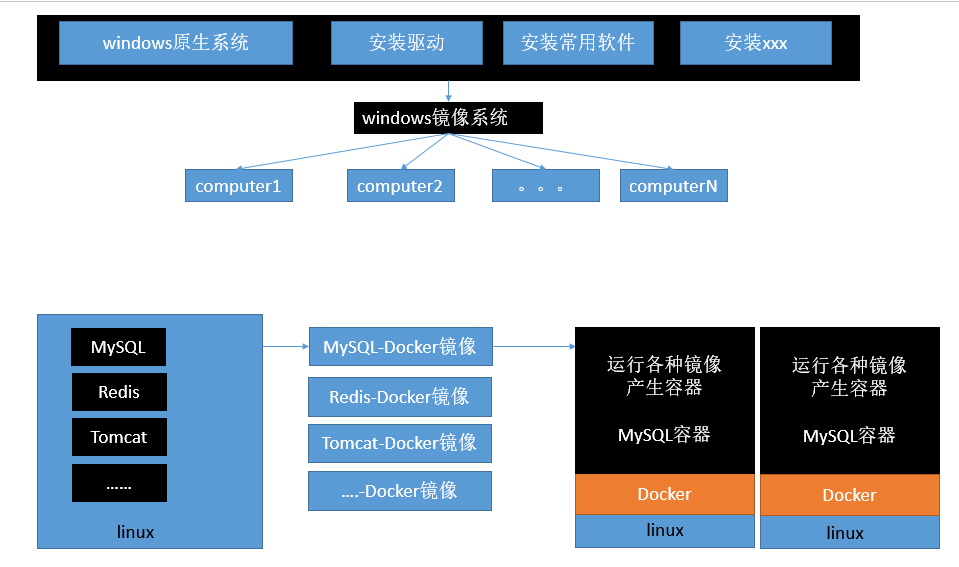
Docker教程

1. **简介**

Docker是一个开源的应用容器引擎；是一个轻量级容器技术；

Docker支持将软件编译成一个镜像；然后在镜像中各种软件做好配置，将镜像发布出去，其他使用者可以直接使用这个镜像；

运行中的这个镜像称为容器，容器启动是非常快速的。



1. **核心概念**

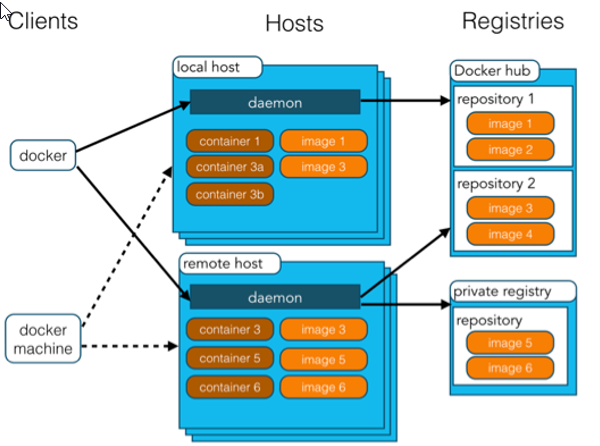
docker主机(Host)：安装了Docker程序的机器（Docker直接安装在操作系统之上）；

docker客户端(Client)：连接docker主机进行操作；

docker仓库(Registry)：用来保存各种打包好的软件镜像；

docker镜像(Images)：软件打包好的镜像；放在docker仓库中；

docker容器(Container)：镜像启动后的实例称为一个容器；容器是独立运行的一个或一组应用



1. **使用Docker的步骤**

1）、安装Docker

2）、去Docker仓库找到这个软件对应的镜像；

3）、使用Docker运行这个镜像，这个镜像就会生成一个Docker容器；

4）、对容器的启动停止就是对软件的启动停止；

1. **docker三要素**

Docker利用容器独立运行的一个或一组应用，容器是用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环境和运行在其中的应用程序。

镜像image：模板（相当于是Java中的类）

容器container：镜像的实例（相对于类的实例对象）

仓库repository：集中存放镜像文件的场所。仓库和仓库注册服务器是有区别的，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

仓库分为公开仓库和私有仓库两种形式

最大的公开仓库是Docker Hub（<https://hub.docker.com/>）

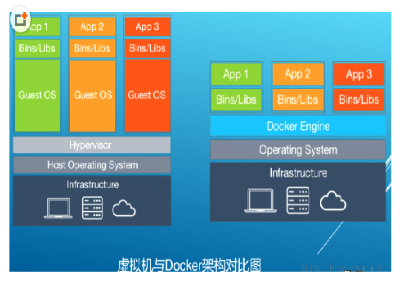
存放了数量庞大的镜像供用户下载，国内的公开仓库包括阿里云、网易云

1. **底层原理**

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。

1）、docker有着比虚拟机更少的抽象层，由于docker不需要Hypervisor实现硬件资源虚拟化，运行在docker容器上的程序直接使用的都是实际物理机的硬件资源。因此在CPU、内存利用率上docker将会在效率上有明显优势。

2）、docker利用的是宿主机的内核，而不需要Guest OS，因此，当新建一个容器时，docker不需要和虚拟机一样重新加载一个操作系统内核。仍而避免引导、加载操作系统内核返个比较费时费资源的过程。当新建一个虚拟机软件需要加载guest OS，返个新建过程是分钟级别的。而docker由于直接利用宿主机的操作系统，则省略了返个过程，因此新建一个docker容器只需要几秒钟。





1. **镜像原理**

镜像是一种轻量级、可执行的独立软件包，用来打包软件运行环境和基于运行环境开发的软件，它包含运行某个软件所需的所有内容：包括代码、运行时、库、环境变量和配置文件。

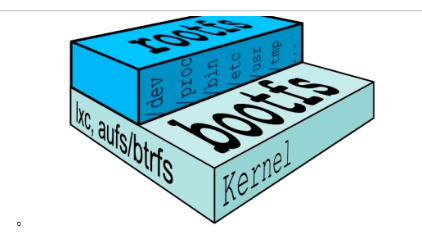
**UnionFS**：联合文件系统，union文件系统是一种分层、轻量级并且高性能的文件系统，它支持对文件系统的修改作为一次提交来一层层的叠加，同时可以将不同目录挂载到同一个虚拟文件系统（unite several directories into a single virtual filesystem）。Union文件系统是docker镜像的基础，镜像可以通过分层来进行继承，基于基础镜像（没有父镜像），可以制作各种具体的应用镜像。

**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

**Docker镜像加载原理：**

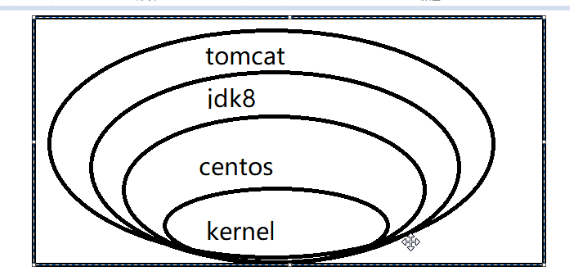
Docker的镜像实际上由一层一层的文件系统组成，这种层级的文件系统UnionFS，bootfs（boot file system）主要包含bootloader和kemel，bootloader主要是引导加载kemel，Linux刚启动时会加载bootfs文件系统，在docker镜像的最底层是bootfs。这一层与我们典型的Linux/Unix系统是一样的，包含boot加载和内核。当boot加载完成之后整个内核就都在内存中了，此时内存的使用权已由bootfs转交给内核，此时系统也会卸载bootfs。

