**Docker操作手册**

|  |  |
| --- | --- |
| **作者姓名：** | **王杉** |
| **项目组：** | **联海创智** |
| **制作日期：** | **2018年12月06日** |

**Docker操作手册**

**摘要**

 Docker是一个开源项目，DocKer一次构建可放在任何地方就可以运行，不需要进行任何改变DocKer 就类似于一个容器。这个容器就好像咱们常用的虚拟机一样，当我们虚拟机里面安装过VS、SQL、浏览器 ......  之后咱们就把虚拟机镜像备份下来、等到下一次需要重新搭一个环境的时候，就可以省去很多事情了，直接把备份的虚拟机运行起来，该有的就都有了，省去了很多事情。

我们基于centos7以上的系统，Docker版本为1.13.1来编写Docker操作手册。后续的镜像我们基于jdk7,tomcat7编写Dockerfile制作Docker镜像。

**关键词：**Docker；Dockerfile；镜像

**目录**

总页数：19页

[1. 初识Docker 1](#_Toc23735)

[1.1 Docker简介 1](#_Toc30975)

[1.2 Docker的特征 1](#_Toc1693)

[1.3 Docker的原理 1](#_Toc8341)

[2. 安装 3](#_Toc2984)

[2.1 安装前提 3](#_Toc23340)

[2.2配置yum源 3](#_Toc18263)

[2.3安装 4](#_Toc3048)

[3.拉取镜像及验证 4](#_Toc5997)

[3.1拉取mysql5.7镜像 4](#_Toc19427)

[3.2启动容器 5](#_Toc16738)

[3.3验证mysql 5](#_Toc8308)

[3.3.1 容器内验证 5](#_Toc7168)

[4. 镜像 7](#_Toc6339)

[4.1查看官方Dockerfile格式 7](#_Toc26419)

[4.2 Dockerfile命令讲解 7](#_Toc13548)

[4.3构建镜像 8](#_Toc19225)

[4.3.1 前提条件 9](#_Toc19257)

[4.3.2创建Dockerfile 9](#_Toc211)

[4.3.3构建镜像 9](#_Toc1091)

[5. 容器 10](#_Toc31403)

[5.1启动容器 10](#_Toc3317)

[5.2验证 10](#_Toc20101)

[6. 仓库 10](#_Toc21954)

[6.1搭建私有仓库 10](#_Toc31922)

[6.2客户端上传镜像到仓库 10](#_Toc12442)

[6.3添加用户名和密码 10](#_Toc16041)

[6.4测试登录 12](#_Toc27480)

[6.5推送镜像 12](#_Toc9260)

[6.6镜像查询 12](#_Toc12986)

[6.7拉取镜像 13](#_Toc15072)

[7. harbor来管理仓库 14](#_Toc10801)

[7.1harbor介绍 14](#_Toc7013)

[7.2docker-compose安装 14](#_Toc18177)

[7.3Harbor 服务搭建 15](#_Toc20593)

[7.3.1下载解压 15](#_Toc12681)

[7.3.2配置Harbor  15](#_Toc28536)

[7.3.3启动 Harbor  17](#_Toc17840)

[7.3.4访问 17](#_Toc14315)

[7.3.4推送 18](#_Toc21443)

[7.3.5拉取 19](#_Toc29971)

# 初识Docker

## 1.1 Docker简介

Docker 是一个开源的应用容器引擎，让开发者可以打包他们的应用以及依赖包到一个可移植的容器中，然后发布到任何流行的[Linux](https://baike.baidu.com/item/Linux)机器上，也可以实现虚拟化，容器是完全使用沙箱机制，相互之间不会有任何接口。

## 1.2 Docker的特征

Docker是一个云开源项目，托管在github，任何人都可以通过 git clone 或者fork参与进来，本身是基于linux的容器技术，采用当时如日中天google新推出的Go语言实现。采用apache 2.0协议开源。

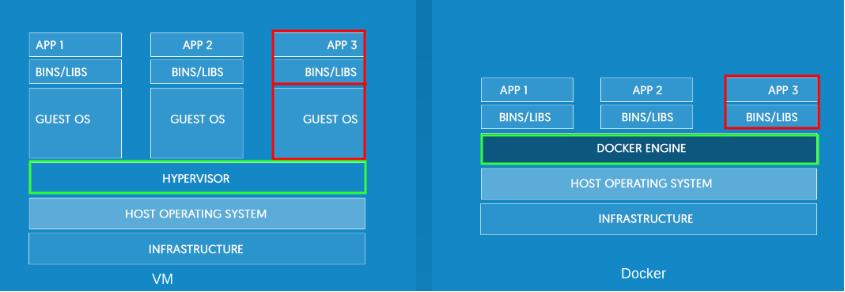
## 1.3 Docker的原理

容器有效的将单个操作系统管理的资源划分到孤立的组中，以便更好的在孤立的组之间平衡有冲突的资源使用需求。与虚拟化相比，这样既不需要指令级模拟，也不需要即时编译。容器可以在核心CPU本地运行指令，而不需要任何专门的解释机制。此外，也避免了准虚拟化（paravirtualization）和系统调用替换中的复杂性。简而言之就是，Docker是一个盒子，一个盒子装一个玩具，无论你丢在哪里，你给他通电(glibc)，他就能运行。你的玩具大就用大盒子，小玩具就用小盒子。

