# docker容器

目录

[docker容器 2](#_Toc5372)

[第一章 容器简介 2](#_Toc18929)

[1 容器技术概述 2](#_Toc26567)

[2 虚拟机和容器之间的区别 2](#_Toc15694)

[3 容器的特性 2](#_Toc16397)

[第二章 Docker容器的介绍 2](#_Toc15493)

[1、docker简介 2](#_Toc19697)

[2、Docker 的优点 2](#_Toc3684)

[3、Docker 的缺点 3](#_Toc9560)

[第三章 docker容器的使用 3](#_Toc29763)

[2、 实验-镜像的基本操作 3](#_Toc6941)

[1 、实验-安装docker 3](#_Toc16111)

[3 、实验-容器命令 4](#_Toc20683)

[4、 实验-镜像的导出和导入 4](#_Toc4077)

[5 、实验-通过Dockerfile构建镜像 4](#_Toc28245)

[Dockerfile 参数 4](#_Toc20960)

[（1）FROM 4](#_Toc29797)

[（2）RUN 5](#_Toc2327)

[RUN ：用来执行命令行命令，格式有两种： 5](#_Toc27439)

[（3）Dockerfile 上下文 5](#_Toc25996)

[（4） COPY 5](#_Toc20728)

[（5）ADD 6](#_Toc2524)

[（6）CMD 6](#_Toc3498)

[（7）ENTRYPOINT 7](#_Toc22272)

[（8）ENV 7](#_Toc7213)

[（9）ARG 8](#_Toc19151)

[（10）VOLUME 8](#_Toc24410)

[（11）EXPOSE 8](#_Toc12180)

[（12）WORKDIR 8](#_Toc1992)

[（14）HEALTHCHECK 8](#_Toc3380)

[（15）ONBUILD 9](#_Toc22725)

[6、 实验-通过容器创建镜像 9](#_Toc4291)

[7、 实验-修改镜像内容 10](#_Toc20013)

[8、 实验-httpd容器 10](#_Toc1393)

[9、 实验-映射文件和文件夹 10](#_Toc29196)

[10、 实验-mysql 11](#_Toc4665)

[11、 实验-容器调用容器 11](#_Toc26659)

[12 、实验-wordpress 11](#_Toc18879)

[13、 实验-搭建本地镜像仓库 11](#_Toc9934)

1. **容器简介**

**1 容器技术概述**

容器是一个允许我们在资源隔离的过程中，运行应用程序和其依赖项的 、轻量的 、操作系统级别的虚拟化技术， 运行应用程序所需的所有必要组件都打包为单个镜像，这个镜像是可以重复使用的。当镜像运行时，它是运行在独立的环境中，并不会和其他的应用共享主机操作系统的内存，CPU或磁盘。这保证了容器内的进程不会影响到容器外的任何进程。

**2 虚拟机和容器之间的区别**

虚拟机通常包括整个操作系统和应用程序。还需要与他们一起运行的虚拟机管理程序来控制虚拟机。  
 因为它们包括操作系统，因此它们的大小是几千兆字节（ 1千兆字节= 1GB）。使用虚拟机的一个缺点是它们需要几分钟的时间才能启动操作系统，和初始化它们托管的应用程序。另一方面，容器则是轻量级的，大部分是兆字节（1兆字节 = 1MB）大小的。容器的性能与虚拟机相比较，容器性能更好，可以立即启动。

**3 容器的特性**

容器技术已经成为应用程序分装和交付的核心技术

容器技术内核组成（Cgroups-资源管理 NameSpace-进程隔离）

由于是在物理机器上实施隔离、启动一个容器、可以像启动一个进程一样快速

1. **Docker容器的介绍**

**1、docker简介**

docker是完整的一套容器管理系统

Docker 可以让开发者打包他们的应用以及依赖包到一个轻量级、可移植的容器中，然后发布到任何流行的 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（类似 iPhone 的 app）,更重要的是容器性能开销极低。