Rootfs（root file system），在bootfs之上，包含的就是典型Linux系统中的/dev，/proc，/bin，/etc等标准目录和文件。Rootfs就是各种不同的操作系统发行版，比如Ubuntu，centos等等。





为什么Tomcat比centos的docker镜像大？



Tomcat镜像里面包含centos镜像。

Docker为什么采用分层结构？

原因：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。同时内存中也只需要加载一份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。

Docker镜像特点：docker镜像都是只读的，当容器启动时，一个新的可写层被加载到镜像的顶部，这一层通常被称作“容器层”，“容器层”之下的都叫“镜像层”。

1. **安装**

安装前还是必须安装gcc

yum –y install gcc

参考官方文档：

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

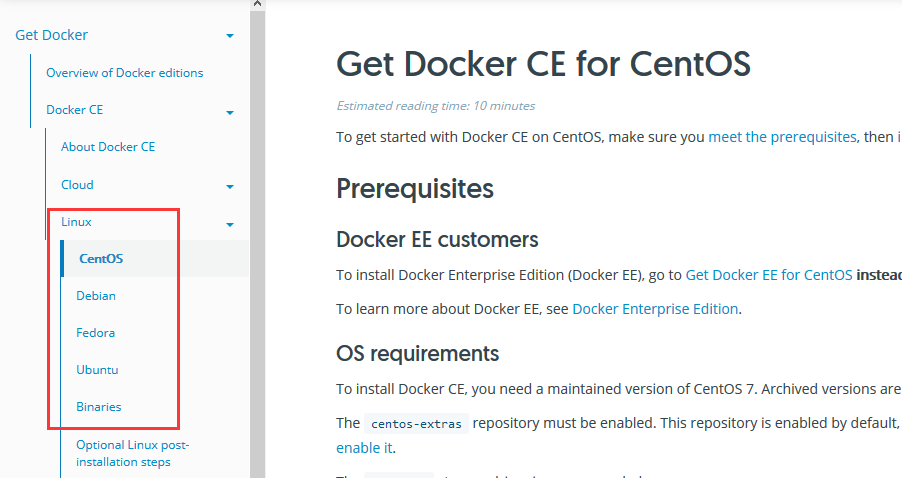
我们的Linux系统是centos，因此安装的是centos版本

此处要注意：由于官方网址的仓库连接比较慢，我们一般都使用阿里云的地址。此处跟官方文档不太一样。



更新yum软件包索引

yum makecache fast



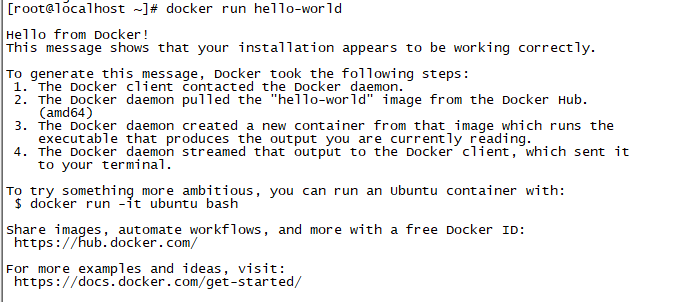
开机自启动

systemctl enable docker

启动

systemctl start docker

检测docker是否安装成功正常运行



**安装条件：**

Docker支持以下的centos版本

--centos7（64-bit），系统内核版本是3.10以上

--centos6.5(64-bit)或更高的版本，系统内核版本为2.6.32-431或更高版本。

1. **常用命令**

以下都是centos版本命令

1. **镜像加速**

提高下载镜像的速度

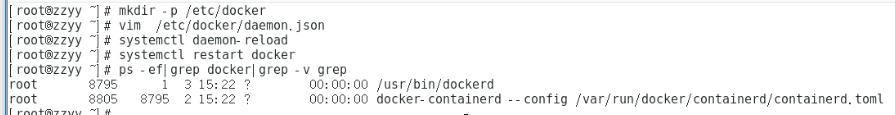
阿里云镜像加速：

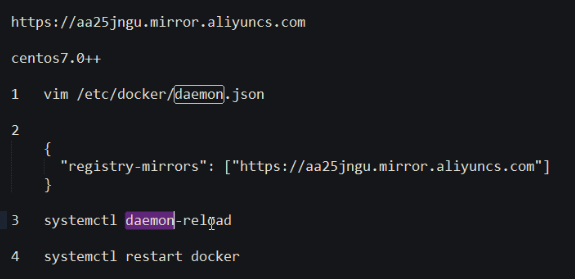
1）、注册一个属于自己的阿里云账号，获得加速器地址链接

https://www.aliyun.com/?utm\_content=se\_1000301881

2）、配置本机docker运行镜像加速器，重新启动docker后台服务

3）、Linux系统下配置完加速器需要检查是否生效





网易云加速：

1. **普通命令**

查看docker版本：docker -v

开机自启动：systemctl enable docker

启动：systemctl start docker

停止：systemctl stop docker

1. **帮助命令**

docker info:

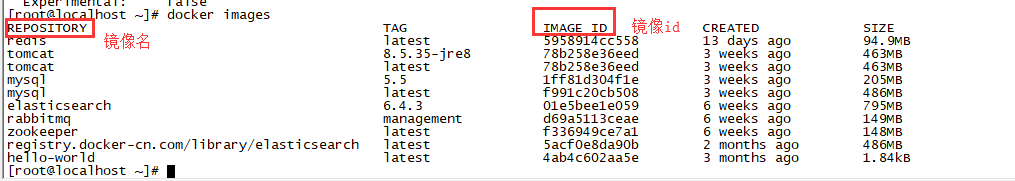
docker version:

docker help:

docker help images:显示images命令的相关信息

1. **镜像命令**

docker images:列出本地已经下载能够运行的存在的镜像，可以通过镜像名和镜像id对镜像进行操作



docker images –a:列出本地所有的镜像（包含中间映射层）

-q:只显示镜像id

-qa:也就是-a和-q的组合功能

-digests:显示镜像的摘要信息

--no-trunc:显示完整的镜像信息

docker search tomcat:从docker线上仓库搜索tomcat镜像

docker pull tomcat:version 从线上库拉取Tomcat镜像到本地，version是Tomcat的版本号，可以不加，默认是获取最新版本

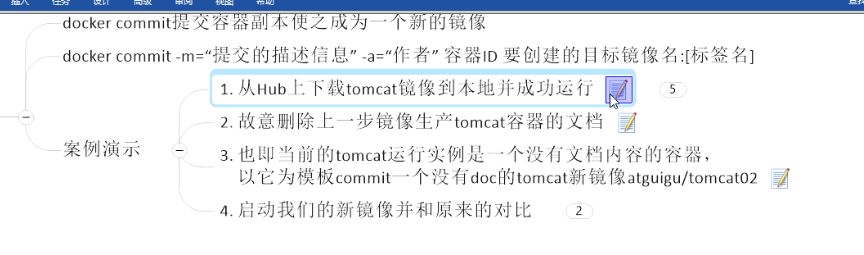
docker rmi 镜像id/镜像名：删除本地某个镜像，此处不加版本号则删除最新版本，而不是删除所有版本

docker rmi –f 镜像id：删除单个镜像

docker rmi –f 镜像id1:version 镜像id2:version 删除多个镜像

docker rmi –f $(docker images -qa)：删除全部镜像id

docker commit



docker commit -a="guohong" -m="my zookeeper design" 689f02727a10 guohong/myzk:1.0