两个应用之间的环境是环境是完全隔离的，建立通信机制来互相调用。容器的创建和停止都十分快速（秒级），容器自身对资源的需求十分有限，远比虚拟机本身占用的资源少。

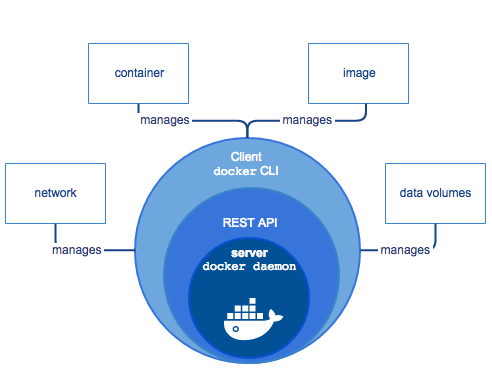
**Docker VS VM**

Docker与虚拟机(虚拟机)的区别可以看成：



左图是虚拟机的工作原理图，对资源进行抽象，着重体现在硬件层面的虚拟化上，这种方式增加了两场调用链，对性能的损耗比较大，而且还会占用大量的内存资源  
有图是Docker的工作原理图，属于OS级别的虚拟化，kernel通过创建多个镜像来隔离不同的app进程，由于kernel是是共享，而且本身linux image也不大，性能损耗几乎可以不计，而且内存占用也不大，大大节约了设备成本。

**Docker架构总览**



最核心的是 Docker Daemon我们称之为Docker守护进程，也就是Server端，Server端可以部署在远程，也可以部署在本地，因为Server端与客户端(Docker Client)是通过Rest API进行通信。  
  
docker CLI 实现容器和镜像的管理，为用户提供统一的操作界面,这个 客户端提供一个只读的镜像，然后通过镜像可以创建一个或者多个容器(container)，这些容器可以只是一个RFS(Root File System),也可以是一个包含了用户应用的RFS。容器在docker Client中只是一个进程，两个进程是互不可见的。  
用户不能与server直接交互，但可以通过与容器这个桥梁来交互，由于是操作系统级别的虚拟技术，中间的损耗几乎可以不计

**注：**  
CLI：command line interface。命令行接口.  
RFS：Root File System 根文件系统.

**Image & Container**

在docker中，我们重点关注的就是镜像和容器了。因为在实际应用中，我们封装好镜像，然后通过镜像来创建容器，在容器运行我们的应用就好了。而server端掌控网络和磁盘，我们不用去关心，启动docker sever 和 docker client都是一条命令的事情。后面会详细讲docker的启动过程。  
  
Image: 一个只读的镜像模板。可以自己创建一个镜像也可以从网站上下载镜像供自己使用。镜像包含了一个RFS.一个镜像可以创建很多容器。  
  
Container:由docker client通过镜像创建的实例，用户在容器中运行应用，一旦创建后就可以看做是一个简单的RFS，每个应用运行在隔离的容器中，享用独自的权限，用户，网络。确保安全与互相干扰  
  
两者在创建后，都是一堆layer的统一视角，唯一的却别是镜像最上面那一层是只读的，不可以修改，但是容器最上面一层是rw的，提供给用户操作  
  
repository:仓库，这个东西没有单独介绍不是因为它不重要，而是因为之前做个比较多的Android源码编译，所以这里就没有仔细往下看，大概就是一个镜像库，最大的是docker hub，类似于google 的aosp，当然也可以本地搭，比如mig事业群就有自己的repo。

# 安装

## 2.1 安装前提

Docker 运行在 CentOS 7 上，要求系统为64位、系统内核版本为 3.10 以上。使用命令 uname -r 查看当前操作系统内核，我当前系统内核版本是3.10.0-693.el7.x86\_64。

## 2.2配置yum源

由于安装centos后的默认yum源为centos的官方地址，所以在国内使用很慢甚至无法访问，所以一般的做法都是把默认的yum源替换成aliyun的yum源或者163等国内的yum源(下文介绍如何配置)。

但是以上的方法都是需要网络的，当没有网络的时候就无法使用了，所以还有一个常用的方法就是用Centos的iso镜像搭建本地yum源，这样安装软件的速度就会飞快，缺点是可能有些包没有。

我这里提供在线联网安装网易的和阿里的，二者选其一即可。

1. 备份

mv /etc/yum.repos.d/CentOs-Base.repo /etc/yum.repos.d/CentOs-Base.repo.bak

1. 配置yum源

网易云：wget -O /etc/yum.repos.d/CentOS-Base.repo <http://mirrors.163.com/.help/CentOS7-Base-163.repo>

阿里云yum源：

cd  /etc/yum.repos.d/

wget  <http://mirrors.aliyun.com/repo/Centos-7.repo>

mv Centos-7.repo CentOs-Base.repo

1. 更新yum，生成缓存

yum clean all

yum makecache

## 2.3安装

1. 安装yum 管理工具

# yum install -y yum-utils device-mapper-persistent-data lvm2

b.添加 docker yum 源

yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo

c.安装docker-ce

yum -y install docker-ce

启动docker并设置为开启自启动

systemctl start docker

systemctl enable docker

查看docker版本

docker version

# 3.拉取镜像及验证

## 3.1拉取mysql5.7镜像

Docker搜索镜像

docker search xxx

如：搜索mysql则docker search mysql

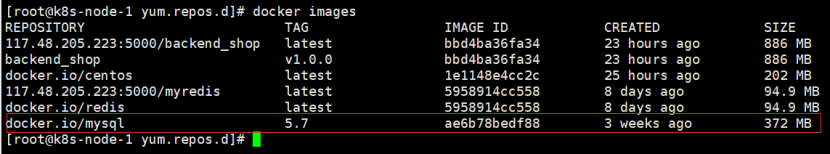
Docker拉取镜像到本地

docker pull xxxx:tag

如：搜索mysql则docker pull mysql,默认为mysql:latest,想要指定mysql版本，例如mysql5.7只需要docker pull mysql:5.7。

