# Docker

## 主要内容

1. docker简介
2. docker的安装和加速配置
3. **docker相关命令详解**
4. docker容器数据卷
5. **Dockerfile详解**
6. docker常用安装
7. **docker四种网络模型及使用**
8. docker部署我们自己的项目
9. 其它说明

## 学习目标

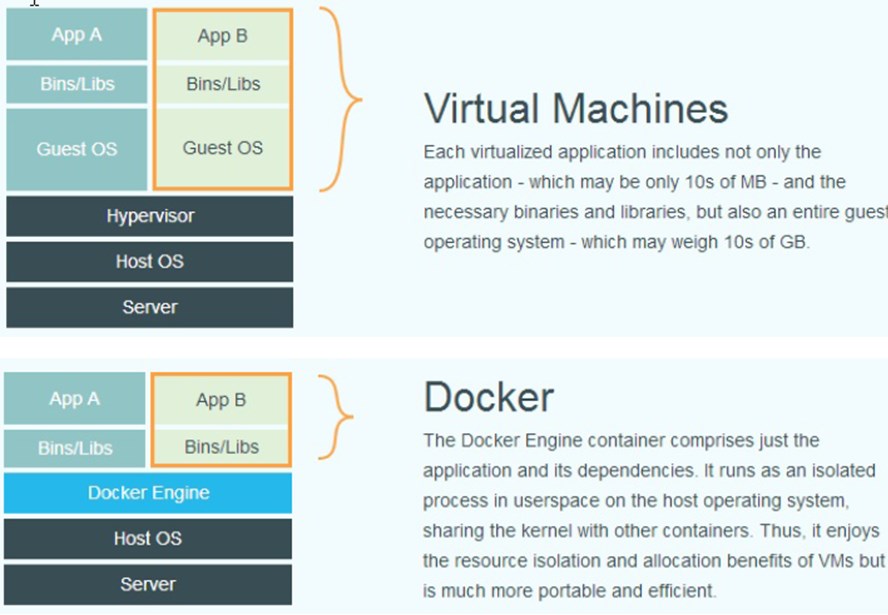
|  |  |
| --- | --- |
| 知识点 | 要求 |
| docker简介 | 掌握 |
| docker的安装和加速配置 | 掌握 |
| docker相关命令详解 | 掌握 |
| docker容器数据卷 | 掌握 |
| Dockerfile详解 | 掌握 |
| docker常用安装 | 掌握 |
| docker四种网络模型及使用 | 掌握 |
| docker部署我们自己的项目 | 掌握 |

# 【熟悉】docker简介

## 什么是docker

        Docker是一个开源项目，诞生于2013年初，最初是dotCloud公司内部的一个业余项目。它基于Google公司推出的Go语言实现。项目后来加入了Linux基金会，遵从了Apache2.0协议，项目代码在GitHub上进行维护。Docker自开源后受到广范的关注和讨论，以至于dotCloud公司后来都改名为Docker Inc。RedHat已经在其RHEL6.5中集中支持Docker;Google也在其PaaS产品中广泛应用。Docker的目标是实现**轻量级的操作系统虚拟化解决方案**。Docker的基础是Linux容器（LXC）等技术。在LXC的基础上Docker进行了进一步的封装，让用户不需要关心容器的管理，使得操作更加简单。用户操作Docker的容器就像操作一个快速轻量级的虚拟机一样简单。

        下图比较了Docker和传统虚拟化方式的不同之处，可见容器是在操作系统层面上实现的虚拟化，直接复用本地主机的操作系统，而传统方式则是在硬件层现实现



## docker的设计思想

        Docker的思想来自于集装箱，集装箱解决了什么问题？在一艘大船上，可以把货物规整的摆放起来。并且各种各样的货物被集装箱标准化了，集装箱和集装箱之间不会互相影响。那么我就不需要专门运送水果的船和专门运送化学品的船了。只要这些货物在集装箱里封装的好好的，那我就可以用一艘大船把他们都运走。docker就是类似的理念。现在都流行云计算了，云计算就好比大货轮。docker就是集装箱。

1.不同的应用程序可能会有不同的应用环境，比如.net开发的网站和php开发的网站依赖的软件就不一样，如果把他们依赖的软件都安装在一个服务器上就要调试很久，而且很麻烦，还会造成一些冲突。比如IIS和Apache访问端口冲突。这个时候你就要隔离.net开发的网站和php开发的网站。常规来讲，我们可以在服务器上创建不同的虚拟机在不同的虚拟机上放置不同的应用，但是虚拟机开销比较高。docker可以实现虚拟机隔离应用环境的功能，并且开销比虚拟机小，小就意味着省钱了。

2.你开发软件的时候用的是Ubuntu，但是运维管理的都是centos，运维在把你的软件从开发环境转移到生产环境的时候就会遇到一些Ubuntu转centos的问题，比如：有个特殊版本的数据库，只有Ubuntu支持，centos不支持，在转移的过程当中运维就得想办法解决这样的问题。这时候要是有docker你就可以把开发环境直接封装转移给运维，运维直接部署你给他的docker就可以了。而且部署速度快。

3.在服务器负载方面，如果你单独开一个虚拟机，那么虚拟机会占用空闲内存的，docker部署的话，这些内存就会利用起来。

总之docker就是集装箱原理。

## 为什么要使用docker

        作为一种新兴的虚拟化方式，Docker 跟传统的虚拟化方式相比具有众多的优势。首先，Docker 容器的启动可以在秒级实现，这相比传统的虚拟机方式要快得多。其次，Docker 对系统资源的利用率很高，一台主机上可以同时运行数千个Docker 容器。

容器除了运行其中应用外，基本不消耗额外的系统资源，使得应用的性能很高，同时系统的开销尽量小。

传统虚拟机方式运行10 个不同的应用就要起10 个虚拟机，而Docker 只需要启动10 个隔离的应用即可。具体说来，Docker 在如下几个方面具有较大的优势。

### 更快速的交付和部署

对开发和运维（devop）人员来说，最希望的就是一次创建或配置，可以在任意地方正常运行。开发者可以使用一个标准的**镜像**来构建一套开发容器，开发完成之后，运维人员可以直接使用这个容器来部署代码。Docker 可以快速创建容器，快速迭代应用程序，并让整个过程全程可见，使团队中的其他成员更容易理解应用程序是如何创建和工作的。Docker 容器很轻很快！容器的启动时间是秒级的，大量地节约开发、测试、部署的时间.

### 更轻松的迁移和扩展

Docker 容器几乎可以在任意的平台上运行，包括物理机、虚拟机、公有云、私有云、个人电脑、服务器等。这种兼容性可以让用户把一个应用程序从一个平台直接迁移到另外一个。

### 更简单的管理

使用Docker，只需要小小的修改，就可以替代以往大量的更新工作。所有的修改都以增量的方式被分发和更新，从而实现自动化并且高效的管理。对比虚拟机



### docker的局限性

1. Docker是基于Linux 64bit的，无法在32bit的linux/Windows/unix环境下使用
2. **LXC**是基于cgroup等linux kernel功能的，因此container的guest系统只能是linux base的
3. 隔离性相比KVM之类的虚拟化方案还是有些欠缺，所有container公用一部分的运行库
4. 网络管理相对简单，主要是基于namespace隔离
5. cgroup的cpu和cpuset提供的cpu功能相比KVM的等虚拟化方案相比难以度量(所以dotcloud主要是按内存收费)
6. Docker对disk的管理比较有限
7. container随着用户进程的停止而销毁，container中的log等用户数据不便收集

## docker在开发、测试、部署中的定位

### 尝试新软件

对开发者而言，每天会催生出的各式各样的新技术都需要尝试，然而开发者却不太可能为他们一一搭建好环境并进行测试。时间非常宝贵，正是得益于 Docker，让我们有可能在一条或者几条命令内就搭建完环境。Docker 有一个傻瓜化的获取软件的方法，Docker 后台会自动获得环境镜像并且运行环境。

并不仅仅是新技术环境搭建用得到 Docker。如果你想快速在你的笔记本上运行一个 MySQL 数据库，或者一个 Redis 消息队列，那么使用 Docker 便可以非常容易地做到。例如 Docker 只需要一条命令便可以运行 MySQL 数据库：**docker run -d -p 3306:3306 tutum/mysql。**

### 进行演示

工作中自己开发的成果对客户或者别人做一两个演示。搭建演示环境的过程非常麻烦。Docker是演示这些工具的最合理的方式。同时，对于客户来说，可以直接将 Docker 镜像提供给他们，而不必去做任何环境配置的工作，工作的效果也会和在他们演示中所看到的一模一样，同时不必担心他们的环境配置会导致我们的产品无法运行。

### 避免“我机器上可以运行”

无论是上一篇介绍的企业部署 Docker 还是本文的个人 Docker 用例，都提到了这个情况。因为环境配置不同，很多人在开发中也会遇到这个情况，甚至开发的软件到了测试人员的机器上便不能运行。但这都不是重点。重点是，如果我们有一个可靠的、可分发的标准开发环境，那么我们的开发将不会像现在这么痛苦。Docker 便可以解决这个问题。Docker 镜像并不会因为环境的变化而不能运行，也不会在不同的电脑上有不同的运行结果。可以给测试人员提交含有应用的 Docker 镜像，这样便不再会发生“在我机器上是可以运行的”这种事情，很大程度上减轻了开发人员测试人员互相检查机器环境设置带来的时间成本。

### 更好地利用资源

虚拟机的粒度是“虚拟出的机器”，而 Docker 的粒度则是“被限制的应用”，相比较而言 Docker 的内存占用更少，更加轻量级。对我来说这是 Docker 的一个优势：因为在如果在电脑中运行多个 Docker 应用，使用 Docker 比使用虚拟机更加简单，方便，粒度更细，也能持续地跟踪容器状态。

### 为微服务定制

我们一直在讲“微服务（Microservices）”的概念。Docker 可以很好地和微服务结合起来。从概念上来说，一个微服务便是一个提供一整套应用程序的部分功能，Docker 便可以在开发、测试和部署过程中一直充当微服务的容器。甚至生产环境也可以在 Docker 中部署微服务。

### 在云服务提供商之间移植

大多数的云主机提供商已经全面支持 Docker。对于开发人员来说，这表示你可以很方便地切换云服务提供商，当然也可以很方便地将你本地的开发环境移动到云主机上，不需要本地上配置一次运行环境、在云主机上还配置一次运行环境。全面部署 Docker (Docker here and Docker there) 作为标准运行环境可以极大地减轻应用上线时的工作量和产生 BUG。

### 技术的创新

Docker 正在快速发展，工具也在不断更新，没有人能预见到未来 Docker 会是什么样子的。你在复杂的系统中 Docker 使用的越多，越是可能会发现技术上的空白和未来技术发展的方向。现在还处在 Docker 的发展期，任何你使用 Docker 创建的工具都有可能成为社区关注的热点。这是 Docker 的机会，也是成就你自己的机会。

## docker和虚拟技术比较

### 之前的虚拟技术

虚拟机（virtual machine）就是带环境安装的一种解决方案。

它可以在一种操作系统里面运行另一种操作系统，比如在Windows 系统里面运行Linux 系统。应用程序对此毫无感知，因为虚拟机看上去跟真实系统一模一样，而对于底层系统来说，虚拟机就是一个普通文件，不需要了就删掉，对其他部分毫无影响。这类虚拟机完美的运行了另一套系统，能够使应用程序，操作系统和硬件三者之间的逻辑不变。



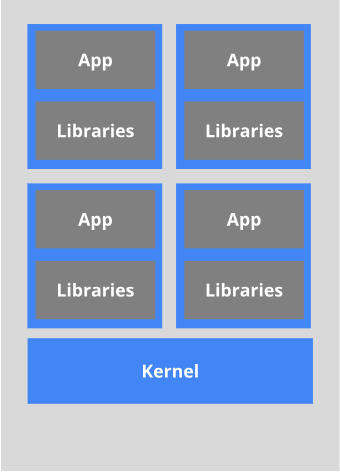
虚拟机的缺点：

1 资源占用多 2 冗余步骤多 3 启动慢

### docker容器化技术

    由于前面虚拟机存在这些缺点，Linux 发展出了另一种虚拟化技术：Linux 容器（Linux Containers，缩写为 LXC）。

Linux 容器不是模拟一个完整的操作系统，而是对进程进行隔离。有了容器，就可以将软件运行所需的所有资源打包到一个隔离的容器中。容器与虚拟机不同，不需要捆绑一整套操作系统，只需要软件工作所需的库资源和设置。系统因此而变得高效轻量并保证部署在任何环境中的软件都能始终如一地运行。



比较了 Docker 和传统虚拟化方式的不同之处：

**\*传统虚拟机技术是虚拟出一套硬件后，在其上运行一个完整操作系统，在该系统上再运行所需应用进程；**

**\*而容器内的应用进程直接运行于宿主的内核，容器内没有自己的内核，而且也没有进行硬件虚拟。因此容器要比传统虚拟机更为轻便。**

**\* 每个容器之间互相隔离，每个容器有自己的文件系统 ，容器之间进程不会相互影响，能区分计算资源。**

# 【掌握】docker说明

## 相关网站

<https://www.docker.com/> docker的官网

<https://docs.docker.com/> 文档

<https://hub.docker.com/> 镜像网站

## Docker 下载

1，官网 <http://www.docker.com>

2，仓库  <https://hub.docker.com>

## Docker 安装前提说明

### CentOS Docker 安装

Docker支持以下的CentOS版本：

CentOS 7 (64-bit)

### 前提条件

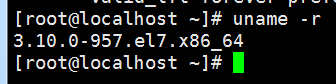
目前，CentOS 仅发行版本中的内核支持 Docker。

Docker 运行在 CentOS 7 上，要求系统为64位、系统内核版本为 3.10 或以上。[我们基于7以上的版本来讲的哦]

### 查看自己的内核

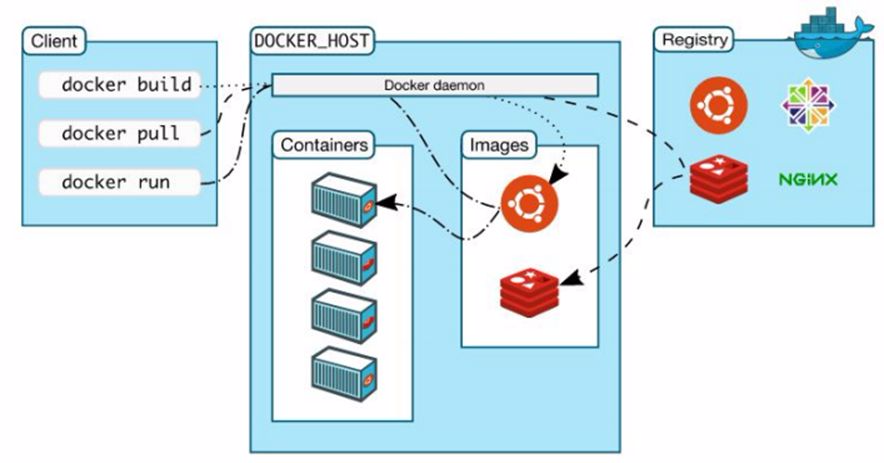
uname命令用于打印当前系统相关信息（内核版本号、硬件架构、主机名称和操作系统类型等）。