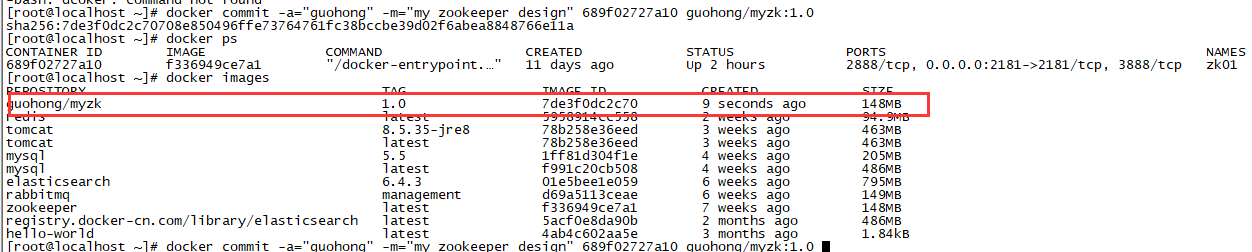
-a :作者名

-m:描述信息

689f02727a10：容器id

guohong/myzk：镜像名

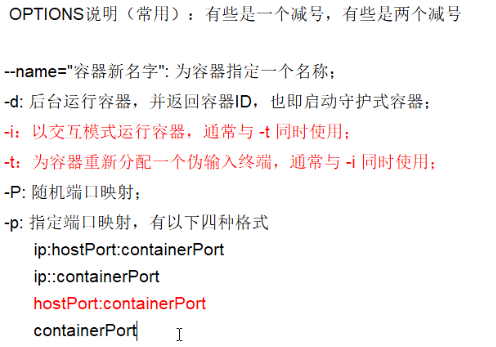
1.0：版本号



此时我们就可以使用我们自己的镜像了。

1. **本地镜像推送到阿里云**
2. **容器命令**

docker run –it imageid:在本地新建并进入容器，如果镜像不存在会先从仓库下载镜像再运行。这是启动交互式容器



docker run –it –p 8080:8080 tomcat 指定端口启动

docker run –d 镜像id

docker run –it centos(镜像名) /bin/bash（进入容器时，默认的路径地址，也就是进入容器后，默认会在该路径下，pwd可查看）

此时镜像以后台进程运行，但是我们docker ps查看docker进程的时候发现不存在该进程

原因：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top、tail），就会自动退出的。

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响应的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，会立即自杀因为它觉得他没事做了。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。

方式：

启动后不停在前台打印hello world



docker ps:查看docker正常里面运行的容器

docker ps –a：查看docker里面的所有容器，包含未运行的容器

退出容器：

exit:容器停止并退出

ctrl+P+Q：容器不停止并退出

启动容器：

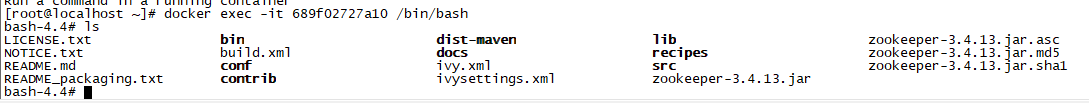
docker start 容器id

进入docker容器：

重新进入docker attach 容器id ：直接进入容器启动命令的终端，不会启动新的进程

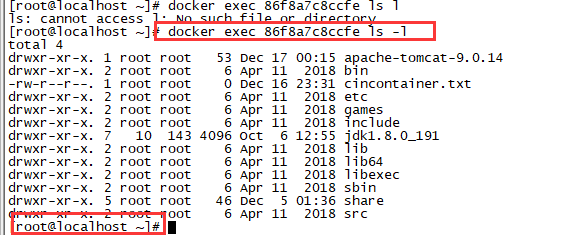
docker exec –it 容器id：在容器中打开新的终端，并且可以启动新的进程