我们安装mysql5.7为例

拉取镜像后，我们执行docker images 就可以看到本地所有镜像信息



## 3.2启动容器

拉取镜像后，我们试着启动一下mysql

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

Docker run 启动容器

--name 指定容器名字

-p 这里是小写的p指定端口映射，第一个3306是宿主机的端口，第二个是容器的端口

-e 指定mysql用户名root密码是123456

-d 后台运行

后面mysql:5.7 镜像名：tag 也可以是imagesId

运行后，我们可以看到docker给我们container随机生成了一个容器id



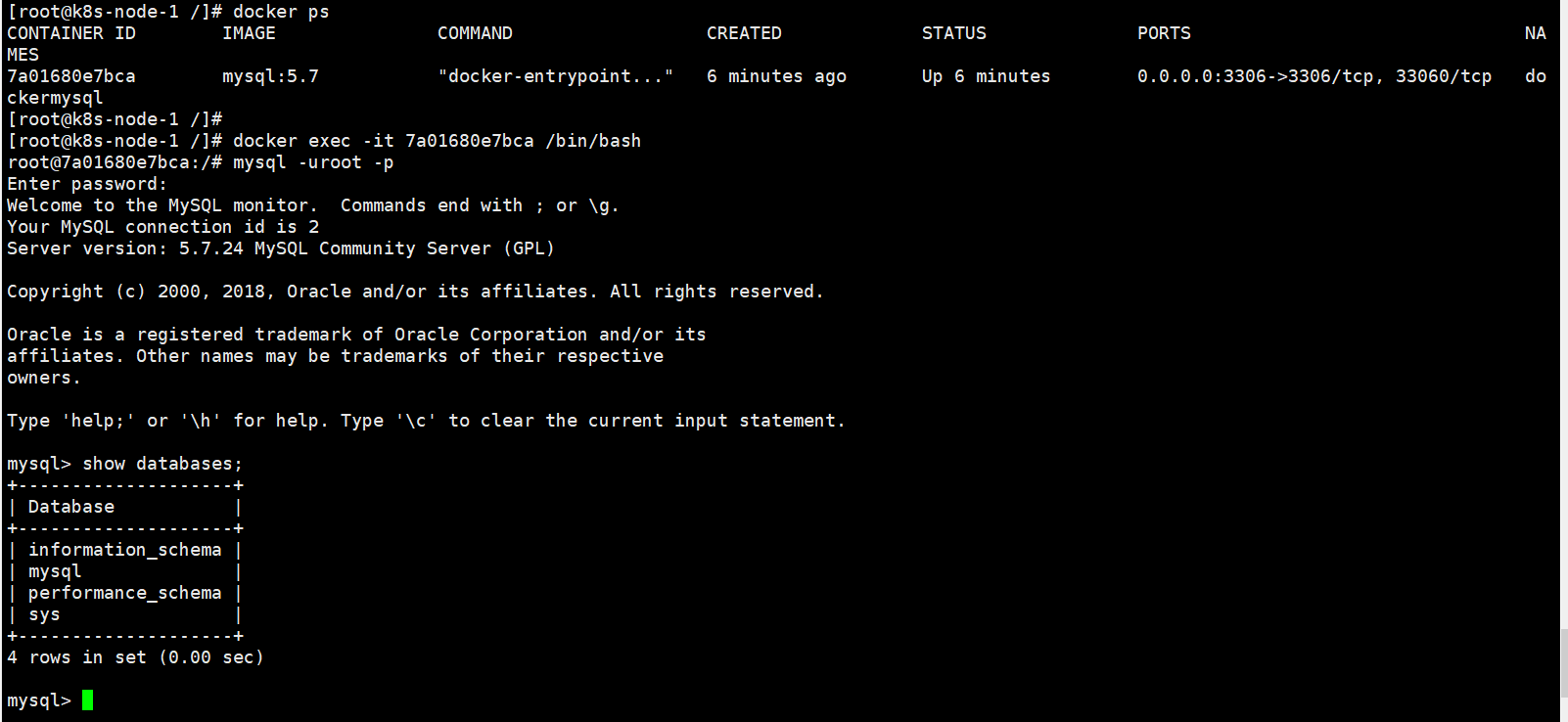
## 3.3验证mysql

### 3.3.1 容器内验证

docker ps 查看正在运行的容器，找到dockermysql对应的containerId

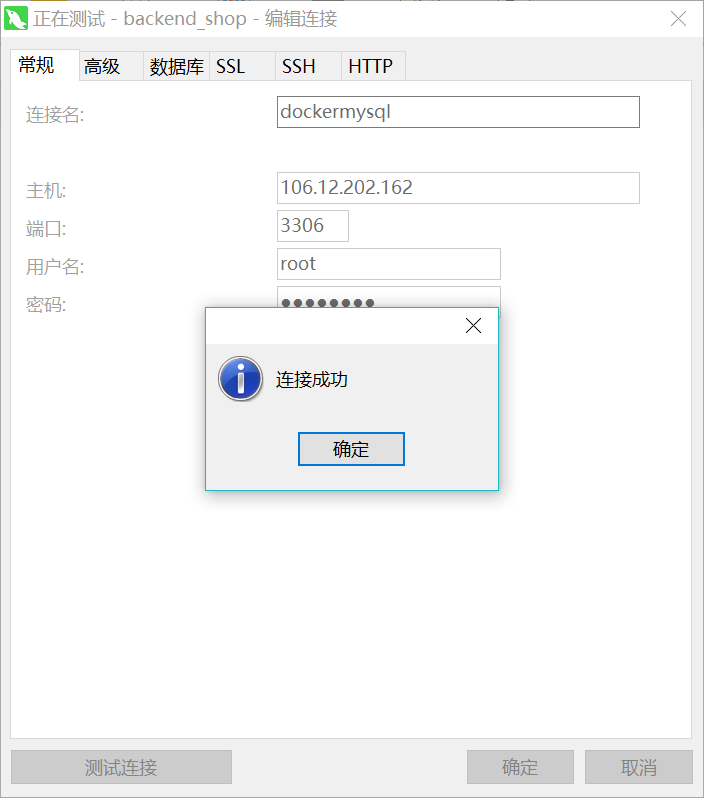
docker exec -it 7a01680e7bca /bin/bash 进入容器