uname -r

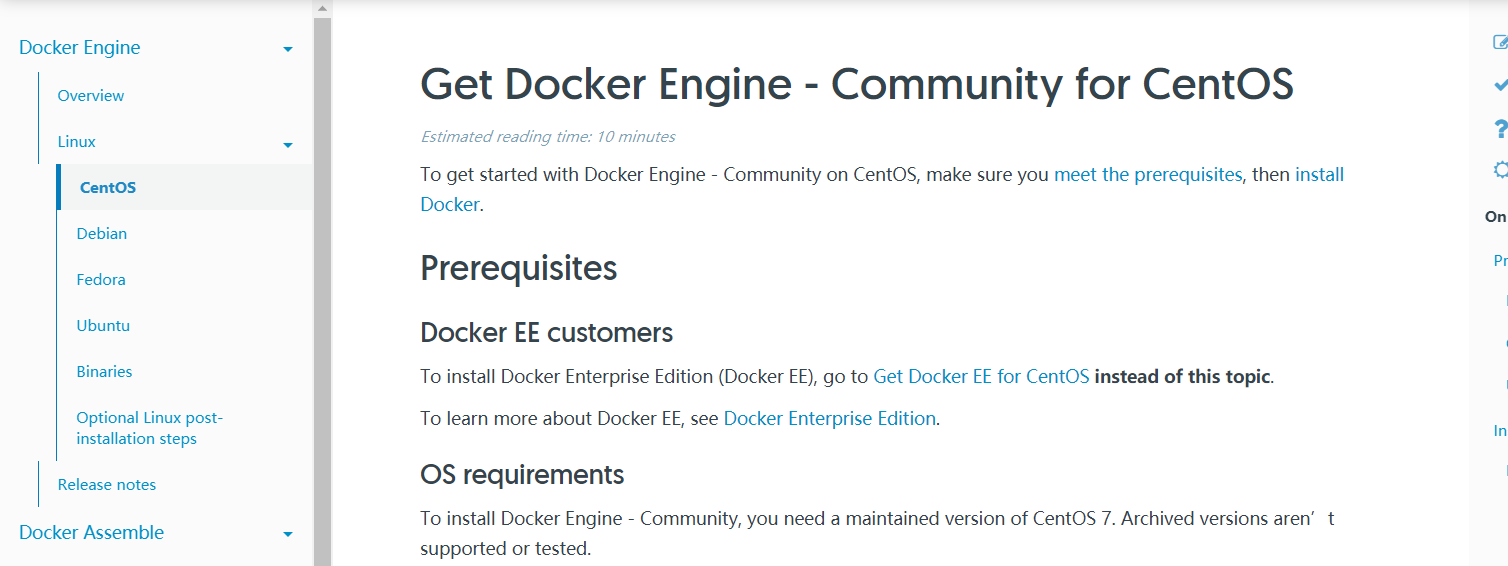


版本OK木有问题

## Docker 的基本组成



# 【掌握】docker安装

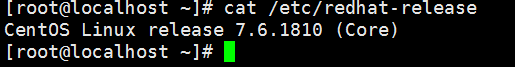


## centos 下安装Docker 文档

https://docs.docker.com/install/linux/docker-ce/centos/

## Docker 安装

### 确定是centos7的版本



### yum安装gcc相关

1，确定centos7能上外网

2，yum -y install gcc

3，yum -y install gcc-c++

### 卸载旧版本

 yum remove docker \

docker-client \

docker-client-latest \

docker-common \

docker-latest \

docker-latest-logrotate \

docker-logrotate \

docker-engine

### 安装方法

 yum install docker

### 启动docker

systemctl start docker

systemctl enable docker 开机自启

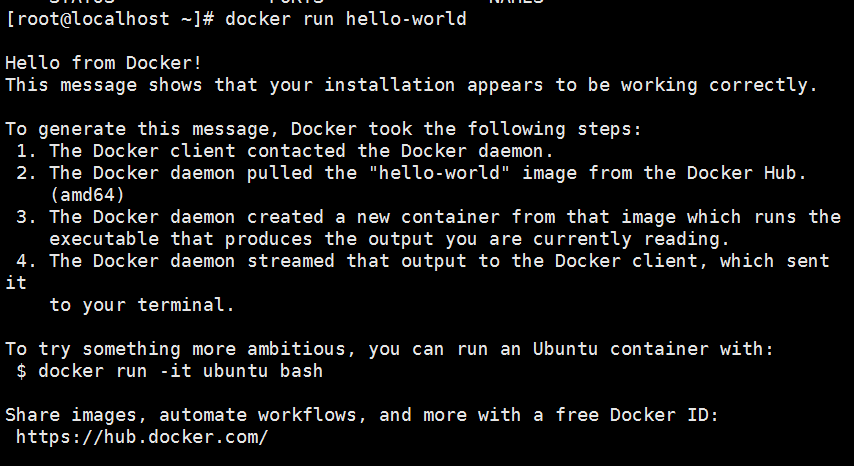
### 查看版本

docker version

### 测试运行 hello-world

 docker run hello-world

由于本地没有hello-world这个镜像，所以会下载一个hello-world的镜像，并在容器内运行。



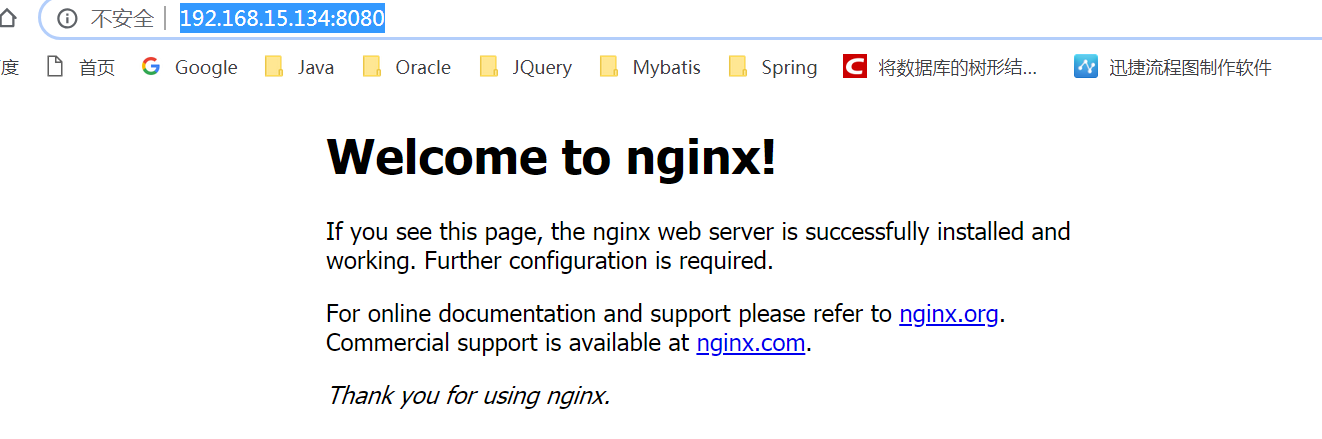
### 测试运行nginx

docker run -p 8080:80 -d docker.io/nginx //将80端口映射为8080



测试

[http://IP:8080/](http://192.168.15.134:8080/)



是不是很爽呀

### 卸载docker

systemctl stop docker

yum -y remover docker

rm -rf /var/lib/docker

# 【掌握】docker加速配置

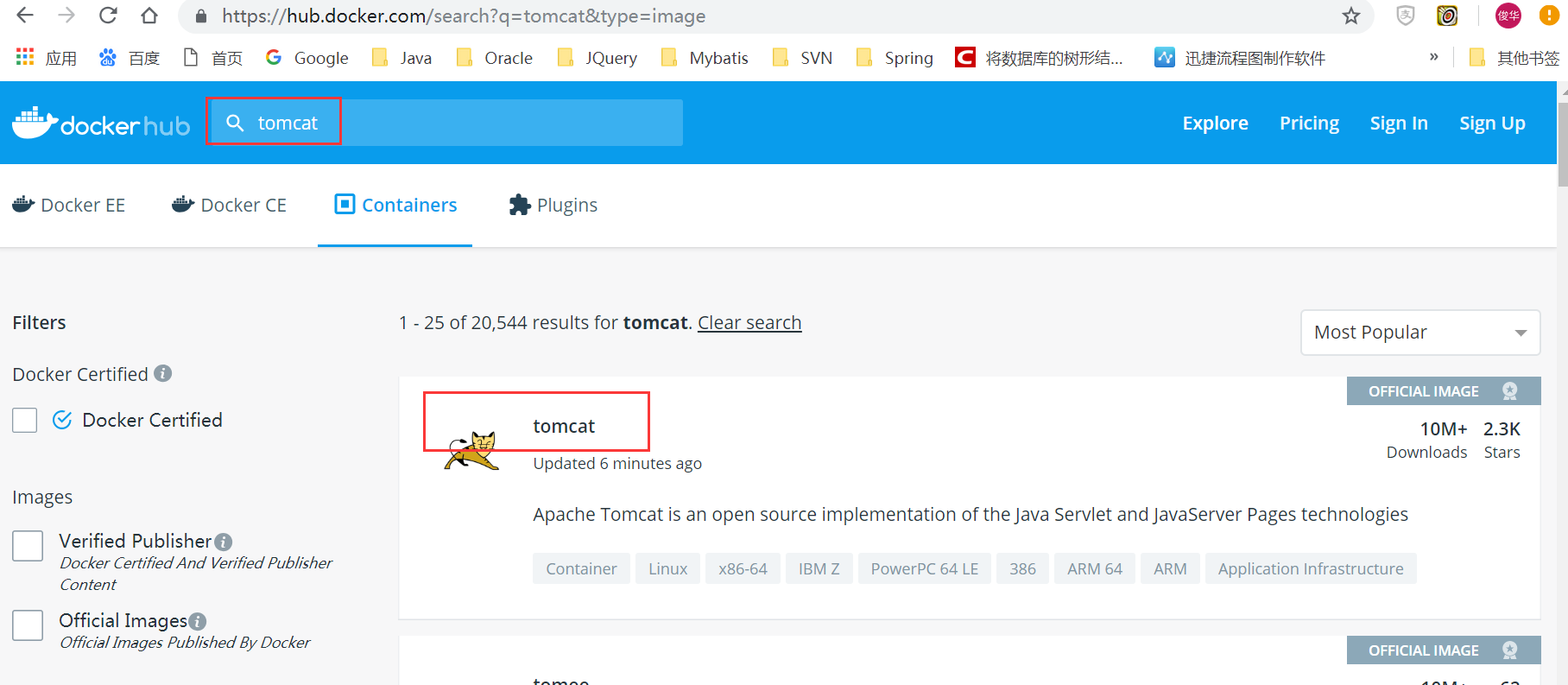
因为直接从dockerhub上下载镜像很慢

## 说明

Docker 运行容器前需要本地存在对应的镜像，如果镜像不存在本地，Docker 会从镜像仓库下载（默认是Docker Hub 公共注册服务器中的仓库）。

## 获取镜像

通过网址可以找到目标镜像 <https://hub.docker.com/explore/>

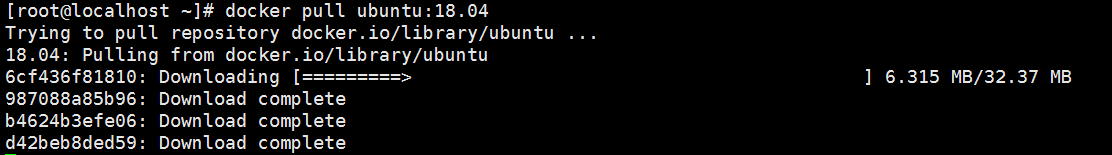


可以使用docker pull 命令来从仓库获取所需要的镜像。

下面的例子将从Docker Hub 仓库下载一个Ubuntu 18.04 操作系统的镜像。

docker pull ubuntu:18.04

该命令实际上相当于sudo docker pull registry.hub.docker.com/ubuntu:18.04

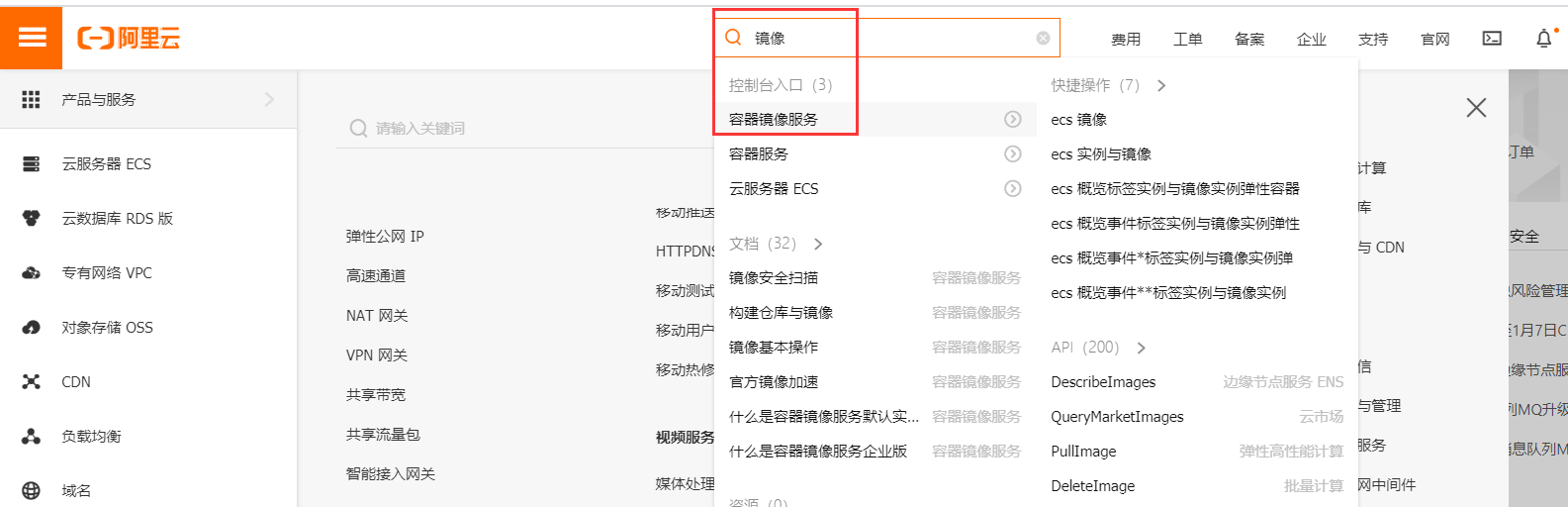


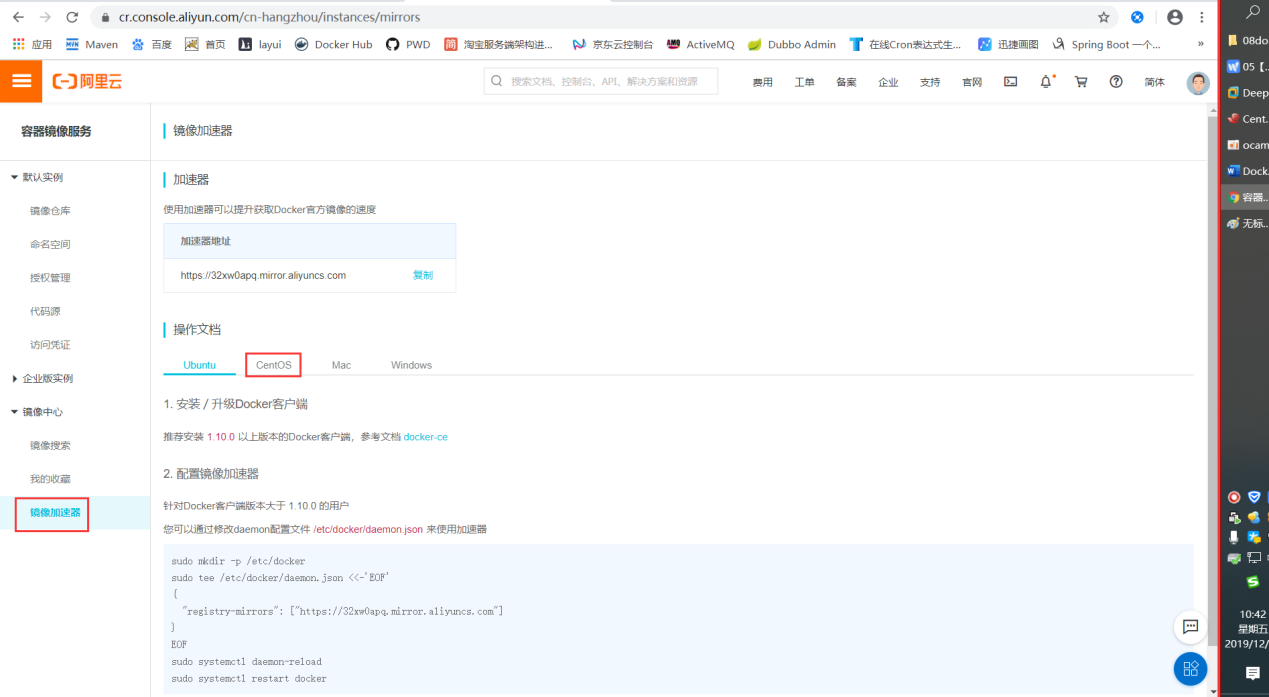
从上面可以看出下载的很慢，因为docker默认是从docker Hub上下载的，而这个镜像地址在国外，所有很慢，我们可要其它镜像地址，如国内的

sudo docker pull registry.docker-cn.com/library/ubuntu:18.04

这样下载就很比刚才快很多了哦

## 阿里云镜像加速





vim /etc/docker/daemon.json

写入

{

"registry-mirrors": ["https://32xw0apq.mirror.aliyuncs.com"]

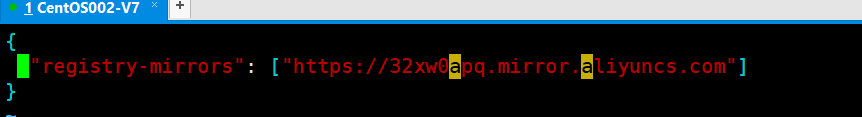
}

刷新配置

systemctl daemon-reload

重启docker

systemctl restart docker





## ~~其它镜像加速~~

文档[https://www.daocloud.io/mirror#accelerator-doc](https://www.daocloud.io/mirror" \l "accelerator-doc)

 http://f1361db2.m.daocloud.io



## 相关命令

### 列出所有镜像

docker images

### 根据镜像id删除镜像

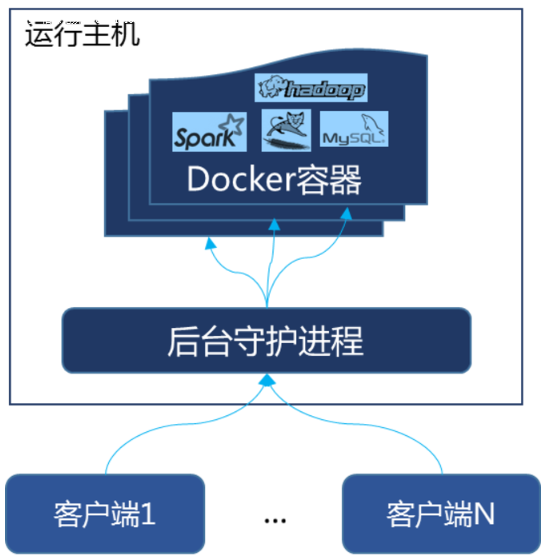
docker rmi 镜像id

docker rmi -f 镜像id

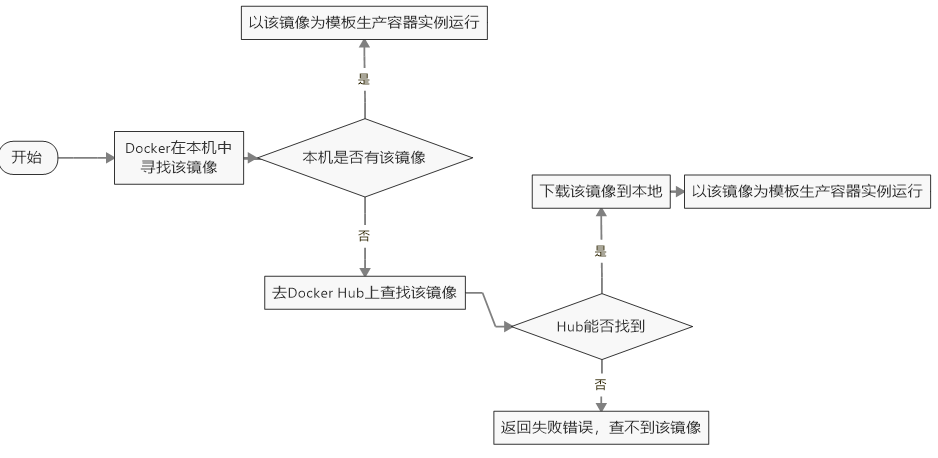
# 【掌握】docker底层原理

## Docker是如何工作的

Docker是一个Client-Server结构的系统，Docker**守护进程**运行在主机上， 然后通过Socket连接从客户端访问，**守护进程**从客户端接受命令并管理运行在主机上的容器。 容器，是一个运行时环境，就是我们前面说到的集装箱。



