# Docker简介

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 Apache 2.0 协议，项目代码在 [GitHub](https://github.com/docker/docker) 上进行维护。

Docker 项目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。

（背景），云计算兴起后，服务器硬件扩展非常便利，软件服务部署成为了瓶颈，docker趁势而兴。

# 为什么用 Docker

容器的启动可以在秒级实现，比传统的虚拟机方式要快得多

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

docker的出现，让开发/测试/线上的环境部署，成为便利一条龙。

## 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。

## 更高效的虚拟化

Docker 容器的运行不需要额外的 hypervisor 支持，它是内核级的虚拟化，因此可以实现更高的性能和效率。

## 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

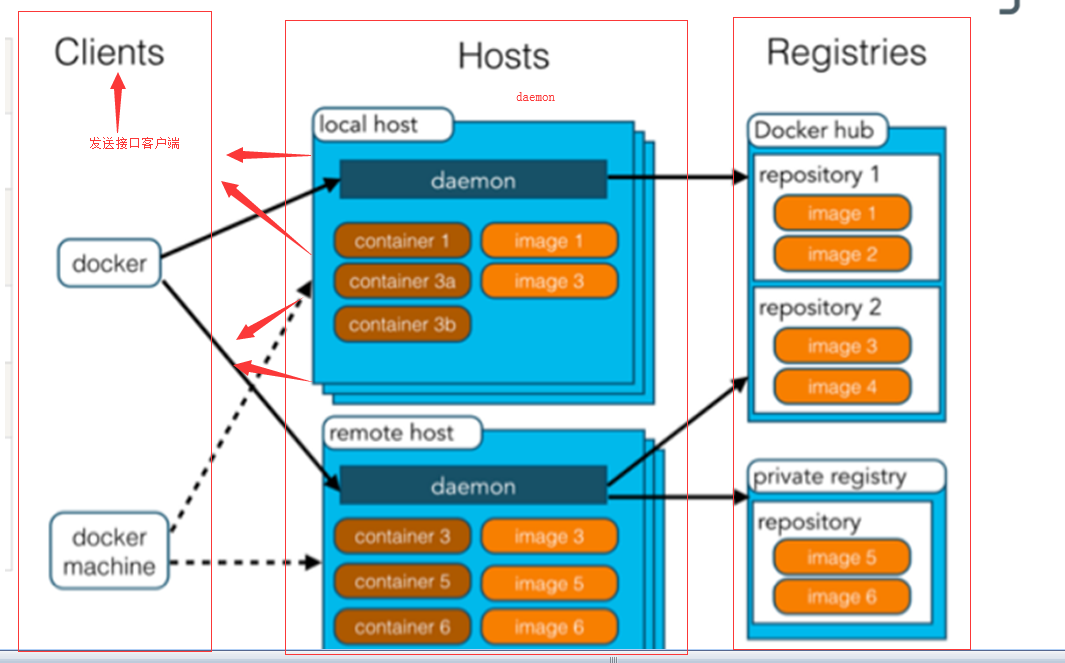
使用 Docker，只需要小小的修改，就可以替代以往大量的更新工作。所有的修改都以增量的方式被分发和更新，从而实现自动化并且高效的管理。

## 对比传统虚拟机总结

| **特性** | **容器** | **虚拟机** |
| --- | --- | --- |
| 启动 | 秒级 | 分钟级 |
| 硬盘使用 | 一般为 MB | 一般为 GB |
| 性能 | 接近原生 | 弱于 |
| 系统支持量 | 单机支持上千个容器 | 一般几十个 |

# Docker基本概念

## Docker架构



host --- 主机载体 == docker安装的地方

继承类比：

Class2 extents Class1 ---------------------- Object o = new Class2

--------------------------------------此时，o对象的结构中，有Class1的成员结构

image2 extents image1 ----------------------Container c = new image2

-------------------------------------此时，c容器中，有image1的文件

## Docker 镜像

Docker 镜像就是一个只读的模板。

例如：一个镜像可以包含一个完整的 ubuntu 操作系统环境，里面仅安装了 Apache 或用户需要的其它应用程序。

镜像可以用来创建 Docker 容器。

Docker 提供了一个很简单的机制来创建镜像或者更新现有的镜像，用户甚至可以直接从其他人那里下载一个已经做好的镜像来直接使用。

## Docker 容器

Docker 利用容器来运行应用。

容器是从镜像创建的运行实例。它可以被启动、开始、停止、删除。每个容器都是相互隔离的、保证安全的平台。

可以把容器看做是一个简易版的 Linux 环境（包括root用户权限、进程空间、用户空间和网络空间等）和运行在其中的应用程序。

## Docker 仓库

仓库是集中存放镜像文件的场所。有时候会把仓库和仓库注册服务器（Registry）混为一谈，并不严格区分。实际上，仓库注册服务器上往往存放着多个仓库，每个仓库中又包含了多个镜像，每个镜像有不同的标签（tag）。

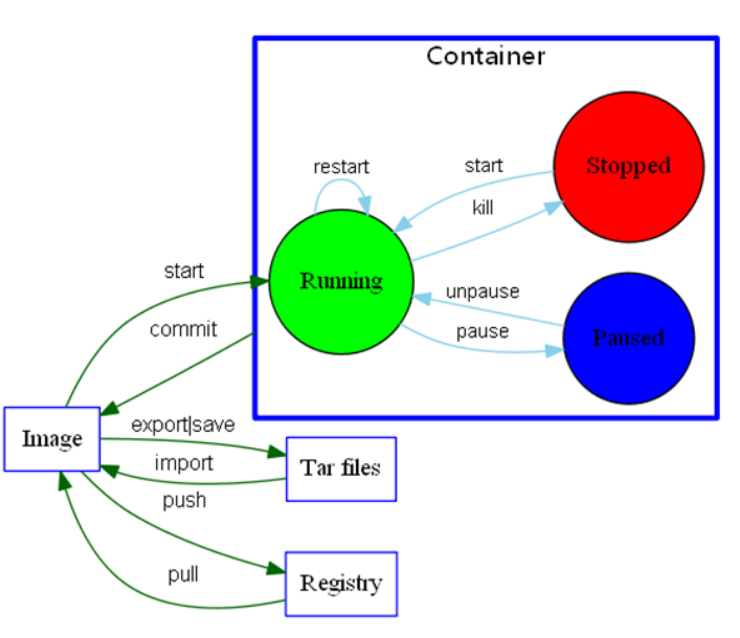
仓库分为公开仓库（Public）和私有仓库（Private）两种形式。

最大的公开仓库是 [Docker Hub](https://hub.docker.com/)，存放了数量庞大的镜像供用户下载。

当然，用户也可以在本地网络内创建一个私有仓库。

当用户创建了自己的镜像之后就可以使用 push 命令将它上传到公有或者私有仓库，这样下次在另外一台机器上使用这个镜像时候，只需要从仓库上 pull 下来就可以了。