docker exec -it 689f02727a10 /bin/bash



docker exec 86f8a7c8ccfe（容器id） ls -l

相当于在容器外部进入容器，在容器里面执行ls -l，此处执行完是没有进入到容器的。



重启容器

docker restart容器id

停止容器：

docker stop 容器id

强制停止容器

docker kill 容器id

删除已停止的容器

docker rm 容器id

docker inspect 容器id：查看容器内部细节

查看容器的日志

docker logs container‐name/container‐id

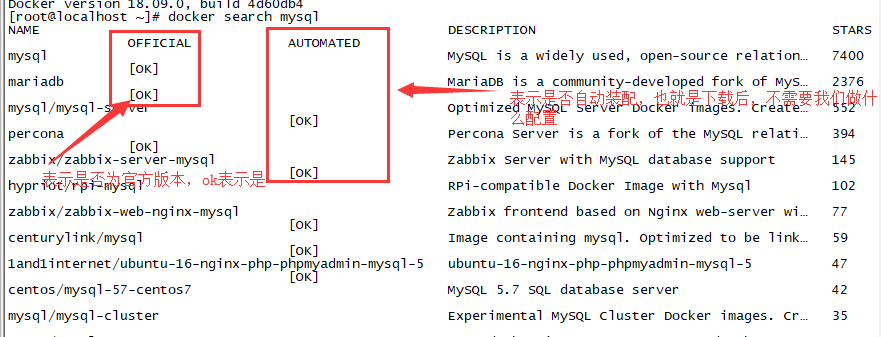
更多命令参看

<https://docs.docker.com/engine/reference/commandline/docker/>

docker top 容器id：查看进程号相关信息

1. **安装软件**

搜索docker中的镜像：docker search 软件名（mysql）

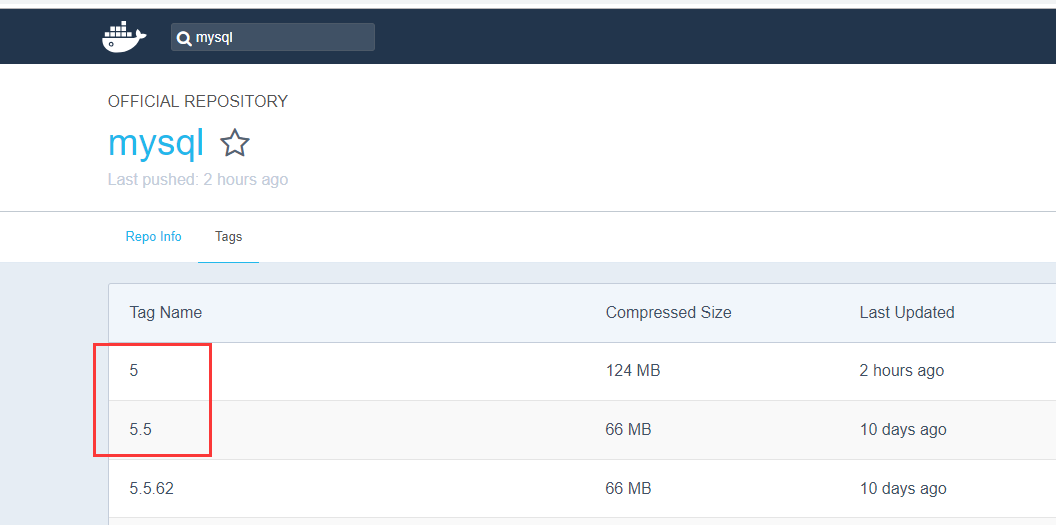


此处我们还可以通过网址搜索软件

<https://hub.docker.com/>，此处还可以查看版本号

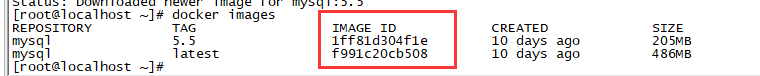
拉取安装包：docker pull mysql

根据版本号拉取：docker pull mysql:5.5



查看所有的镜像：docker images

删除指定镜像：docker rmi image-id（1ff81d304f1e）



我们拉取镜像后，就可以直接运行镜像

docker run --name mytomcat -d tomcat:8.5.35-jre8

--name:设置容器名称，随便取

mytomcat：名字随便取，

-d:后台运行

tomcat：容器名

8.5.35-jre8:tag也就是版本号

查看运行的容器：docker ps



此时如果我们启动Tomcat后，通过http是不能访问的，还必须给容器映射一个外部可以访问的端口号。

docker run -d -p 8081:8080 tomcat:8.5.35-jre8

‐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

查看所有的容器：docker ps ‐a

删除一个容器：docker rm 容器id

为了演示简单关闭了linux的防火墙

service firewalld status ；查看防火墙状态

service firewalld stop：关闭防火墙

1. **安装mysql**

docker search mysql

去此网站<https://hub.docker.com/>，查看mysql版本号

docker pull mysql:5.5拉取mysql镜像

我们也可以通过镜像加速网址：

https://www.docker-cn.com/registry-mirror

运行

docker run -p 3306:3306 --name mysql01 -e MYSQL\_ROOT\_PASSWORD=123456 -d mysql:5.5

-p:映射端口号

MYSQL\_ROOT\_PASSWORD：mysql的密码，默认为123456

5.5：版本号

docker run -p 3306:3306 --name mysql01

-v /zzyyuse/mysql/conf:/etc/mysql/conf.d

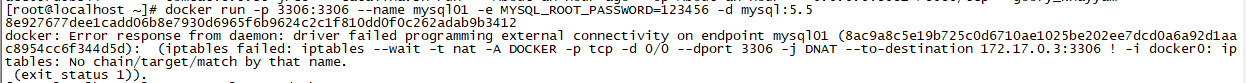
-v /zzyyuse/mysql/logs:/logs

-v /zzyyuse/mysql/data:/var/lib/mysql

-e MYSQL\_ROOT\_PASSWORD=123456 -d mysql:5.6

/zzyyuse/mysql/conf:/etc/mysql/conf.d：将容器内的配置信息位置/etc/mysql/conf.d映射到宿主机的/zzyyuse/mysql/conf下，也就是我们在宿主机的该位置修改配置信息，docker内的mysql就能读取到

注意：我在做mysql端口映射的时候，每次都出现错误，直到重启docker再重新运行mysql就可以了。

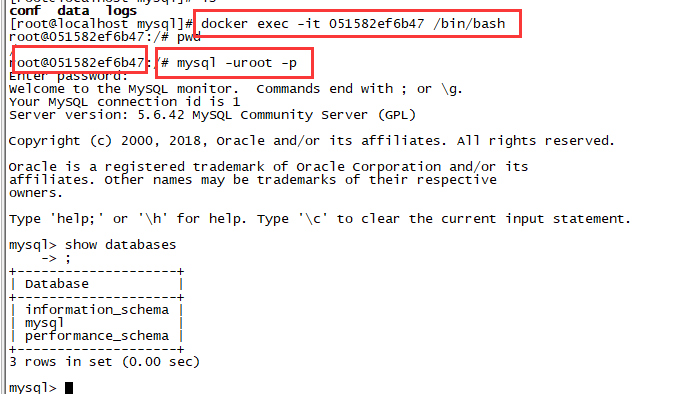




启动软件：docker start 容器id

测试mysql

docker exec -it 051582ef6b47（容器id）/bin/bash



数据备份：

docker exec 051582ef6b47 sh -c 'exec mysqldump --all-databases -uroot -p"123456"' > /zzyyuse/all-databases.sql

将docker容器中的mysql数据备份到宿主机的/zzyyuse/all-databases.sql文件中。

1. **安装Redis**

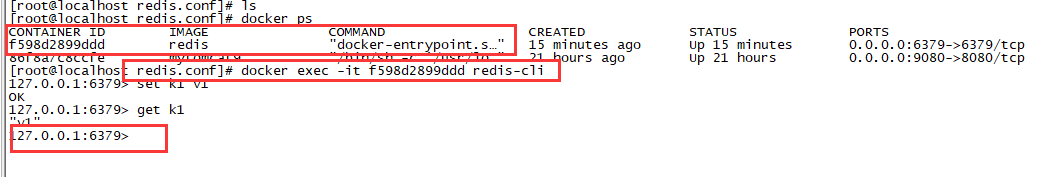
docker search redis

docker pull redis

docker run -p 6379:6379 -v /zzyyuse/myredis/data:/data -v /zzyyuse/myredis/conf/redis.conf:/usr/local/etc/redis/redis.conf -d redis redis-server /usr/local/etc/redis/redis.conf --appendonly yes

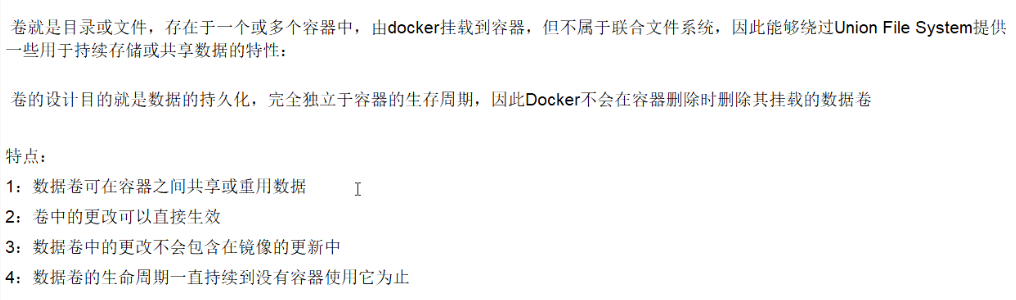
在主机/zzyyuse/myredis/conf/redis.conf目录下新建redis.conf文件