**2、Docker 的优点**

1、简化程序：

Docker 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 机器上，便可以实现虚拟化。Docker改变了虚拟化的方式，使开发者可以直接将自己的成果放入Docker中进行管理。方便快捷已经是 Docker的最大优势，过去需要用数天乃至数周的 任务，在Docker容器的处理下，只需要数秒就能完成。

2、传统虚拟机需要给每个VM安装操作系统,docker不需要

Docker 镜像；Docker 镜像中包含了运行环境和配置，所以 Docker 可以简化部署多种应用实例工作。比如 Web 应用、后台应用、数据库应用、大数据应用比如 Hadoop 集群、消息队列等等都可以打包成一个镜像部署。

3、节省开支：

高性能必然高价格的思维定势。Docker 与云的结合，让云空间得到更充分的利用。不仅解决了硬件管理的问题，也改变了虚拟化的方式。

Docker 架构

Docker 使用客户端-服务器 (C/S) 架构模式，使用远程API来管理和创建Docker容器。

Docker 容器通过 Docker 镜像来创建。

**3、Docker 的缺点**

容器的隔离性没有虚拟化强

共用Linux内核，安全性有先天缺陷

SELinux难以驾驭

监控容器和容器排错是挑战

1. **docker容器的使用**

**2、 实验-镜像的基本操作**

docker镜像查询

docker search uos

docker pull uosproject/uos //下载镜像

docker images //查看镜像

docker save uosproject/uos:latest -o uos.tar //导出uos镜像为uos.tar

docker load -i nginx.tar //倒入本地镜像

docker rmi nginx //删除镜像

docker run -it uosproject/uos bash //以bash环境启动镜像

docker ps -a //#不加-a只显示运行状态的docker进程，每次都随机创建容器名和容器ID

docker logs 597b8cd3ca55 //通过容器ID来查询日志

docker tag uosproject/uos:latest uosproject/txuos:v1 //修改镜像的名称和标签,默认标签为latest

docker inspect uosproject/uos //查看镜像的底层信息

上传镜像 uosproject/txuos:v1

上传镜像之前需要在docker hub注册账户，例如注册账户为uos1

docker login //登录验证

docker tag uosproject/txuos:v1 uos1/uosproject:v1 //更改镜像名称

docker pull uos1/uosproject:v1

**1 、实验-安装docker**

apt update

apt install -y docker.io

systemctl restart docker

systemctl enable docker #docker开机启动

docker version

Client:

Version: 18.09.1

API version: 1.39

Go version: go1.11.6

Git commit: 4c52b90

Built: Tue, 03 Sep 2019 19:59:35 +0200

OS/Arch: linux/amd64

Experimental: false

Server:

Engine:

Version: 18.09.1

API version: 1.39 (minimum version 1.12)

