# Docker

## 安装

### 1.1通过tar安装

通过tar包解压安装docker

以下步骤在centos7和debian10中验证通过

1、通过官网下载安装包

wget https://download.docker.com/linux/static/stable/x86\_64/docker-19.03.5.tgz

2、解压并copy二进制文件到bin目录下

tar -xvf docker-19.03.5.tgz

cp docker/\* /usr/bin/

3、启动docker进程

dockerd &

4、启动hello-world验证

docker run hello-world

### 1.2yum安装

yum install docker

vi /etc/docker/daemon.json

{

"registry-mirrors": ["https://aa25jngun.mirror.aliyuncs.com"]

}

sudo systemctl daemon-reload//重新加载

sudo systemctl restart docker//重启

## 常用命令

启动docker

systemctl start docker

关闭docker

systemctl stop docker

### 2.1镜像命令

docker pull mysql 安装镜像

docker pull mysql:5.7 指定版本安装镜像

查看所有本地镜像

docker images

删除指定的本地镜像

docker rmi -f id 单个

docker rmi -f id id id 多个

docker rmi -f $(docker images -aq) 删除所有的镜像

### 2.2容器命令

docker run [可选参数] image

--name 指定容器的名字

-d 以后台方式运行

-it 使用交互方式运行，进入容器查看内容

-p 指定容器端口

-p 主机端口:容器端口

-p 容器端口

-p 主机ip:端口:容器端口

-P 大写的P 随机映射端口

docker run --name mytomcat2 -d -p 8081:8080 cd097a10ec37

docker run -p 3306:3306 --name mysql01 -e MYSQL\_ROOT\_PASSWORD=123456 -d mysql 根据镜像启动容器

docker run -p 3307:3306 --name mysql02 -e MYSQL\_ROOT\_PASSWORD=123456 -d mysql --character -set -server=utf8mb4 --collation -server=utf8mb4\_unicode\_ci

docker run -d -p 6379:6379 --name myredis redis

启动一个做了端口映射的tomcat

docker run ‐d ‐p 8888:8080 tomcat ‐d：后台运行 ‐p: 将主机的端口映射到容器的一个端口 主机端口:容器内部的端口

docker run --name mytomcat1 -d -p 8081:8080 fabric8/tomcat-8

docker run --name mytomcat2 -d -p 8082:8080 fabric8/tomcat-8

docker stop id 停止容器

docker start id 启动容器

docker restart id 重启容器

docker kill id 杀掉容器

docker rm id 删除容器

docker rm $(docker ps -aq) 删除所有容器

docker ps 查看启动的容器

docker ps -a 查看所有容器

### 2.3常用其他命令

docker logs id 查看容器启动的日志

docker logs -tf --tail 10 id 查看指定条数的日志

docker top id 查看容器的进程信息

docker inspect 容器id 查看容器的元数据

### 2.4进入当前正在运行的容器

1.docker exec -it 容器id /bin/bash (常用)

docker exec -it c7cfb784316e bash

2.docker attach 容器id

两者区别：

docker exec 进入容器后开启一个新的终端

docker attach 进入容器正在执行的终端

### 2.5从容器内拷贝文件到主机

docker cp 容器id:容器内路径 主机路径

将war 包放入tomcat webapps 下

docker cp xxl-job-admin.war id:/opt/apache-tomcat-8.5.32/webapps

## Docker镜像

### Commit镜像

docker commit 提交一个容器成为镜像

docker commit -m=”提交的描述信息” -a=”作者” 容器id 镜像名:[TAG]

docker commit -m="tomcatshibc" -a="shibc" c7cfb784316e tomcatshibc:1.0

实战测试

1. 启动一个默认的Tomcat
2. 默认的 Tomcat没有webapps应用
3. 将默认的启动页面拷贝进去
4. 将操作的容器通过commit提交为一个新的镜像

## 4.容器数据卷

**容器数据源卷的使用**

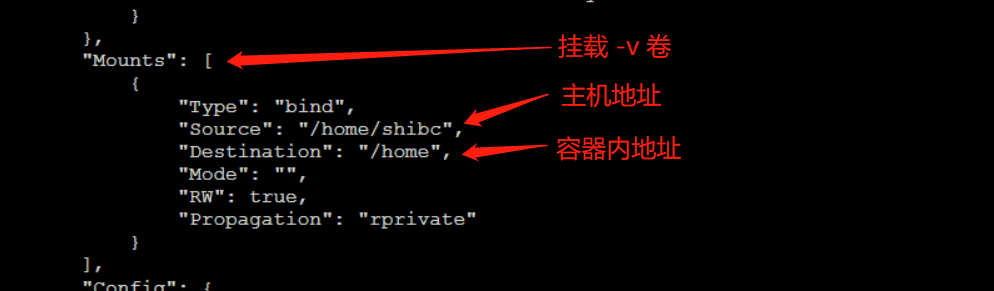
**方式一：直接用命令**

docker run -v 主机目录:容器内的目录

测试：容器和主机是双向绑定的

docker run -it -v /home/shibc:/home 300e315adb2f /bin/bash

启动之后可以通过 docker inpect 容器id 查看容器的信息



**Mysql的持久化**

**-p 端口映射**

**-d 后台运行**

**-v 卷挂载**

**-e 环境配置**

**--name 容器名字**

docker run -p 3310:3306 --name mysql01 -e MYSQL\_ROOT\_PASSWORD=123456 -d -v /home/mysql/conf:/etc/mysql/conf.d -v /home/mysql/data:/var/lib/mysql mysql:5.7

**具名和匿名挂载**

匿名挂载

-v 容器内路径

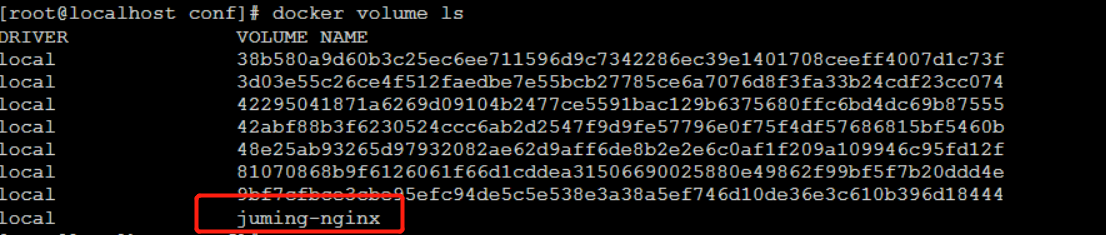
docker run -P -d --name nginx01 -v /etc/nginx nginx