## docker run了啥



# 【掌握】docker命令[帮助+镜像]

## 帮助命令

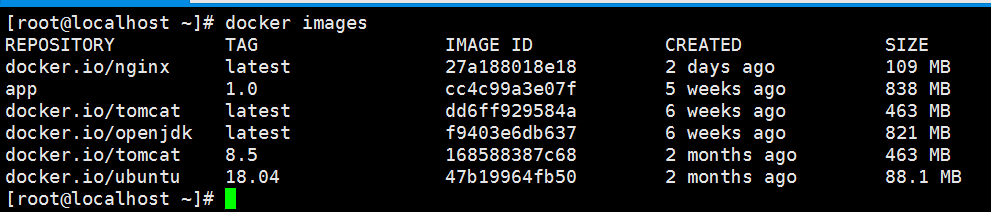
docker version 查看docker版本

docker info 显示全系统信息

docker --help 显示docker相关的所有命令及功能说明

## 镜像命令

### docker images  列表本机上的镜像



REPOSITORY：表示镜像的仓库源

TAG：镜像的标签

IMAGE ID：镜像ID

CREATED：镜像创建时间

SIZE：镜像大小

同一仓库源可以有多个 TAG，代表这个仓库源的不同个版本，我们使用 REPOSITORY:TAG 来定义不同的镜像。

如果你不指定一个镜像的版本标签，例如你只使用 ubuntu，docker 将默认使用 ubuntu:latest 镜像

docker images -[options]的说明

-a 列表本地的所有镜像及子镜像

-q 只显示镜像ID

--digests 显示镜像的摘要信息

--no-trunc 显示完整的镜像信息

​

### docker seach 镜像搜索命令

搜索网站:<https://hub.docker.com>

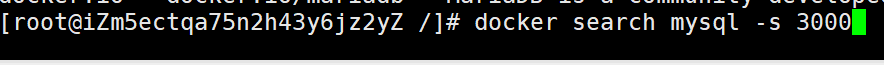
语法

docker search 镜像名称

docker search 镜像名称  -[options]   说明

--no-trunc 显示完整的镜像描述

-s 列出收藏数不少于指定值的镜像



### docker pull 镜像下载命令

语法

docker pull 镜像名称:[TAG]

例如:docker pull tomcat:8.5 下载8.5的镜像版本

dokcer pull tomcat 默认下载最新的tomcat镜像版本 【latest】

### docker rmi 镜像删除命令

删除单个 docker rmi -f 镜像ID

删除多个 docker rmi -f 镜像ID1 镜像ID2

docker rim -f 镜像名:[tag] 镜像名:[tag]

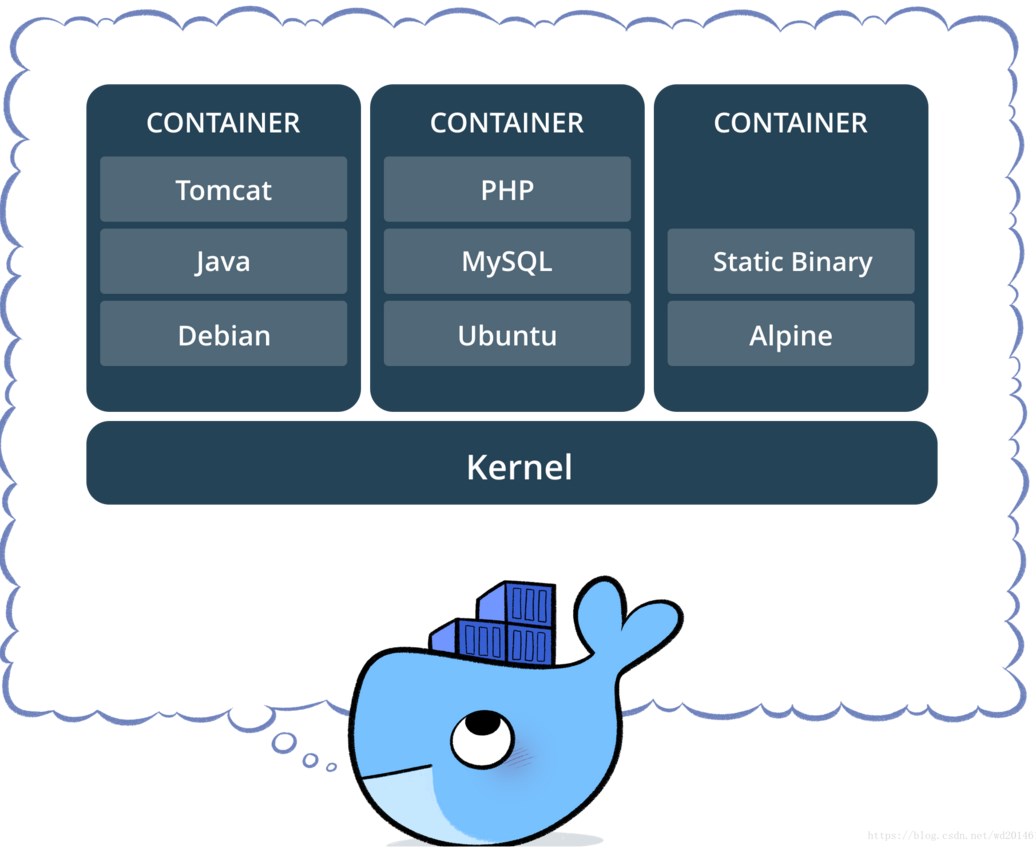
删除全部

docker rmi -f $(docker images -qa)

# 【掌握】docker命令[容器]

## 什么是容器

 容器镜像是一个软件的轻量级独立可执行软件包，包含运行它所需的一切：代码，运行时，系统工具，系统库，设置。不管环境如何，集装箱化软件都可以运行相同的Linux和Windows应用程序。容器将软件与其周围环境隔离开来，例如开发环境和生产环境之间的差异，并有助于减少在同一基础架构上运行不同软件的团队之间的冲突。



1.轻量级

在一台机器上运行的Docker容器共享该宿主机器的操作系统内核; 他们立即开始并使用更少的计算和内存。图像由文件系统层构建并共享公用文件。这最大限度地减少了磁盘使用量，图像下载速度更快。

2.标准

Docker容器基于开放标准，可在所有主要Linux发行版，Microsoft Windows以及任何基础架构（包括虚拟机，裸机和云中）上运行。

3.安全

Docker容器将应用程序彼此隔离并从底层基础架构中分离出来。Docker提供了最强大的默认隔离功能，可以将应用程序问题限制在一个容器中，而不是整个机器上。

## 演示前提

    有镜像才能创建容器，这个是根本要求(下载centos镜像来演示)

docker pull centos

## 创建并启动容器

### 语法

docker run [options] 镜像名:tag/镜像id [命令or 参数]

### options说明

**有些是一个减号，有些是两个减号**

--name="容器新名字": 为容器指定一个名称；

-d: 后台运行容器，并返回容器ID，也即启动守护式容器；

-i：以交互模式运行容器，通常与 -t 同时使用；

-t：为容器重新分配一个伪输入终端，通常与 -i 同时使用；

-P: 随机端口映射；

-p: 指定端口映射，有以下四种格式

      ip:hostPort:containerPort

      ip::containerPort

      hostPort:containerPort

      containerPort

### 交互式运行

#使用镜像centos:latest以交互模式启动一个容器,在容器内执行/bin/bash命令。

docker run -it centos /bin/bash

## 列出当前所有正在运行的容器

### 语法

|  |
| --- |
| docker ps –[options] |

### 语法【options】

-a 列出所有运行和未运行的容器

-q 列出所有运行中容器的ID

-aq 列出所有运行和未运行的容器的ID

## 删除容器

|  |
| --- |
| docker rm [options] 容器ID|容器名称 容器ID|容器名称 容器ID|容器名称 |

Options

-f 强制删除正在运行的容器或没有运行的容器

docker rm -f $(docker ps -aq) 删除所有运行和未运行的容器

## 退出容器

1. exit:停止容器并退出
2. ctrl+P+Q  容器不停止退出

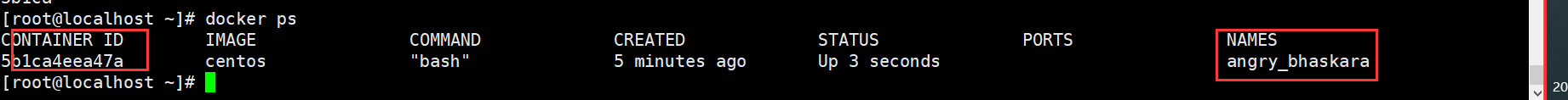
## 进入正在进行的容器并以命令行交互

docker exec -it 容器ID bash 重新开启一个终端进入容器

docker attach 容器ID 直接使用原来的终端进入容器

## 启动停止重启容器

docker start|stop|restart 容器ID或容器名



## 强制停止容器

docker kill 容器ID或容器名称

## 启用守护式容器

#使用镜像centos:latest以后台模式启动一个容器

docker run -d centos

问题：docker ps -a 进行查看, 会发现容器已经退出

很重要的要说明的一点: Docker容器后台运行,就必须有一个前台进程.

容器运行的命令如果不是那些一直挂起的命令（比如运行top，tail），就是会自动退出的。

这个是docker的机制问题,比如你的web容器,我们以nginx为例，正常情况下,我们配置启动服务只需要启动是应的service即可。例如  service nginx start

但是,这样做,nginx为后台进程模式运行,就导致docker前台没有运行的应用,

这样的容器后台启动后,会立即自杀因为他觉得他没事可做了.

所以，最佳的解决方案是,将你要运行的程序以前台进程的形式运行

## 查看容器日志

**语法(docker logs 容器ID|名称)**

如果是后台运行的程序，那么如何查看运行的日志呢？

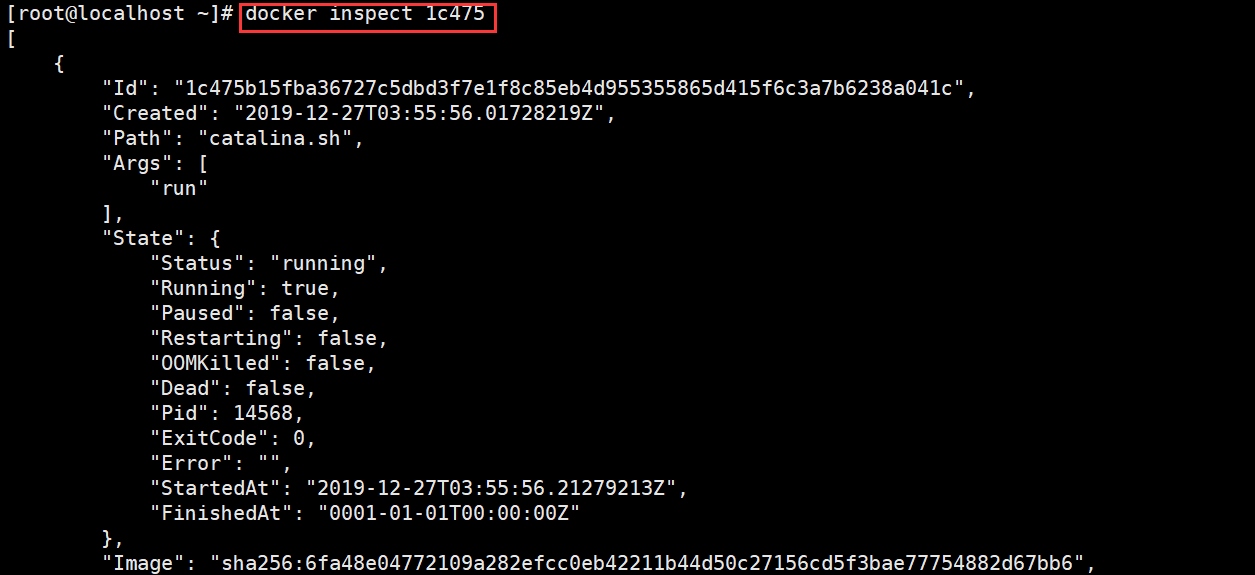
## 查看容器运行进程

docker top 容器ID

## 查询容器内部细节

docker inspect 容器ID|名称

显示了容器里面所有内部细节



## 进入容器拷贝文件到主机

语法

docker cp 容器ID(容器名称):容器内文件或文件夹路径 宿主机的路径



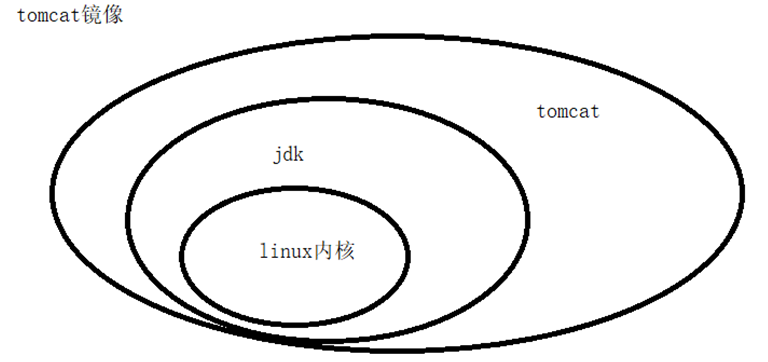
## 在主机拷贝文件到容器

docker cp 宿主机的路径 容器ID(容器名称):容器内文件或文件夹路径



# 【掌握】docker镜像详解

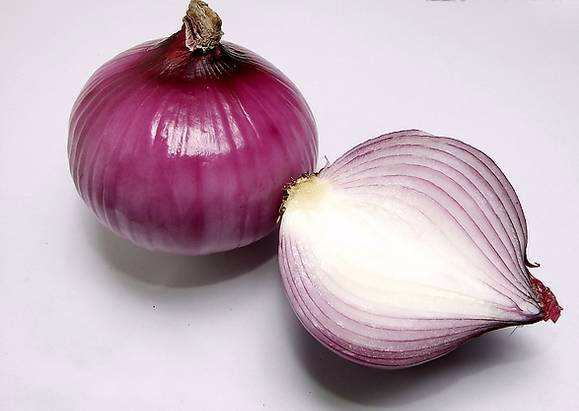
## 为什么下载的tomcat镜像很大



## 什么是镜像

### UnionFS（联合文件系统）

UnionFS（联合文件系统）：Union文件系统（UnionFS）是一种分层、轻量级并且高性能的文件系统，它支持对文件系统的修改作为一次提交来一层层的叠加，同时可以将不同目录挂载到同一个虚拟文件系统下(unite several directories into a single virtual filesystem)。Union 文件系统是 Docker 镜像的基础。镜像可以通过分层来进行继承，基于基础镜像（没有父镜像），可以制作各种具体的应用镜像。



特性：一次同时加载多个文件系统，但从外面看起来，只能看到一个文件系统，联合加载会把各层文件系统叠加起来，这样最终的文件系统会包含所有底层的文件和目录

### 分层的镜像

以我们的pull为例，在下载的过程中我们可以看到docker的镜像好像是在一层一层的在下载



### 为什么docker要使用上面的这种分层结构

最大的一个好处就是 - 共享资源

比如：有多个镜像都从相同的 base 镜像构建而来，那么宿主机只需在磁盘上保存一份base镜像，

同时内存中也只需加载一份 base 镜像，就可以为所有容器服务了。而且镜像的每一层都可以被共享。

## 镜像的特点

Docker镜像都是只读的

当容器启动时，一个新的可写的镜像被加载到镜像层的顶部。

这一层通常被叫做容器层，容器层之下的都叫镜像层

## 镜像的commit操作

### 作用

当镜像运行之后可以修改容器里面的内容，再提交成一个新的镜像

### 命令语法

**docker commit -m='新的镜像的描述信息' -a='作者' 容器ID 要创建的目标镜像名:[标签名]**

### 案例演示

#### 从hub上拉一下tomcat镜像运行

 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

运行后发现webapps里面没有那个5个默认项目

修改的思路

1，解压本的地tomcat

1. 运行tomcat的镜像成容器
2. 把本地的tomcat的Webapps里面的所有文件夹cp 到 运行的tomcat的容器里面

docker cp ./tomcat/webapps/ROOT 45ec37:/usr/local/tomcat/webapps/

docker cp ./tomcat/webapps/docs/ 45ec37:/usr/local/tomcat/webapps/

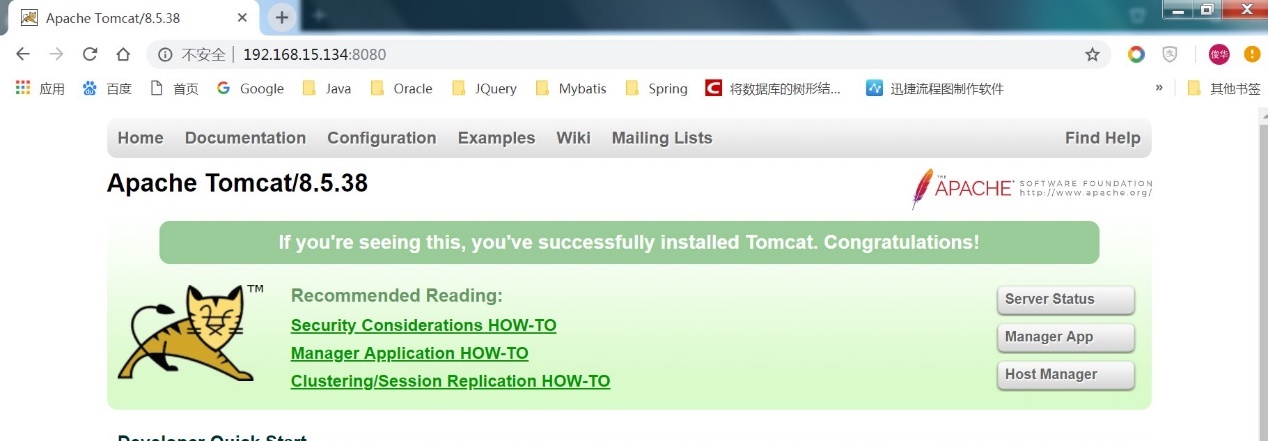
docker cp ./tomcat/webapps/manager 45ec37:/usr/local/tomcat/webapps/

