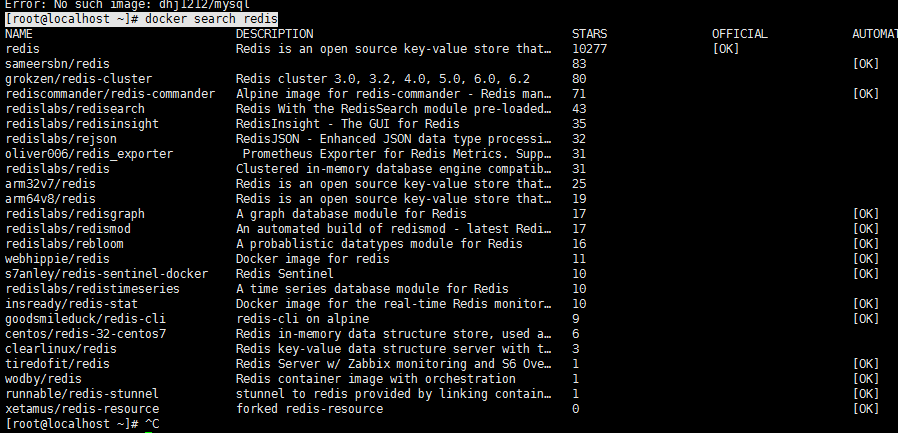
## 退出

[root@localhost test]# docker logout

# 十三、Docker安装redis

## 查看搜索redis镜像

[root@localhost ~]# docker search redis



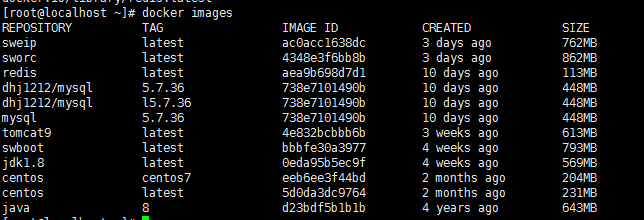
## 拉取redis镜像

docker pull redis

[root@localhost ~]# docker pull redis

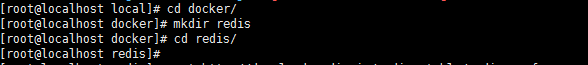


docker images



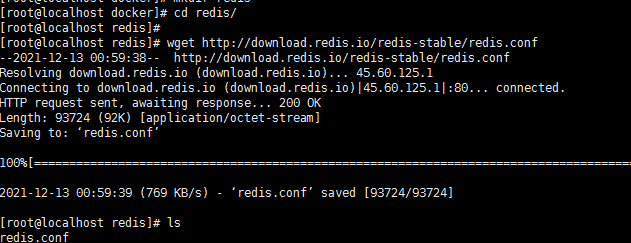
## 从官网获取 redis.conf 配置文件

mkdir /usr/local/docker/redis



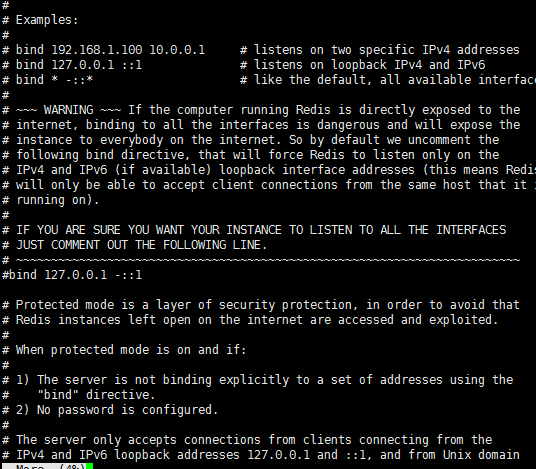
cd /usr/local/docker/redis //进入目录

wget http://download.redis.io/redis-stable/redis.conf //下载redis配置文件



修改配置文件

vim redis.conf



* bind 127.0.0.1 #注释掉这部分，这是限制redis只能本地访问
* protected-mode no #默认yes，开启保护模式，限制为本地访问
* daemonize no#默认no，改为yes意为以守护进程方式启动，可后台运行，除非kill进程（可选），改为yes会使配置文件方式启动redis失败
* dir  ./ #输入本地redis数据库存放文件夹（可选）
* appendonly yes #redis持久化（可选）