查看所有volume的情况

docker volume ls

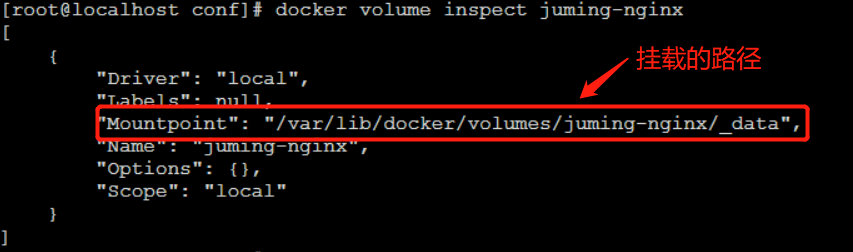
具名挂载

docker run -P -d --name nginx02 -v juming-nginx:/etc/nginx nginx



查看挂载的路径

docker volume inspect juming-nginx



-v 容器内路径 匿名挂载

-v 卷名:容器内路径 具名挂载

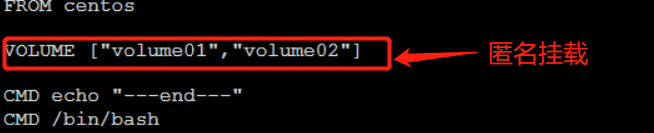
-v /宿主机路径:容器内路径 指定路径挂载

**初识dockerfile**

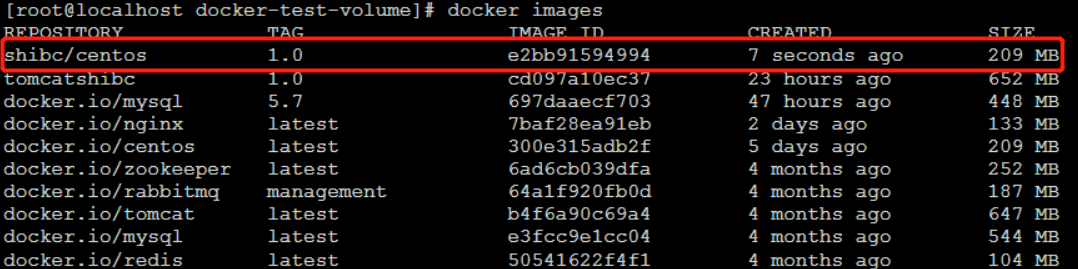
dockerfile就是用来构建docker镜像的构建文件！命令脚本

通过脚本可以生产镜像，镜像是一层一层的，脚本一个个的命令，每个命令都是一层

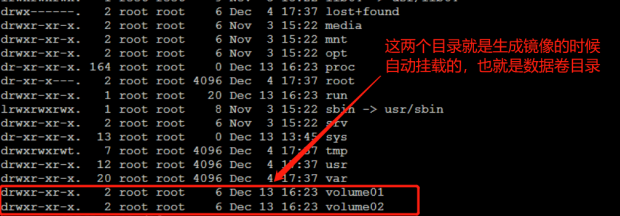
新建dockerfile



docker build -f /home/docker-test-volume/dokcerfile1 -t shibc/centos:1.0 .

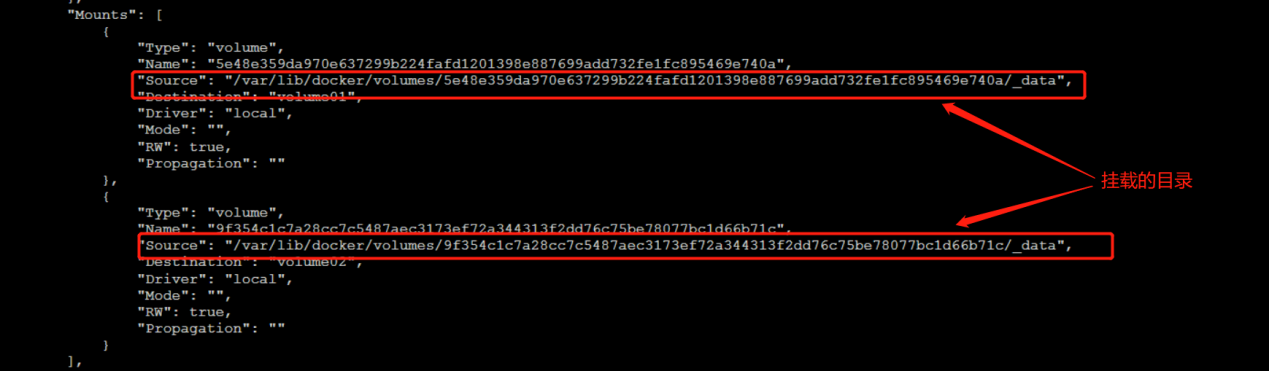


docker run -it e2bb91594994 /bin/bash



查看卷挂载的路径

docker inspect 容器id



**数据卷容器**

docker run -p 3310:3306 --name mysql01 -e MYSQL\_ROOT\_PASSWORD=123456 -d -v /home/mysql/conf:/etc/mysql/conf.d -v /home/mysql/data:/var/lib/mysql mysql:5.7

docker run -p 3311:3306 --name mysql02 -e MYSQL\_ROOT\_PASSWORD=123456 -d --volumes-from mysql01 mysql:5.7

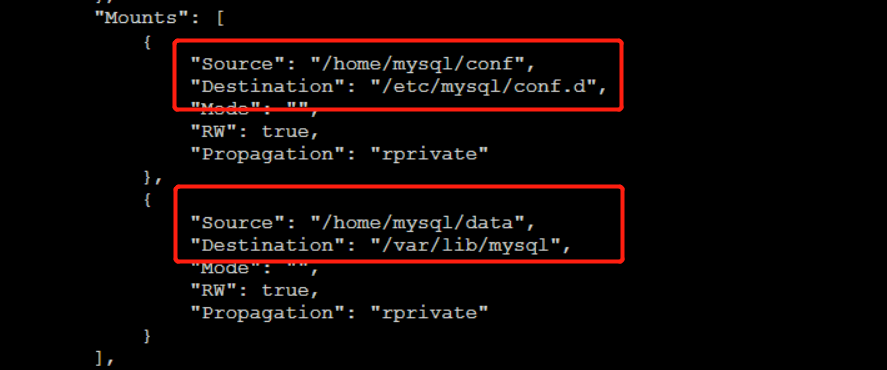
**Mysql01**

**docker inspect mysql01**



**Mysql02**

**docker inspect mysql02**



**多个mysql数据同步**



**结论:**

**容器之间配置信息传递，数据卷容器的生命周期一直持续到没有容器为止**

## 5.DockerFile

dockerfile就是用来构建docker镜像的构建文件！

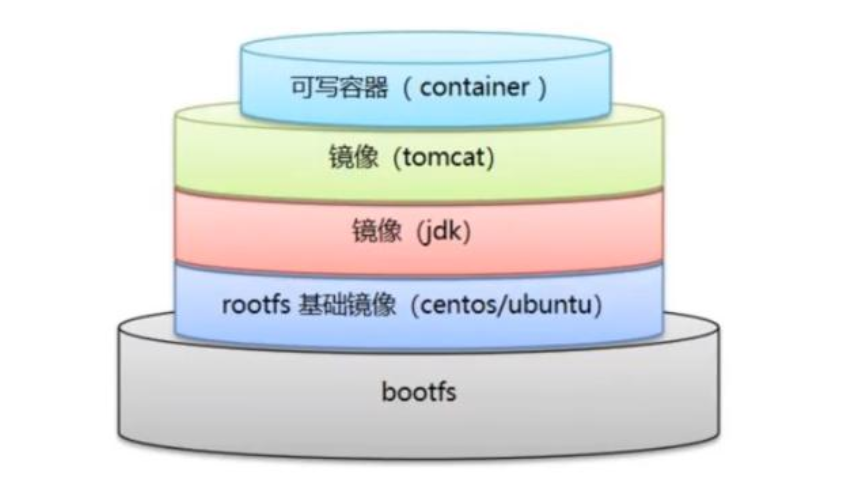
构建步骤

1. 编写一个dockerfile文件
2. docker bulid 构建成一个镜像
3. docker run 运行镜像
4. docker push 发布镜像