## 容器、镜像的运行关系



# 安装 Docker

## CentOS7

CentOS7 系统 CentOS-Extras 库中已带 Docker，可以直接安装：

$ sudo yum install docker ##不是最新版本



#最新版安装

sudo yum install -y yum-utils

sudo yum-config-manager --add-repo https://[mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo](http://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo)

sudo yum install docker-ce

## 查看docker版本

docker version

docker info

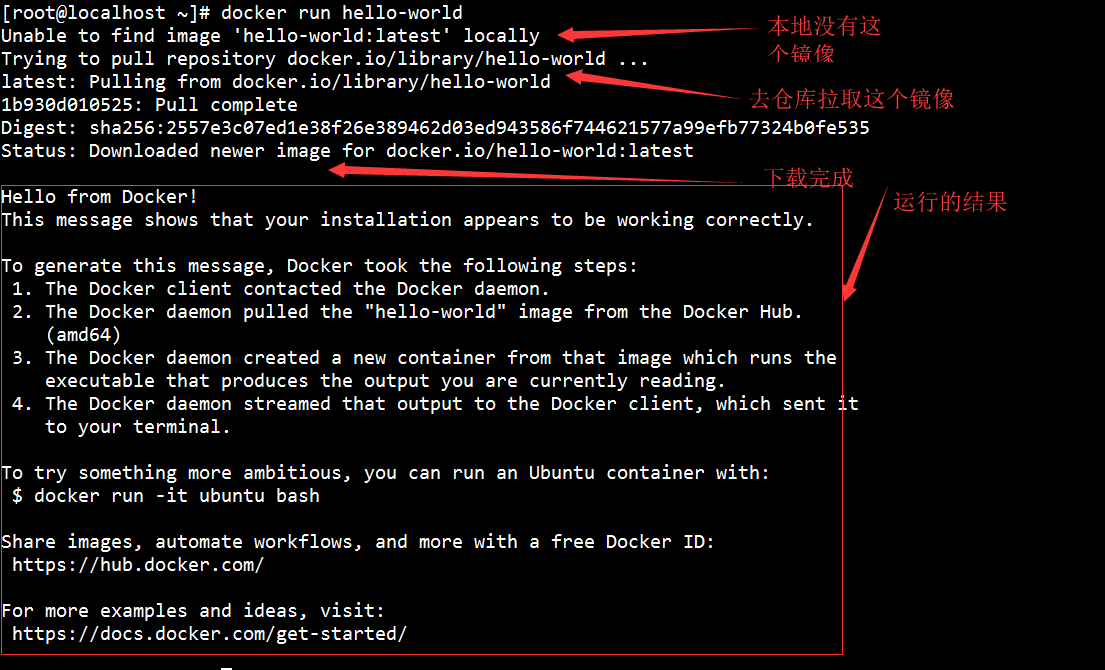
## 启动docker

sudo service docker start

## 设置随系统启动

sudo chkconfig docker on

## Docker初体验

docker run hello-world ##进入docker世界 

# Docker基本操作

## 容器操作

### docker [run|start|stop|restart|kill|rm|pause|unpause]

* [run](http://www.runoob.com/docker/docker-run-command.html)/[create](http://www.runoob.com/docker/docker-create-command.html)[镜像名]： 创建一个新的容器并运行一个命令
* [start/stop/restart](http://www.runoob.com/docker/docker-start-stop-restart-command.html)[容器名]：启动/停止/重启一个容器
* [kill](http://www.runoob.com/docker/docker-kill-command.html) [容器名]： 直接杀掉容器，不给进程响应时间
* [rm](http://www.runoob.com/docker/docker-rm-command.html)[容器名]：删除已经停止的容器
* [pause/unpause](http://www.runoob.com/docker/docker-pause-unpause-command.html)[容器名]：暂停/恢复容器中的进程

### docker [ps|inspect|exec|logs|export|import]

* [ps](http://www.runoob.com/docker/docker-ps-command.html)：查看容器列表（默认查看正在运行的容器，-a查看所有容器）
* [inspect](http://www.runoob.com/docker/docker-inspect-command.html)[容器名]：查看容器配置元数据
* exec -it [容器名] /bin/bash：进入容器环境中交互操作
* logs --since="2019-02-01" -f --tail=10 [容器名]:查看容器日志
* **cp** path1 **[容器名]:path 容器与主机之间的数据拷贝**
* **export -o test.tar** [容器名] **/ docker export** [容器名]**>test.tar : 文件系统作为一个tar归档文件**
* **import test.tar [镜像名:版本号]:导入归档文件，成为一个镜像**

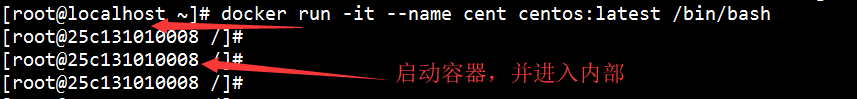
## 镜像操作

### docker images|rmi|tag|build|history|save|load]

* [images](http://www.runoob.com/docker/docker-images-command.html)：列出本地镜像列表
* [rmi](http://www.runoob.com/docker/docker-rmi-command.html) [镜像名：版本]：删除镜像
* tag [镜像名：版本] [仓库]/[镜像名：版本]：标记本地镜像，将其归入某一仓库
* [build](http://www.runoob.com/docker/docker-build-command.html) -t [镜像名：版本] [path]：Dockerfile 创建镜像
* [history](http://www.runoob.com/docker/docker-history-command.html) [镜像名：版本]: 查看指定镜像的创建历史
* **save -o xxx.tar** [镜像名：版本] **/  save** [镜像名：版本]**>xxx.tar** : **将镜像保存成 tar 归档文件**
* **load --input  xx.tar / docker load<xxx.tar : 从归档文件加载镜像**