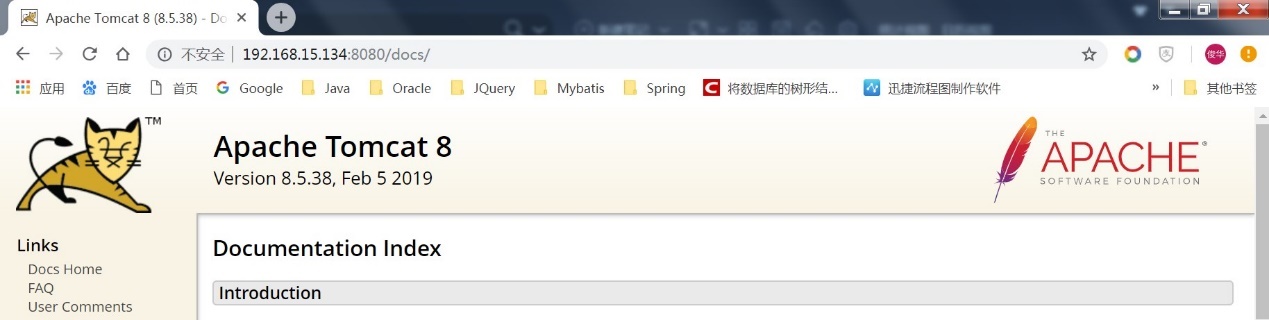
docker cp ./tomcat/webapps/host-manager 45ec37:/usr/local/tomcat/webapps/

docker cp ./tomcat/webapps/examples 45ec37:/usr/local/tomcat/webapps/

1. 重启tomcat的容器
   1. docker restart 容器ID或名称
2. 访问
3. 把新修改的容器提交成一个新的镜像
   1. docker commit -a='leige' -m='有5个项目的tomcat' 45ec37bbf021 mytomcat:9.0
4. 运行这个新的镜像
   1. docker run -it -d -p 83:8080 mytomcat:9.0





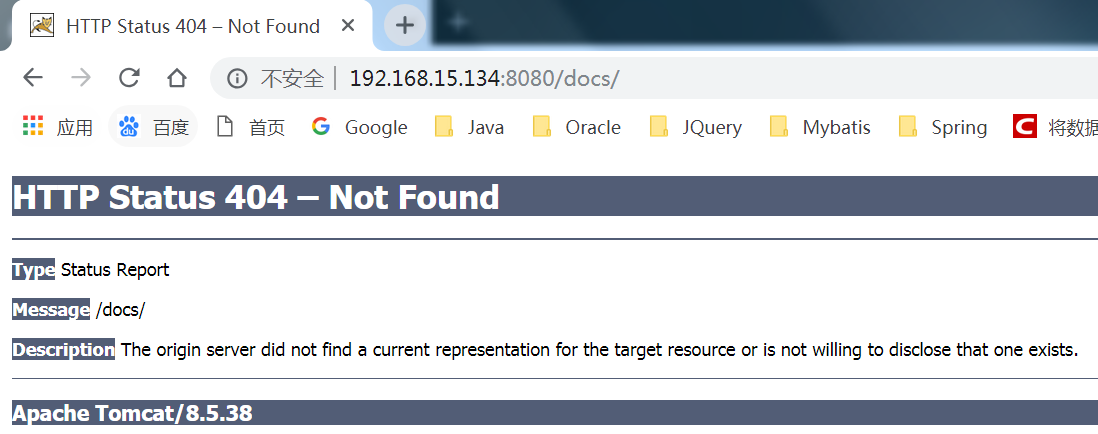


从上面可以看出docs是可以访问的哦

#### 删除tomcat里面webapps 里面的docs项目



刷新页面docs不存在了哦



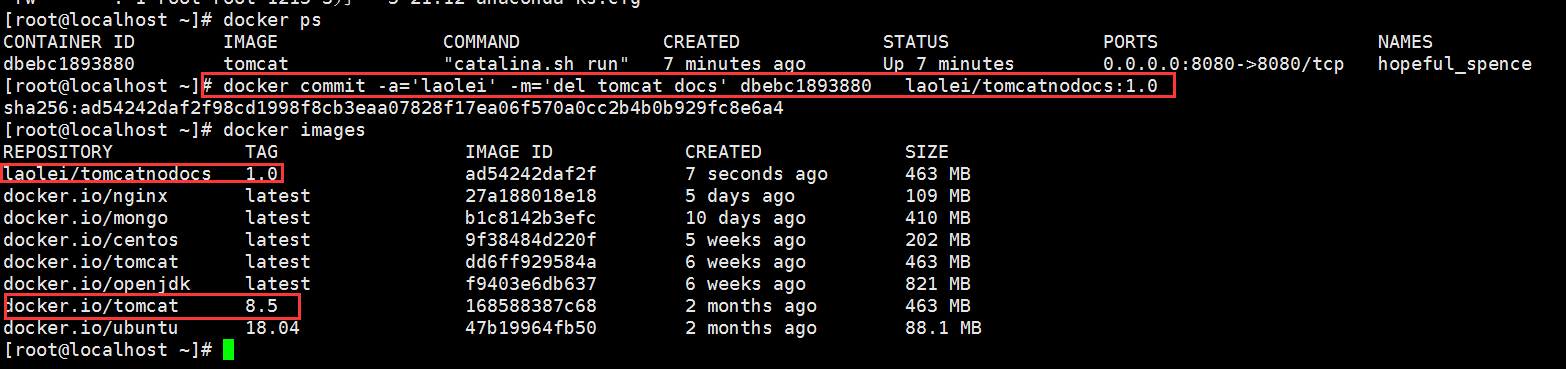
#### 把当前运行的这个没有docs的容器生成一个新的镜像

语法

docker commit -a='作者' -m='镜像描述' 容器ID 新的镜像名/名称:版本

案列

docker commit -a='laolei' -m='del tomcat docs' dbebc1893880 laolei/tomcatnodocs:1.0



#### 启动自己创建的镜像和之前的对比

1 ，删除所有容器

 docker rm -f $(docker ps -aq)

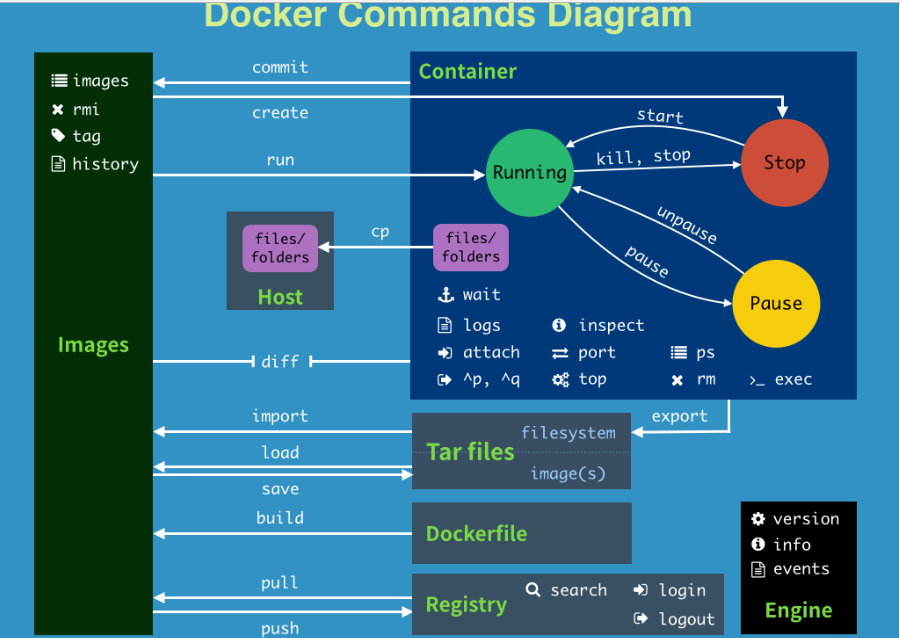
2，启动之前的镜像

 docker run -d -p 8888:8080 镜像ID或仓库ID+版本

3，启动自己的镜像

 docker run -d -p 9999:8080 镜像ID

# 【掌握】docker命令总结



# 【掌握】docker容器数据卷

## 什么是容器数据卷

先来看看Docker的理念：

\* 将应用与运行的环境打包形成容器运行 ，运行可以伴随着容器，但是我们对数据的要求希望是持久化的

\* 容器之间希望有可能共享数据

Docker容器产生的数据，如果不通过docker commit生成新的镜像，使得数据做为镜像的一部分保存下来，

那么当容器删除后，数据自然也就没有了。

为了能保存数据在docker中我们使用卷。

## 容器数据卷能做什么

1，容器数据的持久化

2，容器之间**继承**和共享数据

## 添加数据卷的方式

### 直接使用命令添加

①语法

  docker run -it -v /宿主机目录:/容器内目录 centos /bin/bash

②查看容器卷是否挂载成功

 docker inspect 容器ID



③查看容器卷和宿主机的数据共享

在宿主机的mycentos001中创建hello.txt文件并写入数据mycentos001

进入容器查看container001里面有hello.txt文件数据和mycentos001里面的一样

注意：在以上的例子中，默认的只能在宿主机里面写数据。

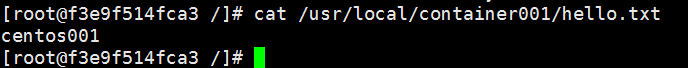
如果出以下的问题



解决办法：在挂载目录后多加一个--privileged=true参数即可



④查看容器停止后，主机修改数据是否同步



⑤带权限的处理方式

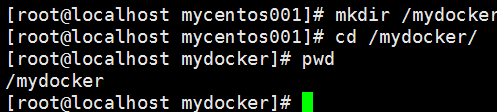


docker run -it -v /宿主机目录:/容器内目录 centos /bin/bash

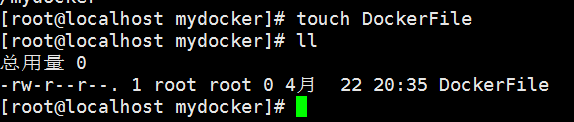
bind

### 使用DockerFile添加

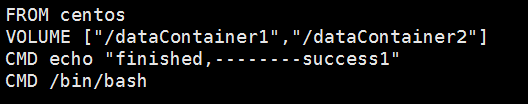
①在宿主机的根目录下创建mydocker文件夹并进入



②在当前目录创建一个Dockerfile的文件



③编写Dockerfile



FROM centos

VOLUME ["/dataContainer1","/dataContainer2"]

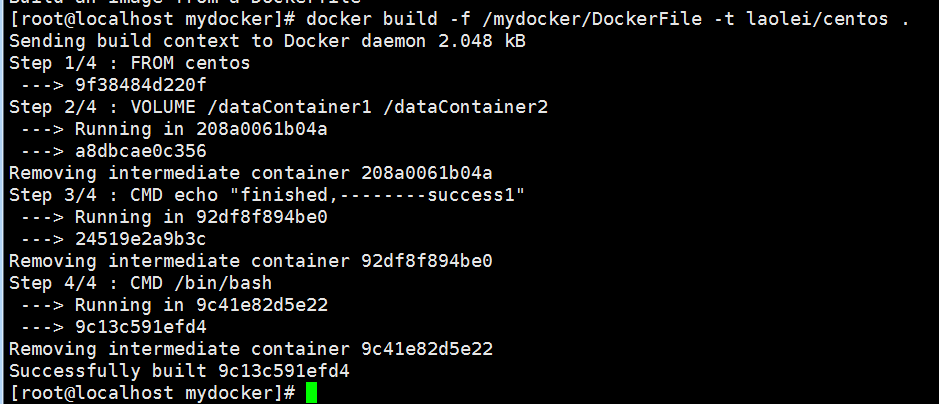
CMD echo "finished,--------success1"

CMD /bin/bash

④build生成一个新的镜像

docker build ~~-f /mydocker/Dockerfile~~ -t laolei/centos .