**DockerFile的构建过程**

**基础知识:**

1. 每一个保留关键字(指令)都必须是大写字母
2. 执行是从上到下顺序执行
3. #表示注释
4. 每一个指令都会创建提交一个新的镜像层，并提交



Dockerfile是面向开发的，我们以后要发布项目，做镜像，就需要编写dockerfile文件

DockerFile:构建文件，定义了一切的步骤，源代码

DockerImages:通过DockerFile构建生成的镜像，最终发布和运行的产品

Docker容器: 容器就是镜像运行起来提供服务

**DockerFile的指令**

**FROM #基础镜像**

**MAINTAINER #镜像是谁写的 姓名+邮箱**

**RUN #镜像构建的时候运行的命令**

**ADD #步骤，tomcat镜像，添加内容**

**WORKDIR #镜像的工作目录**

**VOLUME #挂载的目录**

**EXPOSE #暴露端口位置**

**CMD #指定这个容器启动的时候要运行的命令,只有最后一个会生效，可被替代**

**ENTRYPOINT #指定这个容器启动的时候要运行的命令，可以追加命令**

**ONBUILD #当构建一个被继承dockerfile这个时候就会运行ONBUILD 的指令，触发指令**

**COPY #类似ADD，将文件拷贝到镜像中**

**ENV #构建的时候设置环境变量**

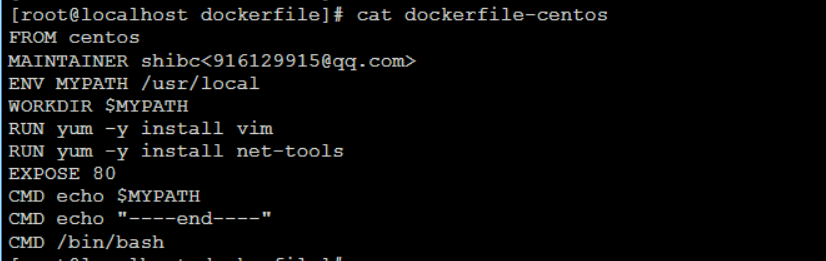




**实战测试**

构建自己的centos

1.编写dockerfile文件



1. 通过文件构建镜像

docker build -f dockerfile文件路径 -t 镜像名:[tag] .

docker build -f dockerfile-centos -t shibc/centos:2.0 .

可以通过docker history 镜像id 查看镜像的历史信息



CMD和ENTRYPOINT的区别

CMD 是替换

vi docker-CMD-test

FROM centos

CMD [“ls”,”a’]

docker build -f docker-CMD-test-t CMD-test .

docker run CMD-test -l #报错ls -l 不是命令

ENTRYPOINT 是追加

vi docker-ENTRYPOINT -test

FROM centos

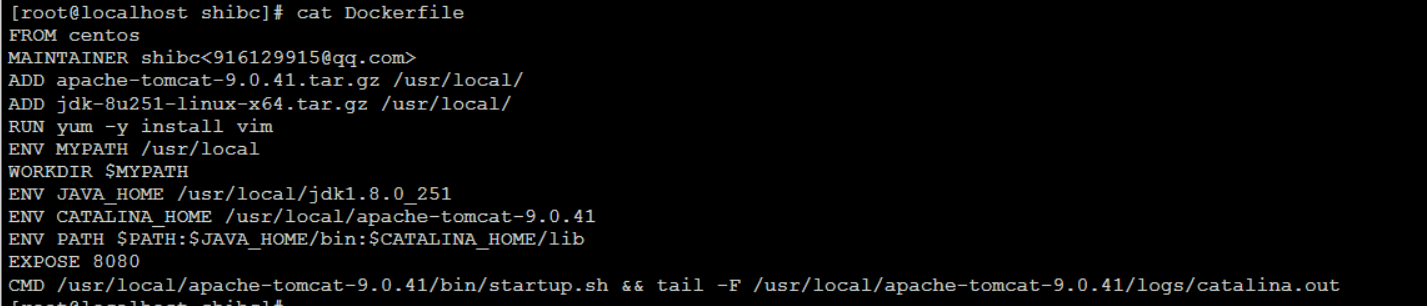
ENTRYPOINT [“ls”,”a’]

docker build -f docker-ENTRYPOINT -test ENTRYPOINT -test .

docker run ENTRYPOINT -test -l #不报错ls -al

**实战Tomcat**

docker build -f Dockerfile -t mytomcat:0.1 .



FROM centos

MAINTAINER cheng<1204598429@qq.com>

COPY README /usr/local/README #复制文件

ADD jdk-8u231-linux-x64.tar.gz /usr/local/ #复制解压

ADD apache-tomcat-9.0.35.tar.gz /usr/local/ #复制解压

RUN yum -y install vim

ENV MYPATH /usr/local #设置环境变量

WORKDIR $MYPATH #设置工作目录

ENV JAVA\_HOME /usr/local/jdk1.8.0\_231 #设置环境变量

ENV CATALINA\_HOME /usr/local/apache-tomcat-9.0.35 #设置环境变量

ENV PATH $PATH:$JAVA\_HOME/bin:$CATALINA\_HOME/lib #设置环境变量 分隔符是：

EXPOSE 8080 #设置暴露的端口

CMD /usr/local/apache-tomcat-9.0.35/bin/startup.sh && tail -F /usr/local/apache-tomcat-9.0.35/logs/catalina.out # 设置默认命令

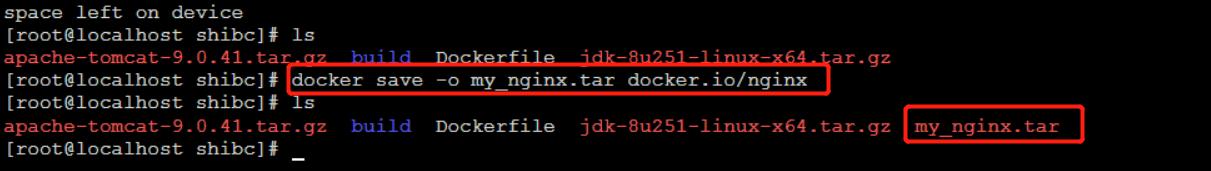
**运行镜像**

docker run -d -p 8080:8080 --name tomcat01 -v /home/shibc/build/tomcat/test:/usr/local/apache-tomcat-9.0.41/webapps/test -v /home/shibc/build/tomcat/tomcatlogs/:/usr/local/apache-tomcat-9.0.41/logs mytomcat:0.1