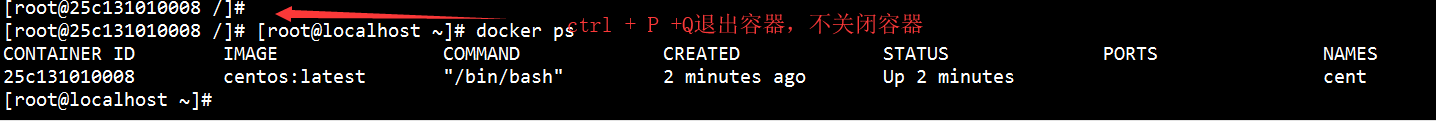
## 容器创建详解

### 交互式创建容器并进入：



docker run -it --name centos centos /bin/bash（前台进程）

------------------------exit退出也关闭容器; Ctrl+P+Q退出不关闭容器



### 后台启动容器：



docker run -d --name nginx nginx

### 进入已运行的容器：

docker exec -it nginx /bin/bash

查看容器的元数据： docker inspect nginx

### 绑定容器端口到主机：

docker run -d -p 8080:80 --name nginx nginx:latest

### 挂载主机文件目录到容器内：

docker run -dit -v /root/peter\_dir/:/pdir --name cent centos

### 复制主机文件到容器内：

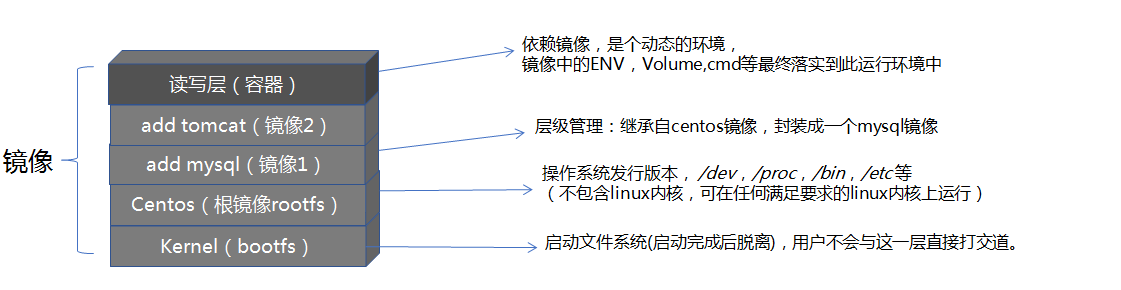
docker cp anaconda-ks.cfg cent:/var

# 镜像的创建

## 由容器提交镜像

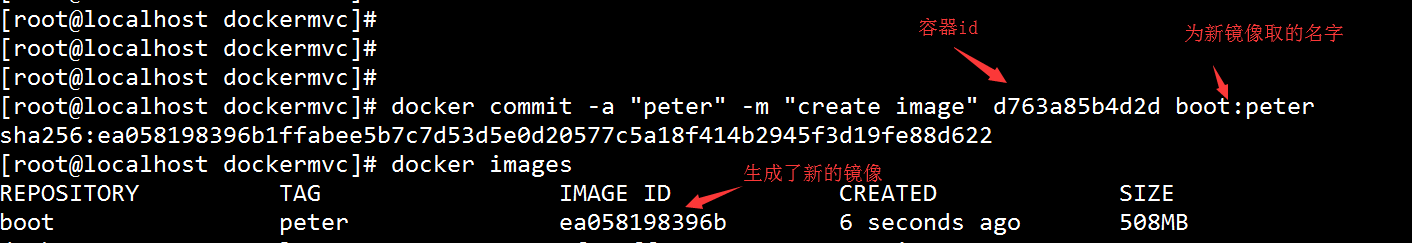
docker镜像与容器，使用联合文件技术管理文件。

镜像与容器由一层层的layer文件组装而成，如下图：



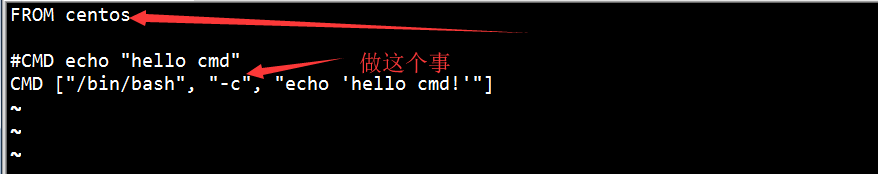
a、当用镜像模板创建容器时，是直接在镜像的文件层级上，加一层容器读写层

b、反过来，如果想要创建一个新的镜像，直接把容器对应的所有文件层，转为只读层即可。commit命令可达到这个效果：

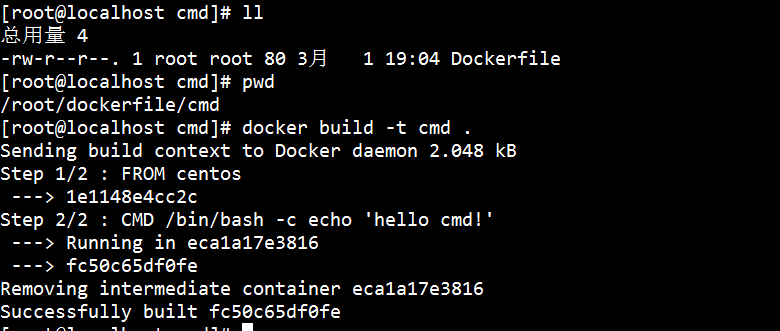


## dockerfile方式创建容器

最简单的dockerfile



创建镜像



使用此镜像运行一个容器



### dockerfile基本要素



### dockerfile指令

#### FROM：

　　FROM {base镜像}

　　必须放在DOckerfile的第一行，表示从哪个baseimage开始构建

#### MAINTAINER：

可选的，用来标识image作者的地方

#### RUN

RUN都是启动一个容器、执行命令、然后提交存储层文件变更。

第一层 RUN command1 的执行仅仅是当前进程，一个内存上的变化而已，其结果不会造成任何文件。

而到第二层的时候，启动的是一个全新的容器，跟第一层的容器更完全没关系，自然不可能继承前一层构建过程中的内存变化。

而如果需要将两条命令或者多条命令联合起来执行需要加上&&。

如：cd /usr/local/src && wget xxxxxxx

#### CMD：

　　CMD的作用是作为执行container时候的默认行为（容器默认的启动命令）

　　当运行container的时候声明了command，则不再用image中的CMD默认所定义的命令

一个Dockerfile中只能有一个有效的CMD，当定义多个CMD的时候，只有最后一个才会起作用

#### EXPOSE

EXPOSE 指令是声明运行时容器提供服务端口，这只是一个声明，在运行时并不会因为这个声明应用就会开启这个端口的服务。在 Dockerfile 中写入这样的声明有两个好处，一个是帮助镜像使用者理解这个镜像服务的守护端口，以方便配置映射；另一个用处则是在运行时使用随机端口映射时，也就是 docker run -P 时，会自动随机映射 EXPOSE 的端口。

#### entrypoint：

entrypoint的作用是，把整个container变成可执行的文件，且不能够通过替换CMD的方法来改变创建container的方式。但是可以通过参数传递的方法影响到container内部

