## **容器**

镜像（Image）和容器（Container）的关系，就像是面向对象程序设计中的 类 和 实例 一样，镜像是静态的定义，容器是镜像运行时的实体。容器可以被创建、启动、停止、删除、暂停等。

容器的实质是进程，但与直接在宿主执行的进程不同，容器进程运行于属于自己的独立的 命名空间。因此容器可以拥有自己的 root 文件系统、自己的网络配置、自己的进程空间，甚至自己的用户 ID 空间。容器内的进程是运行在一个隔离的环境里，使用起来，就好像是在一个独立于宿主的系统下操作一样。这种特性使得容器封装的应用比直接在宿主运行更加安全。

前面讲过镜像使用的是分层存储，容器也是如此。每一个容器运行时，是以镜像为基础层，在其上创建一个当前容器的存储层，我们可以称这个为容器运行时读写而准备的存储层为容器存储层。

容器存储层的生存周期和容器一样，容器消亡时，容器存储层也随之消亡。因此，任何保存于容器存储层的信息都会随容器删除而丢失。

按照 Docker 最佳实践的要求，容器不应该向其存储层内写入任何数据，容器存储层要保持无状态化。所有的文件写入操作，都应该使用 数据卷（Volume）、或者绑定宿主目录，在这些位置的读写会跳过容器存储层，直接对宿主（或网络存储）发生读写，其性能和稳定性更高。

数据卷的生存周期独立于容器，容器消亡，数据卷不会消亡。因此，使用数据卷后，容器删除或者重新运行之后，数据却不会丢失。

# **docker 镜像操作**

## **下载 CentOS 镜像**

docker pull centos:7

## **查看centos7镜**

docker images

或

docker image ls

## **运行 centos7**

docker run -it xxxx bash

xxxx - 镜像名, 或 image id 的前几位

-it 这是两个参数，一个是 -i：交互式操作，一个是 -t 终端。我们这里打算进入 bash 执行一些命令并查看返回结果，因此我们需要交互式终端。

bash 放在镜像名后的是命令，这里我们希望有个交互式 Shell，因此用的是 bash。

## **删除镜像**

501 镜像 id 前几位，一般三位以上，足够区分即可

docker image rm 501

删除指定仓库的镜像

docker image rm centos

## **镜像导出**

docker save mysql:5.7 node:8 | gzip > app.tar.gz

## **镜像导入**

docker load < apps.tar.gz

# **容器操作**

## **启动容器**

docker run -it centos:7 bash

当利用 docker run 来创建容器时，Docker 在后台运行的标准操作包括：

* 检查本地是否存在指定的镜像，不存在就从公有仓库下载
* 利用镜像创建并启动一个容器
* 分配一个文件系统，并在只读的镜像层外面挂载一层可读写层
* 从宿主主机配置的网桥接口中桥接一个虚拟接口到容器中去
* 从地址池配置一个 ip 地址给容器
* 执行用户指定的应用程序
* 执行完毕后容器被终止

## **后台运行**

docker run -dit centos:7

-d 后台运行容器

容器是否会长久运行，是和 docker run 指定的命令有关，和 -d 参数无关。

## **查看后台运行的容器输出结果**

docker container logs 802

## **查看容器**

docker container ls -a

# 或

docker ps -a

-a all, 全部

## **终止容器**

docker container stop 802

## **重新启动容器**

docker container restart 802

## **进入容器**

在使用 -d 参数时，容器启动后会进入后台。

某些时候需要进入容器进行操作，可以使用 docker exec 命令

docker exec -it 802 bash

## **删除容器**

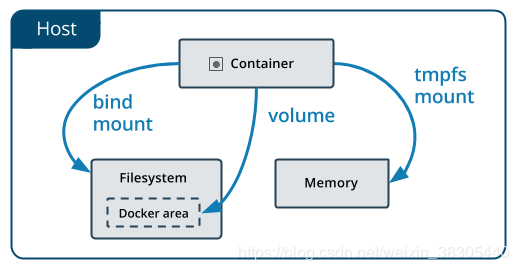
docker container rm 802

如果删除运行中的容器，需要添加 -f 参数

## **清理所有终止状态容器**

docker container prune

# **数据管理**



在容器中管理数据主要有两种方式：

数据卷（Volumes）

挂载主机目录 (Bind mounts)