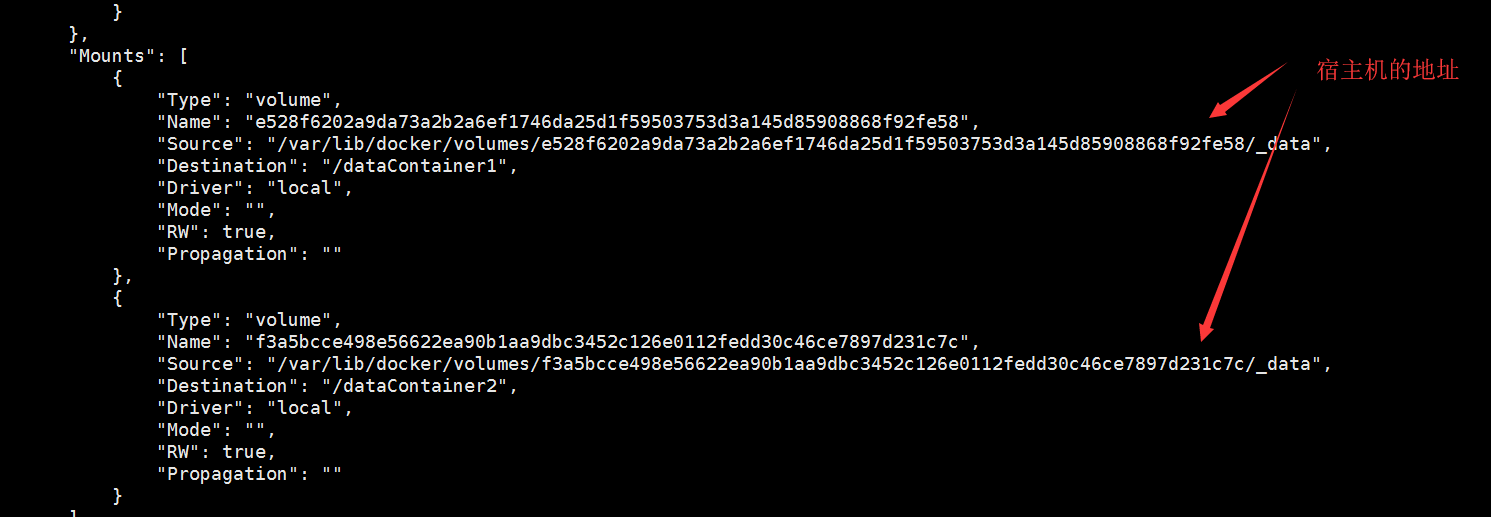
#注意后面有一个点哦

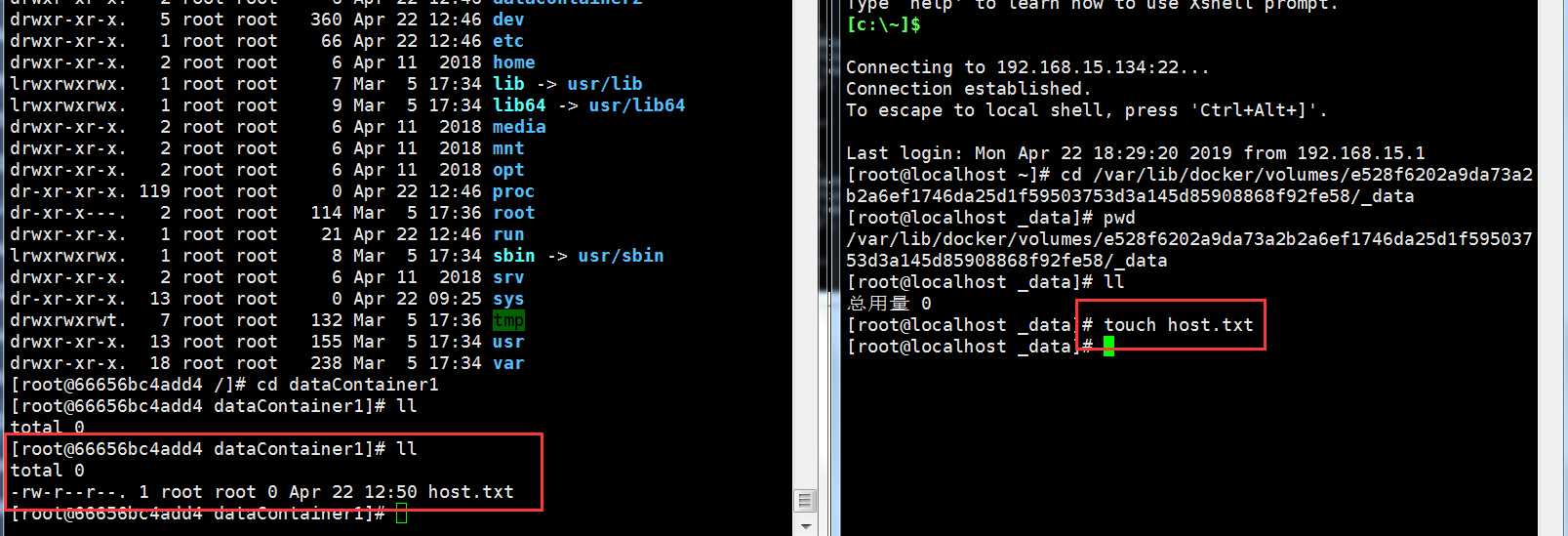


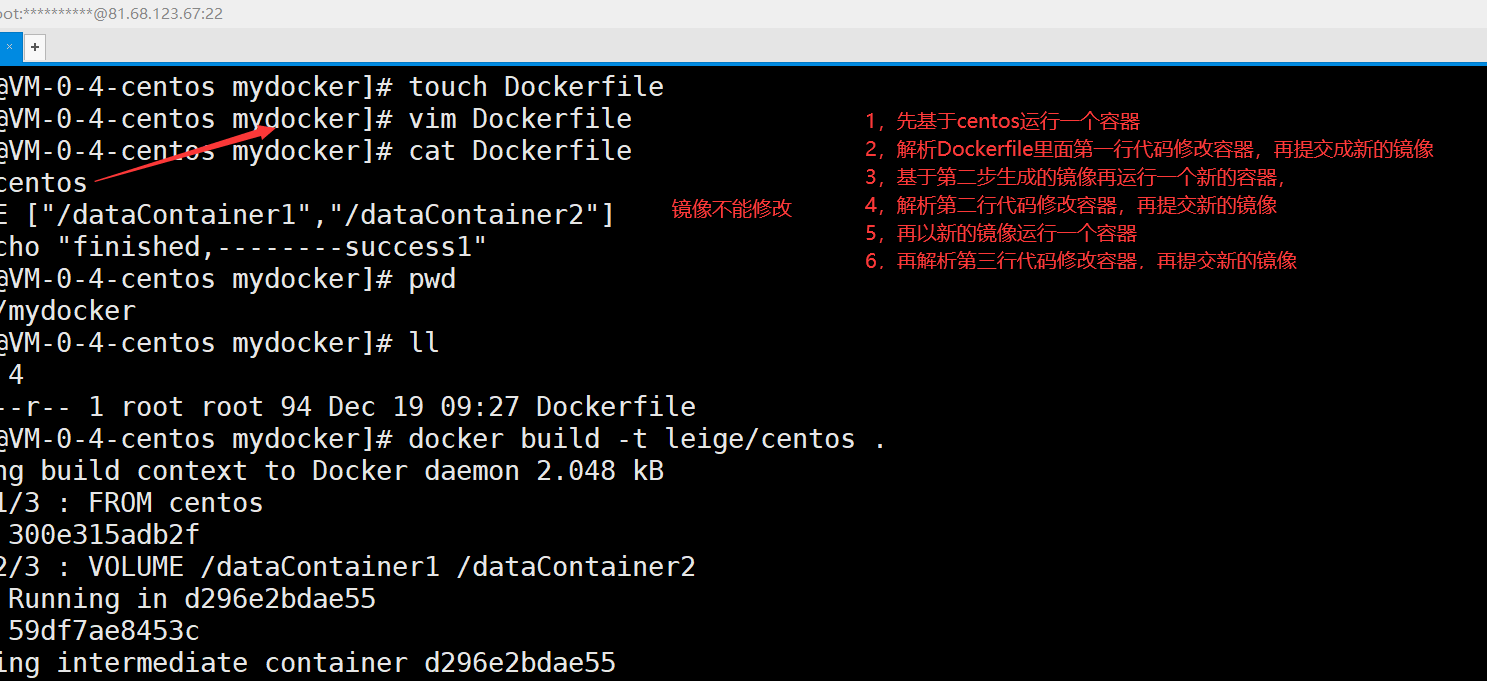
⑤启动容器

docker run -it --name='mycentos' laolei/centos

⑥查看容器里面有两个容器卷  
docker inspect 容器ID







## 数据卷容器

docker run -d -v /root/webapps:/usr/local/tomcat/webapps -p 8001:8080 tomcat

4.1，作用：实现容器之间的数据共享

4.2，操作思路

以上面的laolei/centos为镜像，里面有dataContainer1和dataContailer2

启动一个容器dc1 在dataContailer1里面添加dc1.txt

docker run -it --name='dc1' laolei/centos

cd /dataContailer1

touch dc1.txt

启动一个容器dc2 继承dc1 在dataContailer1里面添加dc2.txt

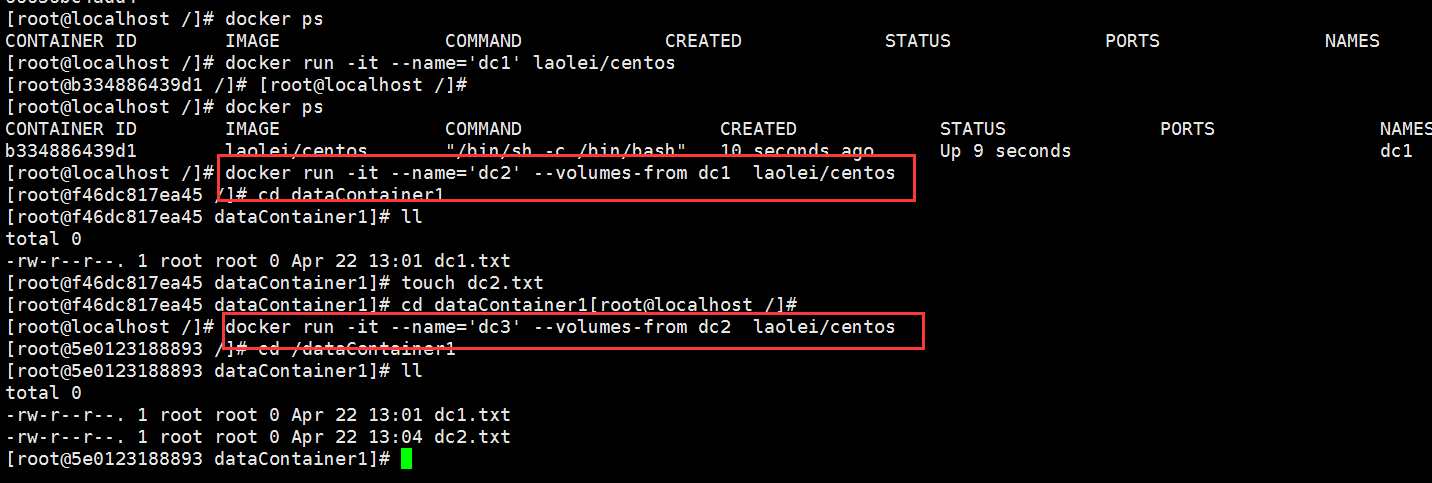
启动一个容器dc3 继承dc2 在dataContailer1里面添加dc3.txt

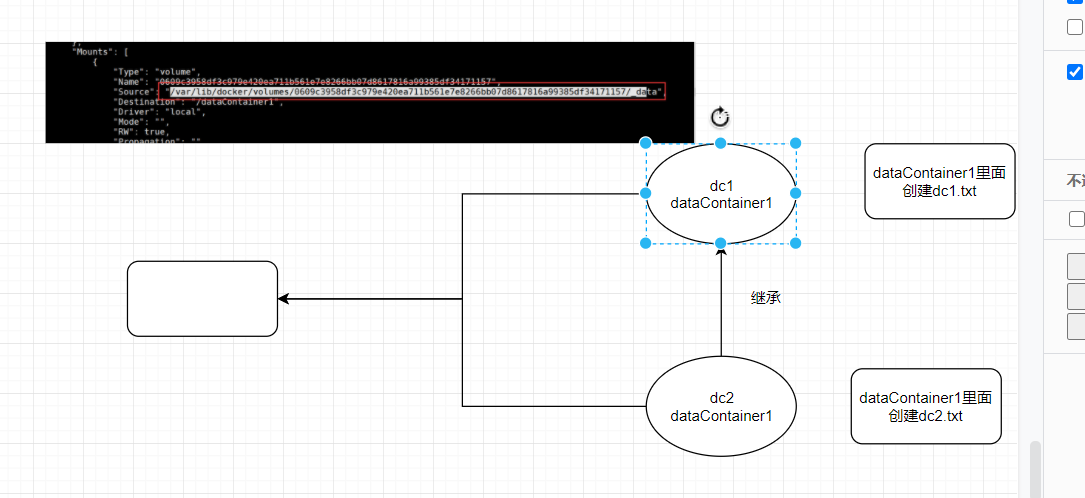
发现在dc3可以看到dc1.txt dc2.txt dc3.txt

dc1里面可以看到dc1.txt dc2.txt dc3.txt

删除dc1容器之后在dc2和dc3里面还是可以看到dc1.txt dc2.txt dc3.txt

说明数据卷容器的生命周期一直持续到没有容器使用它为止





# 【掌握】Dockerfile详解【1】

## 什么是Dockerfile -docker build

1，Dockerfile是用来构建Docker镜像的构建文件，是由一系列的命令和参数构成的脚本

2，Dokcerfile的构建步骤

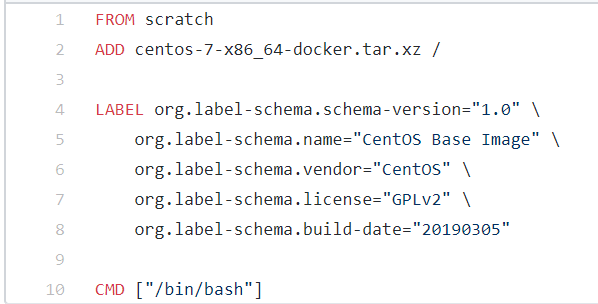
        编写Dokcerfile文件

        docker build 生成新的镜像

        docker run 运行镜像

3，以centos的镜像为例来说明 <https://hub.docker.com/_/centos>





## DockerFile构建过程解析

### 基础知识

1，每条保留字指令都必须为大写字母后面要跟随至少一个参数

2，指令从上到下顺序执行

3，#表示注释

4，每条指令都会创建一个新的镜像层，并对镜像进行提交

### 大致流程

1，docker从基础镜像运行一个容器

2，执行一条指令并对容器进行修改

3，执行类似于docker commit的操作提交一个新的镜像

4，docker再基于刚提交的新的镜像运行一个新的容器

5，执行Dockerfile的下一个指令再从执行第2点直到没有指令

### 总结

 从应用软件的角度来看，Dockerfile、Docker镜像与Docker容器分别代表软件的三个不同阶段，

\* Dockerfile是软件的原材料

\* Docker镜像是软件的交付品

\* Docker容器则可以认为是软件的运行态。

Dockerfile面向开发，Docker镜像成为交付标准，Docker容器则涉及部署与运维，三者缺一不可，合力充当Docker体系的基石。

1 Dockerfile，需要定义一个Dockerfile，Dockerfile定义了进程需要的一切东西。

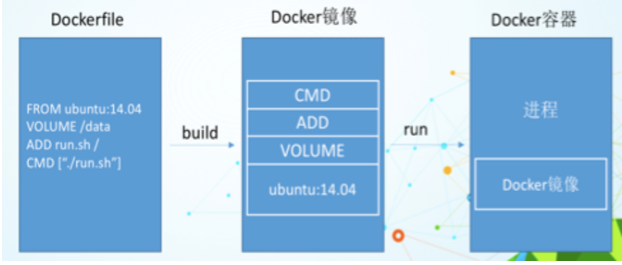
Dockerfile涉及的内容包括执行代码或者是文件、环境变量、依赖包、运行时环境、动态链接库、操作系统的发行版、

服务进程和内核进程(当应用进程需要和系统服务和内核进程打交道，这时需要考虑如何设计namespace的权限控制)等等;

2 Docker镜像，在用Dockerfile定义一个文件之后，docker build时会产生一个Docker镜像，

当运行 Docker镜像时，会真正开始提供服务;

3 Docker容器，容器是直接提供服务的。



## Dockerfile体系结构（关键字---重点啊）

**FROM** 基础镜像，当前新镜像是基于哪个镜像的

**MAINTAINER**  镜像维护者的姓名和邮箱地址

**RUN**  容器构建时需要运行的linux命令

**EXPOSE** 当前容器对外暴露的端口[**只是提示作用**，]

**WORKDIR** 指定在创建容器后，终端默认登陆进来的工作目录

**ENV** 用来在构建镜像过程中设置环境变量

**ADD** 将宿主机目录下的文件拷贝进镜像并且ADD命令会自动处理URL和解压tar包

**COPY** 类似ADD，拷贝文件和目录到镜像中 ，语法COPY src dest  COPY [''src","dest"]

**VOLUME** 容器数据卷，用于数据保存和持久化工作

**CMD** 指定一个容器启动时要运行的命令格式CMD['ls']   run -it cenos -lh

        shell:  CMD  <命令>

        exec  CMD ['可执行文件',"参数1","参数2"]

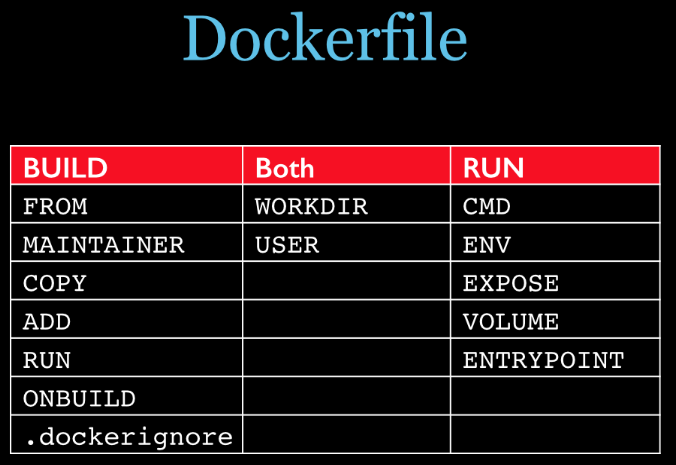
        DockerFile中可以有多个CMD指令，但只有最后一个生效，CMD会被docker run之后的参数替换

**ENTEYPONT**  指定一个容器启动时要运行的命令 ENTEYPORT ['ls']   run -it cenos -lh

            ENTRYPOINT的目地和CMD一样，都是在指定容器启动程序及参数

**~~OBBUILD~~** ~~当构建一个被继承的Dockerfile时运行命令，父镜像在被子镜像继承后触发父镜像的onbuild~~

### 总结



# 【掌握】Dockerfile详解【2】

## 制作一个带vim centos镜像

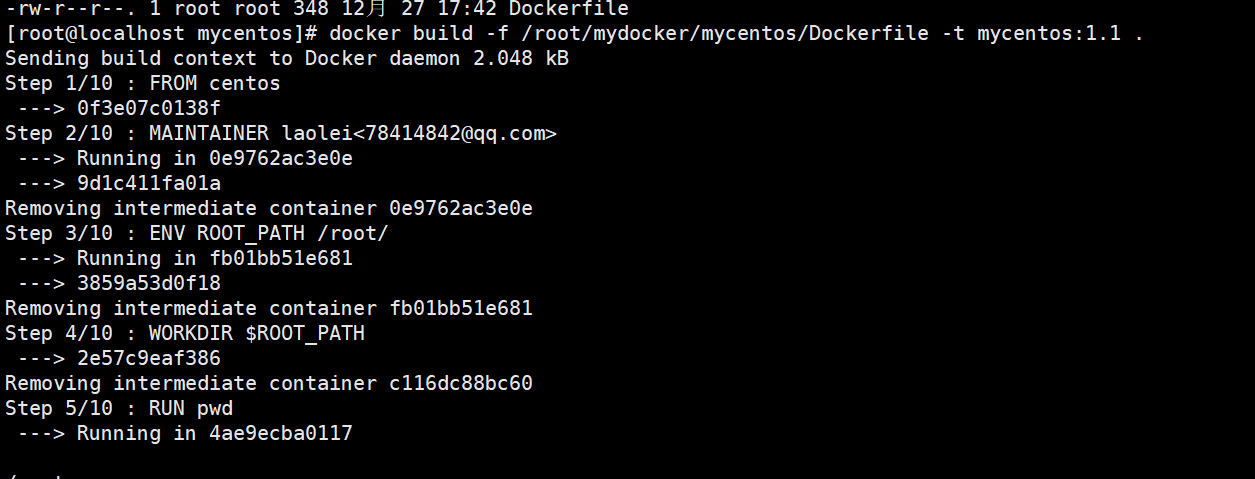
Vim

指定工作目录

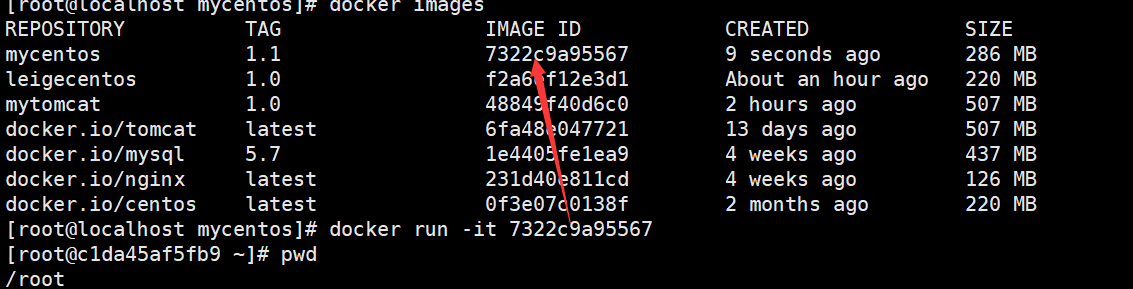
## 编写Dockerfile

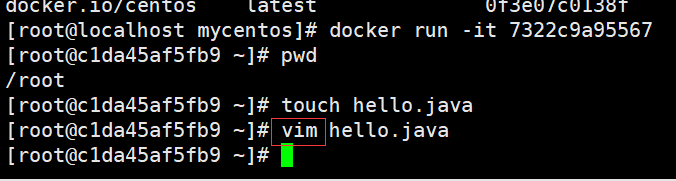


## 使用build+Dockerfile构建镜像



## 运行





# 【掌握】Dockerfile详解【3】

## 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

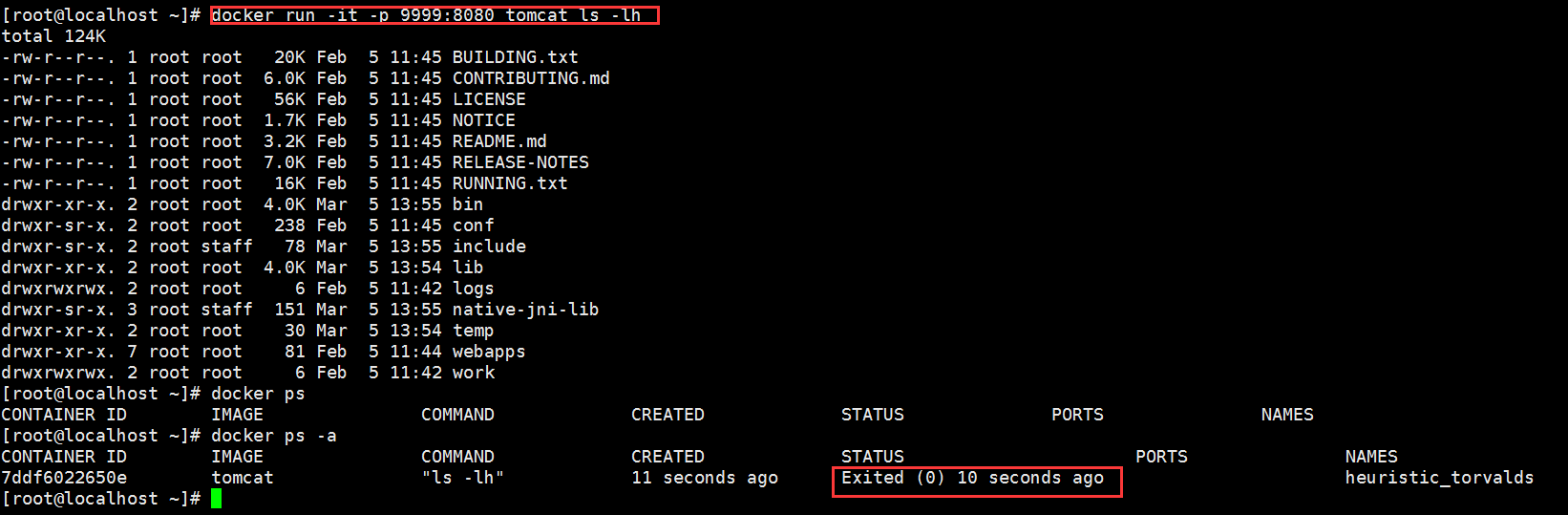
可以使用tomcat的案例演示

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"]