Go version: go1.11.6

Git commit: 4c52b90

Built: Tue Sep 3 17:59:35 2019

OS/Arch: linux/amd64

Experimental: false

**3 、实验-容器命令**

docker stop 597b8cd3ca55 //通过容器ID关闭容器

docker start 597b8cd3ca55 //启动容器

docker attach 597b8cd3ca55 //连接到容器

docker ps //查看容器以关闭

docker start 597b8cd3ca55

docker exec -it 597b8cd3ca55 bash

docker ps //容器不会关闭

docker top 597b8cd3ca55 //查看容器进程列表

//过滤mac地址和ip地址

docker inspect -f '{{.NetworkSettings.MacAddress}}' 597b8cd3ca55

02:42:ac:11:00:02

docker inspect -f '{{.NetworkSettings.IPAddress}}' 597b8cd3ca55

172.17.0.2

**4、 实验-镜像的导出和导入**

docker save uosproject/uos > /tmp/uos.tar #将镜像导出为tar文件

docker rmi uosproject/uos

docker load < /tmp/uos.tar #删除原有的uos镜像，通过文件导入镜像

docker images #列出所有镜像

**5 、实验-通过Dockerfile构建镜像**

使用Dockerfile工作流程

mkdir build; cd build

vim Dockerfile

docker build -t imagename Dockerfile文件目录

mkdir uosdocker #创建Dockerfile文件目录

vim uosdocker/Dockerfile #编辑Dockerfile内容

**Dockerfile 参数**

**（1）FROM**

FROM指定基础镜像，必须是第一条指令

# 定制 nginx 镜像的 Dockerfile

FROM nginx

RUN echo '<h1>Hello,Docker!</h1>' > /usr/share/nginx/html/index.html

注： Docker Hub 上有很多高质量的服务类的官方镜像可以拿来直接使用，比如：nginx 、redis 、mysql 、php 、mongo \ tomcat 等，可以在其中找最符合的一个进行定制

        另外也有一些方便开发、构建、运行各种语言的镜像，比如：node 、python 、golang 等

        如果没有找到对应服务的镜像，官方镜像中还提供了一些更为基础的操作系统镜像，比如：ubuntu 、debian 、fedora 、centos 、 uos等，也可以利用这些操作系统提供的软件库

**（2）RUN**

RUN ：用来执行命令行命令，格式有两种：

     1, shell 格式： RUN <命令>，就像直接在命令行中输入的命令一样

     2, exec 格式：RUN ["可执行文件"，“参数1”，“参数2”]，更像是函数调用中的格式

warning:每一个RUN命令都会在 docker镜像中新建一层，所以应该尽量少用 RUN 命令，而且要在RUN 的最后要做必要的清除工作

# 构建层次太多，未做清理工作

FROM uos:stretch

RUN apt-get update

RUN apt-get install -y gcc libc6-dev make wget

RUN wget -O redis.tar.gz "http://download.redis.io/releases/redis-5.0.3.tar.gz"

RUN mkdir -p /usr/src/redis

RUN tar -xzf redis.tar.gz -C /usr/src/redis --strip-components=1

RUN make -C /usr/src/redis

RUN make -C /usr/src/redis install

# 一层构建，并在最后清理压缩包等缓存文件

FROM uos:stretch

RUN buildDeps='gcc libc6-dev make wget' \

&& apt-get update \

&& apt-get install -y $buildDeps \

&& wget -O redis.tar.gz "http://download.redis.io/releases/redis-5.0.3.tar.gz" \

&& mkdir -p /usr/src/redis \

&& tar -xzf redis.tar.gz -C /usr/src/redis --strip-components=1 \

&& make -C /usr/src/redis \&& make -C /usr/src/redis install \

&& rm -rf /var/lib/apt/lists/\* \

&& rm redis.tar.gz \

&& rm -r /usr/src/redis \

&& apt-get purge -y --auto-remove $buildDeps

**（3）Dockerfile 上下文**

# 构建新的镜像

# -t ：指定镜像名称和 tag

# . : 上下文，表示将本路径下的所有文件打包上传到 docker daemon,进行定制镜像

$ docker build -t nginx:v3 .

**（4） COPY**

用来从构建上下文目录中<原路径>的文件/目录复制到新一层镜像内的 <目标路径>位置，格式有两种：

    1，shell 格式：COPY [--chown=<user>:<group>] <原路径>...<目标路径>

    2，exec 合适：COPY[--chown=<user>:<group>] ["原路径1"，... "<目标路径>"]

原路径：可以是多个，甚至可以是通配符

目标路径：可以是容器内的绝对路径，也可以是相对于工作目录的相对路径(工作目录可以用 WORKDIR 指令来指定，不需要事先创建，会自动创建)

# 构建新的镜像

# -t ：指定镜像名称和 tag

# . : 上下文，表示将本路径下的所有文件打包上传到 docker daemon,进行定制镜像

$ docker build -t nginx:v3 .