mysql -uroot -p 回车输入密码，可以看到我们已经成功安装了mysql



3.3.1远程验证

我们使用navicat远程连接一下mysql



# 镜像

## 4.1查看官方Dockerfile格式

我们打开dockerhub官网<https://hub.docker.com/>

随便找一个镜像，我们以tomcat8.5为例





## 4.2 Dockerfile命令讲解

FROM

格式为 FROM <image>或FROM <image>:<tag>。

ENV

格式为 ENV <key> <value>。 指定一个环境变量，会被后续 RUN 指令使用，并在容器运行时保持。

MAINTAINER

格式为 MAINTAINER <name>，指定维护者信息。

WORKDIR

格式为 WORKDIR /usr/local。

为后续的 RUN、CMD、ENTRYPOINT 指令配置工作目录。

RUN

格式为 RUN <command> 或 RUN ["executable", "param1", "param2"]。

COPY

格式为 COPY <src> <dest>。

复制本地主机的 <src>（为 Dockerfile 所在目录的相对路径）到容器中的 <dest>。当使用本地目录为源目录时，推荐使用 COPY。

ADD

格式为 ADD <src> <dest>。

该命令将复制指定的 <src> 到容器中的 <dest>。 其中 <src> 可以是Dockerfile所在目录的一个相对路径；也可以是一个 URL；还可以是一个 tar 文件（自动解压为目录）。

EXPOSE

格式为EXPOSE PORT

CMD

支持三种格式

    CMD ["executable","param1","param2"] 使用 exec 执行，推荐方式；

    CMD command param1 param2 在 /bin/sh 中执行，提供给需要交互的应用；

    CMD ["param1","param2"] 提供给 ENTRYPOINT 的默认参数；

ENTRYPOINT

两种格式：

    ENTRYPOINT ["executable", "param1", "param2"]

    ENTRYPOINT command param1 param2（shell中执行）。

**注意**：CMD有多个时候，CMD最后一个有效，其他被覆盖，ENTRYPOINT 则是以追加的形式。我们这里讲解的是常用命令，不常用的还有VOLUME，USER,ONBUILD...

## 4.3构建镜像

### 4.3.1 前提条件

下载apache-tomcat-7.0.92.tar.gz,jdk-7u80-linux-x64.tar.gz并上传到服务器

上传一个可运行的war包到服务器。

这里我们直接用rz命令上传，网速相当于当前服务器网速。安装也简单

yum -y lrzsz

### 4.3.2创建Dockerfile

命令：vim Dockerfile

Dockerfile内容

|  |
| --- |
| FROM centos  ADD jdk-7u80-linux-x64.tar.gz /usr/local/  ADD apache-tomcat-7.0.92.tar.gz /usr/local/  RUN cd /usr/local/ && mv apache-tomcat-7.0.92 tomcat  RUN cd /usr/local/ && mv jdk1.7.0\_80 jdk  ADD backend\_shop.war /usr/local/tomcat/webapps/  ENV JAVA\_HOME /usr/local/jdk  ENV CLASSPATH $JAVA\_HOME/lib/dt.jar:$JAVA\_HOME/lib/tools.jar  ENV CATALINA\_HOME /usr/local/tomcat  ENV CATALINA\_BASE /usr/local/tomcat  ENV PATH $PATH:$JAVA\_HOME/bin:$CATALINA\_HOME/lib:$CATALINA\_HOME/bin  ENTRYPOINT ["/usr/local/tomcat/bin/catalina.sh","run"]  EXPOSE 8080 |

这里的apache-tomcat-7.0.92.tar.gz,jdk-7u80-linux-x64.tar.gz,backend\_shop.war

和Dockerfile必须在同一个路径下，因为上面的路径写的相对路劲。

### 4.3.3构建镜像

docker build -t backend\_shop:v1.0.0 .

-t tag简写 打标签+版本号

注意我后面有个点，代表当前路径。默认会去找当前的Dockerfile，如果你当前的不叫Dockerfile,你可以指定-f参数指定Dockerfile文件名。

如：docker build -f Dockerfile001 -t backend\_shop:v1.0.0 .

镜像制作完成后，我们查看镜像

docker images

backend\_shop v1.0.0 bbd4ba36fa34 5 seconds ago 886 MB

# 容器

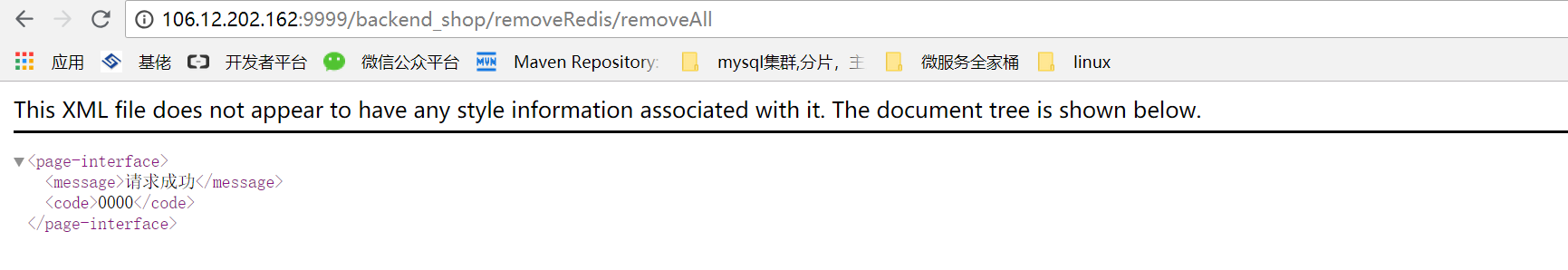
## 5.1启动容器

启动容器，指定我们自己制作的镜像

docker run --name="backend" -d -p 9999:8080 backend\_shop:v1.0.0

## 5.2验证

打开浏览器，访问我们的项目地址信息。



# 仓库

## 6.1搭建私有仓库

docker pull registry

docker run -d -v /usr/local/yangyao/docker/registry:/var/lib/registry -p 5000:5000 --restart=always --privileged=true --name registry registry:latest