## 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

## 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

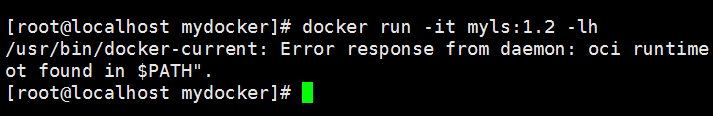
CMD [ "ls", "-a" ]

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

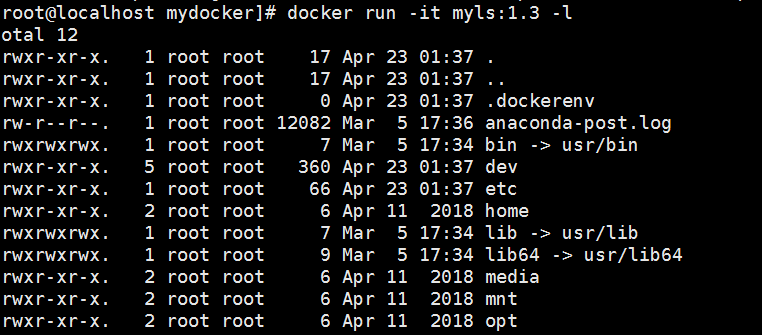
ENTRYPOINT [ "ls", "-a" ]

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

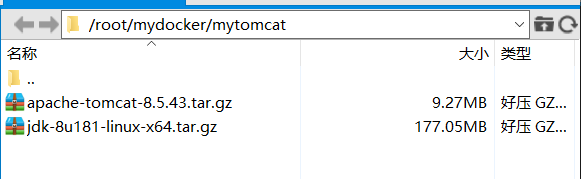
# 【掌握】Dockerfile详解【4】

## 自定义tomcat镜像

### 准备工作

下载tomcat8

下载jdk8linux



#创建目录

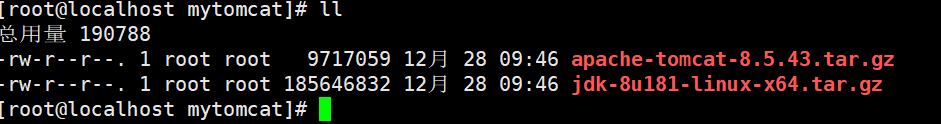
mkdir -p /mydocker/mytomcat

#进入目录

cd /mydocker/ mytomcat

#创建文件

#把下载好的tomcat和jdk放到/mydocker/mytomcat里面



### 编写Dockerfile

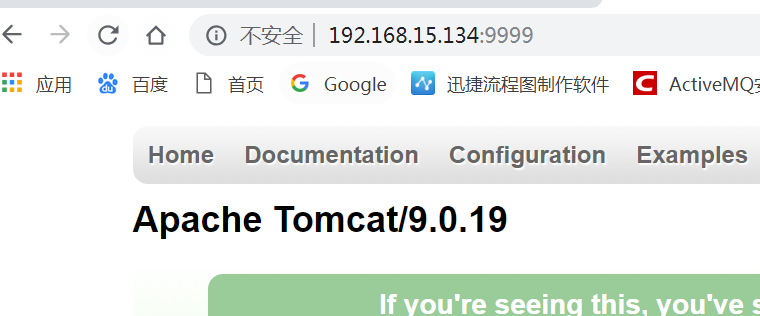
|  |
| --- |
|  |
|  |

### 构建镜像

  docker build -t mytomcat9:1.0 .

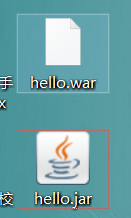
### 运行镜像测试

docker run -it --name 'mytomcat-1' -p 9999:8080 mytomcat:1.0



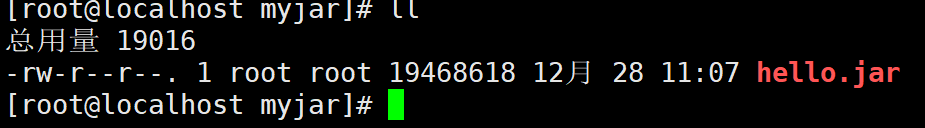
# 【掌握】Dockerfile详解【5】

## 准备jar和war



## Jar包的打包方式？

### 上传jar包



### 编写Dockerfile

思路：

java-jar hello.jar

docker run -itd -p 8080:8080 myjar:1.0

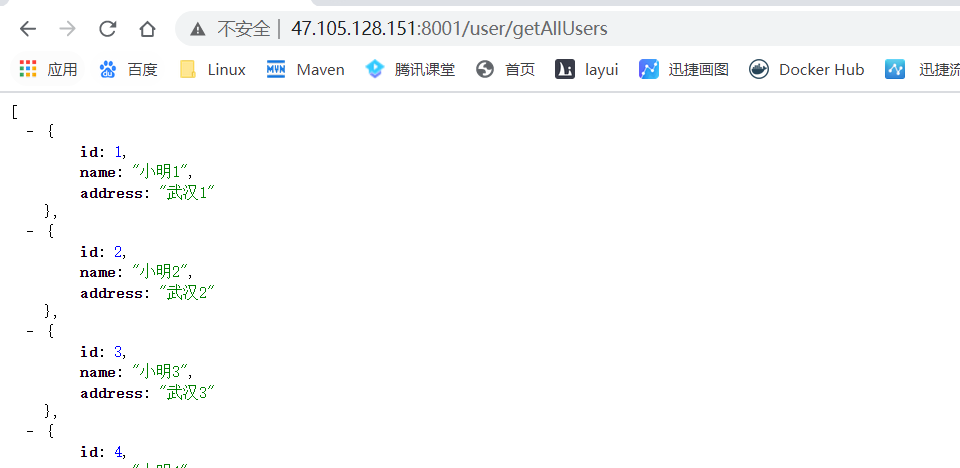


### 使用Dockerfile构造镜像



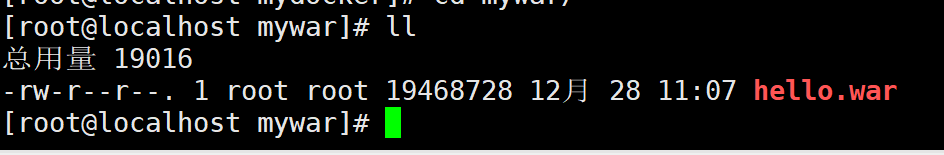
### 运行镜像





## war包的打包方式

### 上传war包



### 编写Dockerfile



### 使用Dockerfile构造镜像



### 运行镜像





# 【了解】本地镜像发布到阿里云

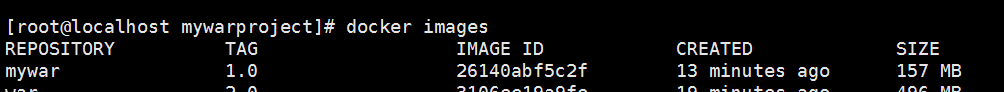
## 镜像的生成方法

1，docker commit

2，Dockerfile

## 本地镜像推送到阿里云【了解】

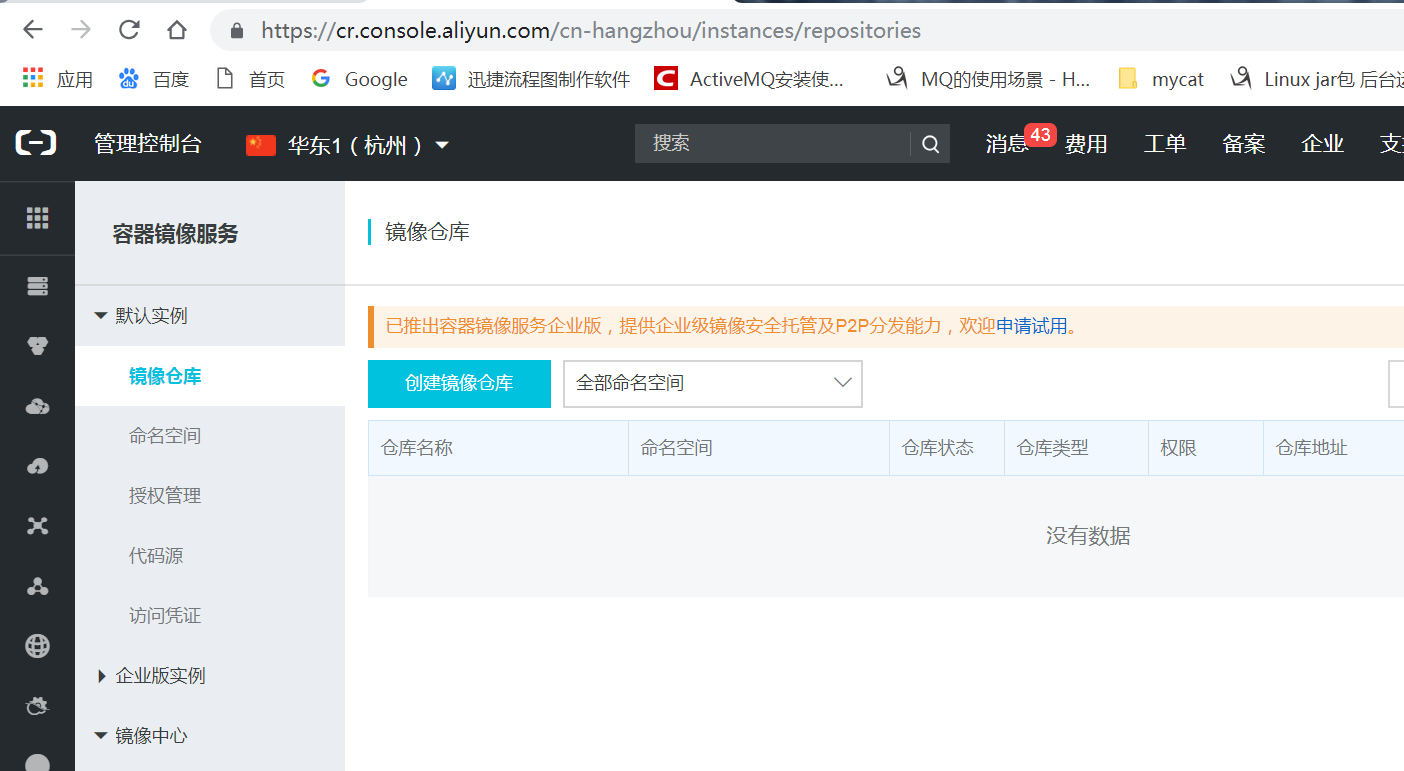
### 以修改的mywar的image为例来说明



### 登陆阿里开发平台

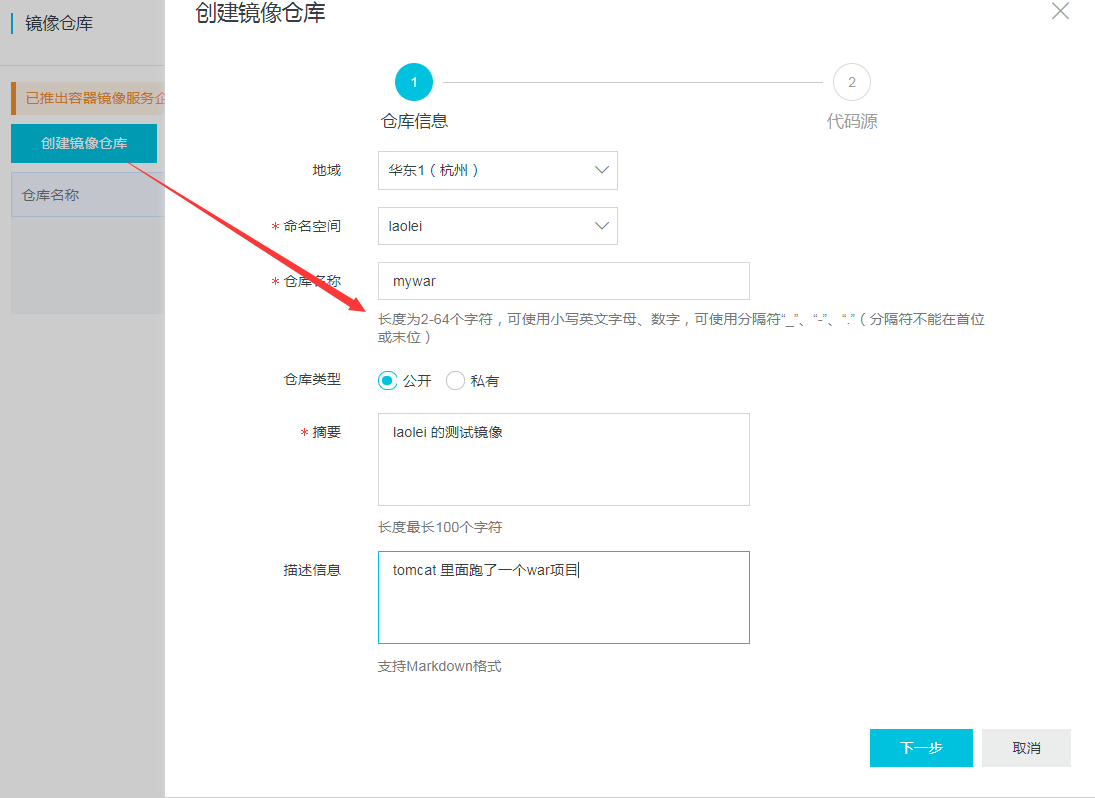
<https://dev.aliyun.com/search.html>   完成登陆

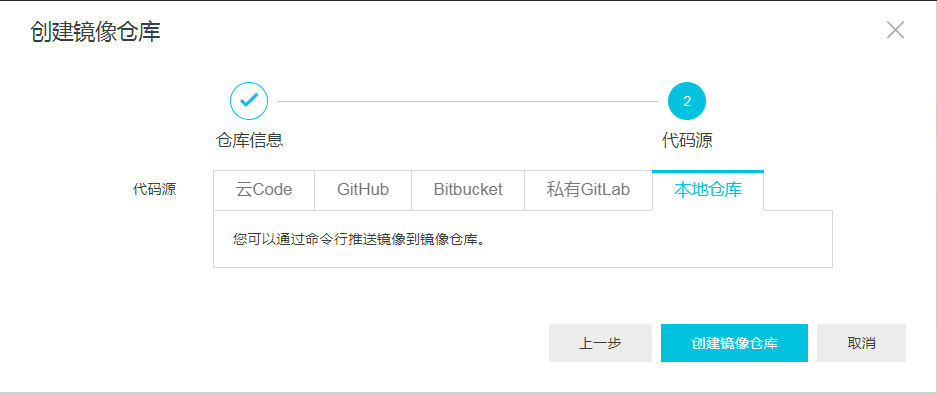
打开<https://cr.console.aliyun.com/cn-hangzhou/instances/repositories>