每个Dockerfile只能够包含一个entrypoint，多个entrypoint只有最后一个有效

当定义了entrypoint以后，CMD只能够作为参数进行传递

#### ADD & COPY：

　　把host上的文件或者目录复制到image中（能够进行自动解压压缩包）

#### ENV：

　　用来设置环境变量，后续的RUN可以使用它所创建的环境变量

#### WORKDIR：

　　用来指定当前工作目录（或者称为当前目录）

#### USER：

　　运行RUN指令的用户

#### VOLUME：

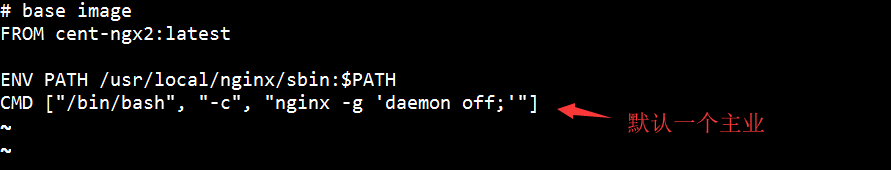
　　用来创建一个在image之外的mount point

### docker容器的主业

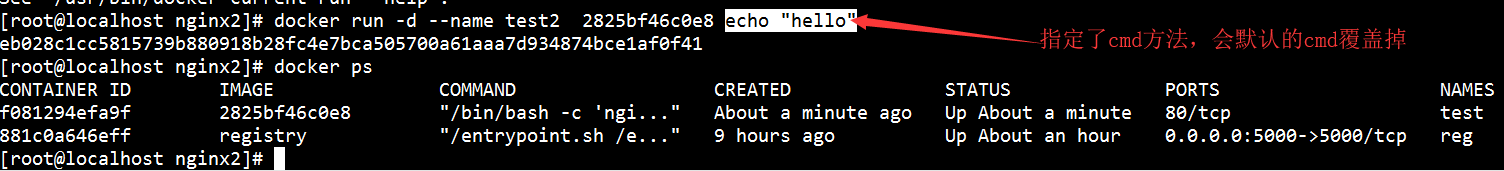
docker理念里，容器启动时，应当为它指定主业是什么，如nginx容器主业就是nginx代理服务，tomcat容器就是web服务等等

1、容器创建时，必须指定主业任务，如不指定，则容器无事可干立即退出。

2、在dockerfile打包镜像时，可以使用cmd命令来指定一个默认的主业，如下：



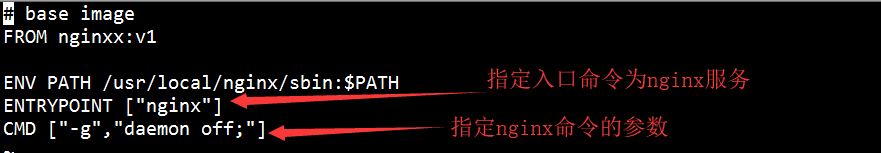
3、既然镜像里是默认主业，即意味着创建容器时，可以覆盖此默认命令，如下



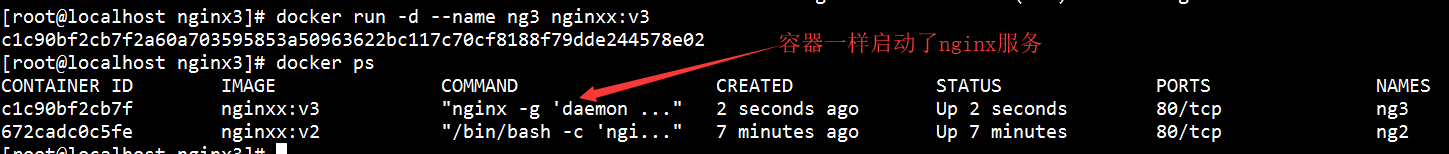
#### 推荐的ENTRYPOINT方式

1、镜像本身应该有稳定的主业，应当指定后即不能更改用途，于是引入ENTRYPOINT

2、使用ENTRYPOINT字义即容器入口，它不能被run中cmd覆盖，如下例：



执行：docker build -t nginxx:v3 .



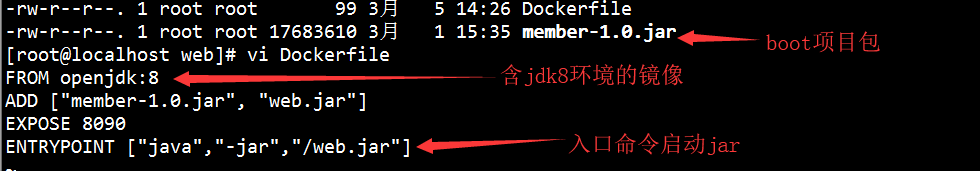
以后使用nginxx:v3这个镜像时，只能做nginx服务来使用啦

#### 手动打包springboot镜像

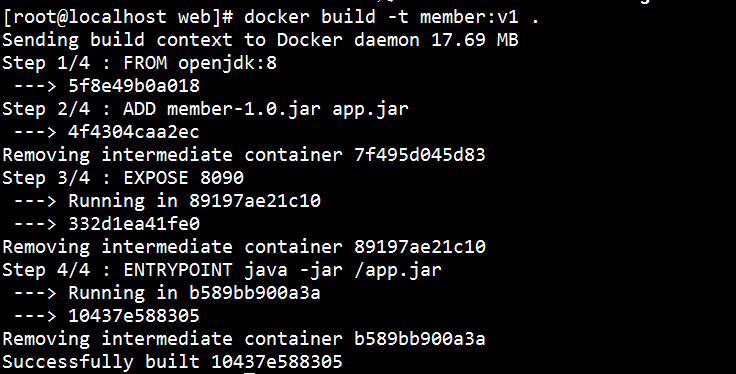
我们需要对业务项目打包发布，一样需要制作成为业务镜像，供运维使用，下面讲述springboot的制作过程：

1、将springboot打好的jar包上传

2、在同级目录下，创建Dockerfile文件，内容如下：

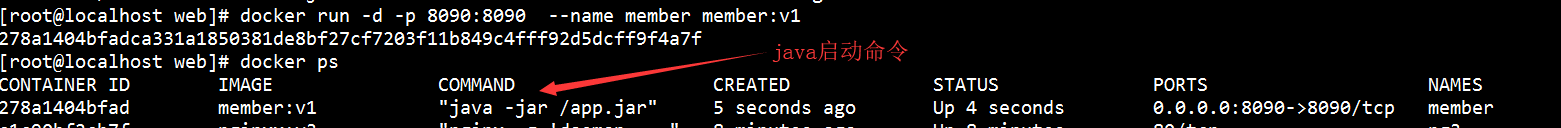


3、dockerfile打包业务镜像

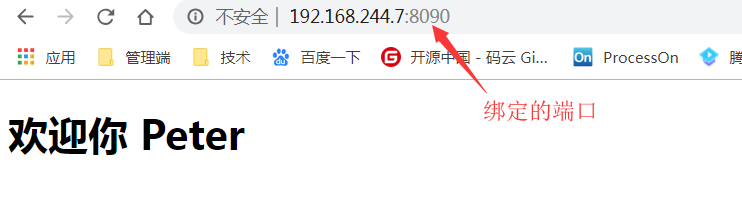


4、启动镜像，即得到业务运行

docker run -d -p 8090:8090 --name member member:v1



5、浏览器打开页面校验：http://192.168.244.7:8090/



### maven源码打包用法

更多的情况，我们是直接在运维环境里，上传源码，直接maven打包jar，然后再进一步打包成镜像，与手动打包过程类似

如果环境中没有安装maven，请手动安装，脚本如下：

sudo yum install -y yum-utils

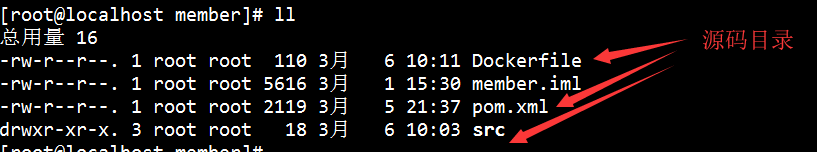
# yum-config-manager --add-repo http://repos.fedorapeople.org/repos/dchen/apache-maven/epel-apache-maven.repo