## 6.2客户端上传镜像到仓库

打标签

sudo docker tag tomcat 仓库ip:5000/tomcat

推送到仓库

docker push 仓库ip:5000/tomcat

## 6.3添加用户名和密码

1. mkdir -p /usr/local/yangyao/docker/registry-var/auth/

创建用户名：yangyao 密码：123456

1. docker run --entrypoint htpasswd registry -Bbn yangyao 123456 >> /usr/local/yangyao/docker/registry-var/auth/htpasswd

设置配置文件，启用删除镜像功能（也可以不启用，看业务需要，修改 storage - delete - enable 为 false 即可）

mkdir -p /usr/local/yangyao/docker/registry-var/config

vim /usr/local/yangyao/docker/registry-var/config/config.yml

|  |
| --- |
| version: 0.1  log:  fields:  service: registry  storage:  delete:  enabled: true  cache:  blobdescriptor: inmemory  filesystem:  rootdirectory: /var/lib/registry  http:  addr: :5000  headers:  X-Content-Type-Options: [nosniff]  health:  storagedriver:  enabled: true  interval: 10s  threshold: 3 |

启动服务（这里将镜像路径映射到宿主机的 /usr/local/yangyao/docker/registry-var/ 文件夹下，可以根据需要修改）

|  |
| --- |
| docker run -d -p 5000:5000 --restart=always --name=registry -v /usr/local/yangyao/docker/registry-var/config/:/etc/docker/registry/ -v /usr/local/yangyao/docker/registry-var/auth/:/auth/ -e "REGISTRY\_AUTH=htpasswd" -e "REGISTRY\_AUTH\_HTPASSWD\_REALM=Registry Realm" -e REGISTRY\_AUTH\_HTPASSWD\_PATH=/auth/htpasswd -v /usr/local/yangyao/docker/registry-var/:/var/lib/registry/ registry |

客户端操作：

**开启 http 形式访问私有仓库模式**

vim /etc/docker/daemon.json

{"insecure-registries":["仓库ip:5000"]

}

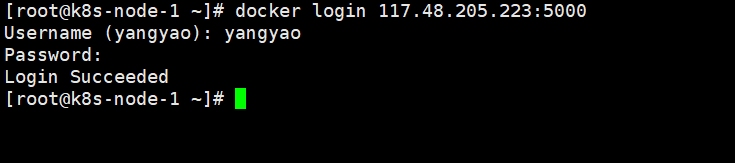
systemctl daemon-reload

systemctl restart docker

## 6.4测试登录

docker login 117.48.205.223:5000

按照提示输入用户名密码即可，看到login successed表示登录成功



## 6.5推送镜像

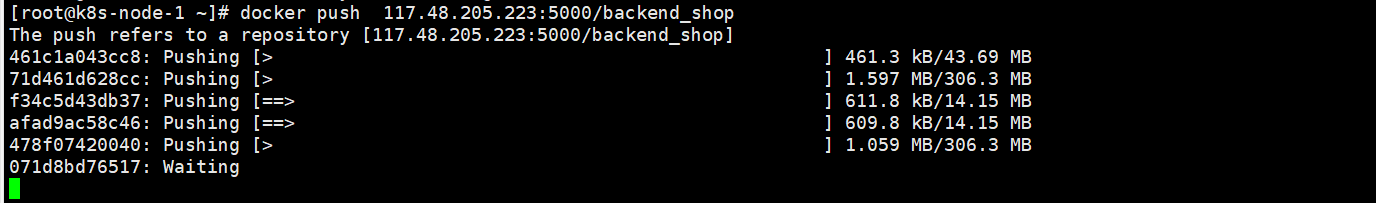
给镜像打标签

docker tag backend\_shop 117.48.205.223:5000/backend\_shop

推送镜像到仓库

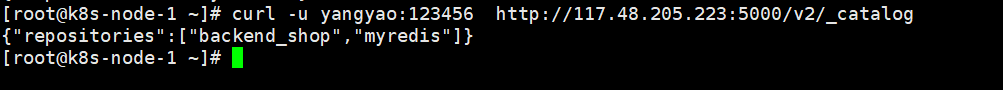
docker push 117.48.205.223:5000/backend\_shop