## 启动redis容器

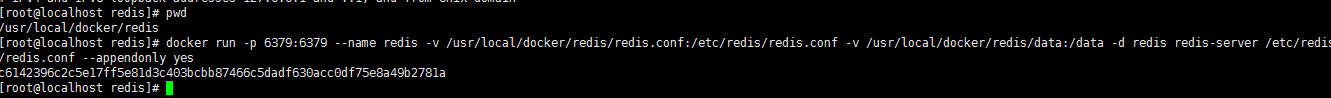
docker run -p 6379:6379 --name redis -v /usr/local/docker/redis/redis.conf:/etc/redis/redis.conf -v /usr/local/docker/redis/data:/data -d redis redis-server /etc/redis/redis.conf --appendonly yes

命令说明：

* p 6379:6379 端口映射：前表示主机部分，：后表示容器部分。
* --name myredis  指定该容器名称，查看和进行操作都比较方便。
* -v 挂载目录，规则与端口映射相同。
* -d redis 表示后台启动redis
* redis-server /etc/redis/redis.conf  以配置文件启动redis，加载容器内的conf文件，最终找到的是挂载的目录/usr/local/docker/redis.conf
* appendonly yes 开启redis 持久化

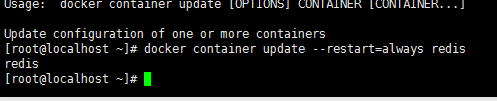
[root@localhost redis]# docker run -p 6379:6379 --name redis -v /usr/local/docker/redis/redis.conf:/etc/redis/redis.conf -v /usr/local/docker/redis/data:/data -d redis redis-server /etc/redis/redis.conf --appendonly yes

|  |
| --- |
| docker run -p 6379:6379 --name redis -v /usr/local/docker/redis/redis.conf:/etc/redis/redis.conf -v /usr/local/docker/redis/data:/data -d redis redis-server /etc/redis/redis.conf --appendonly yes |



设置开机自动重启

docker container update --restart=always redis



## 查看redis容器

[root@localhost redis]# docker ps

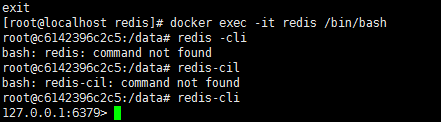
## 进入redis容器

docker exec -it redis /bin/bash

登录

redis-cli

root@c6142396c2c5:/data# redis-cli

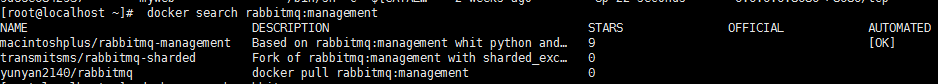


# 十四、Docker安装RabbitMQ

## 查看查询镜像

我们安装带有management的，即有控制台界面的。

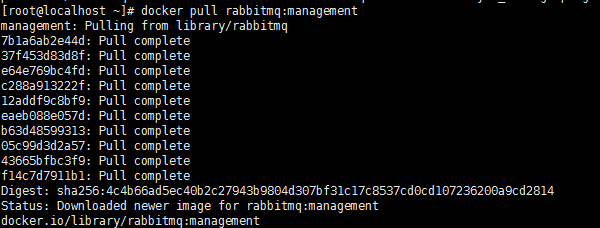
docker search rabbitmq:management



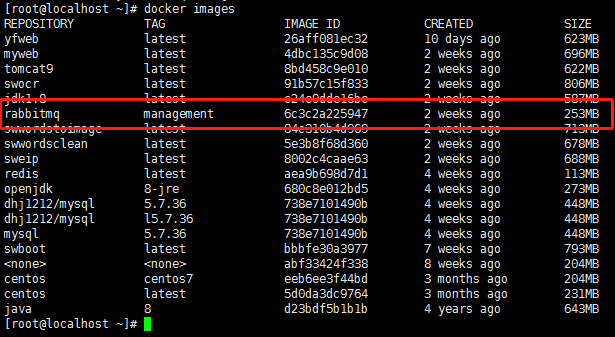
## 拉取镜像

docker pull rabbitmq:management

[root@localhost ~]# docker pull rabbitmq:management



安装成功后，查看



## 运行启动镜像

方式1：默认guest 用户，密码也是 guest

[root@localhost ~]# docker run -d --hostname my-rabbit --name rabbit -p 15672:15672 -p 5672:5672 rabbitmq:management