# yum-config-manager --enable epel-apache-maven

// 安装maven

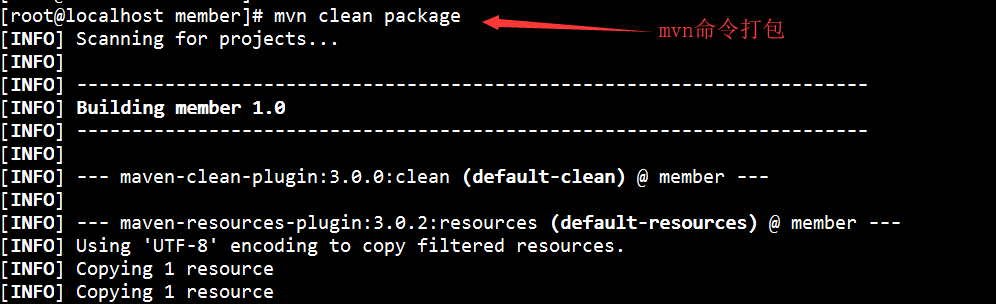
# yum install -y apache-maven

1、上传原码到docker环境中（一般是git/svn直接拉取源码）

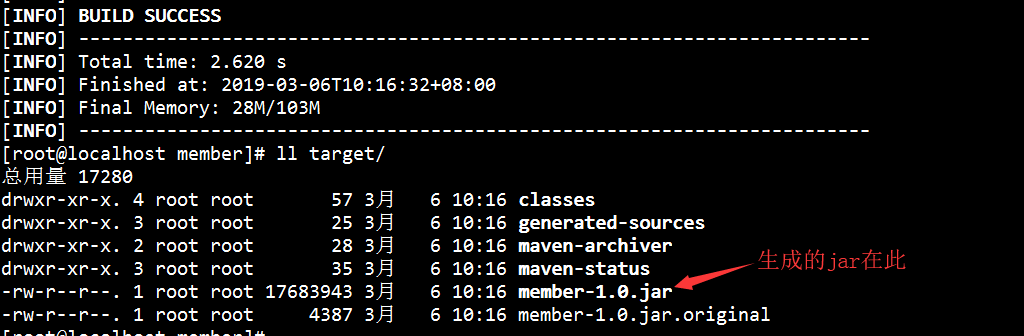


2、maven打包

mvn clean package

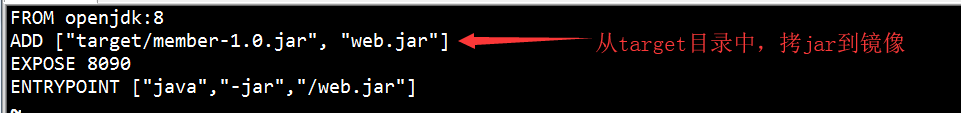


生成的jar在同级target目录下

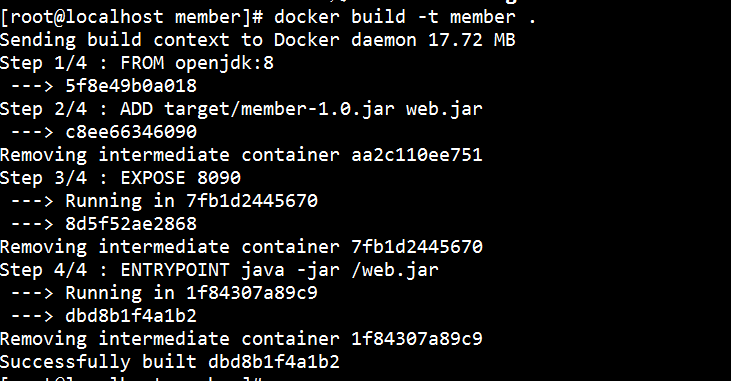


3、执行docker命令生成镜像

dockerfile文件内容



命令创建镜像



#### maven插件打包

前面打springboot包的方式，需要手动上传项目jar或者源码到服务器（违和感很强），这对于开发人员日常发布开发环境项目，极为不便

下面，演示一个maven插件：docker-maven-plugin用法，来打通环境。

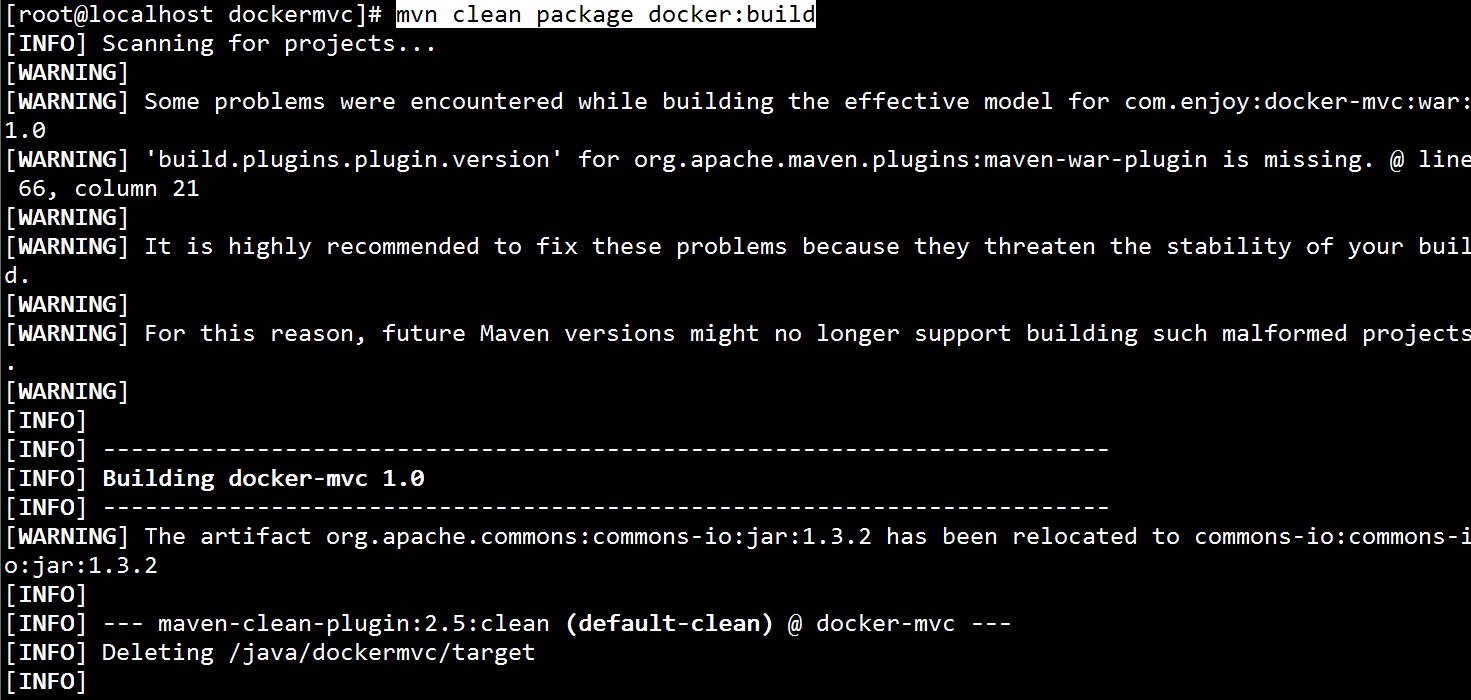
##### 项目环境配置maven插件

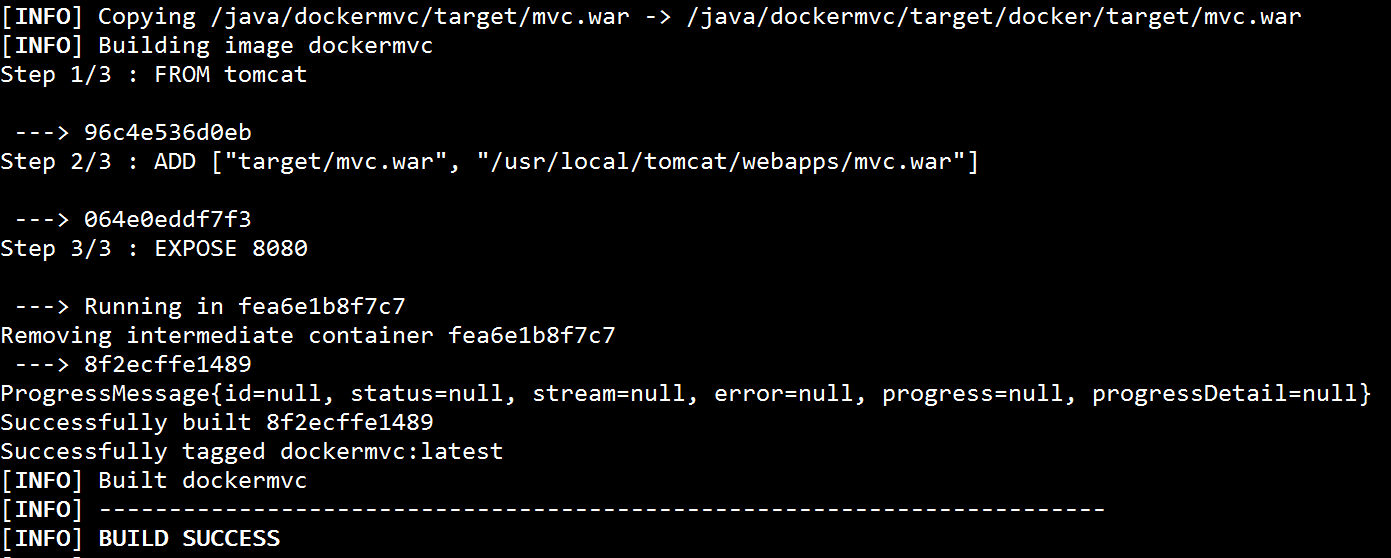
在我们的工程pom中加入docker-maven-plugin插件的配置，如下



现在，我们可以使用mvn命令，直接编译项目，打包镜像了

mvn clean package docker:build



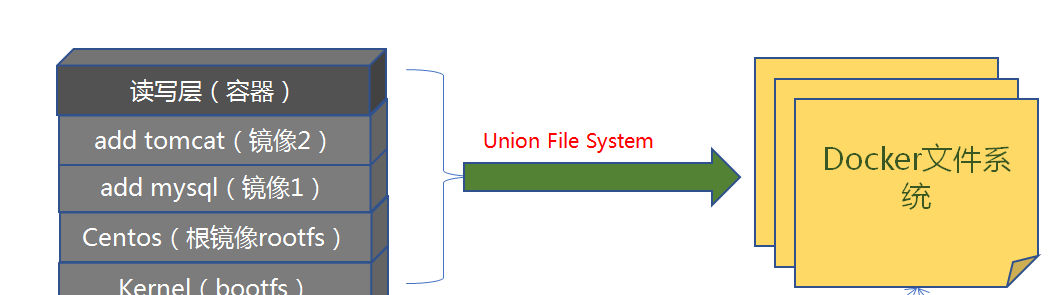


至此，我们的服务器环境，已经可以直接运行docker run 镜像得到容器服务了

# 数据管理

docker容器运行，产生一些数据/文件/等等持久化的东西，不应该放在容器内部。应当以挂载的形式存在主机文件系统中。

## docker的文件系统



1. 镜像与容器读写层，通过联合文件系统，组成系统文件视角
2. 容器服务运行中，一定会生成数据