note : COPY 会将原文件的各种数据都保留，比如 读、写、执行权限，可以通过 --chown=<user>:<group> 选项来改变文件的所属用户及所属组。

**（5）ADD**

ADD : 和 COPY 指令的功能，性质基本一致，也可以通过 --chown 改变文件所属用户和所属组，但是在 COPY 的基础上增加了一些功能：

    1，原路径为 URL ： Docker 会试图下载这个文件放到 目标路径去，默认下载后的文件权限为 600，如果想要修改权限或者下载的是压缩包，需要解压，则还需要额外的一层 RUN 进行调整，还不如直接用 RUN 指令用 wget 进行下载，处理权限，解压缩，然后清理无用文件更合理，所以该命令不常用，而且不推荐使用。

    2，原路径为 tar 压缩包 ： 如果压缩文件格式为 gzip , bzip2 以及 xz 的情况下，ADD 指令将自动解压这个压缩文件到 <目标路径> 去，只有此种情况适合使用 ADD 指令。

note: ADD 指令可能会使镜像构建缓存失效，从而可能会令镜像的构建变的比较缓慢，镜像构造缓存点击[这里查看](https://docs.docker.com/develop/develop-images/dockerfile_best-practices/" \t "/root/文档\\x/_blank)

**（6）CMD**

CMD : 和 RUN 指令相似，也是两种格式：

    1，shell 格式：CMD <命令>

    2，exec 格式 ： CMD ["可执行文件"，“参数1”，“参数2” ...]

    3，参数格式列表：在指定了 ENTRYPOINT 指令后，用 CMD 指定具体的参数

    CMD 指令用于指定默认的容器主进程的启动命令的，例如 ubuntu 默认的 CMD 是 bash ，我们也可以在容器运行时指定运行别的命令，如：

# 直接进入 bash

$ docker run -it ubuntu

# 修改默认的 CMD

# docker run -it ubuntu cat /etc/os-release

note1: 在指令格式上，一般推荐使用 exec 格式，这类格式在解析时会被解析为 JSON 数组，因此一定要用 双引号 “ 而不要使用单引号 。

# 如果执行

CMD echo $HOME

# 实际执行会变更为：

CMD ["sh" "-c" "echo $HOME"]

note2 : 容器的前台执行和后台执行问题

　　注：Docker 不是虚拟机，容器中的应用都应该以前台执行，而不能像虚拟机用 systemd 去启动后台服务，容器内没有后台服务的概念。例如：

# 错误代码

# 目的：启动 nginx 在后台以守护进程的形式在运行

CMD service nginx start

# 实际上执行

# sh 为主进程，执行完成进程退出，导致容器也会退出

CMD ["sh" "-c" "service nginx start"]

# 正确做法

# nginx :可执行文件

CMD ["nginx", "-g", "daemon off;"]

**（7）ENTRYPOINT**

ENTRYPOINT：格式和 RUN 指令格式一样，分为 exec 格式和 shell 格式，目的和 CMD 一样，都是在指定容器启动程序及参数;当指定了 ENTRYPOINT 后，CMD 的含义就发生了变化，不再是直接的运行其命令，而是将 CMD 的内容作为参数传给 ENTRYPOINT 指令，换句话说实际执行时，将变为： <ENTRYPOINT>"<CMD>"

用处 1 ： 让镜像变成向命令一样使用：

ENTRYPOINT 的使用

ENTRYPOINT ["curl", "-s", "https://ip.cn"]

用处 2 ： 应用运行前的准备工作：比如数据库配置，初始化工作，此时可以传 ENTRYPOINT 一个脚本，然后通过 CMD 指定参数，在脚本最后执行

FROM aduser

RUN groupadd -S redis && useradd -S -G redis redis

.......