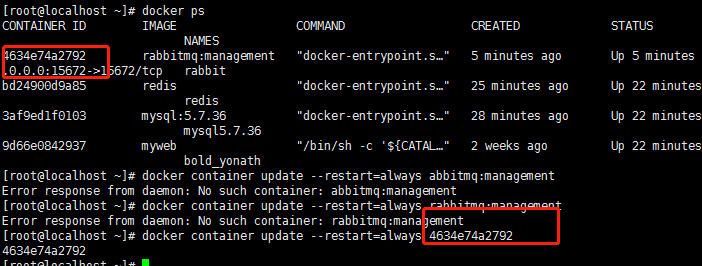


方式2：设置用户名和密码

docker run -d --hostname my-rabbit --name rabbit -e RABBITMQ\_DEFAULT\_USER=admin -e RABBITMQ\_DEFAULT\_PASS=123456 -p 15672:15672 -p 5672:5672 rabbitmq:management

**设置开机自启动：**

[root@localhost ~]# docker container update --restart=always 4634e74a2792

****

参数说明：

-d #后台运行

-- homename my-rabbit #主机名

RABBITMQ\_DEFAULT\_USER=admin #可视化界面登录用户名

RABBITMQ\_DEFAULT\_PASS=admin #可视化界面登录密码

-p 15672:15672 #端口映射

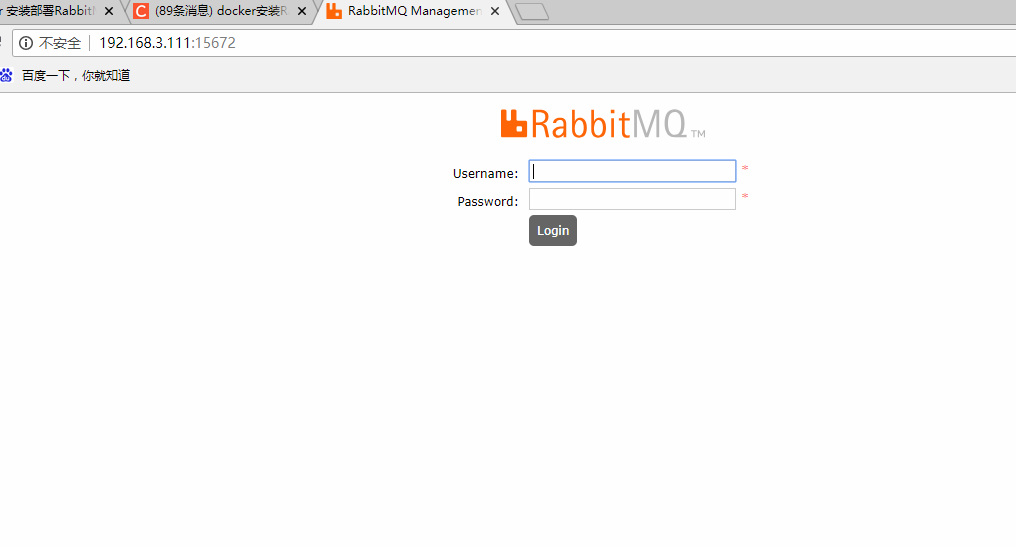
## 开通5672和15672端口

根据各个操作系统开通

## 访问管理界面

访问管理界面的地址就是 http://[宿主机IP]:15672，如果是方式一可以使用默认的账户登录，用户名和密码都guest，如：

http://192.168.3.111:15672/



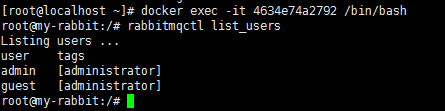
## 基本命令

### 6.1进入docker中的rabbitmq中

[root@localhost ~]# docker exec -it 4634e74a2792 /bin/bash

### 6.2查看用户列表

root@my-rabbit:/# rabbitmqctl list\_users



### 6.3新建用户

rabbitmqctl add\_user username password

例如：

rabbitmqctl add\_user admin1 123456



### 6.4删除用户

rabbitmqctl delete\_user username

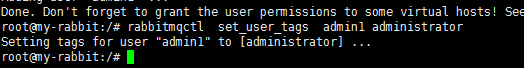
例如：rabbitmqctl delete\_user admin1

### 6.5授权角色

rabbitmqctl set\_user\_tags User Tag

例如：rabbitmqctl set\_user\_tags admin1 administrator

root@my-rabbit:/# rabbitmqctl set\_user\_tags admin1 administrator



