1.

启动docker服务： **systemctl start docker**

查看docker服务状态： **systemctl status docker**

重启docker: **systemctl restart docker**

停止docker服务: **systemctl stop docker**

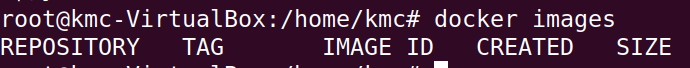
开机启动docker: **systemctl enable docker**

1. 镜像相关

docker镜像和docker容器的关系就像类与对象

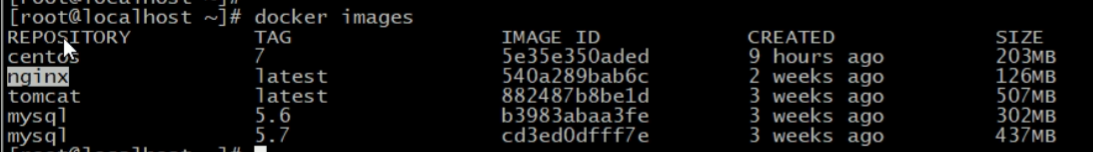
(1 查看本地镜像：**docker images** #image是镜像的意思

**docker images -q**  #查看所有镜像ID



这几列依次是：仓库(镜像)、版本号、镜像ID、创建时间、镜像大小

镜像是软件及其运行环境的文件



(2 从远程仓库(dockerhub)搜索镜像

**docker search 镜像名称**

(3 下载镜像

**docker pull 镜像名称:版本号(tag)** #默认最新版本

如： docker pull redis:3.2

(4 删除镜像

**docker rmi 镜像ID;** 或者 **docker rmi 镜像名称:版本号** #rmi是remove image的意思

如：docker rmi 7614ae9453d1 或 docker rmi redis:5.0

**docker rmi ‘docker images -q**’ #用引号删除查到的所有镜像ID

1. docker容器相关

利用镜像创建容器，之后容器内自动运行对应的软件

(1 创建容器 **docker run**

如： **docker run -it --name=cl centos:7 /bin/bash**

参数**-i** : 保持容器一直运行，否则在没有客户端连接时会自动关闭

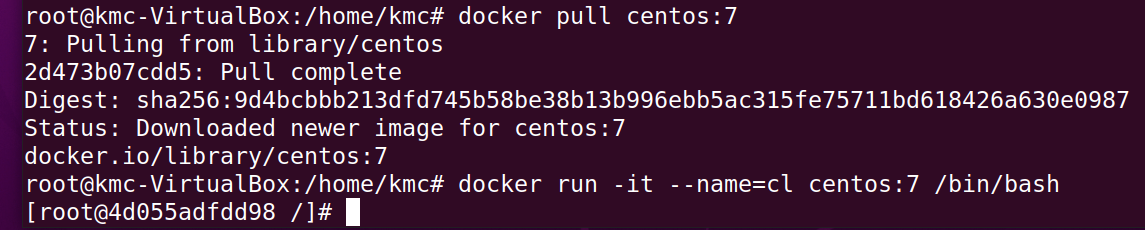
**-t**: 给容器分配终端，可以用于输入命令

**--name=名称** 或 **--name 名称** #定义容器的名称

指定镜像： **镜像名:版本号**

指定容器终端使用的Shell: **/bin/bash**

创建完之后会自动进入容器终端：



exit可退出终端,这时通过-it创建的容器会stop

docker -id --name=c2 centos:7

#**-d**可以后台启动容器，创建完后不会立即进入容器终端。这时可以省略/bin/bash。

进入创建的容器：**docker exec -it c2 /bin/bash** #用docker exec执行进入指令，/bin/bash指定使用的Shell。-i表示交互(interactive)