3. 容器只是运行态的服务器，是瞬时的，不承载数据的持久功能

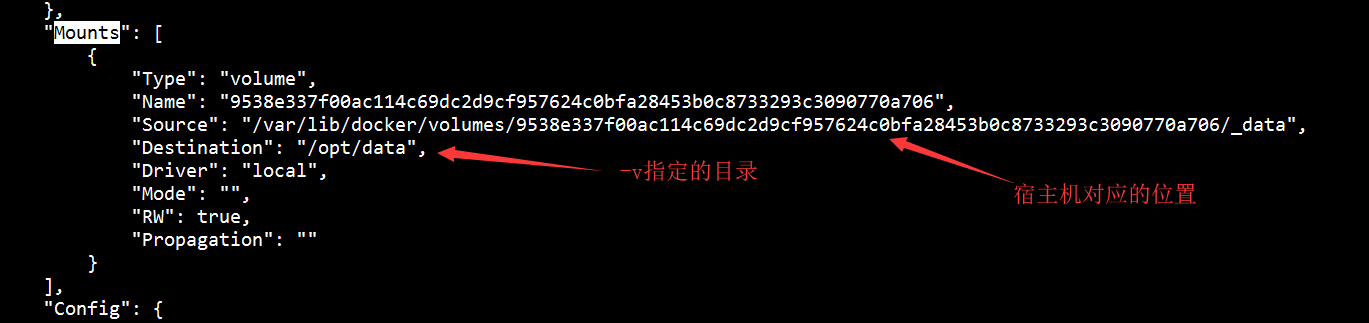
## volume文件挂载的探究

1、volume参数创建容器数据卷

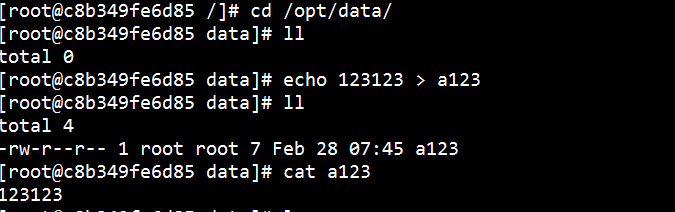


2、我们通过docker inspect data查看容器元数据，可看到挂载信息

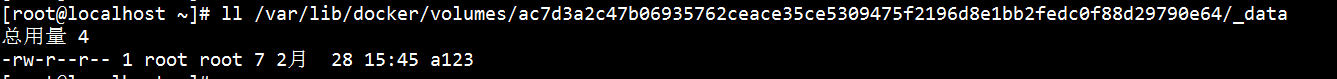




3、在容器端添加一个文件



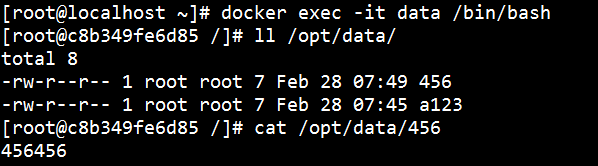
回主机目录查看，果然存在此文件：



4、在主机方添加一个文件



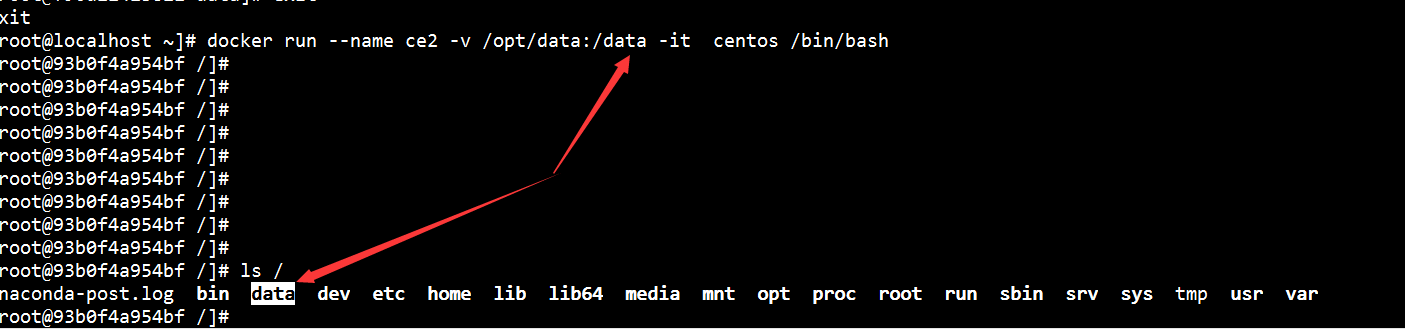
回容器里查看，果然也同步增加了此文件



5、指定主机目录方式挂载文件

格式：-v path1：path2

如下命令，容器方会自动增加一个data目录



宿主机方，同样自动增加一个/opt/data目录

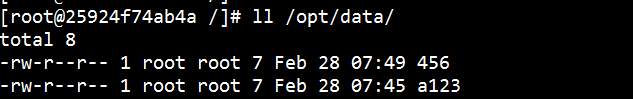


## volumes-from引用数据卷

新启一容器，引入上一步的data容器目录



自动得到同一个目录，内容与data容器里挂载一样



## 删除数据卷：

docker rm -v data

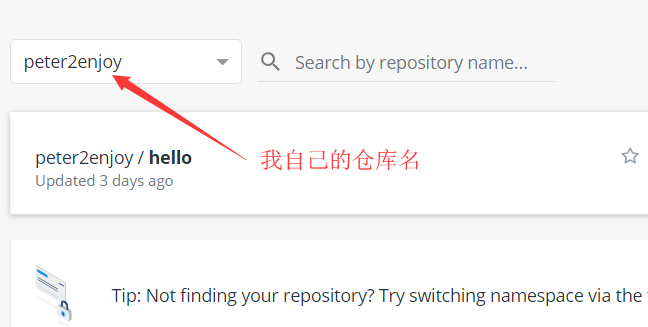
# 仓库使用

## docker官方仓库

### 注册

https://hub.docker.com

自由注册，邮件激活即可使用



### 命令使用

Docker pull/search/login/push/tag

tag [镜像名：版本] [仓库]/[镜像名：版本]：标记本地镜像，将其归入某一仓库

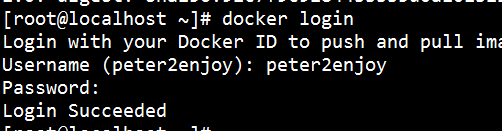
Push [仓库]/[镜像名：版本]: 推送镜像到仓库 --需要登陆

Search [镜像名]：在仓库中查询镜像 – 无法查询到tag版本