### 6.6授权多角色

rabbitmqctl set\_user\_tags admin1 monitoring policymaker

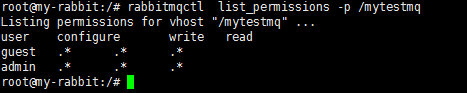


### 6.7查看(指定hostpath)所有用户的权限信息

rabbitmqctl list\_permissions [-p VHostPath]

例如：

rabbitmqctl list\_permissions -p /mytestmq



### 6.8设置用户权限

rabbitmqctl set\_permissions -p VHostPath User ConfP WriteP ReadP

例如：

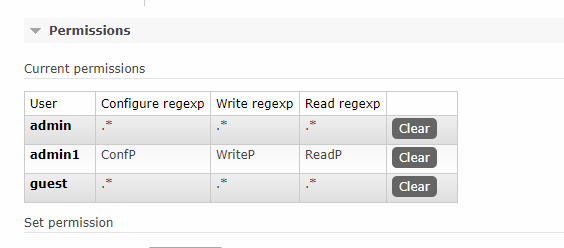
rabbitmqctl set\_permissions -p /mytestmq admin1 ConfP WriteP ReadP

或者

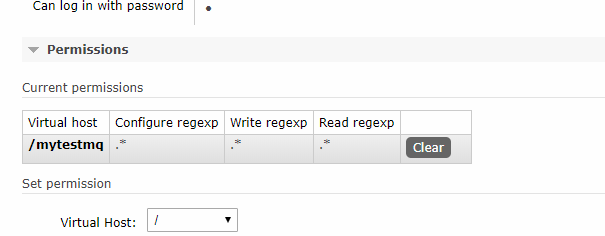
rabbitmqctl set\_permissions -p /mytestmq admin1 ".\*" ".\*" ".\*"

给用户admin1授权 vhost名称为/mytestmq的 配置 写入 读取的权限





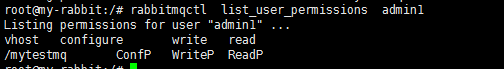
或者



### 6.9查看指定用户的权限信息

rabbitmqctl list\_user\_permissions User

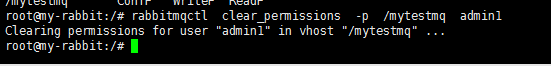
例如：rabbitmqctl list\_user\_permissions admin1

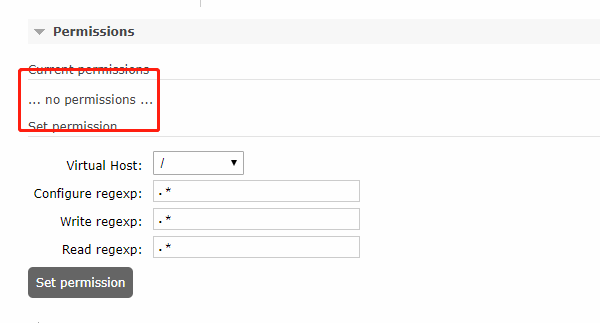


### 6.10清除用户的权限信息

rabbitmqctl clear\_permissions [-p VHostPath] User

例如：rabbitmqctl clear\_permissions -p /mytestmq admin1





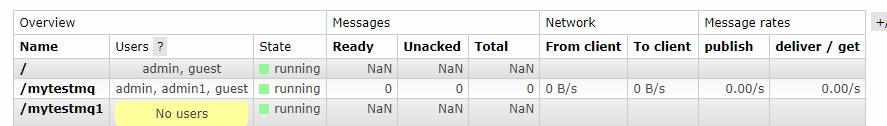
### 6.11新建virtual\_host

rabbitmqctl add\_vhost xxx

例如：

rabbitmqctl add\_vhost /mytestmq1





### 6.12撤销virtual\_host

rabbitmqctl delete\_vhost xxx

例如：rabbitmqctl delete\_vhost /mytestmq1



### 6.13查看virtual\_host列表

rabbitmqctl list\_vhosts