**将镜像打成tar包**

docker save -o tar包名字 镜像名或id

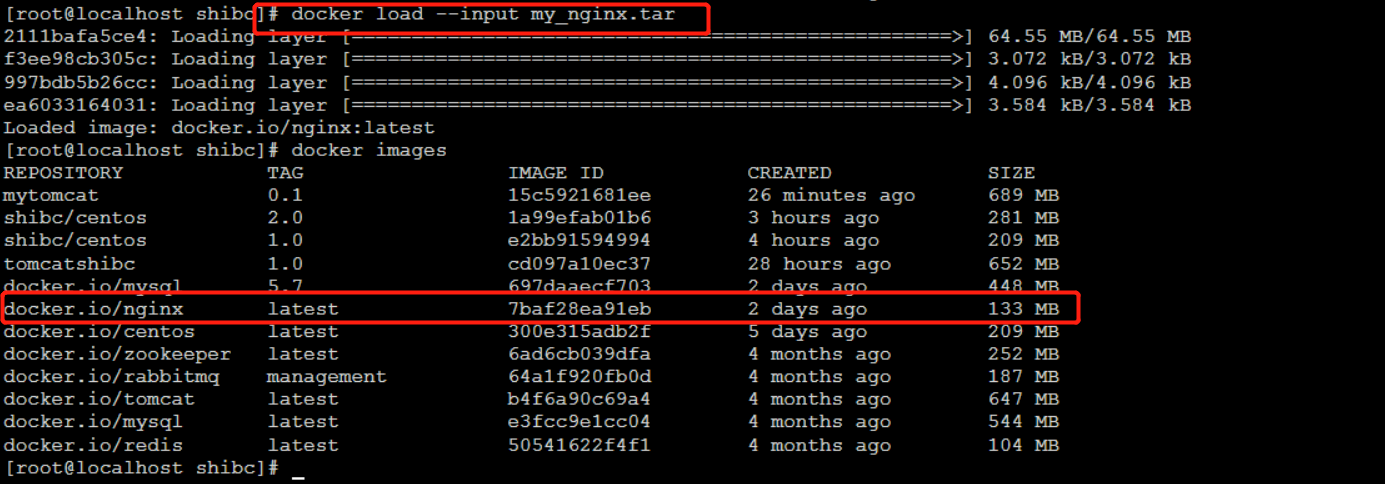
docker save -o my\_nginx.tar docker.io/nginx



**将tar包打成镜像**

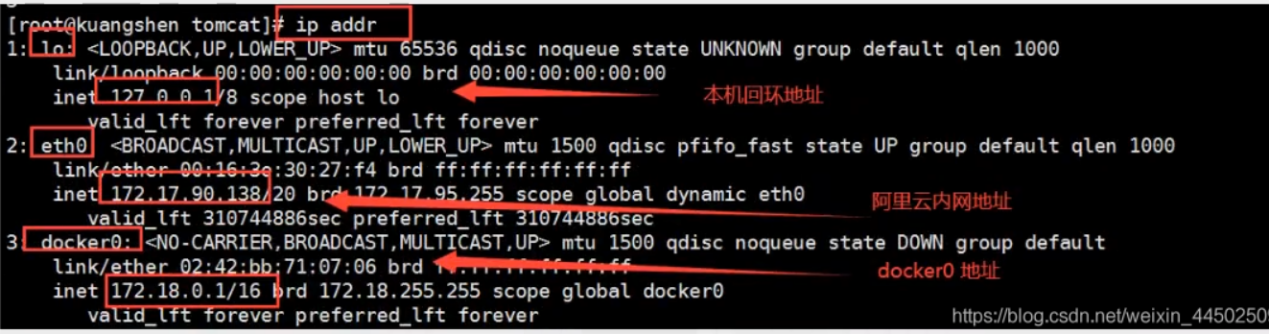
docker load --input tar名字

docker load --input my\_nginx.tar



## Docker网路

理解docker0

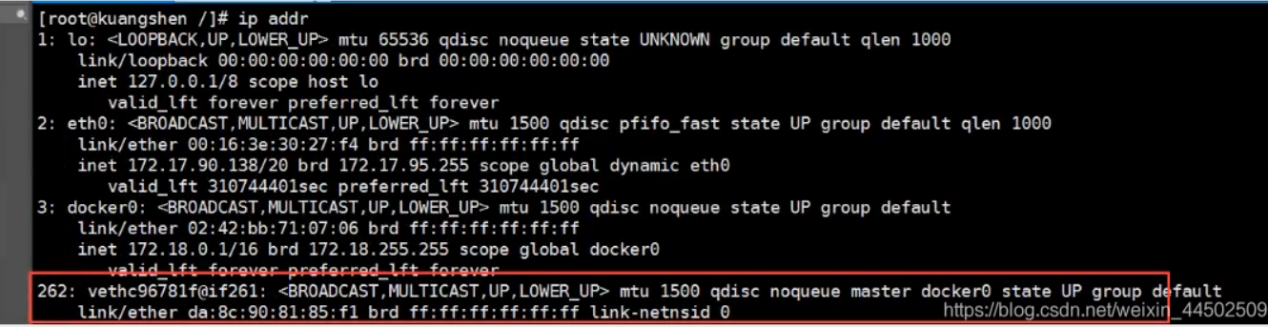




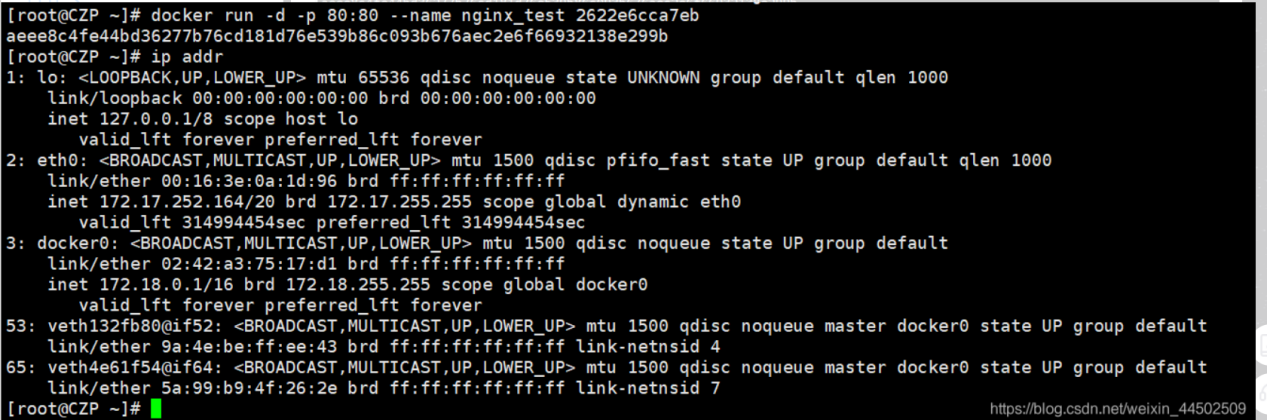
**原理**

我们每启动一个docker容器,docker就会给docker容器分配一个ip,我们只要安装了docker,就会有一个网卡docker0,桥接模式,使用的技术是evth-pair技术!