修改redis.conf配置信息，启动redis就能生效



1. **Docker容器数据卷**





容器内添加数据卷两种方式：直接命令添加；dockerFile添加

1. **命令添加数据卷**

输入命令：

拉取centos镜像

docker pull centos

添加数据卷

docker run -it -v /myDataVolume:/dataVolumeContainer centos

或

docker run -it -v /myDataVolume:/dataVolumeContainer:ro centos

说明：

ro:表示容器卷只读，也就是在容器卷里面不能写和修改，只能查看

/myDataVolume：宿主机文件夹，如果不存在则新建

/dataVolumeContainer：docker运行centos容器的文件夹，不存在则新建

centos：镜像名，可以换成镜像id

运行上面命令后可以发型在宿主机上的根目录下存在myDataVolume文件夹

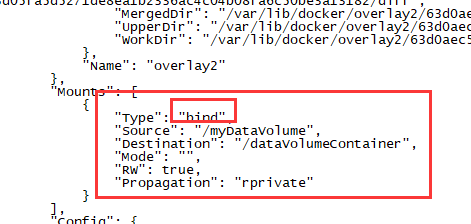


在docker容器里面存在dataVolumeContainer文件夹



查看是否挂载成功，还可以使用docker inspect 容器id命令

docker inspect 3db129e6ee56



此时我们在宿主机的myDataVolume文件夹下执行touch a.txt命令，创建a.txt文件。进入到容器dataVolumeContainer文件夹下也发现该a.txt文件也存在了。说明两边的文件数据是同步共享的。

当容器退出后（exit）,我们在宿主机上再新建b.txt文件，然后重新启动容器，docker start 容器id

docker start 3db129e6ee56

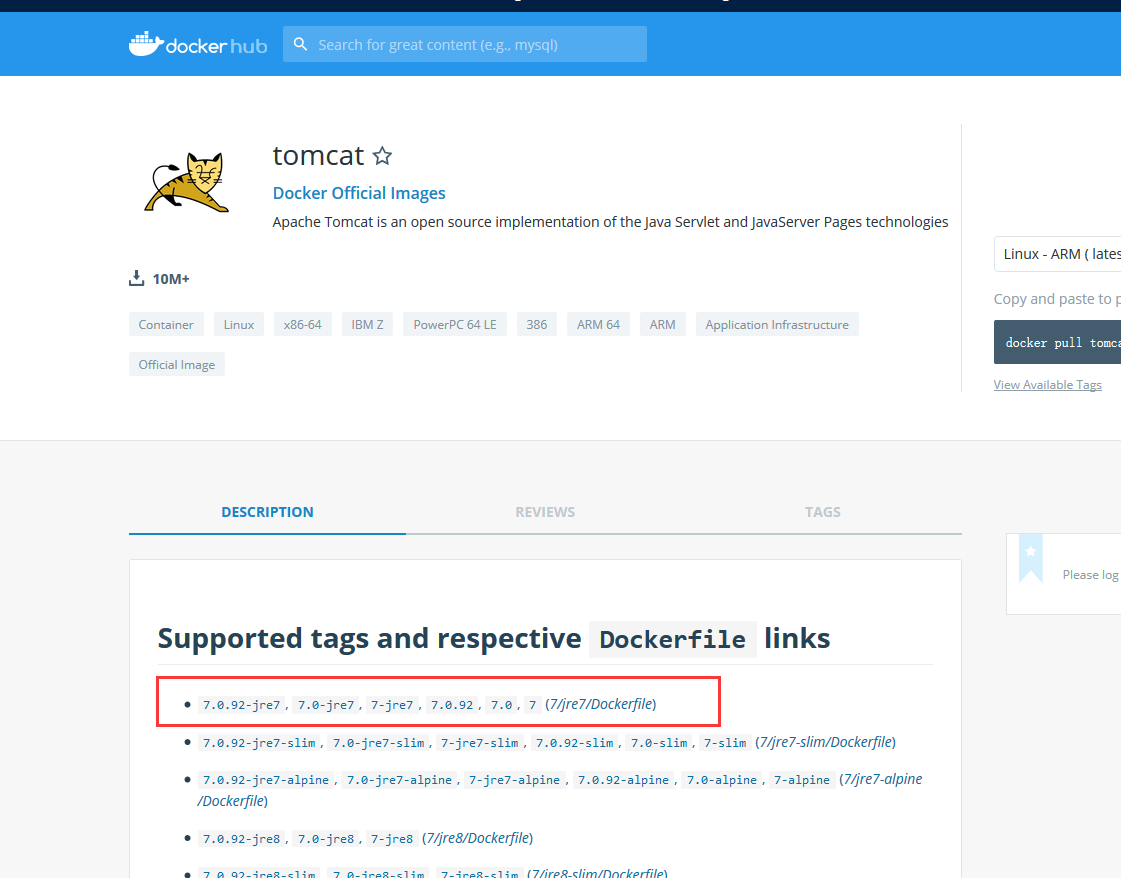
重新进入容器

docker attach 3db129e6ee56

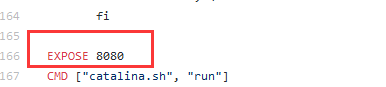
发现dataVolumeContainer文件夹下也存在b.txt文件。

1. **DockerFile添加容器卷**

Dockerfile就相当于Java的class文件。也就是docker编程，如下是去<https://hub.docker.com/>网址下搜索Tomcat，可以点击红圈的东西，里面就是dockerfile的编写语法。

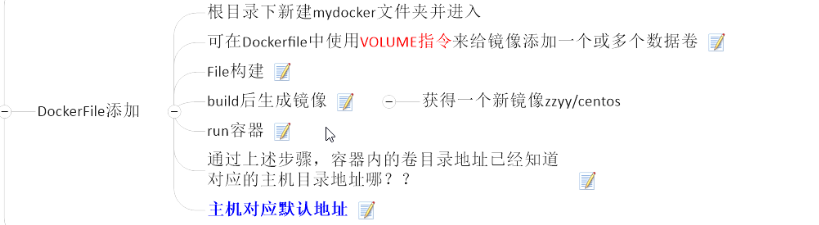






出于可移植和分享的考虑，用-v主机目录：容器目录这种方法（也就是命令添加数据卷方式）不能够直接在dockerfile中实现，由于宿主机目录是依赖于特定宿主机的，并不能够保证在所有宿主机上都存在这样的特定目录。

**具体步骤**：



在宿主机根目录下新建mydocker文件夹

mkdir mydocker

cd mydocker/

vim DockerFile,内容如下

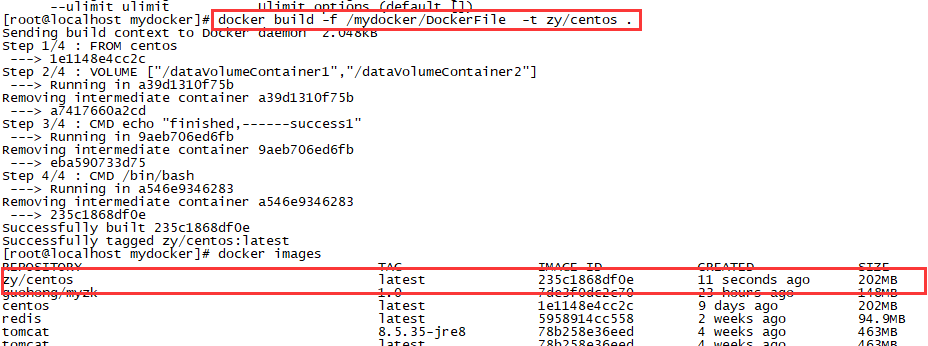
|  |
| --- |
| # volume test  FROM centos  VOLUME ["/dataVolumeContainer1","/dataVolumeContainer2"]  CMD echo "finished,------success1"  CMD /bin/bash |