可以看到我们镜像正在上传



## 6.6镜像查询

curl -u 用户名:密码 <http://117.48.205.223:5000/v2/_catalog>

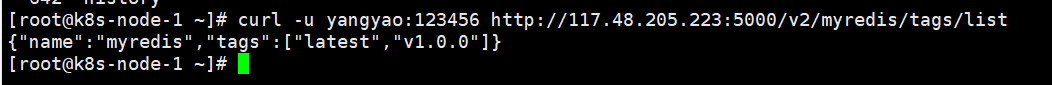


网页也可以访问，按照要求输入用户信息



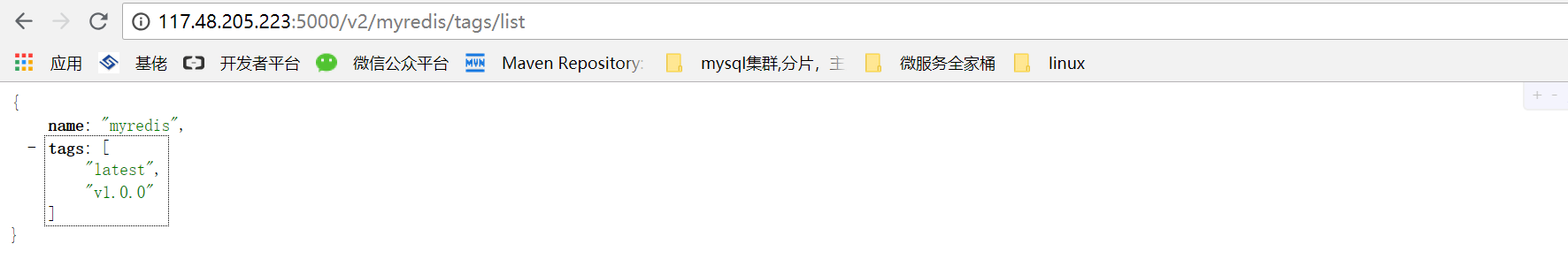
查看某个具体的镜像又那些版本

curl -u yangyao:123456 http://117.48.205.223:5000/v2/myredis/tags/list



网页执行方式

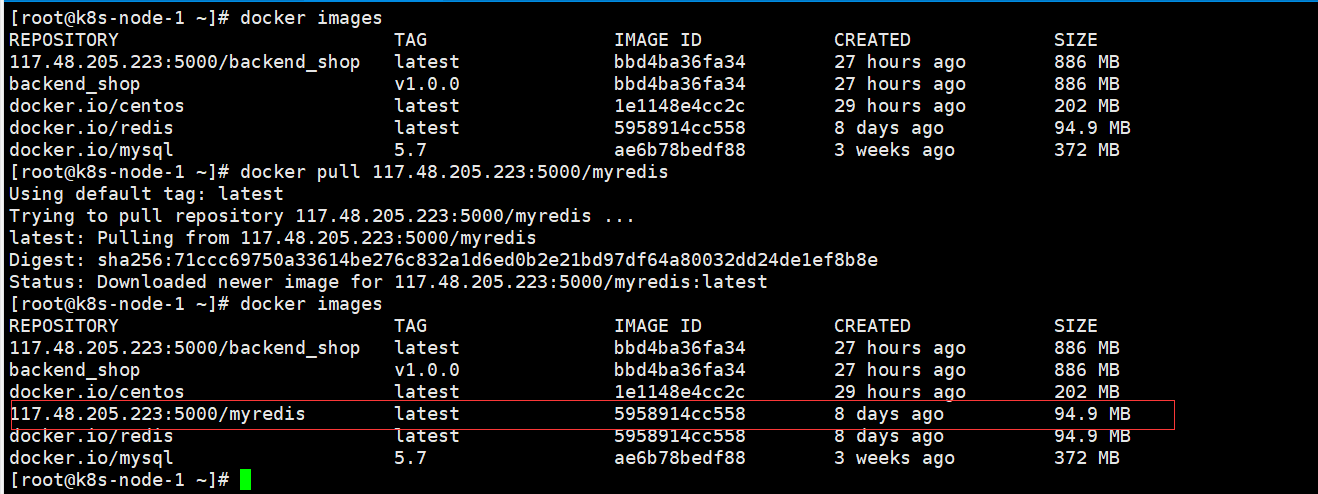
<http://117.48.205.223:5000/v2/myredis/tags/list>



## 6.7拉取镜像

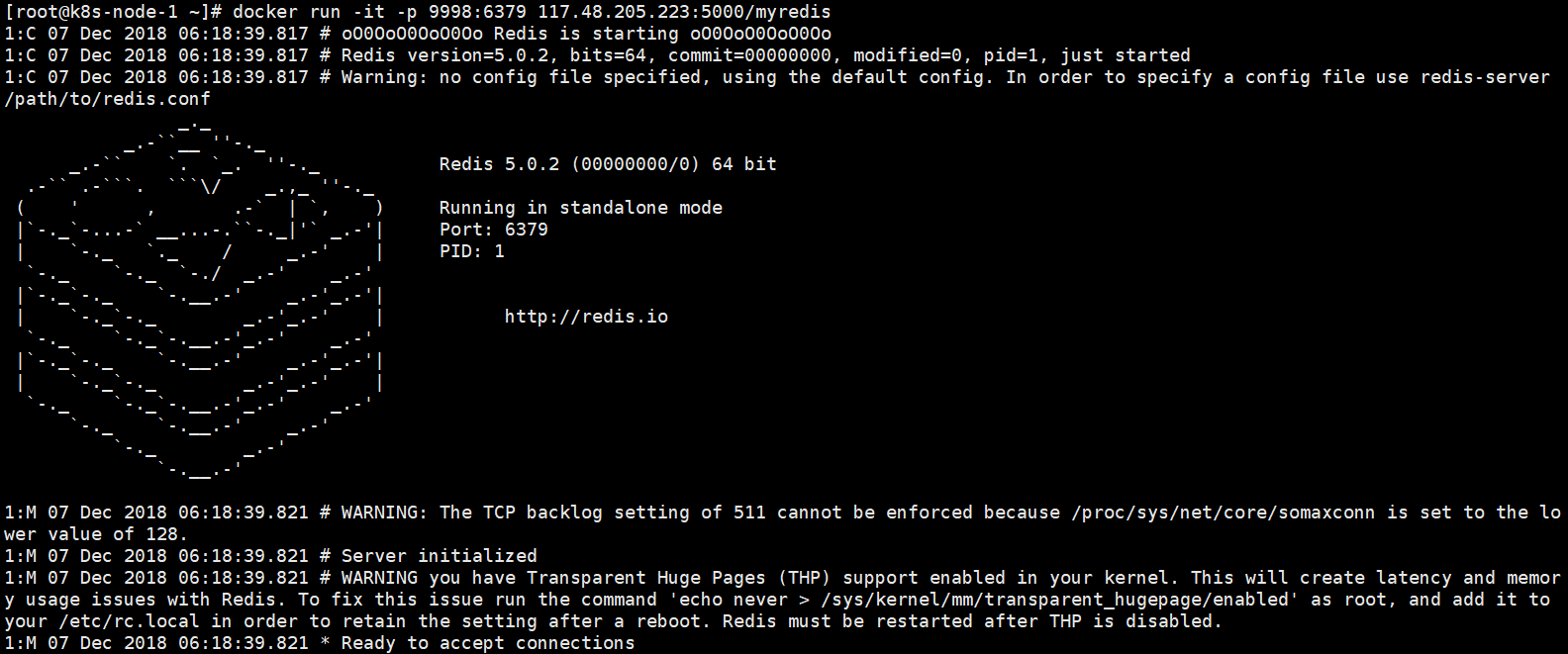
我这里服务器带宽较小，bankend\_shop上传较慢，我之前上传了一个myredis我们试着拉取做一下测试。