再次测试 ip addr



再启动一个容器

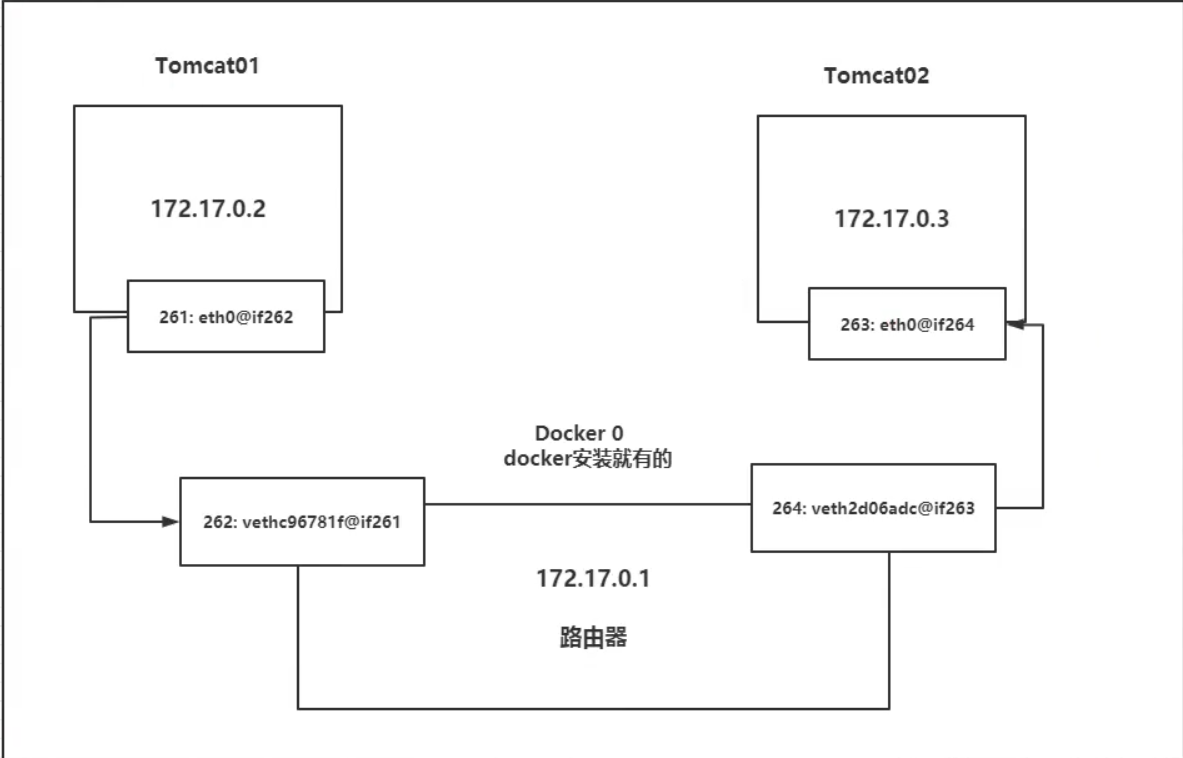


# 我们发现这个容器带来网卡, 都是一对对的

# evth-pair 就是一对虚拟机设备接口,他们都是成对出现的,一端连着协议,一端彼此相连

# 正因为有这个特性,veth-pair 充当桥梁,连接各种虚拟网络设备的

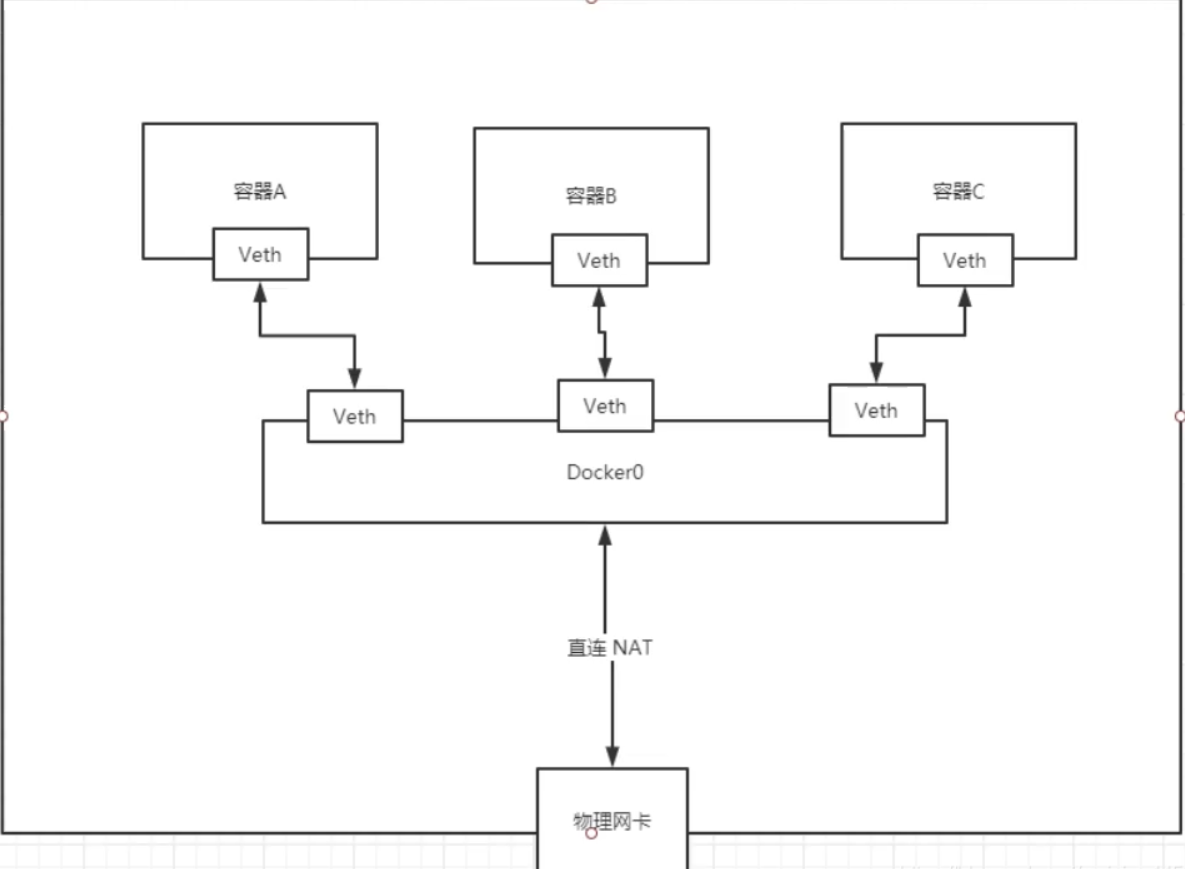
# openStac,Docker容器之间的连接,OVS的连接,都是使用 evth-pair 技术



结论: tomcat01和tomcat02是共用的一个路由器,docker0

所有的容器不指定网络的情况下,都是docker0路由的,docker会给我们的容器分配一个默认的可用IP

Docker 使用的是Linux的桥接,宿主机中是一个Docker容器的网桥,docker0



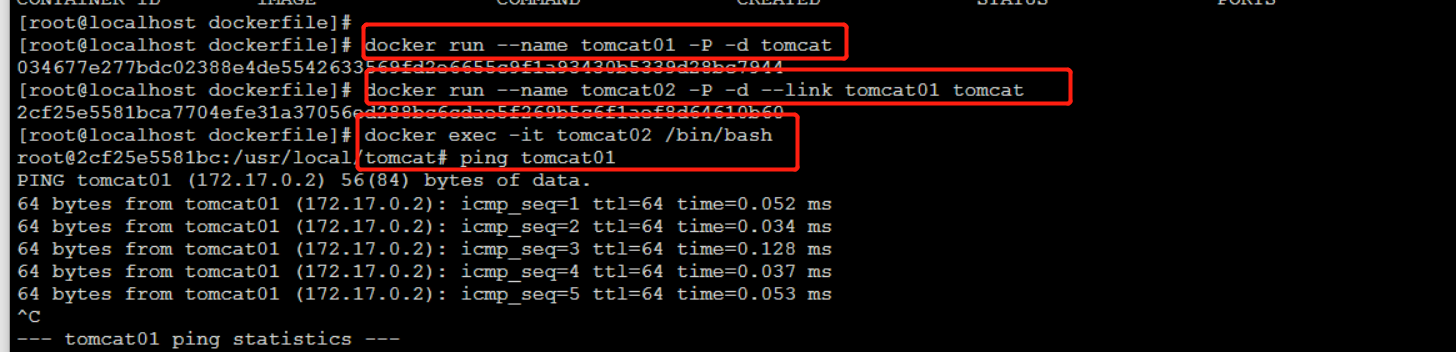
## **–link**

思考一个场景,我们编写了一个微服务,database url=ip: 项目不重启,数据库IP换掉了,我们希望可以通过名字来访问服务

docker run --name tomcat01 -P -d tomcat

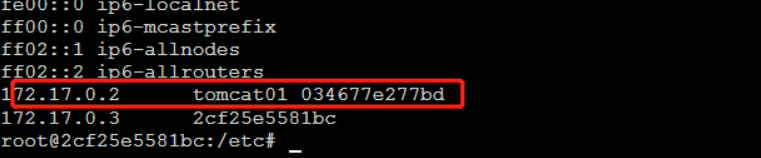
tomcat02是可以ping通tomcat01的，但是tomcat01是ping不通tomcat02