构建

docker build -f /mydocker/DockerFile -t zy/centos .

zy/centos：镜像名

效果：



运行容器

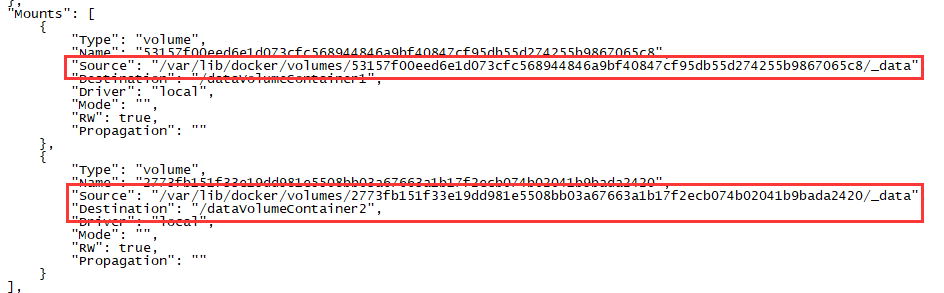
docker run -it zy/centos

进入容器，查看ls,容器内已经创建了2个容器卷



当我们没有指定宿主机的目录时，系统会指定默认的目录，可以通过如下查看默认目录

docker inspect 容器id



我们在容器内的dataVolumeContainer1容器卷下创建a.txt文件

进入如上宿主机地址

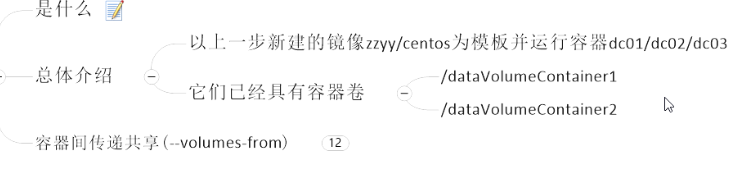
cd /var/lib/docker/volumes/53157f00eed6e1d073cfc568944846a9bf40847cf95db55d274255b9867065c8/\_data

ll命令就可以看到刚才新创建的a.txt文件了



1. **数据卷容器**

命名的容器挂载数据卷，其他容器通过挂载这个（父容器）实现数据共享，挂载数据卷的容器，称之为数据卷容器。也就是两个活动硬盘，实现数据传递共享的意思。



首先运行容器dc01

docker run -it --name dc01 zy/centos

此处的zy/centos是由上面DockerFile添加容器卷的时候添加的镜像。

此时在dc01容器内就能看到dataVolumeContainer1和dataVolumeContainer2文件夹

此时运行容器dc02

docker run -it --name dc02 --volume-from dc01 zy/centos

--volume-from dc01:相当于容器dc02继承自dc01容器，也就是dc01和dc02共享数据文件夹dataVolumeContainer1和dataVolumeContainer2。

此时在容器dc02内文件夹dataVolumeContainer1下添加a.txt文件，再进入容器dc01容器的dataVolumeContainer1文件夹，也能发现a.txt文件。

2个容器中的dataVolumeContainer1和dataVolumeContainer2数据是共享的

注：容器之间配置信息的传递，数据卷的生命周期一直持续到没有容器使用它为止，只要有容器使用，容器卷的数据都是最新不和共享的。

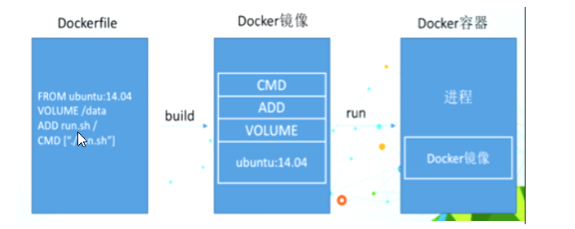
1. **DockerFile**

DockerFile是用来构建docker镜像的构建文件，是由一系列命令和参数构成的脚本。

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。

Dockerfile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

Docker容器：容器是直接提供服务的。



**步骤**：

1）、手动编写一个DockerFile文件，必须符合file规范

2）、有这个文件后，直接docker build 命令执行，获得一个自定义的镜像

3）、run 容器

1. **基础知识**

每条保留字指令都必须为大写字母且后面要跟随至少一个参数

指令按照从上到下，顺序执行

#表示注释

每条指令都会创建一个新的镜像层，并对镜像进行提交

1. **Docker执行dockerFile的流程**

Docker从基础镜像运行一个容器

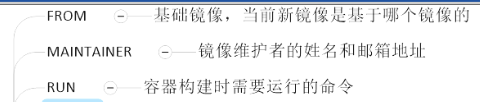
执行一条指令并对容器作出修改

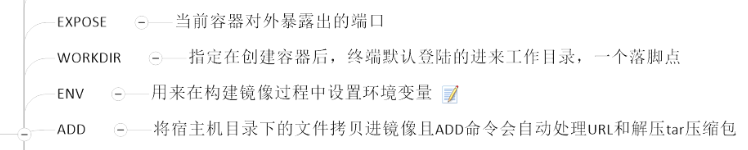
执行类似docker commit的操作提交一个新的镜像层

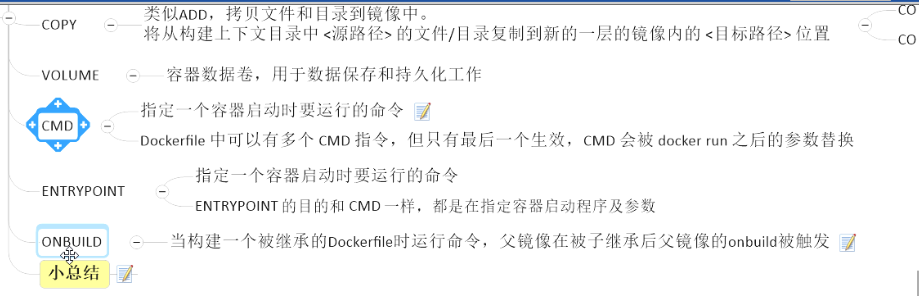
Docker再基于刚提交的镜像运行一个新容器

执行dockerfile中的下一条指令直到所有指令都执行完成

1. **保留字指令**

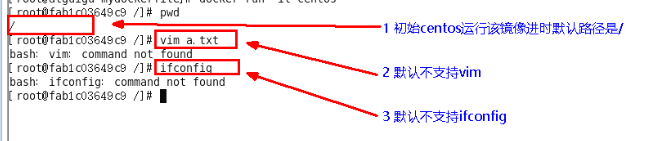






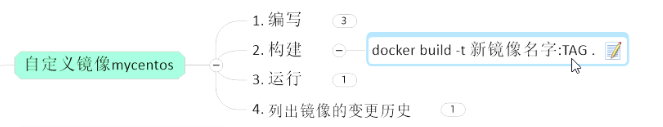
Base镜像scratch（相当于Java中的object类），docker hub中99%的镜像都是通过在base镜像中安装和配置需要的软件构建出来的。