（2 查看运行中的容器：**docker ps**

**-a** 参数可查看所有容器（包括已关闭的容器）

**-q** 只查看所有容器的ID

-t创建的是**交互式容器**，-d创建的是**守护式容器**

（3 停止容器: **docker stop 容器名**

启动容器： **docker start 容器名**

删除容器： **docker rm 容器ID或容器名** #不能删除运行中的容器

查看容器信息： **docker inspect 容器名**

1. docker容器的数据卷

**数据卷**是宿主机(容器所在的主机)中的一个**目录或文件**，容器中的目录或文件**挂载于**该目录

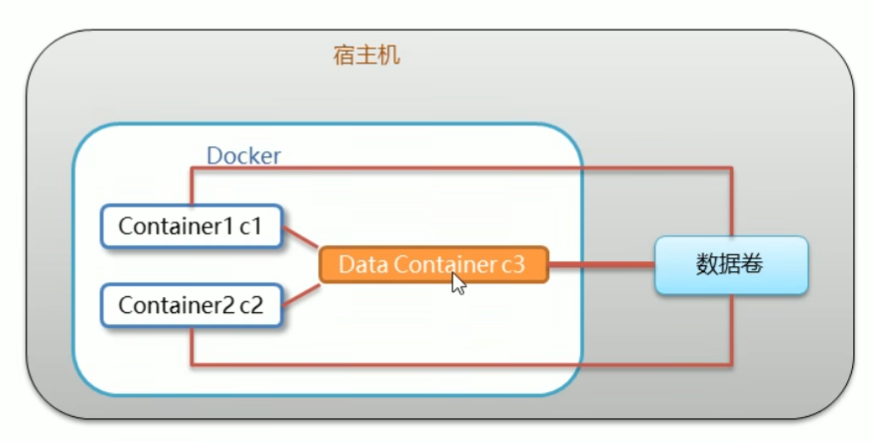
配置数据卷： docker run ...