docker run --name tomcat02 -P -d --link tomcat01 tomcat



在tomcat2中

/etc/hosts 配置端口和域名的绑定



本地探究 – link 就是我们在host配置中增加了一个172.18.0.3 tomcat02 312857784cd4

我们现在玩Docker已经不建议使用–link了!

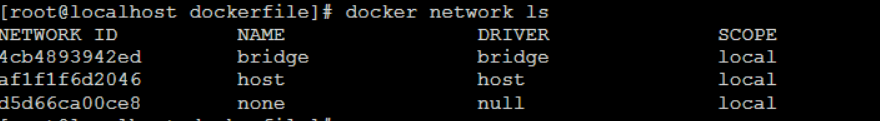
自定义网络,不使用docker0!

docker0问题: 它不支持容器名连接访问!

## 自定义网络

查看所有的docker网络

docker network ls



****网络模式****

bridge : 桥接 docker 大桥

none: 不配置网络

host: 和宿主机共享网络

container: 容器内网络联通!

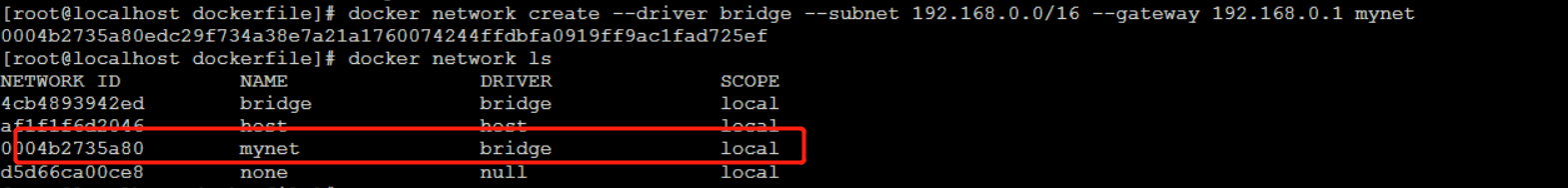
# 直接启动的命令 --net brodge,默认docker0

docker run -d -P --name tomcat01 --net bridge tomcat

# docker0的特点: 默认的,域名是不能访问的, --link可以打通连接

**创建自定义网络**

docker network create --driver bridge --subnet 192.168.0.0/16 --gateway 192.168.0.1 mynet