1. **自定义镜像mycentos**



docker run –it zy/centos

自定义mycentos目的使我们自己的镜像具备如下：登陆后的默认路径、有vim编辑器、查看网络配置ifconfig支持



在/mydocker目录下新建DockerFile2文件

|  |
| --- |
| from centos  #设置环境变量  ENV mypath /tmp  #引用环境变量定义的变量  WORKDIR $mypath  #容器构建时需要的命令  RUN yum -y install vim  RUN yum -y install net-tools  #暴露出端口号  EXPOSE 80  CMD /bin/bash |

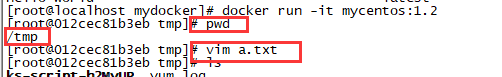
构建

docker build -f /mydocker/DockerFile2 -t mycentos:1.2 .

docker images就可以看见刚才我们构建的docker镜像

运行该镜像

docker run -it mycentos:1.2



此时发现pwd的时候，默认的目录就是我们设置的环境变量/tmp,vim命令也能正常使用，ifconfig也支持了。

1. **自定义Tomcat9**

步骤



mkdir -p zzyyuse/mydockerfile/tomcat9

cd zzyyuse/mydockerfile/tomcat9

touch c.txt

拷贝两个文件到tomcat9文件夹下

apache-tomcat-9.0.14.tar.gz

jdk-8u191-linux-x64.tar.gz

vim DockerFile

|  |
| --- |
| FROM centos  MAINTAINER zzyy<982452937@qq.com>  #把宿主机当前上下文的c.txt拷贝到容器/usr/local/路径下，并重命名为cincontainer.txt  COPY c.txt /usr/local/cincontainer.txt  #把Java与Tomcat添加到容器中  ADD apache-tomcat-9.0.14.tar.gz /usr/local  ADD jdk-8u191-linux-x64.tar.gz /usr/local  #安装vim编辑器  RUN yum -y install vim  #设置工作访问时候的WORKDIR路径，登录落脚点  ENV MYPATH /usr/local  WORKDIR $MYPATH  #配置Java和Tomcat环境变量  ENV JAVA\_HOME /usr/local/jdk1.8.0\_191  ENV CLASSPATH $JAVA\_HOME/lib/dt.jar:$JAVA\_HOME/lib/tools.jar  ENV CATALINA\_HOME /usr/local/apache-tomcat-9.0.14  ENV CATALINA\_BASE /usr/local/apache-tomcat-9.0.14  ENV PATH $PATH:$JAVA\_HOME/bin:$CATALINA\_HOME/lib:$CATALINA\_BASE/bin  #容器运行时监听的端口  EXPOSE 8080  #启动时运行Tomcat  #ENTRYPOINT ["/usr/local/apache-tomcat-9.0.14/bin/startup.sh"]  #CMD ["/usr/local/apache-tomcat-9.0.14/bin/catalina.sh","run"]  CMD /usr/local/apache-tomcat-9.0.14/bin/startup.sh && tail -F /usr/local/apache-tomcat-9.0.14/bin/logs/catalina.out |

构建

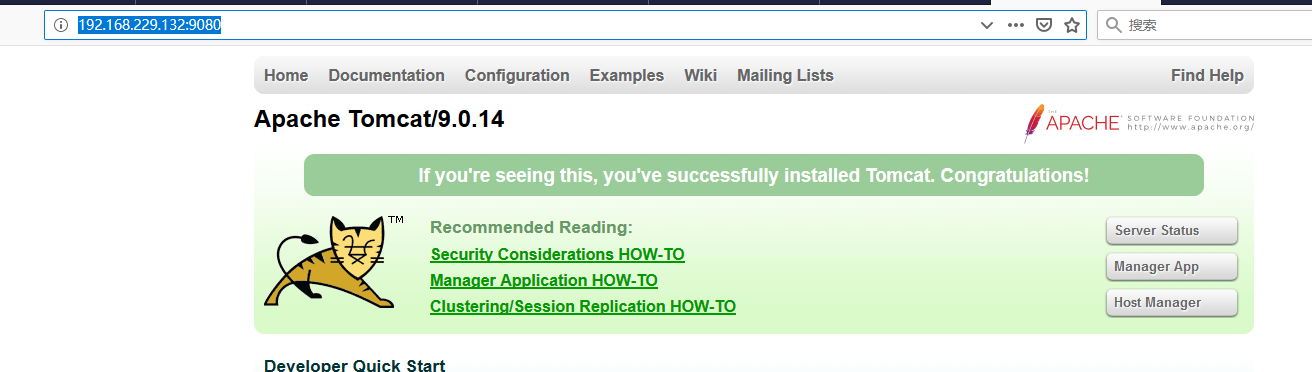
docker build -f DockerFile -t mytomcat9 .

运行

docker run -d -p 9080:8080 --name myt9 -v /zzyyuse/mydockerfile/tomcat9/test:/usr/local/apache-tomcat-9.0.14/webapps/test mytomcat9

此时访问

<http://192.168.229.132:9080/>



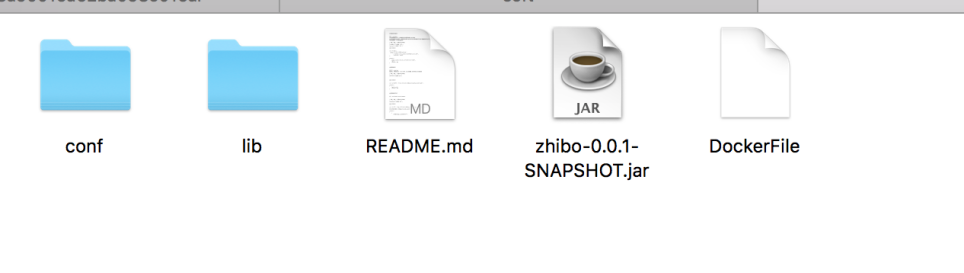
1. **部署自己开发的web应用**

步骤1：

编写DockerFile文件

|  |
| --- |
| #此处是基于镜像openjdk:latest来的，因为他里面有jdk  FROM openjdk:latest  VOLUME /Users/hongguo/Documents/conf:/conf  #如下是将jar、依赖的jar、配置文件加入到docker镜像中  ADD test-0.0.1-SNAPSHOT.jar /test.jar  ADD conf/ /conf  ADD lib/ /lib  #如下是解决乱码问题  ENV LANG C.UTF-8  ENV LANG en\_US.UTF-8  ENV LANGUAGE en\_US:en  ENV LC\_ALL en\_US.UTF-8  #执行运行jar包命令  CMD java -jar /test.jar  #对外暴露端口  EXPOSE 9080 |