docker pull 117.48.205.223:5000/myredis

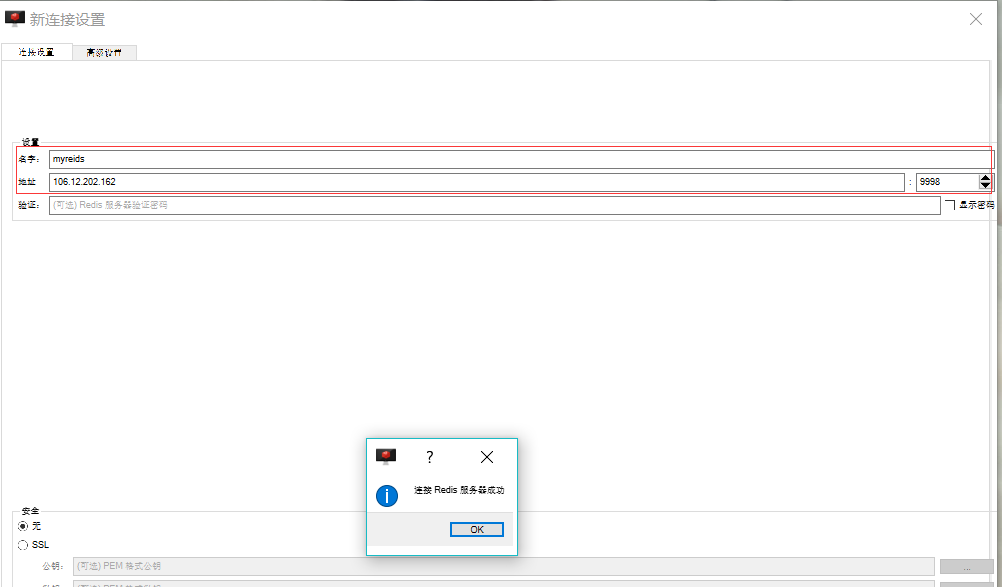


我们启动一下试试

docker run -it -p 9998:6379 117.48.205.223:5000/myredis



我们远程连接一下,可以看到我们自己制作的镜像已经成功上传下载并且正常启动了



# harbor来管理仓库

## 7.1harbor介绍

Docker容器应用的开发和运行离不开可靠的镜像管理，虽然Docker官方也提供了公共的镜像仓库，但是从安全和效率等方面考虑，部署我们私有环境内的Registry也是非常必要的。Harbor是由VMware公司开源的企业级的Docker Registry管理项目，它包括权限管理(RBAC)、LDAP、日志审核、管理界面、自我注册、镜像复制和中文支持等功能。

## 7.2docker-compose安装

安装docker-compose,我们这里安装1.13.0

curl -L https://github.com/docker/compose/releases/download/1.13.0/docker-compose-`uname -s`-`uname -m` > /usr/local/bin/docker-compose

对文件赋予可执行权限

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

验证

docker-compose --version

## 7.3Harbor 服务搭建

### 7.3.1下载解压

wget <https://github.com/vmware/harbor/releases/download/v1.1.2/harbor-online-installer-v1.1.2.tgz>

tar xvf harbor-online-installer-v1.1.2.tgz

### 7.3.2配置Harbor

解压缩之后，目录下回生成harbor.conf文件，该文件就是Harbor的配置文件。

|  |
| --- |
| ## Configuration file of Harbor  # hostname设置访问地址，可以使用ip、域名，不可以设置为127.0.0.1或localhost  hostname = 本机ip地址  # 访问协议，默认是http，也可以设置https，如果设置https，则nginx ssl需要设置on  ui\_url\_protocol = http  # mysql数据库root用户默认密码root123，实际使用时修改下  db\_password = root123  max\_job\_workers = 3  customize\_crt = on  ssl\_cert = /data/cert/server.crt  ssl\_cert\_key = /data/cert/server.key  secretkey\_path = /data  admiral\_url = NA  # 邮件设置，发送重置密码邮件时使用  email\_identity =  email\_server = smtp.mydomain.com  email\_server\_port = 25  email\_username = sample\_admin@mydomain.com  email\_password = abc  email\_from = admin <sample\_admin@mydomain.com>  email\_ssl = false  # 启动Harbor后，管理员UI登录的密码，默认是Harbor12345  harbor\_admin\_password = Harbor12345  # 认证方式，这里支持多种认证方式，如LADP、本次存储、数据库认证。默认是db\_auth，mysql数据库认证  auth\_mode = db\_auth  # LDAP认证时配置项  #ldap\_url = ldaps://ldap.mydomain.com  #ldap\_searchdn = uid=searchuser,ou=people,dc=mydomain,dc=com  #ldap\_search\_pwd = password  #ldap\_basedn = ou=people,dc=mydomain,dc=com  #ldap\_filter = (objectClass=person)  #ldap\_uid = uid  #ldap\_scope = 3  #ldap\_timeout = 5  # 是否开启自注册  self\_registration = on  # Token有效时间，默认30分钟  token\_expiration = 30  # 用户创建项目权限控制，默认是everyone（所有人），也可以设置为adminonly（只能管理员）  project\_creation\_restriction = everyone  verify\_remote\_cert = on |