ENTRYPOINT ["docker-entrypoint.sh"] 7

EXPOSE 6379

CMD ["redis-server"]

# docker-entrypoint.sh 脚本文件

#!/bin/bash

if [ "$1" = 'redis-server' -a "$(id -u)" = '0' ]; then

chown -R redis

exec su-exec redis "$0" "$@"

fi

exec "$@"

**（8）ENV**

ENV : 用来设置环境变量，格式有两种：

　　1，ENV <key> <value>

　　2，ENV <key1>=<value1> <key2>=<value2>...

在设置了环境变量之后，无论是后面的其它指令，如 RUN ，还是运行时的应用，都可以直接使用这里定义的环境变量

# 定义环境变量

ENV VERSION=1.0 DEBUG=ON \

NAME="Happy Feet"

**（9）ARG**

ARG : 构建参数，格式：

　　1，ARG <参数名>[=<默认值>]

构建参数和 ENV 的效果一样，都是设置环境变量，所不同的是，ARG 所设置的是构建环境的环境变量，在将来容器运行时是不会存在这些环境变量的。

**（10）VOLUME**

VOLUME：定义匿名卷，格式为：

　　1，VOLUME ["<路径1>”，”<路径2>"...]

　　2，VOLUME <路径>

　　之前说过，容器运行时应该尽量保持容器存储层不发生写操作，对于数据库类需要保存动态数据的应用，其数据库文件应该保存在卷中，为了防止运行时用户忘记将动态文件所保存目录挂载为卷，在 Dockerfile 中，我们可以事先指定某些目录挂载为匿名卷，这样在运行时如果用户不指定挂载，其应用也可以正常运行，不会向容器存储层写入大量数据。

# /data 目录会在运行时自动挂载为匿名卷

VOLUME /data

# 运行时也可以覆盖这个挂载设置

# 用 mydata 这个命名卷挂载到了 /data 这个位置，代替 Dockerfile 中的匿名卷的挂载配置

docker run -d -v mydata:/data xxxx

**（11）EXPOSE**

EXPOSE：声明端口，格式为：

　　EXPOSE <端口1> [<端口2>...]

　　该条指令是声明运行时容器提供的服务端口，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。这样声明带来两个好处：

　　1，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

　　2，在运行时使用随机端口映射，也就是 docker run -P 时，会自动随机映射 EXPOSE 的端口

note: 要将 EXPOSE 和在运行时使用 -p <宿主端口>:<容器端口> 区分开来。-p 是映射宿主端口和容器端口，就是将容器的对应端口服务公开给外界访问，而 EXPOSE 仅仅是声明容器打算使用什么端口而已，并不会在宿主进行端口映射。

**（12）WORKDIR**

WORKDIR : 指定工作目录，格式为：

　　WORKDIR <工作目录路径>

　　该条指令可以来指定工作目录（或者称为当前目录），以后各层的当前目录就被改为指定的目录，如果该目录不存在，则会自动建立。

USER

13，USER：指定当前用户，格式为：

　　USER <用户名>[:<用户组>]

　　该条指令和 WORKDIR 相似，都是改变环境状态并影响以后的层，WORKDIR 是改变工作目录， USER 是改变之后层的执行 RUN ，CMD 以及 ENTRYPOINT 这类命令的身份。如果以 root 执行的脚本，在执行期间希望改变身份，比如希望以某个已经建立好的用户来运行某个服务进程，不要使用 su 或者 sudo ,这些都需要比较麻烦的配置，而且在 TTY 缺失的情况下经常出错，建议使用 gosu 。

**（14）HEALTHCHECK**

HEALTHCHECK：健康检查，格式为：

　　HEALTHCHECK [选项] CMD <命令> ：设置检查容器健康状况的命令

　　HEALTHCHECK  NONE ： 如果基础镜像有健康检查指令，使用这行可以屏蔽掉其健康检查指令

options:

　　--interval=<间隔> ：两次健康检查的间隔，默认为 30s;

　　--timeout=<时长>： 健康检查命令运行超时时间，如果超过这个时间，本次健康检查就被视为失败，默认 30s;