### 创建镜像仓库



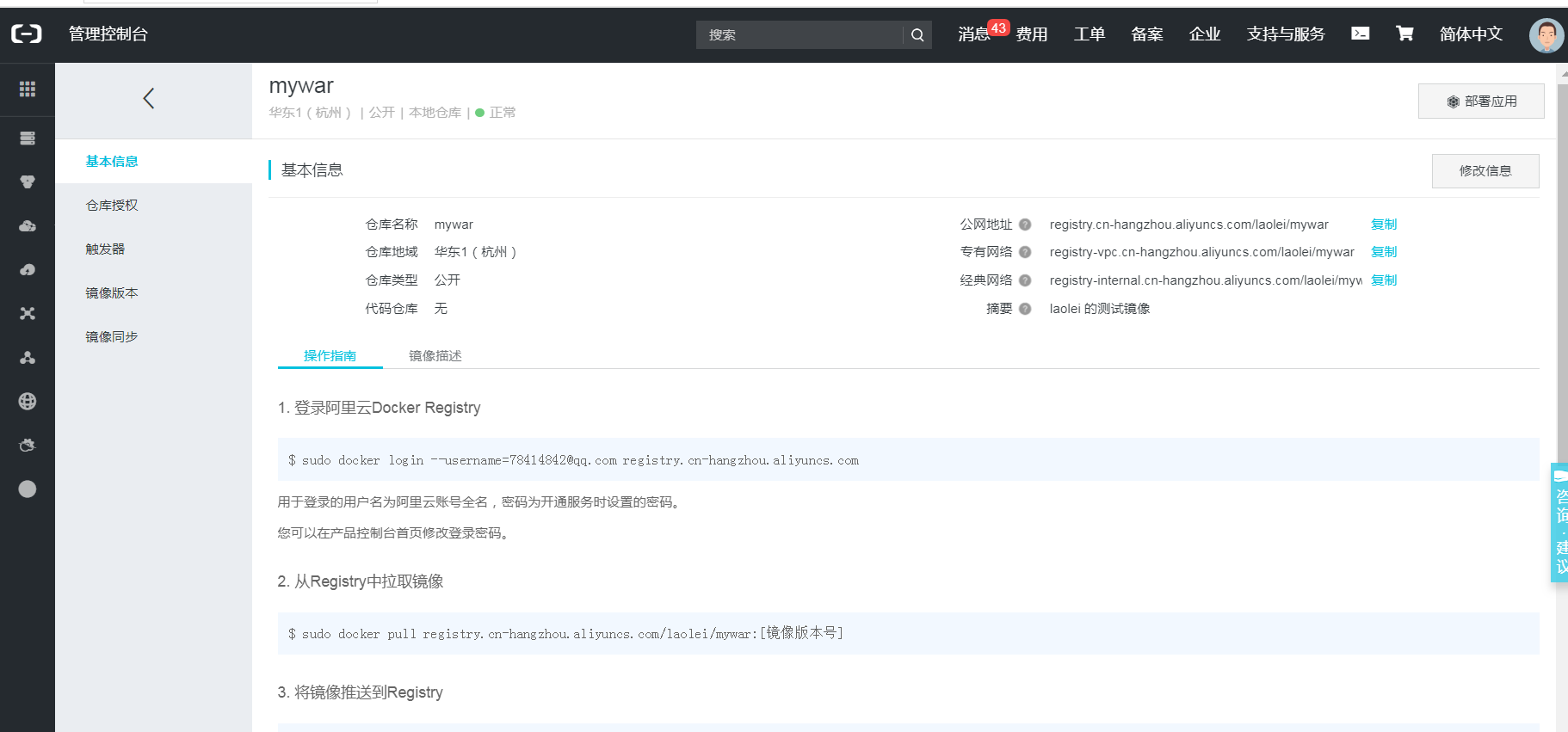




### 设置镜像仓库的访问密码



### 打开阿像页面

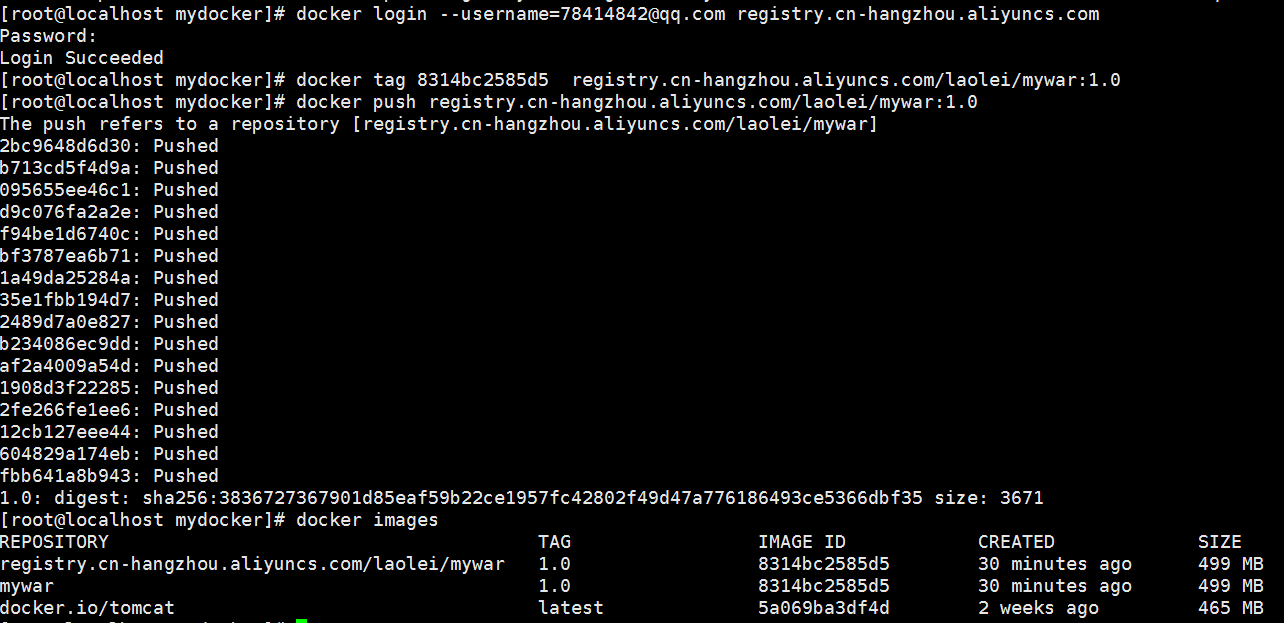


按里面的操作来就行了

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

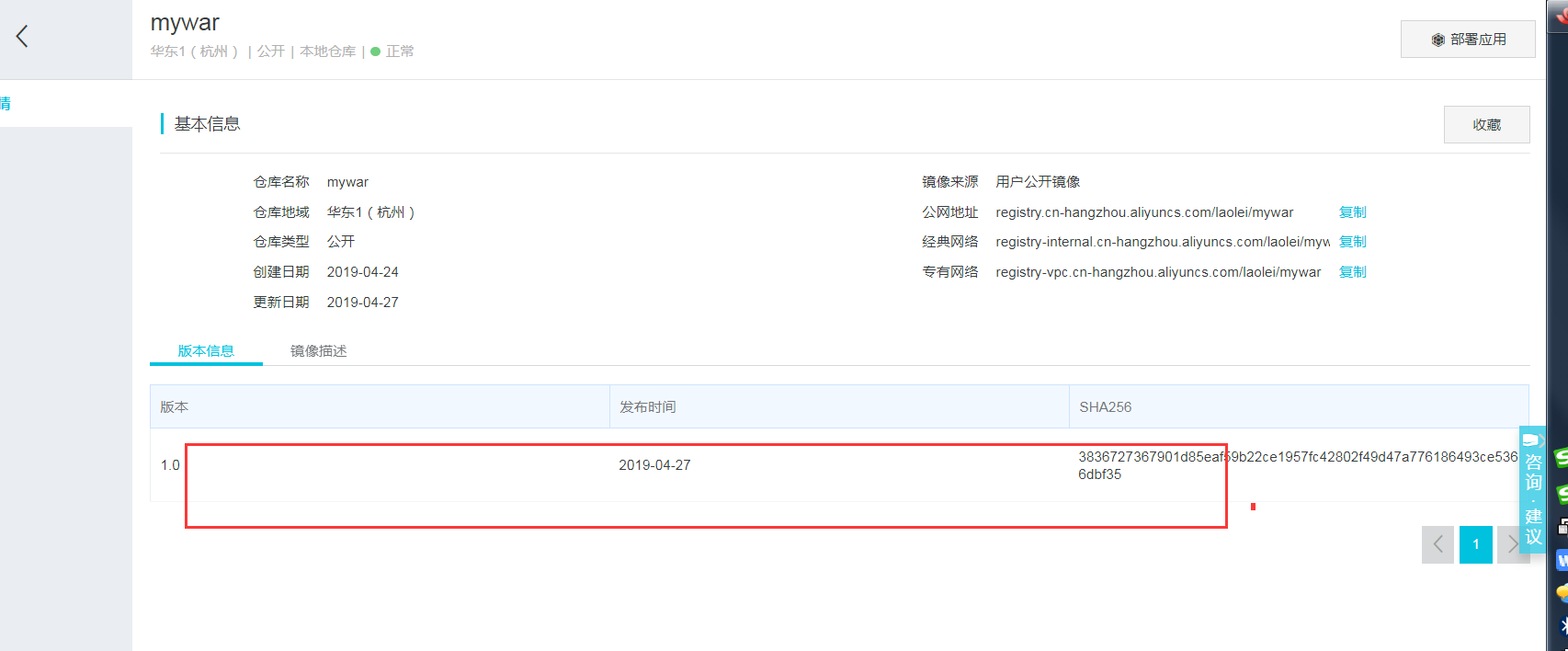
docker tag [ImageId] registry.cn-hangzhou.aliyuncs.com/laolei/mywar:[镜像版本号]

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



### 的阿里的公有仓库里面查询试试



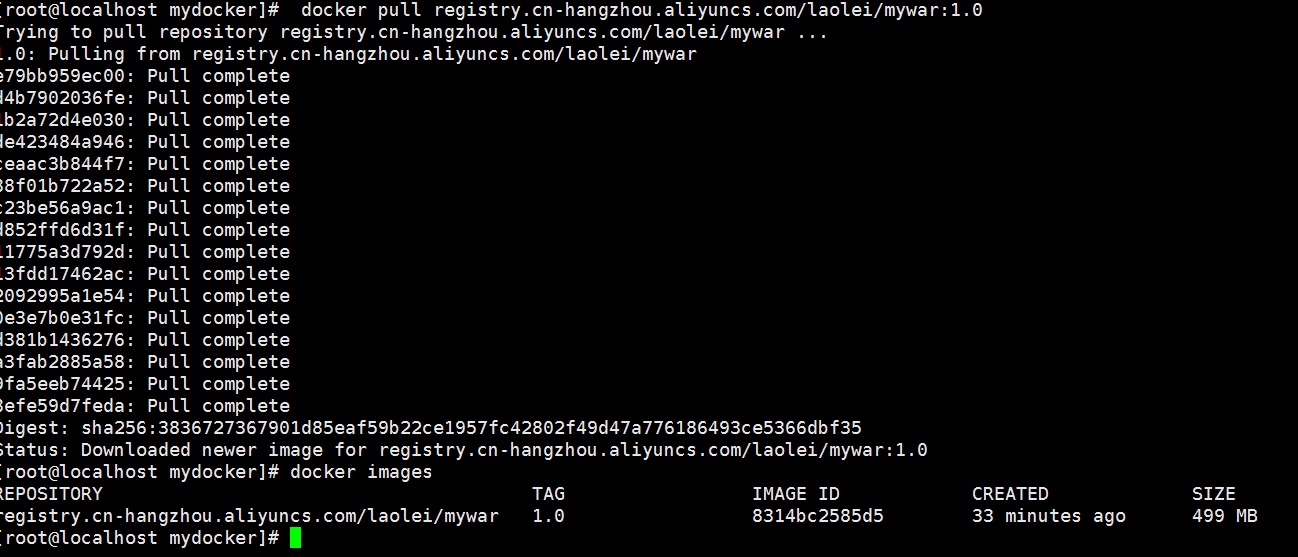


## 把阿里云的镜像下载到本地

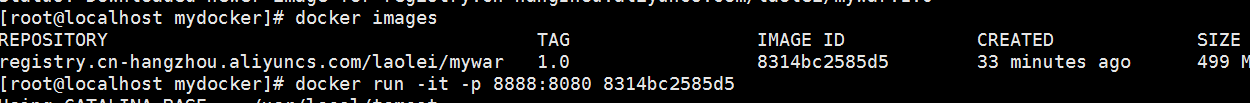
### 语法

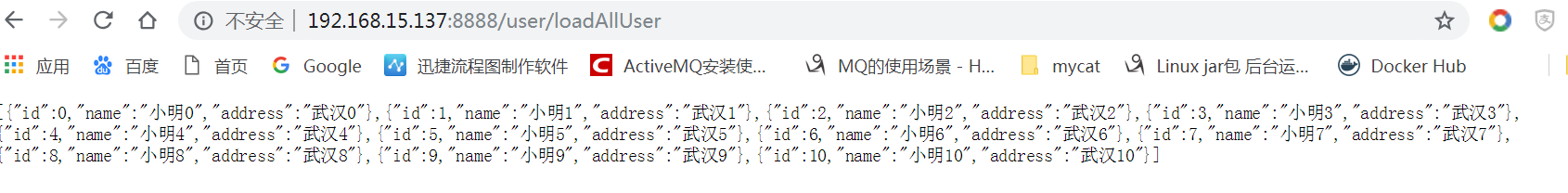
docker pull registry.cn-hangzhou.aliyuncs.com/laolei/mywar:[镜像版本号]

### 案例



### 运行测试





# 【掌握】docker常用安装

## 安装步骤梳理

1. 搜索镜像 docker search mysql
2. 拉取镜像  docker pull
3. 查看镜像  docker images
4. 启动镜像   docker run
5. 停止容器   docker stop 容器ID
6. 移除容器  docker rm -f 容器ID

## 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

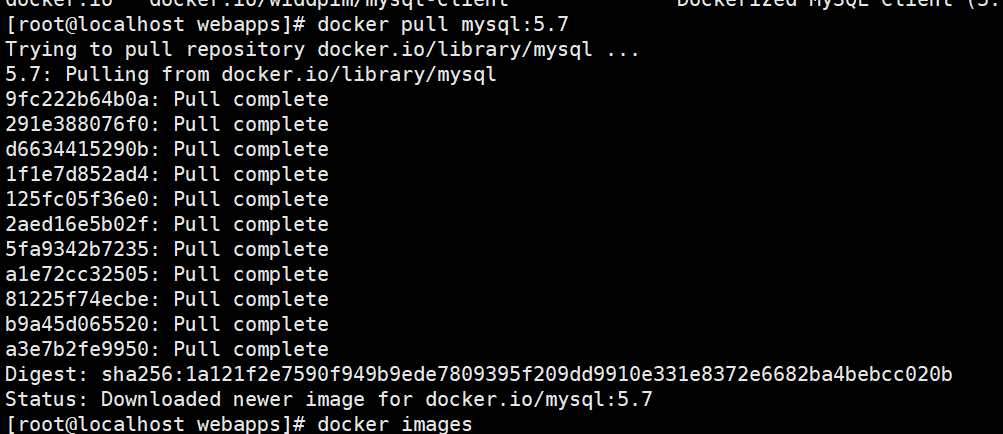
                -t 终端

## 安装mysql

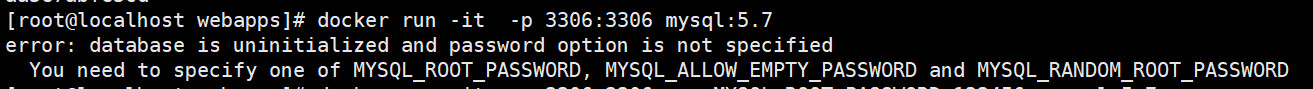
docker hub上查找mysql镜像  docker search mysql



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



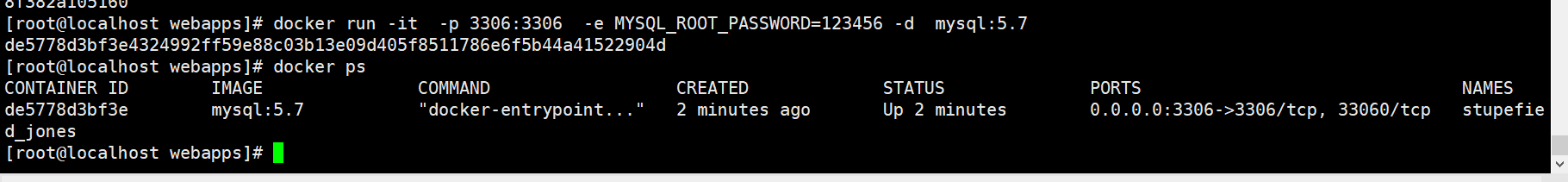
使用mysql5.7镜像创建容器运行存在问题



这是因为mysql镜像一启动中在加设置用户名和密码

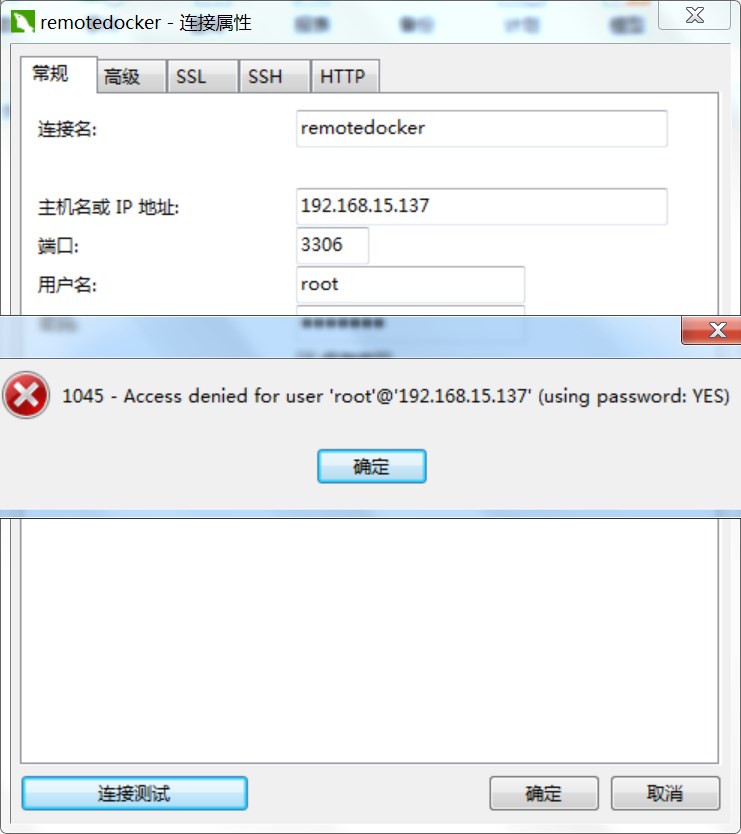
修改成下面的命令

docker run --name mysql -p 3306:3306 -e MYSQL\_ROOT\_PASSWORD=123456 -d mysql:5.7 --lower\_case\_table\_names=1