参数 **-v 宿主机目录:容器内目录** #连接两个目录，目录必须是绝对路径；目录不存在会自动创建，可以**挂载多个**数据卷

容器删除后，宿主机的数据卷还存在

5.数据卷容器

多容器(C1、C2)进行数据交换：为C1和C2创建一个容器C3，**将C3挂载到宿主机**上，将C1和C2挂载到C3上。此时C3被称为**数据卷容器**。



(1 创建C3容器，用-v设置数据卷

docker run -it --name=c3 -v **/volume** centos: /bin/bash

#只指定容器目录，此时宿主机会自动分配一个目录

(2 创建c1、c2容器，使用--volumes-from参数设置数据卷

**docker run -it --name=c1 --volumes-from c3 centos:7 /bin/bash**

**docker run -it --name=c2 --volumes-from c3 centos:7 /bin/bash** #此时C1、C2会与C3有相同的目录volume

6.Docker应用部署——MySQL部署

在docker容器中部署MySQL，并通过外部客户端连接服务器

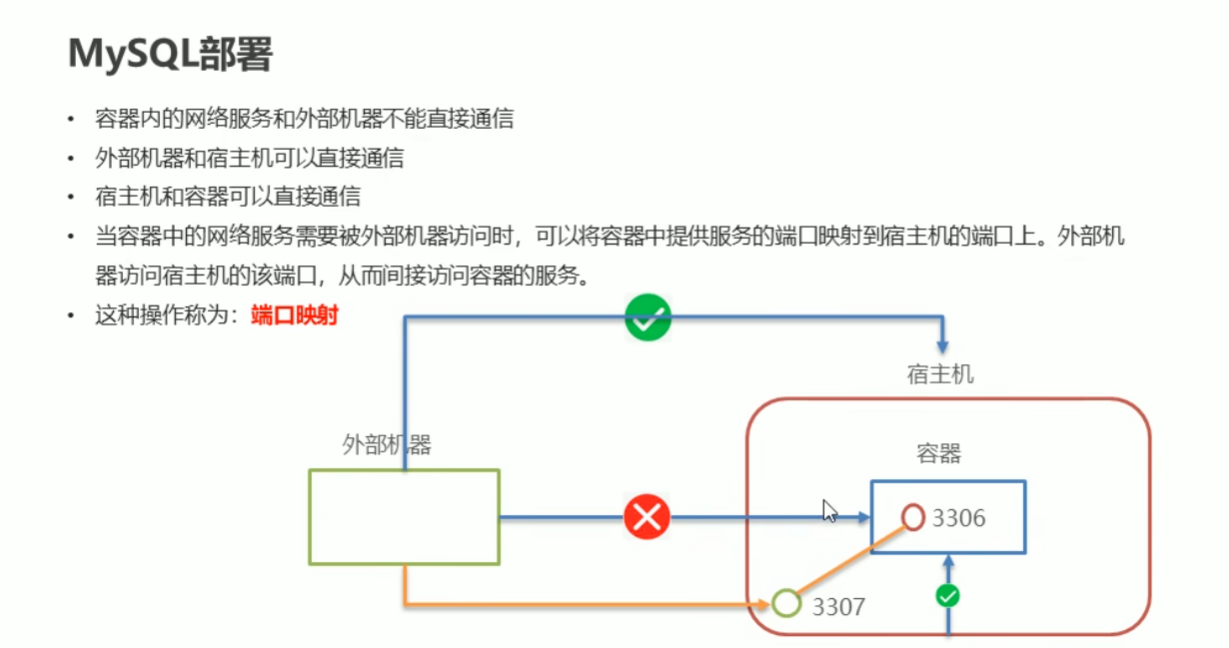
步骤：**(1 搜索镜像**

**(2 拉取镜像**

**(3 创建容器**

#不需要创建环境等

外部机器与容器连接：端口映射



**创建**MySQL容器：

**docker run -id \**

**-p 3307:3306 \**  #使用-p端口映射，将容器的3306端口映射到宿主机的3307端口

**--name=c\_mysql \**

**-v $PWD/conf:/etc/mysql/conf.d \**

**-v $PWD/logs:/logs \**

**-v $PWD/data:/var/lib/mysql \**

**-e MYSQL\_ROOT\_PASSWORD=123456 \**

**mysql:5.6**

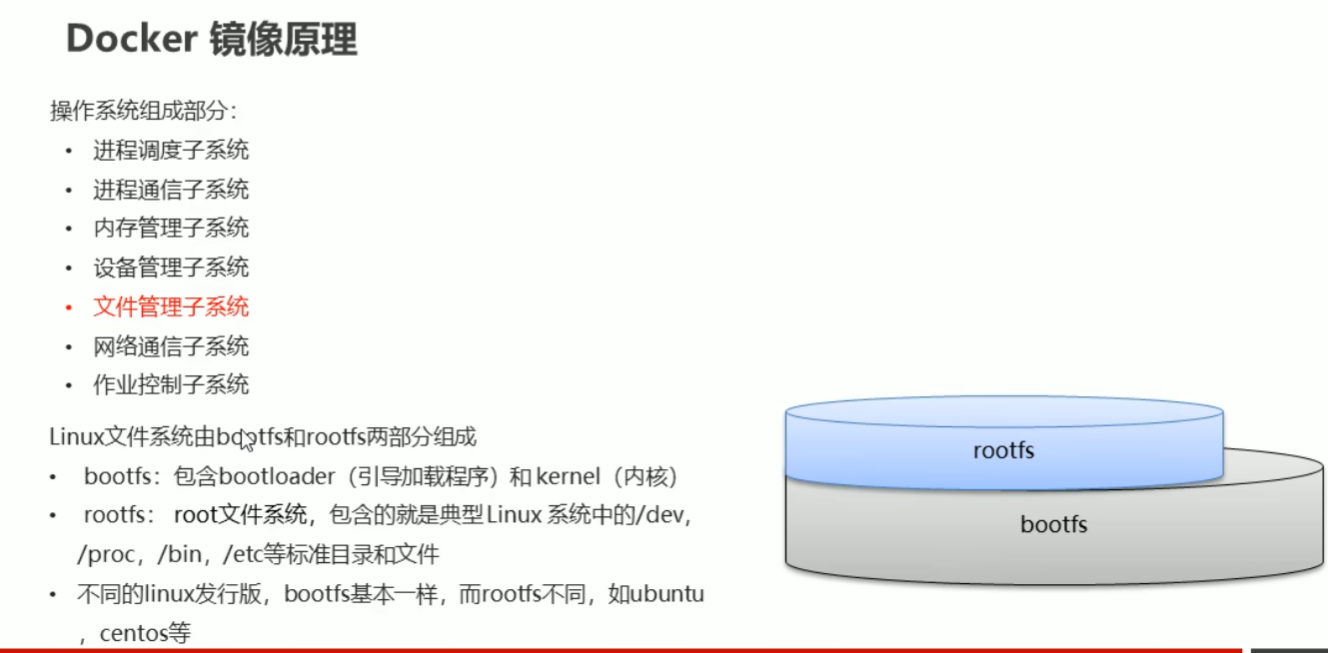
进入MySQL容器客户端:

**docker exec -it c\_mysql /bin/bash**

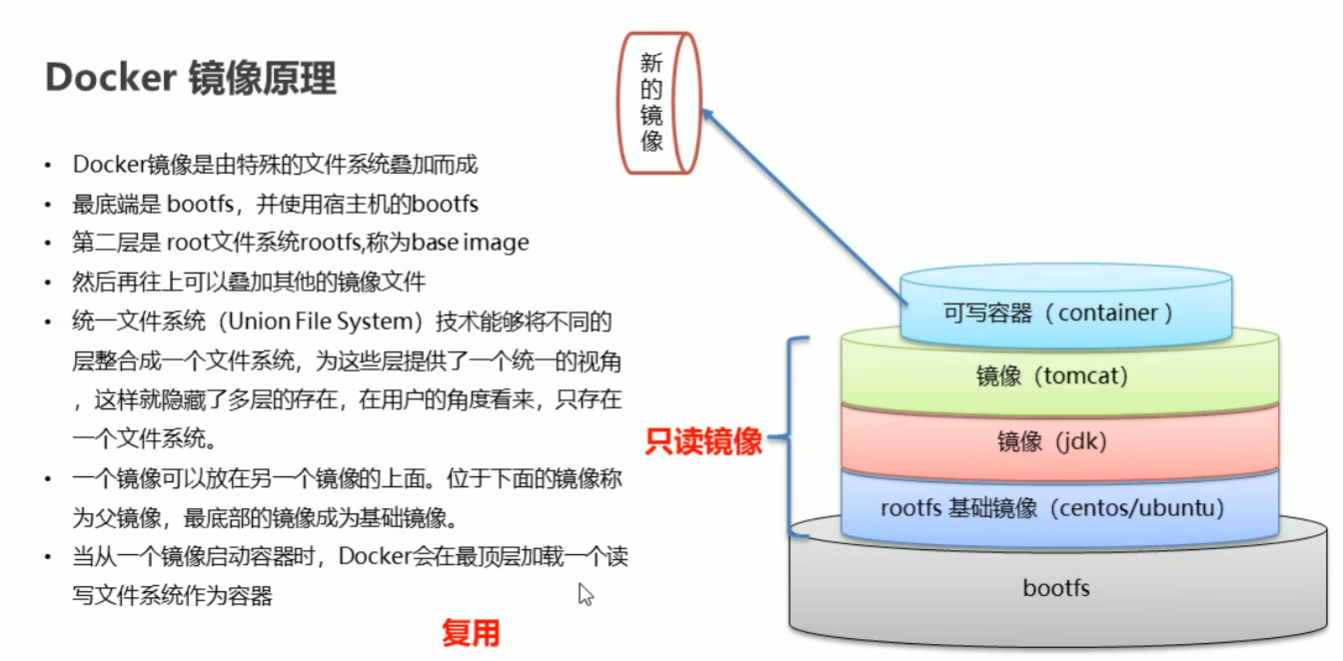
1. docker镜像原理

Dockerfile用于制作镜像

Linux操作系统的**文件管理系统**：



Docker镜像原理：分层的文件系统，利用了Linux的bootfs



8.制作Docker镜像

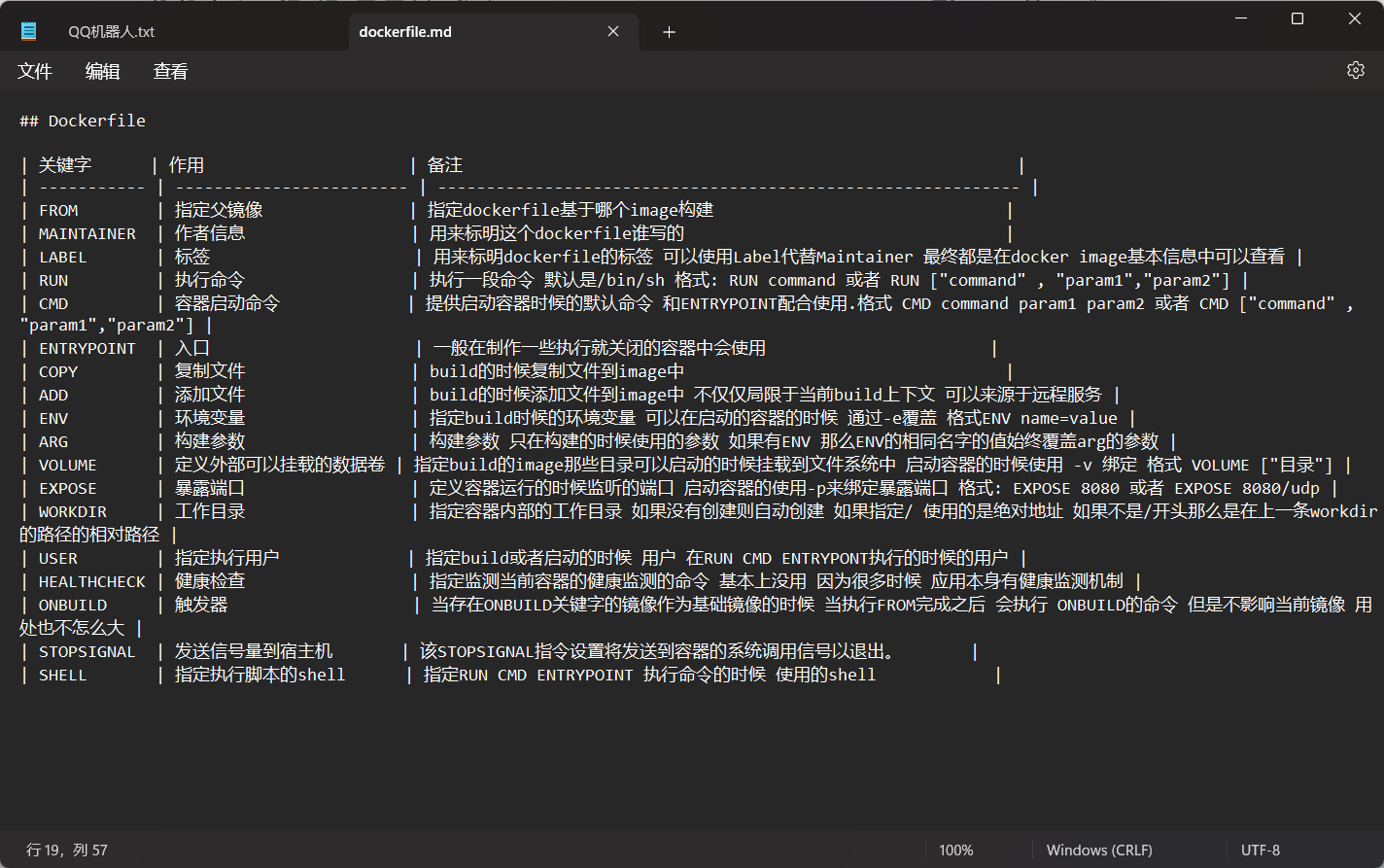
**docker commit 容器ID 镜像号:版本号**  #将容器转换为镜像，挂载的目录不会被转换为镜像