　　--retries=<次数> ：  当连续失败指定次数后，则将容器状态视为 unhealthy ,默认 3 次；

return value:

　　0 : 成功

　　1：失败

　　2：保留（不要使用这个值）

**（15）ONBUILD**

ONBUILD：后构建指令，格式为：

　　ONBUILD <其它指令>

　　ONBUILD 是一个特殊的指令，它后面跟的是其它指令，比如 RUN，COPY 等，而这些指令，在当前镜像构建时并不会被执行。只有当以以前镜像为基础镜像，去构建下一级镜像的时候才会被执行。

实验用dockerfile构建镜像

mkdir uosdocker #创建Dockerfile文件目录

vim uosdocker/Dockerfile #编辑Dockerfile内容

FROM uosproject/uos #镜像名称

MAINTAINER uos #标记

RUN useradd uos #运行命令

RUN /bin/echo 'uos:uos'|chpasswd

RUN /bin/echo -e "LANG=\"en\_US.UTF-8\"" >/etc/default/locale

EXPOSE 22 #暴露22端口

CMD /use/sbin/sshd -D #执行命令

docker build -t uos:20 uosdocker/ #构建镜像，指定镜像名称和版本

docker images #列出刚刚创建的uos镜像

REPOSITORY TAG IMAGE ID CREATED SIZE

uos 20 e4644e68d11e 3 minutes ago

**6、 实验-通过容器创建镜像**

docker run -it --name "uos" uosproject/uos

echo uos > /uos

exit

docker commit -m ="uos update v1" uos uos:v20 #从uos容器创建 -m 描述信息

docker images #可以看到提交的镜像

docker run -it uos:v20

cat /uos #可以查看之前容器新建的uos文件内容

docker export -o /tmp/uos-`date +%Y%m%d`.tar uos #将容器导出到/tmp下

docker import -m="uos update v1" /tmp/uos-`date +%Y%m%d`.tar uos:v21

docker events --since=1467302400 #查看2016年7月1日后的相关事件

docker history uos:v20 #查看uos镜像创建的历史

docker rm uos #删除uos容器

docker rmi uos:v20 #删除uos:v20镜像

**7、 实验-修改镜像内容**

docker pull nginx:latest

docker run -it nginx:latest bash

docker ps -a

docker start 9725ff87872e

docker exec -it 9725ff87872e bash

#echo 'UOS web' > /usr/share/nginx/html/index.html

#nginx

开启新的终端执行以下命令：

docker inspect 9725ff87872e

apt install -y curl

curl 172.17.0.2 //可以访问到容器提供的web页面

**8、 实验-httpd容器**

docker pull docker.io/httpd:2.4 #下载httpd镜像

docker run -d -P --name=uoshttpd docker.io/httpd:2.4 #运行httpd容器

ps aux | grep httpd #查看httpd进程

docker top uoshttpd #查看uoshttpd容器内进程

docker inspect uoshttpd | grep IP #查看容器内部IP

http://172.17.0.2/

docker port uoshttpd #获取uoshttpd的映射端口

docker stop uoshttpd #停止httpd容器

docker rm uoshttpd #删除容器

docker run -d -p 18888:80 --name="uoshttpd" --restart=always docker.io/httpd:2.4

#-P自动分配端口，-p手动分配端口，-d后台运行，默认是协议为tcp可不写，udp协议可用-p 18888:80/udp格式，--restart=always重启后自动启动