查看创建好网络的详细信息

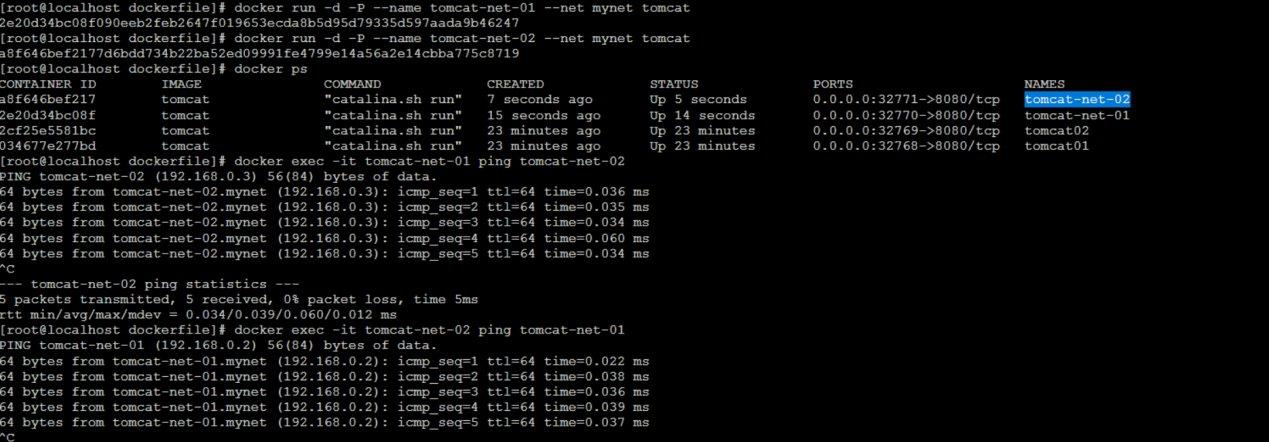
docker network inspect mynet

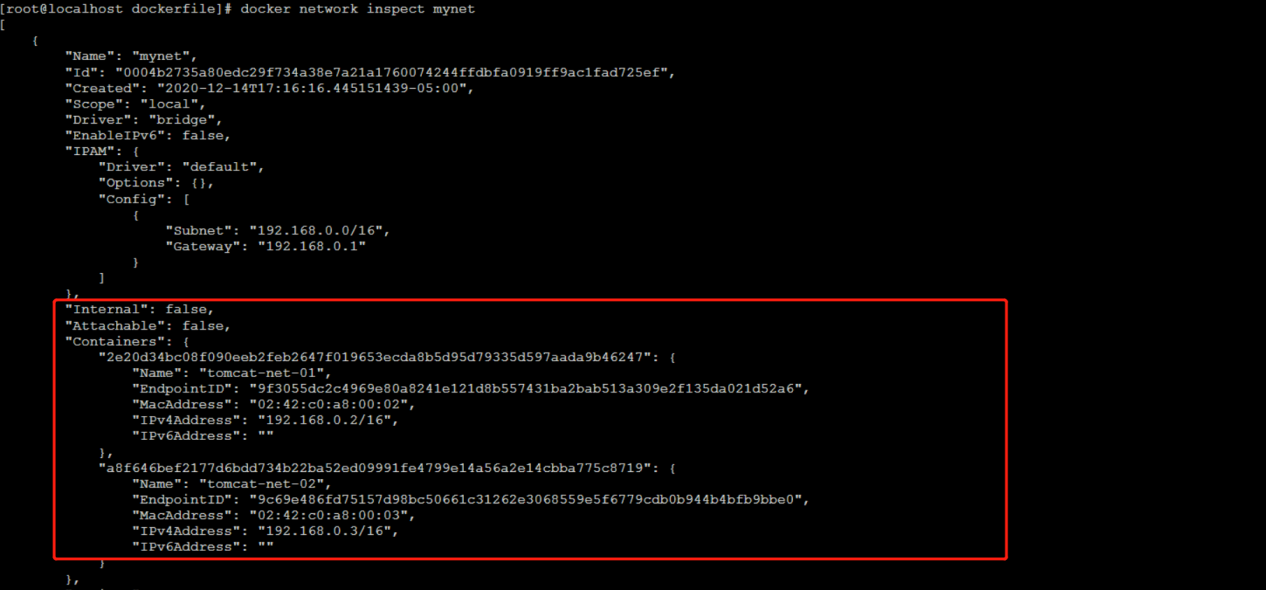


docker run -d -P --name tomcat-net-01 --net mynet tomcat

docker run -d -P --name tomcat-net-02 --net mynet tomcat

网络ping通是双向的

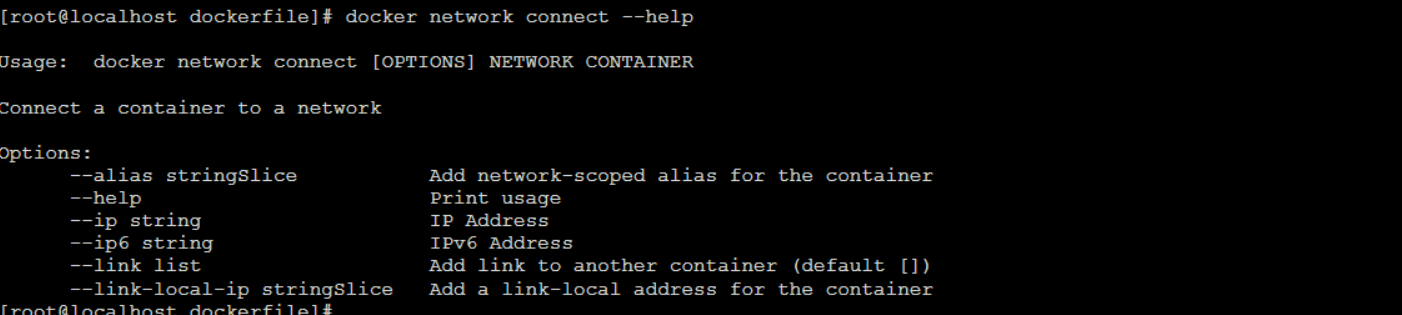




**网络连通**

#测试打通 tomcat01到tomcat-net-01

# 连通之后就是将 tomcat01 放到了mynet网络下



docker network connect 网络 容器名

docker network connect mynet tomcat01

docker network inspect mynet



## Docker Compose

**#官方下载**

sudo curl -L "https://github.com/docker/compose/releases/download/1.26.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

**#授权**

chmod +x /usr/local/bin/docker-compose

**# 加速下载**

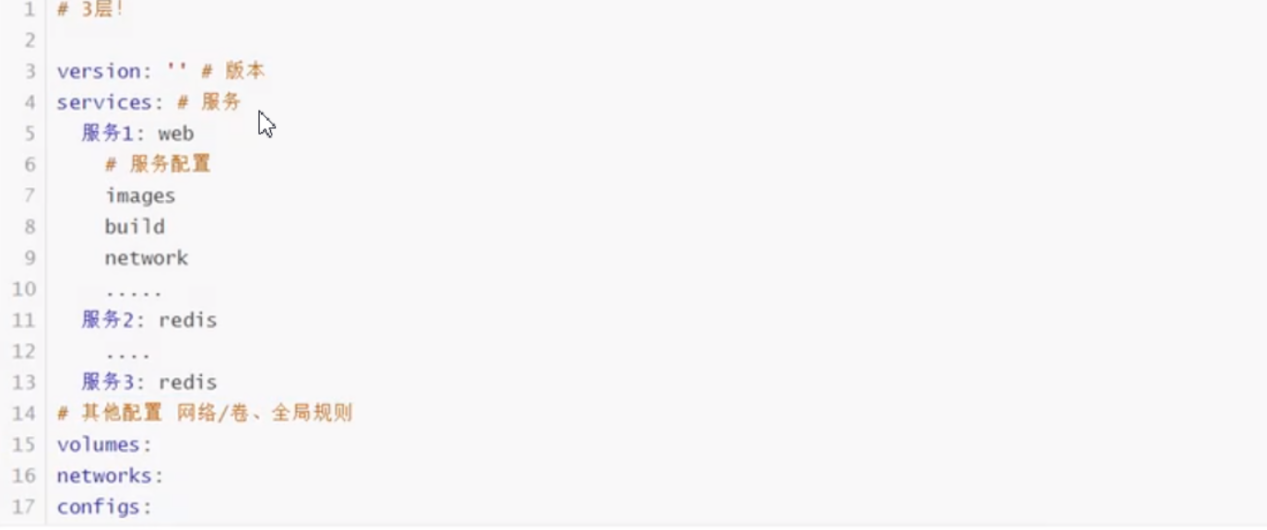
curl -L https://get.daocloud.io/docker/compose/releases/download/1.26.2/docker-compose-`uname -s`-`uname -m` > /usr/local/bin/docker-compose

**#授权**

chmod +x /usr/local/bin/docker-compose



**compose配置编写规则**





**实战:自己编写微服务上线**

**Dockerfile**

*FROM* java:8  
*COPY \**.jar */*app.jar  
*CMD* ["--server.port=8080"]  
*EXPOSE* 8080  
*ENTRYPOINT* ["java","-jar","app.jar"]

**docker-compos.yml**

version: '3.8'  
services:  
 shiapp:  
 build: .  
 image: shiapp  
 depends\_on:  
 - redis  
 ports:  
 - "8080:8080"  
 redis:  
 image: redis