### 7.3.3启动 Harbor

修改完配置文件后，在的当前目录执行./install.sh，Harbor服务就会根据当期目录下的docker-compose.yml开始下载依赖的镜像，检测并按照顺序依次启动各个服务。

### 7.3.4访问

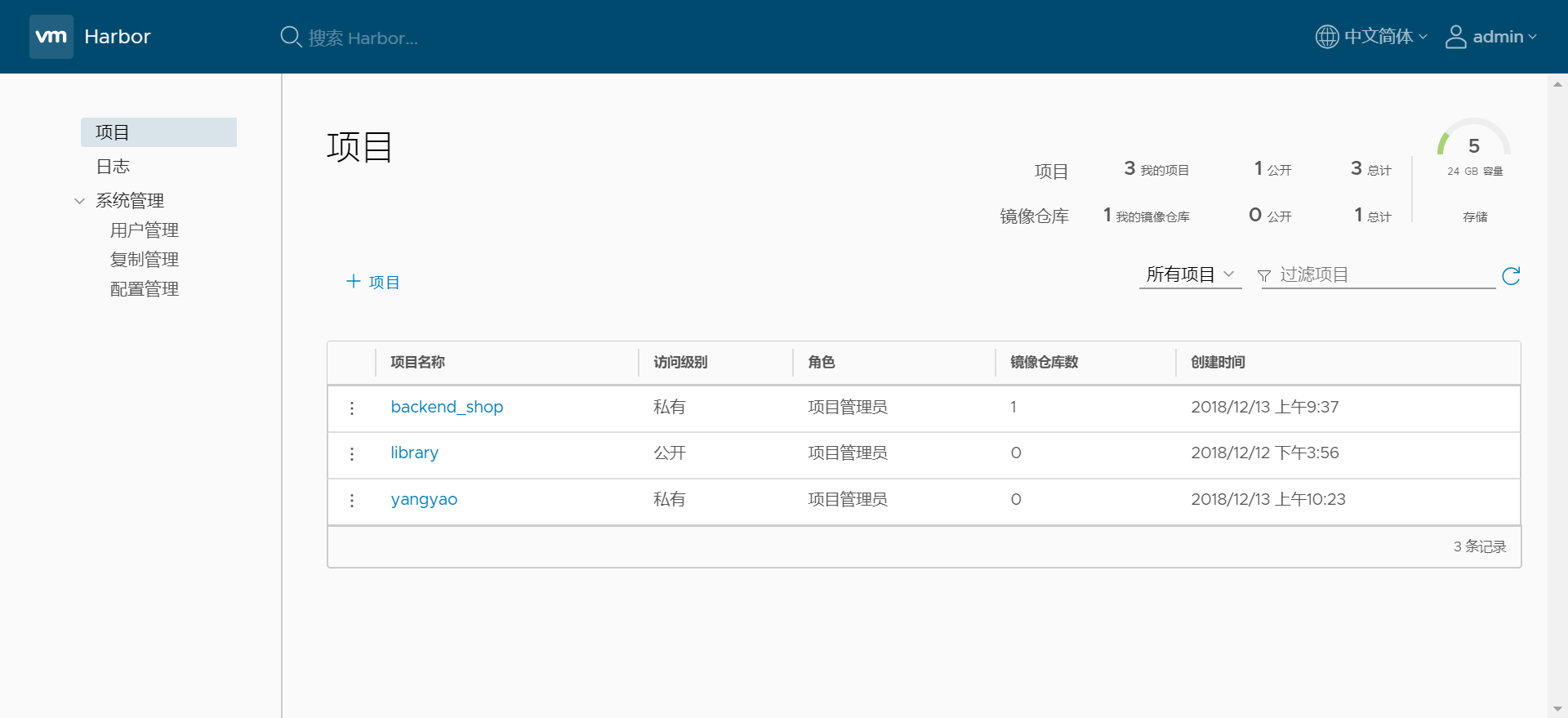
启动完成后，我们访问刚设置的hostname即可 [http://你的ip/](http://10.236.63.76/)，默认是80端口，如果端口占用，我们可以去修改docker-compose.yml文件中，对应服务的端口映射。

用户名admin

密码是你修改的密码



登录成功后页面如下图



我们可以看到系统各个模块如下：

项目：新增/删除项目，查看镜像仓库，给项目添加成员、查看操作日志、复制项目等

日志：仓库各个镜像create、push、pull等操作日志

系统管理

用户管理：新增/删除用户、设置管理员等

复制管理：新增/删除从库目标、新建/删除/启停复制规则等

配置管理：认证模式、复制、邮箱设置、系统设置等

其他设置

用户设置：修改用户名、邮箱、名称信息

修改密码：修改用户密码

注意：非系统管理员用户登录，只能看到有权限的项目和日志，其他模块不可见。

### 7.3.4推送

推送之前我们必须新建项目

新建后我们选择不公开，表示没有分配人员的情况下只有admin用户能推送，公开表示所有用户都可以推送。

同时客户端也需要配置

开启 http 形式访问私有仓库模式

vim /etc/docker/daemon.json

{"insecure-registries":["仓库ip"]

}

systemctl daemon-reload

systemctl restart docker

1、admin登录

$ docker login ip

2、给镜像打tag

$docker tag 镜像名ip/项目名/取的名字:版本号

3、push到仓库

$ docker push ip/项目名/名称：版本号

页面登录harbor查看



### 7.3.5拉取

docker pull ip/项目名/名称：版本号