**docker save -o 压缩文件名称 镜像名称:版本号** #将镜像变为归档/压缩文件(.tar、.gz、.bz后缀)

**docker load -i 压缩文件名称** #将归档/压缩文件还原为镜像

9. Dockerfile

用于制作镜像，是一个**文本文件**，包含了一条条**指令**。**每一条指令构建一层镜像**，基于基础镜像最终构建出一个新的镜像。



写完后通过dockerfile构建镜像：**docker build -f DockerFile文件路径 -t 镜像名称:版本 .**

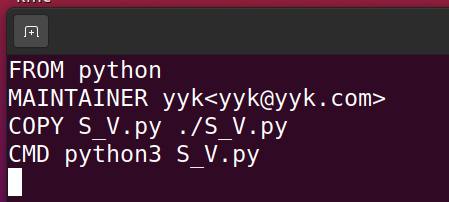
**-f :**指定要使用的Dockerfile路径；

**--tag, -t:** 镜像的名字及标签

如：

**docker build -f ./DockerFile1 -t python1:1.0 .** #末尾的.表示在当前目录下寻找dockerfile文件

Dockerfile的内容为：



这里的原S\_V必须与Dockerfile在同一目录内

**docker run -id python1:1.0**  #利用该镜像创建容器

带有vim和默认工作路径为./usr的centos7的dockerfile：

**FROM centos:7**  #父镜像

**MAINTAINER YYK <YYK@COM>** #作者信息

**RUN yum install -y vim** #centos中使用yum安装

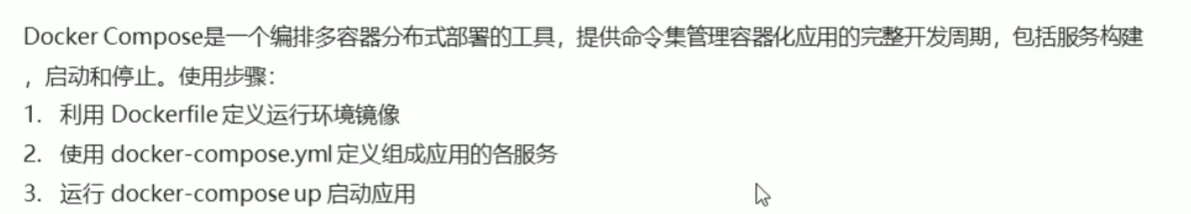
**WORKDIR ./usr** #workdir设置默认工作目录

**cmd /bin/bash** #启动客户端时的默认命令

1. 服务编排

按照一定的业务规则批量管理容器，docker-compose是一个编排**多容器**分布式部署的工具

使用步骤：



1. docker私有仓库

12.

13.

14.

15.

16.

17.

18.

19.

20.

21.

22.

23.

24.

25.

26.

27.

28.

29.

30.

31.

32.

33.

34.

35.

36.

37.

38.

39.

40.

41.

42.

43.

44.

45.

46.

47.