**启动**

**docker-compose up -d**

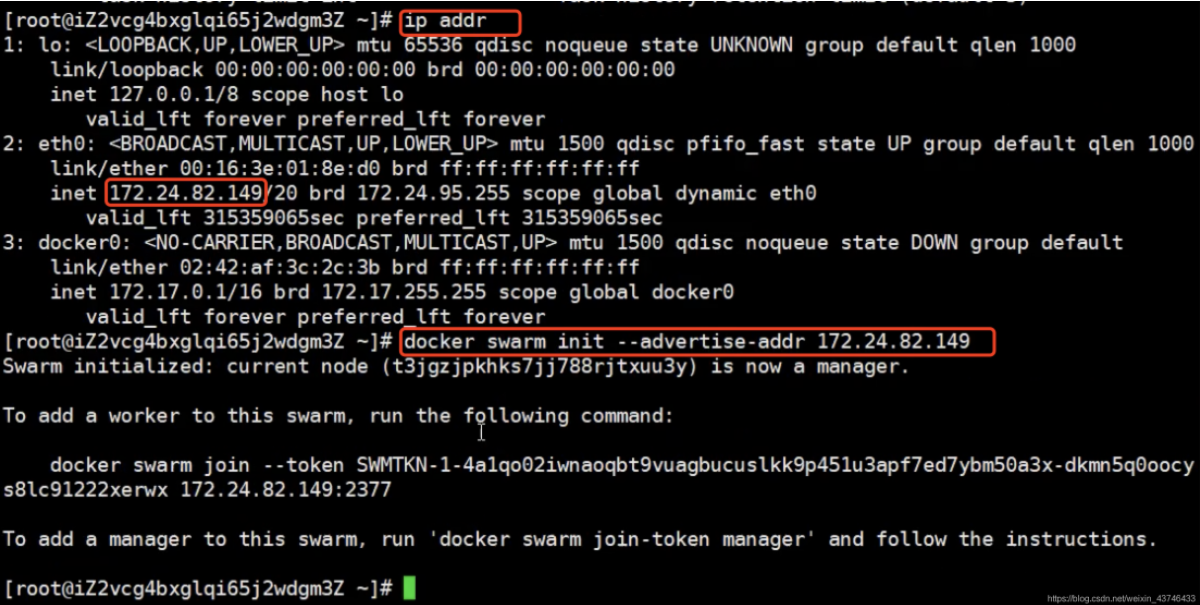
**关闭**

**docker-compose down**

## Docker swarm

### 搭建集群

### **初始化节点**



### **Docker swarm 加入节点**

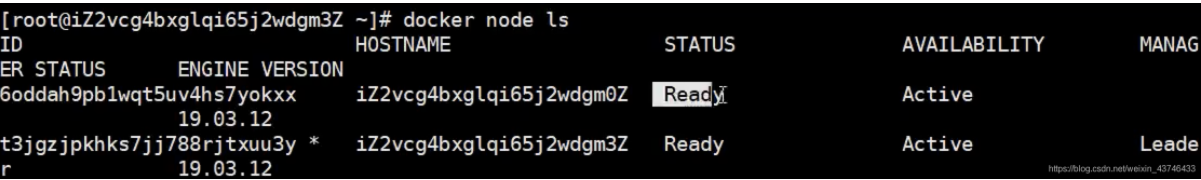
#获取令牌

docker swarm join-token manager

docker swarm join-token worker

### **查看节点**

docker node ls

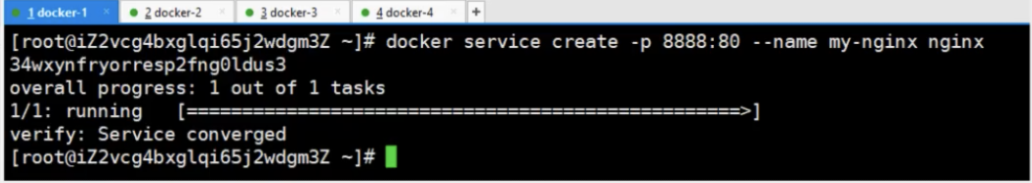


### Raft一致性协议

保证大多数节点存活,才可以使用,高可用

## **Swarm集群弹性创建服务**

### **创建service**



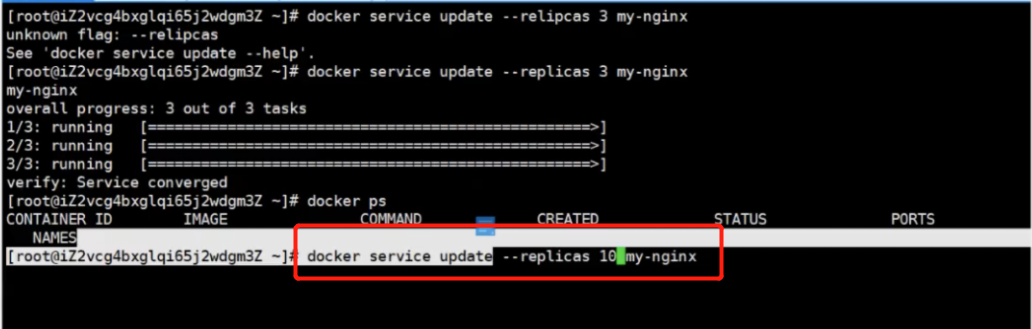
docker run 容器启动,不具有扩缩容器

docker service 服务,具有扩缩容器,滚动风险

### **动态扩缩容**

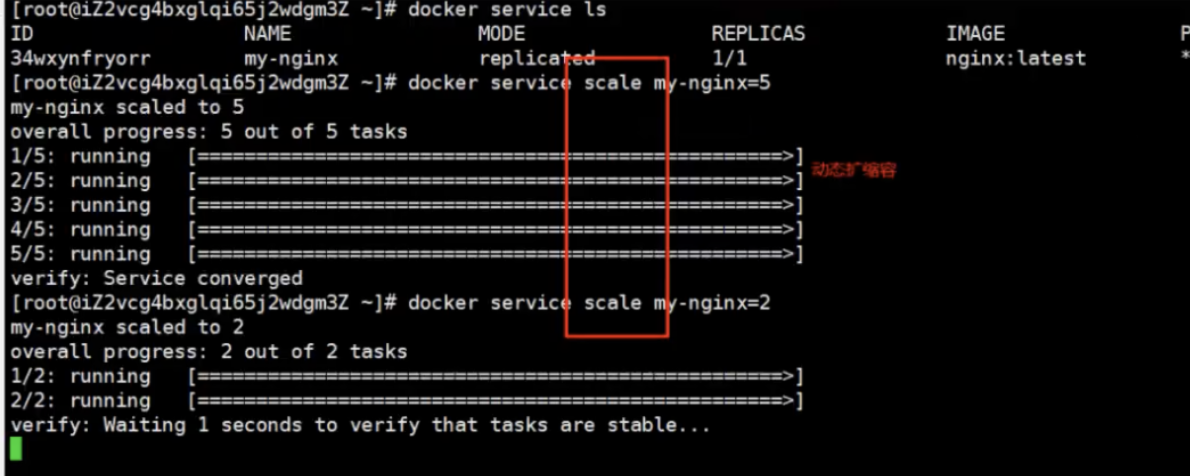
第一种方式

docker service update --replicas 10 my-nginx



第二种方式

docker service scale my-nginx=10

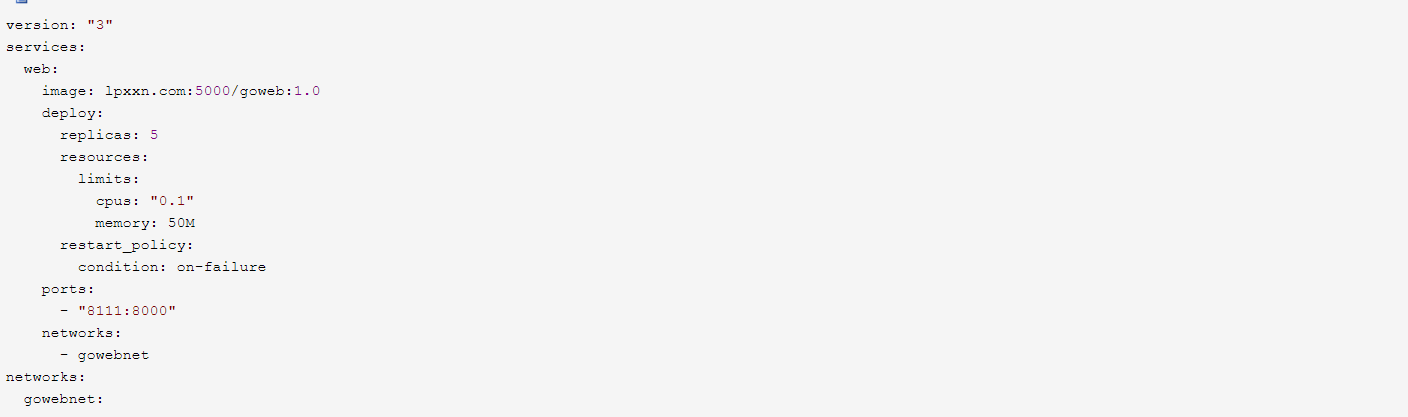


## **调整service 方式运行**



### Docker Stack

****docker-compose 单机部署  
docker stack 集群部署****



### Docker Secret