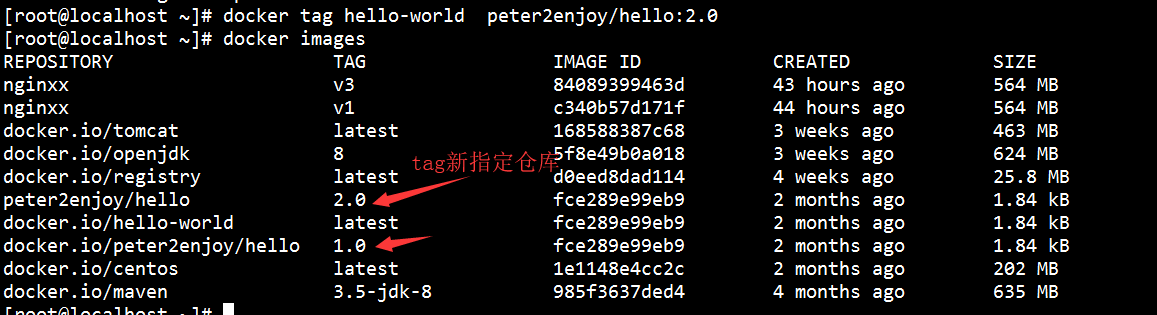
Pull [镜像名：版本]： 下载镜像到本地

Login：登陆仓库

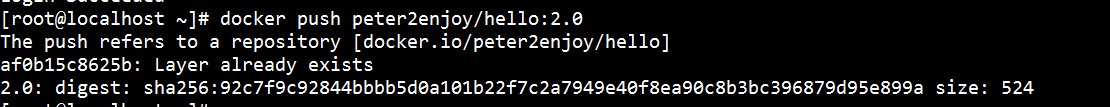
1、命令登陆dockerhub



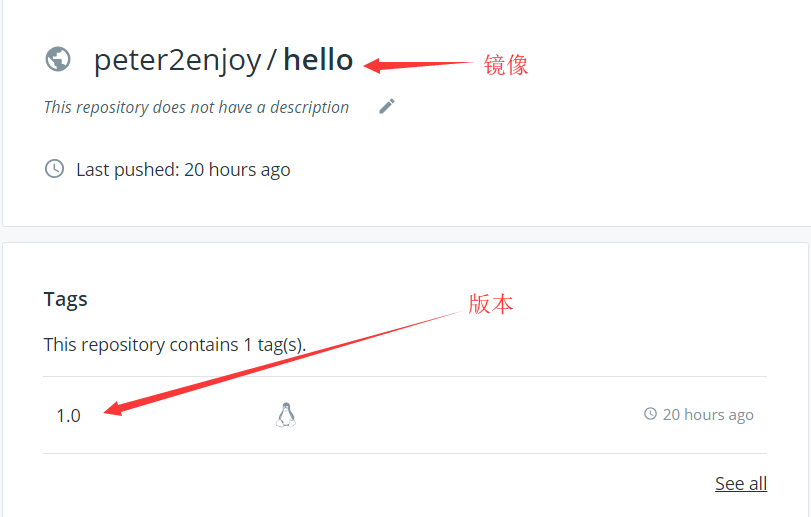
2、再使用tag命令标记一个镜像，指定自己的仓库



3、使用push命令推送此镜像到仓库里



4、打开查询自己仓库的镜像

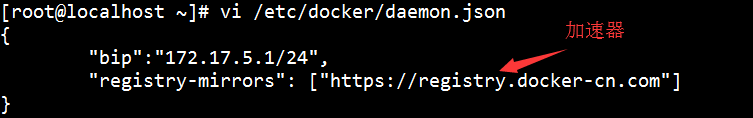


## 私有仓库

### 搭建

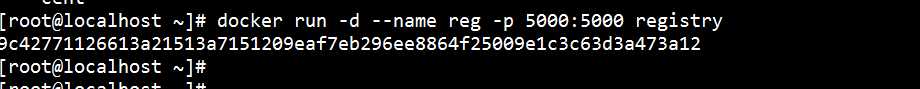
下载registry镜像：docker pull registry

-----可配置加速器加速下载

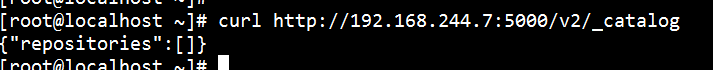


### 启动

docker run -d --name reg -p 5000:5000 registry

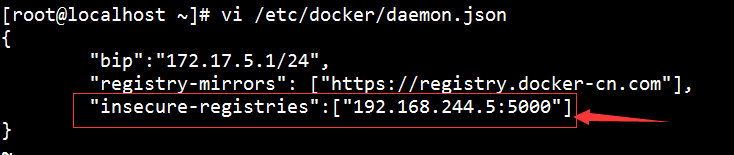


然后可以通过restful接口查看仓库中的镜像（当前仓库是空的）



### 配置http传输

私服默认只能使用https，需要配置开放http



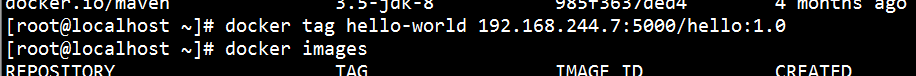
配置完毕重启下docker服务

systemctl daemon-reload

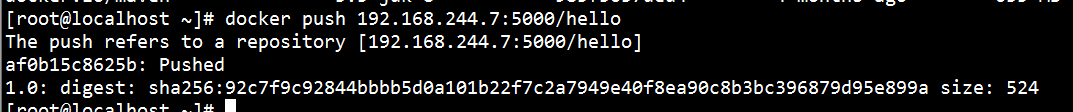
systemctl restart docker

### 私服仓库推送镜像

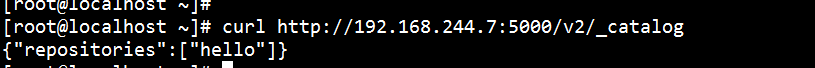
docker tag hello-world 192.168.244.7:5000/hello-world



docker push 192.168.244.7:5000/hello-world



查询镜像：http://192.168.244.5:5000/v2/\_catalog