我们的工程结构目录



conf是配置文件所在地

lib是依赖jar所在文件夹

此处要将刚才新建的DockerFile文件放到jar的同级目录下

步骤2：

终端进入DockerFile所在目录下，执行如下命令

docker build -f ./DockerFile -t test:1.0 .

test:1.0 镜像名

步骤3：

执行如下命令启动docker容器，此时我们的应用就跑起来了

docker run -d -p 9080:9080 -v /Users/hongguo/Documents/conf:/conf -v /Users/hongguo/Documents/log:/log -it test:1.0

-d 以后台方式启动

-p 9080:9080 对外暴露端口

-v /Users/hongguo/Documents/conf:/conf 让docker容器去读取/Users/hongguo/Documents/conf下的配置信息，这样我们可以方便修改配置信息，然后重启docker容器

-v /Users/hongguo/Documents/log:/log 让docker容器的日志文件输出到/Users/hongguo/Documents/log，方便我们查看

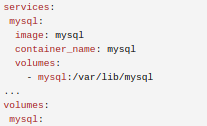
1. **Docker-compose**
2. **数据挂载**

docker-compose里两种设置方式都是可以持久化的

1）绝对路径



1. 卷标的



第一种情况路径直接挂载到本地，比较直观，但需要管理本地的路径，而第二种使用卷标的方式，比较简洁，但你不知道数据存在本地什么位置，下面说明如何查看docker的卷标。



1. **物理机执行容器内部命令**

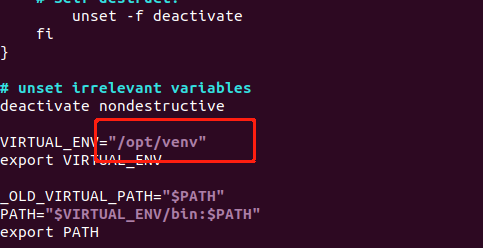
有的时候我们需要在物理机上使用脚本直接执行容器内部命令,而不需要进入容器.

|  |
| --- |
| docker exec -it ocr\_service bash -c “rm -rf test.txt && rm -rf ../conf/wvcm/” |

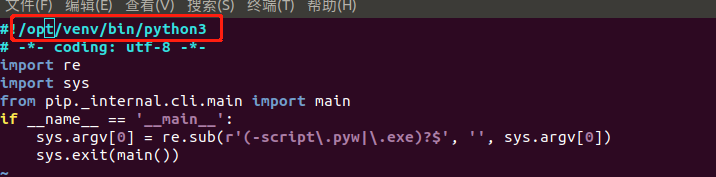
如上命令是进入ocr容器内部,删除test.txt文件以及删除conf文件夹下的wvcm文件夹

1. **Python构建docker镜像和部署**

1.修改venv文件夹下的bin的active文件

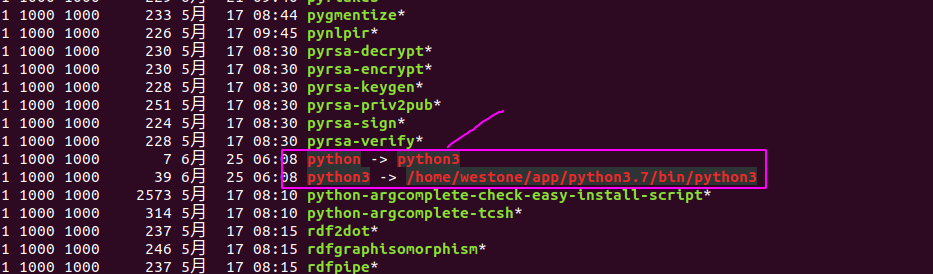


修改/venv/bin/pip和pip3



如果还不对，查看python的软连接，在venv/bin目录下使用如下命令,

ll,如下软连接是有问题的，修改软连接



ln -sf /usr/local/python/python3/bin/python3.7 python3

2.容器内的服务无法访问物理机的服务接口

可能是防火墙的问题,需要在物理机上关闭防火墙

systemctl stop firewalld.service

**部署步骤:**

修改management项目下的docker-compose.yml文件,修改挂载目录等(management下的SERVICE\_URL环境变量修改为物理机的IP地址)

本地测试python项目,需要配置本机的环境变量

vi ~/.bashrc

在文件末尾添加如下环境变量

export MYSQL\_HOST=127.0.0.1

export MYSQL\_PORT=13306

export RABBITMQ\_HOST=127.0.0.1

export RABBITMQ\_PORT=5672

export MANAGEMENT\_HOST=127.0.0.1

export MANAGEMENT\_PORT=9080

source ~/.bashrc(不行就重启电脑)

Redis

启动docker命令

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

添加密码：

config set requirepass westone@2020

1. **镜像仓库**

自建镜像仓库

<https://www.cnblogs.com/ilifeilong/p/11708183.html>

1. **物理机和容器数据拷贝**

物理机和docker容器数据拷贝

<https://www.cnblogs.com/areyouready/p/8973495.html>

1. **运行rabbitMQ**

运行rabbitmq

docker run -d --name rabbitmq3.7.7 -p 5672:5672 -p 15672:15672 --hostname myRabbit -e RABBITMQ\_DEFAULT\_USER=westone\_admin -e RABBITMQ\_DEFAULT\_PASS=westone@2020 2888deb59dfc

1. **运行nginx**

启动nginx

https://www.cnblogs.com/770880-lele/p/13828052.html

docker run --name ai\_nginx -p 18080:80 -p 10443:443 -v /data/westone/nginx/conf/:/etc/nginx/ -v /data/westone/common/:/opt/ -d daee903b4e43

1. **Docker run运行容器内部脚本**

问题描述:启动一个docker镜像,但是想通过docker run命令运行镜像里面的脚本

方法:

docker run -itd --name news\_classification\_forward\_service --restart always -p 8096:8096 news\_classification\_forward\_service:v01.00 sh /opt/init.sh

init.sh:就是容器内部的脚本,内容如下:

**#!/bin/bash**source /etc/profile  
. /home/news\_venv/bin/activate  
cd /opt/news\_classification\_forward/  
mkdir log  
touch log/news.log  
cd /opt/news\_classification\_forward/code/  
uwsgi --ini uwsgi.ini

1. **Docker容器编码问题**

描述:我们在启动docker容器时,经常发生容器内部编码乱七八糟的问题,如下异常:

docker本地化异常：/bin/sh: warning: setlocale: LC\_ALL: cannot change locale (en\_US.UTF-8)

docker中经常设置不了 环境变量$LC\_ALL, 导致报很多奇怪的编码错误：

解决方法：

sudo localedef -i en\_US -f UTF-8 en\_US.UTF-8

其次如果docker容器内部python在print出现编码问题,则需要设置编码,最好是docker容器初始化的时候就设置编码为utf-8

|  |
| --- |
| ENV *LANG* en\_US.UTF-8 ENV *LANGUAGE* en\_US.UTF-8 ENV *LC\_ALL* en\_US.UTF-8 |

或者终端输入命令:

export LANG=en\_US.UTF-8

export LANGUAGE=en\_US.UTF-8

export LC\_ALL=en\_US.UTF-8