发现启动起来了

## 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='%'

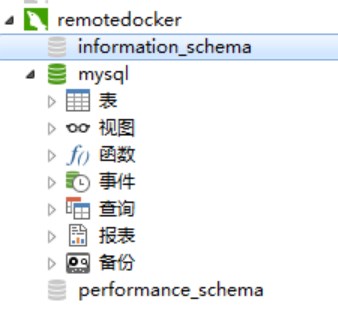
5,重启mysql的docker 容器

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

## 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

# 【掌握】Docker的四种网络模式

## 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：

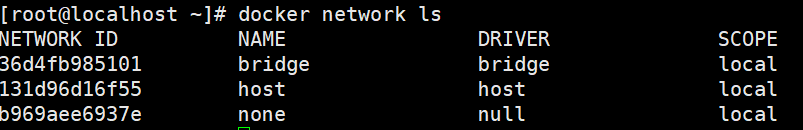
**bridge模式：使--net =bridge指定，默认设置；**

**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



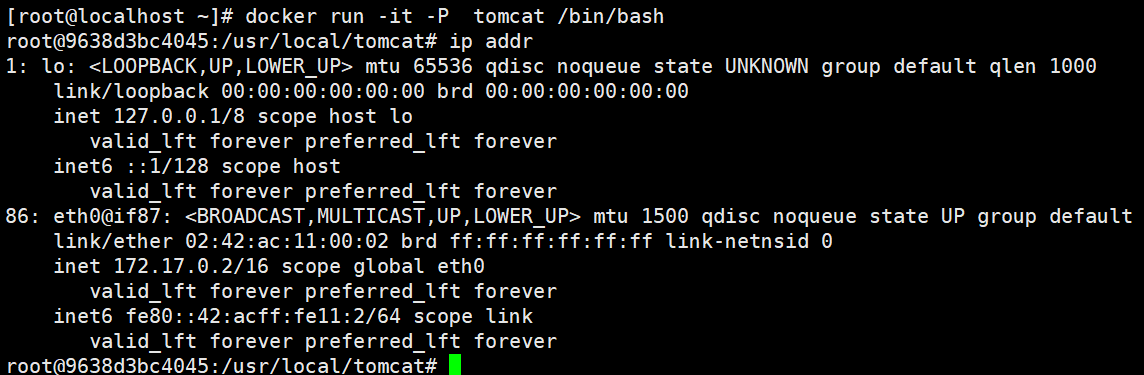
下面我们分别来讲解docker的四种网络模式：

## bridge模式

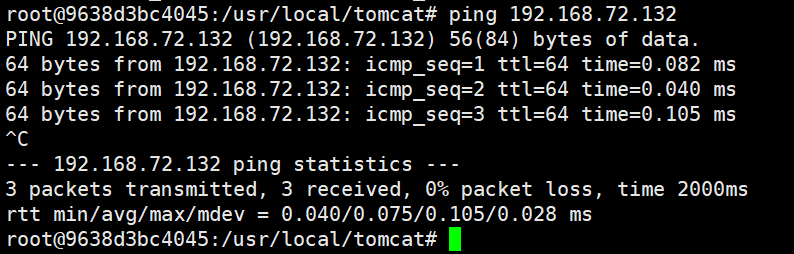
bridge模式是Docker默认的网络设置，此模式会为每一个容器分配Network Namespace、设置IP等，并将并将一个主机上的Docker容器连接到一个虚拟网桥上。当Docker server启动时，会在主机上创建一个名为docker0的虚拟网桥，此主机上启动的Docker容器会连接到这个虚拟网桥上。虚拟网桥的工作方式和物理交换机类似，这样主机上的所有容器就通过交换机连在了一个二层网络中。接下来就要为容器分配IP了，Docker会从RFC1918所定义的私有IP网段中，选择一个和宿主机不同的IP地址和子网分配给docker0，连接到docker0的容器就从这个子网中选择一个未占用的IP使用。如一般Docker会使用172.17.0.0/16这个网段，并将172.17.0.1/16分配给docker0网桥（在主机上使用ip addr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：

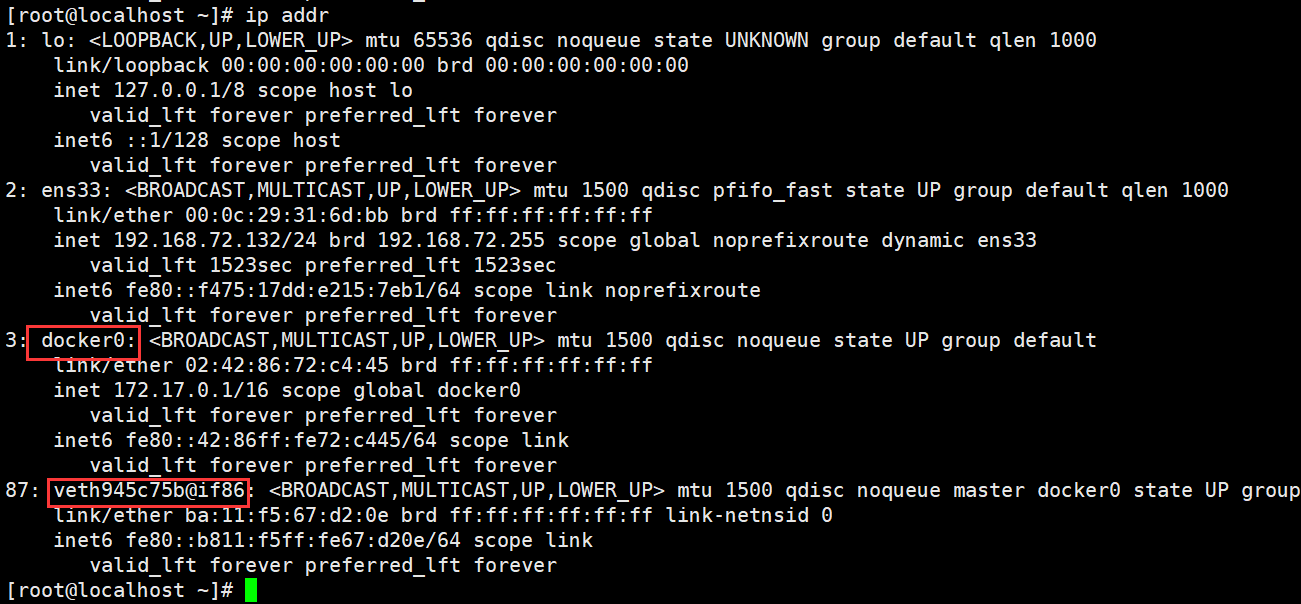
启动容器：（由于是默认设置，这里没指定网络–net =bridge，可以看到容器内创建了eth0）



使用ping命令连接Host网络发现，容器与Host网络是连通的：



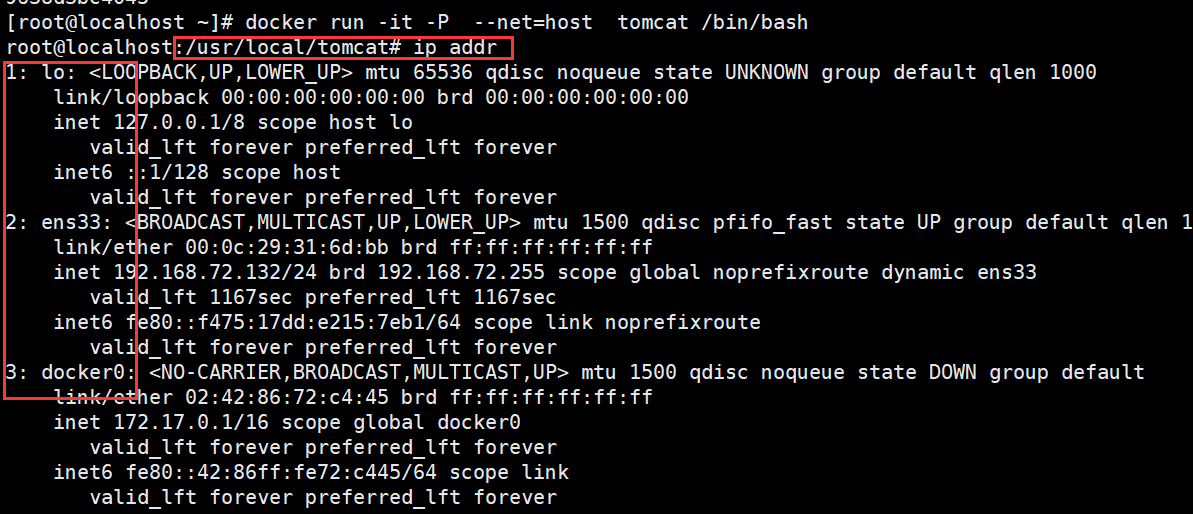
eth0实际上是veth pair的一端，另一端（veth945c）连接在docker0网桥上：



## 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：



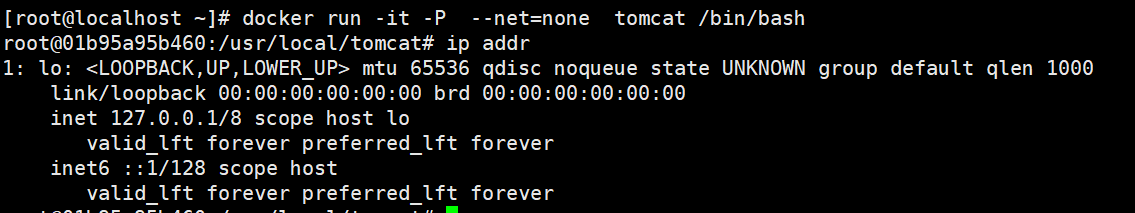
使用host模式启动容器后可以发现，使用ip addr查看网络环境时，看到的都是宿主机上的信息。这种方式创建出来的容器，可以看到host上的所有网络设备。

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。

## 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。

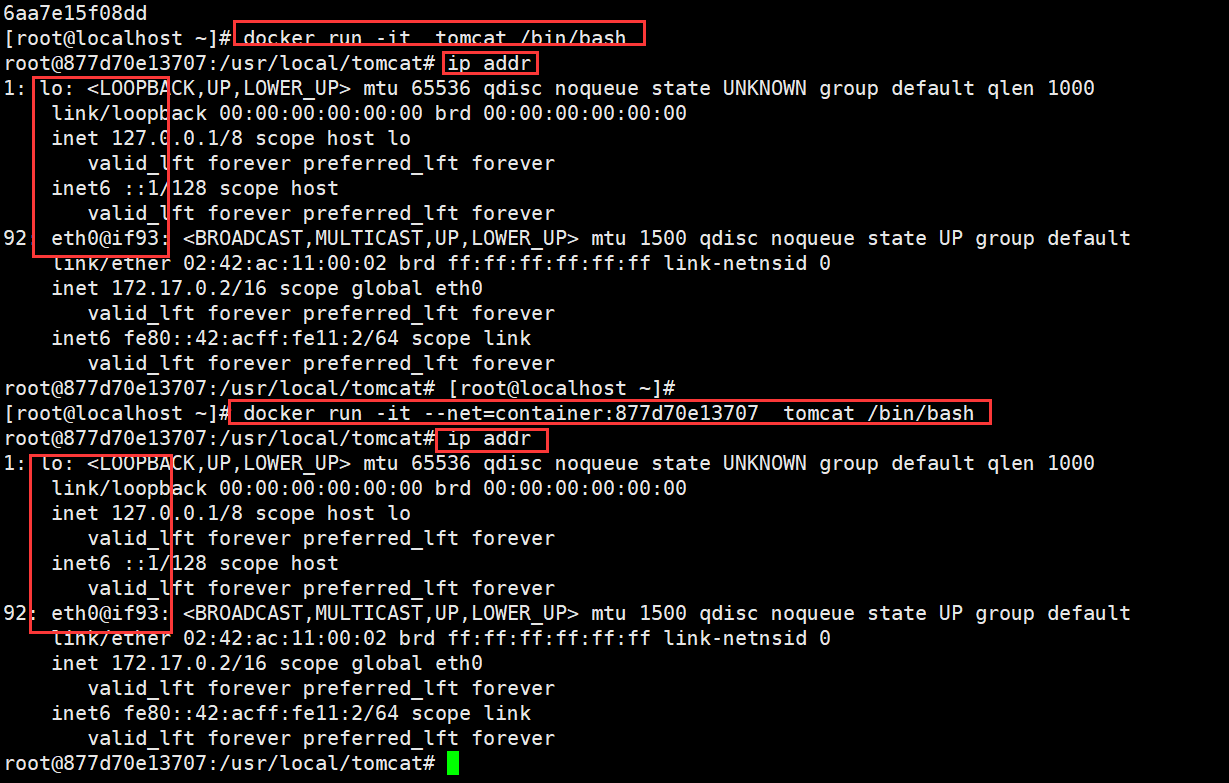
使用--net =none模式启动容器：



## 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：



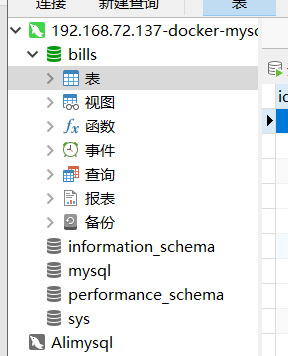
通过该例子可以看出来，两者的网络完全相同。

# 【掌握】Docker部署记账本

## 思路

1. 使用docker 运行一个mysql 把bills.sql导入
2. 打包war
3. 上传war
4. 编写Dockerfile
5. 构造镜像
6. 运行?

## 使用docker 运行mysql 并导入bills.sql



## 上传war



## 创建Dockerfile



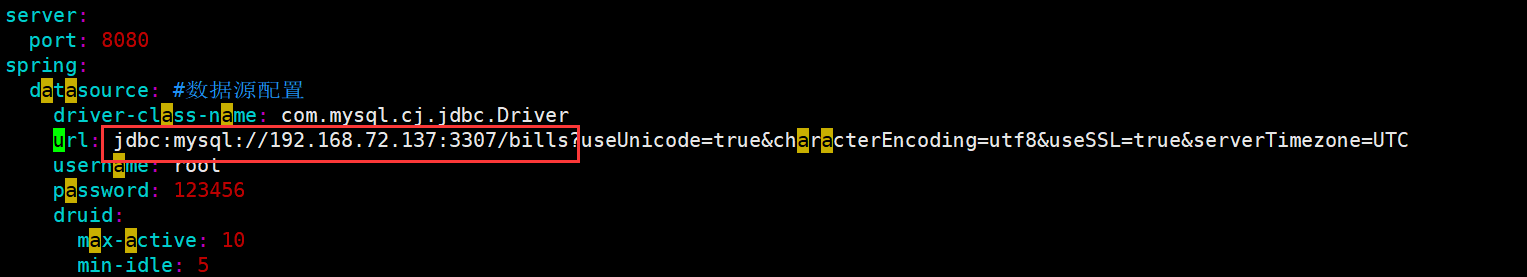
## 运行bills镜像

|  |
| --- |
| docker run -itd -p 8080:8080 --name=bills bills:1.0 |

## 复制容器里面的application.yml

|  |
| --- |
| docker cp bills:/usr/local/tomcat/webapps/ROOT/WEB-INF/classes/application.yml ./ |

## 修改application.yml



## 把application.yml 放回容器

|  |
| --- |
| docker cp application.yml bills:/usr/local/tomcat/webapps/ROOT/WEB-INF/classes/application.yml |

## 重启容器

docker restart bills

## 如果是jar 包

### 方法一，使用配置文件的优先级去处理

file:config

file:./

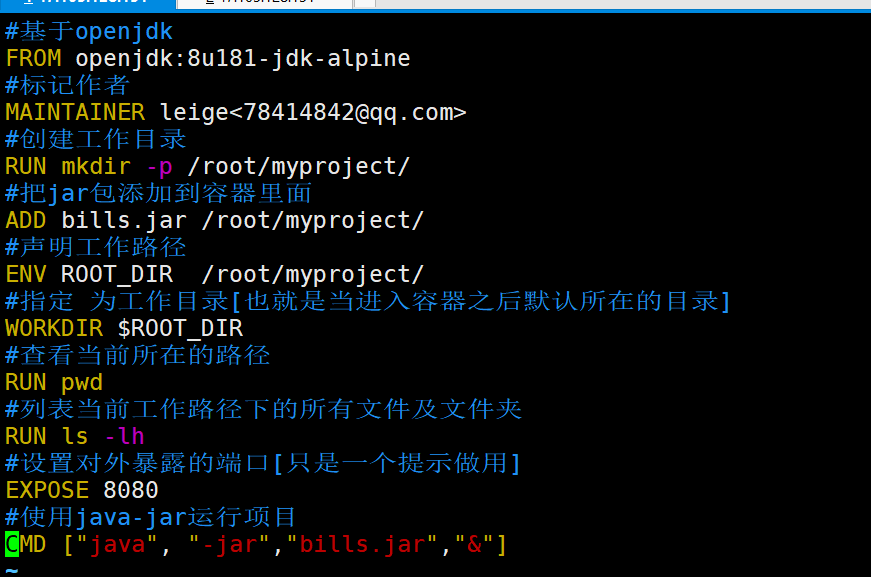
classpath:config

classpath:./

如容器里面的jar包放到了root/bills这个目录

docker cp application.yml bills:/root/bills/application.yml

编写dockerFile



上传jar包

修改application.yml

构造镜像

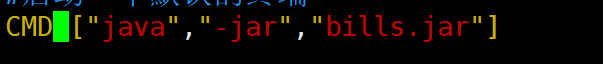
运行镜像

把修改好的application.yml放到容器里面和jar包的同级目录

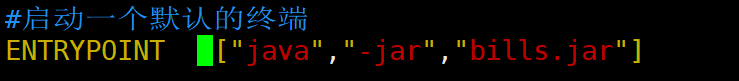
重启docker 容器

### 方法二，使用运行参数去处理

CMD



ENTRYPOINT



启动时

docker run -it -d -p 84:8080 mybillsjar:1.1 --spring.datasource.url= jdbc:mysql://192.168.72.137:3307/bills?useUnicode=true&characterEncoding=utf8&useSSL=true&serverTimezone=UTC

# 【掌握】关于防火墙的问题

1. 开启，重启docker 运行mysql 外面能不能访问？ 能！
2. 关闭防火墙 运行mysql 报错！！！
3. 关闭防火墙 重启docker 运行mysql 正常

如果系统的防火墙有关闭或开启，请重新启动docker

systemctl restart docker 重启

systemctl start docker 启动

systemctl stop docker 停止

systemctl enable docker 开机自启