查询hello版本： http://192.168.244.5:5000/v2/hello/tags/list



# Docker-Compose使用

当项目涉及容器较多时，需要一个管理容器的工具

## docker-compose安装

### curl方式安装

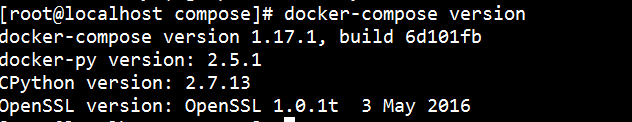
sudo curl -L https://github.com/docker/compose/releases/download/1.17.1/docker-compose-`uname -s`-`uname -m` > /usr/local/bin/docker-compose

### 增加可执行权限

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

### 查看版本

docker-compose version



## docker-compose.yaml命令

docker-compose的命令与docker命令极为相似，用法上没有区别，下面列出它特有的几种命令：

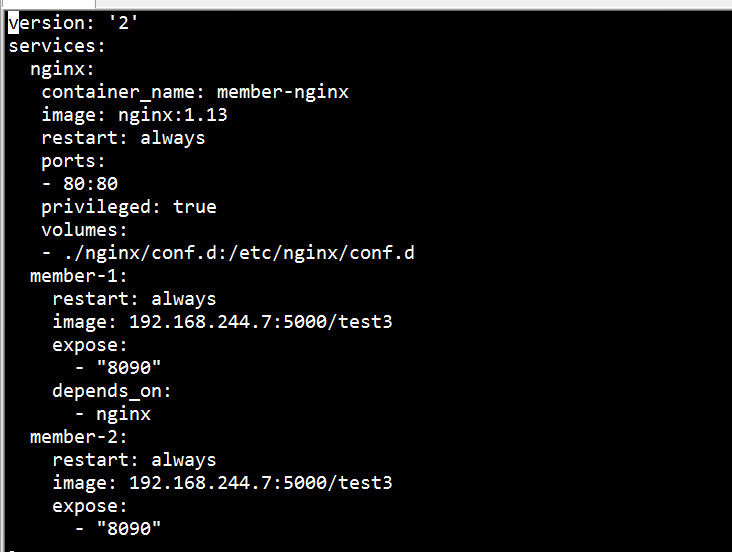
up 创建并启动容器：docker-compose up -d --scale 服务名=数字

---------- d表示后台运行，scale是表示对应的服务同时启动几个容器

down 停止并删除容器： docker-compose down

---------- 会停掉容器，并删除掉容器。如果不希望删除容器，请使用stop

## docker-compose实战

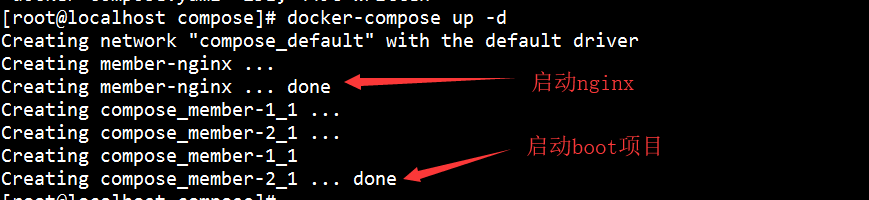


编写一个项目整体服务，一个网关nginx + springboot的集群，如上图

其中nginx服务，将配置文件挂载在主机当前项目目录的路径下：nginx/conf.d/



命令：docker-compose up -d



docker-compose up -d --scale member-1=2

把member-1服务启动两个容器