**9、 实验-映射文件和文件夹**

docker run -d -p 18888:80 -v /uoshttpd/html:/usr/local/apache2/htdocs/ --name=uoshttpd docker.io/httpd:2.4

echo uos > /uoshttpd/html/index.html/

#真机的/uoshttpd文件夹映射到容器的/usr/local/apache2/htdocs/文件夹

docker exec -it uoshttpd bash #通过交互终端进入运行中的容器，后面可加脚本

cat /usr/local/apache2/htdocs/index.html

exit

docker cp uoshttpd:/usr/local/apache2/conf/httpd.conf /tmp/ #拷贝容器内文件到真机

cat /tmp/httpd.conf

docker stop uoshttpd

docker rm uoshttpd

docker rmi httpd:2.4

rm -rf /uoshttpd/

**10、 实验-mysql**

docker pull mysql:5.6

docker run -d -p 33060:3306 -e MYSQL\_ROOT\_PASSWORD=uos -e MYSQL\_USER=uos -e MYSQL\_PASSWORD=uos -e MYSQL\_DATABASE=uos -v /uosmysql:/var/lib/mysql --name=uosmysql docker.io/mysql:5.6

#设置mysql数据库root密码uos，新建用户uos（密码uos），新建数据库uos

配置mysql源 /etc/apt/source.list

deb [by-hash=force] https://packages.chinauos.cn/uos eagle main contrib non-free

deb-src https://packages.chinauos.cn/uos eagle main contrib non-free

deb [arch=amd64] https://download.docker.com/linux/debian stretch stable

deb http://mirrors.163.com/deepin/ unstable main contrib non-free

apt-get install -y mysql-server #安装mysql客户端

docker inspect qinmysql | grep IP

mysql -h172.17.0.2 -P33060 -uroot -puos

MySQL> show databases;

**11、 实验-容器调用容器**

docker run -d -p 33060:3306 -e MYSQL\_ROOT\_PASSWORD=uos -v /uosmysql:/var/lib/mysql --name=uosmysql docker.io/mysql:5.6

docker run -it --name=uoscentos --link=uosmysql:uosdb docker.io/centos --link 关联容器uosmysql

ping uosdb #容器内部通过容器IP进行通讯

**12 、实验-wordpress**

准备好数据库：

docker pull mysql:5.6

docker run -d --name db.wordpress -e MYSQL\_ROOT\_PASSWORD=admin mysql:5.6

-d: 启动的容器在后台运行

--name: 给启动的容器起个名字，这里叫做 db.wordpress

-e MYSQL\_ROOT\_PASSWORD: 这里是设置 MySQL 的 root 密码

:5.6: 指定 MySQL 的版本

docker run -d -p 80:80 --name wordpress --link db.wordpress:mysql wordpress:4.9.4

docker ps -a

docker inspect 06e09e14660b

浏览器打开 172.17.0.3

**13、 实验-搭建本地镜像仓库**

在uos1上

docker pull docker.io/registry

docker run -it --name=uosregistry docker.io/registry sh

#有些容器没有bash环境，只有sh环境

find / -name registry #查询registry仓库放在/var/lib/registry

exit

docker stop uosregistry

docker rm uosregistry

docker run --name=uosregistry --restart=always -d -p 5000:5000 -v /uosregistry:/var/lib/registry docker.io/registry

vim /etc/docker/daemon.json

{"registry-mirrors": ["https://registry.docker-cn.com"],"insecure-registries":["192.168.200.10:5000"]}

#必须把192.168.200.10:5000添加为信任的安全registry，否则报错，多个registry可在192.168.200.10:5000后面继续写，用逗号分隔

systemctl restart docker

docker tag docker.io/httpd:2.4 192.168.200.10:5000/uosimages/uoshttpd:2.4

docker push 192.168.200.10:5000/uosimages/uoshttpd:2.4

#将镜像推入docker.io/registry容器，作为内部仓库的镜像

curl 192.168.200.201:5000/v2/\_catalog #查看uos1仓库内的镜像

在uos2上安装docker环境

curl 192.168.200.201:5000/v2/\_catalog

firefox http://192.168.200.201:5000/v2/\_catalog

docker pull 192.168.200.201:5000/uosimages/uoshttpd:2.4 #下载本地仓库镜像很快

#如果报错Get https://192.168.200.201:5000/v2/\_catalog: http: server gave HTTP response to HTTPS client，需要在uos2上的/etc/docker/daemon.json文件中添加本地仓库信任