## **数据卷**

数据卷 是一个可供一个或多个容器使用的特殊目录

* 数据卷 可以在容器之间共享和重用
* 对 数据卷 的修改会立马生效
* 对 数据卷 的更新，不会影响镜像
* 数据卷 默认会一直存在，即使容器被删除

### **创建数据卷**

docker volume create my-vol

### **查看所有数据卷**

docker volume ls

### **查看指定 数据卷 的信息**

docker volume inspect my-vol

查询的结果：

[

{

"Driver": "local",

"Labels": {},

"Mountpoint": "/var/lib/docker/volumes/my-vol/\_data",

"Name": "my-vol",

"Options": {},

"Scope": "local"

}

]

### **启动挂载数据卷的容器**

docker run -it --mount source=my-vol,target=/webapp centos:7 bash

# 或者：

docker run -it -v my-vol:/webapp centos:7 bash

-v my-vol:/webapp 把数据卷 my-vol 挂载到容器的 /webapp 目录

### **删除数据卷**

删除指定的数据卷，如果数据卷被容器使用则无法删除

docker volume rm my-vol

清理无主数据卷

docker volume prune

## **挂载主机目录**

docker run -it --mount type=bind,source=/usr/app,target=/opt/app centos:7 bash

# 或

docker run -it -v /usr/app:/opt/app centos:7 bash

* -v 如果本地目录不存在 Docker 会自动为你创建一个文件夹
* --mount 参数时如果本地目录不存在，Docker 会报错

### **查看挂载目录信息**

docker inspect 91a

显示结果：

...

"Mounts": [

{

"Type": "bind",

"Source": "/usr/app",

"Destination": "/opt/app",

"Mode": "",

"RW": true,

"Propagation": "rprivate"

}

],

...

# **网络**

## **自动分配映射端口**

docker run -d -P tomcat

# 查看容器信息

docker container ls -a

显示结果：

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

d03480b1a781 tomcat "catalina.sh run" 3 minutes ago Up 3 minutes 0.0.0.0:32768->8080/tcp trusting\_gagarin

## **映射指定端口**

docker run -d -p 8080:8080 tomcat

8080:8080 本机端口:容器端口

## **映射多个端口**

docker run -d \

-p 5000:5000 \

-p 80:8080 tomcat

## **映射指定端口（指定网卡）**

docker run -d -p 192.168.64.150:8080:8080 tomcat

## **自动分配映射端口（指定网卡）**

docker run -d -p 192.168.64.150::8080 tomcat

## **查看端口配置**

docker port 8af

# **容器互联**

## **新建网络**

docker network create -d bridge my-net

-d driver,网络类型，默认 bridge，也可以是 overlay（Swarm mode）

## **列出网络**

docker network ls

## **查看网络信息**

docker inspect 67d

## **连接容器**

docker run -it --name app1 --network my-net centos:7

新开终端执行：

docker run -it --name app2 --network my-net centos:7

在两个终端中分别执行：

ping app1

ping app2

显示如下：

[root@35569c623c4c /]# ping app1

PING app1 (172.18.0.2) 56(84) bytes of data.

64 bytes from 35569c623c4c (172.18.0.2): icmp\_seq=1 ttl=64 time=0.577 ms

64 bytes from 35569c623c4c (172.18.0.2): icmp\_seq=2 ttl=64 time=0.061 ms

64 bytes from 35569c623c4c (172.18.0.2): icmp\_seq=3 ttl=64 time=0.066 ms

......

多个容器互联，推荐使用 Docker Compose

# **Dockerfile**

## **准备**

* centos:7镜像
* jdk压缩包 jdk-8u212-linux-x64.tar.gz
* tomcat7压缩包 apache-tomcat-7.0.94.tar.gz

## **Dockerfile文件**

#以centos7为基础,安装oracle jdk8和tomcat7

FROM centos:7

#ADD命令将压缩包传入镜像中的指定目录,并同时解压缩

ADD jdk-8u212-linux-x64.tar.gz /opt/

ADD apache-tomcat-7.0.96.tar.gz /usr/

#为了方便,把文件夹名称改得简单一点

RUN mv /usr/apache-tomcat-7.0.96 /usr/tomcat

#设置环境变量

ENV JAVA\_HOME=/opt/jdk1.8.0\_212 \

CATALINA\_HOME=/usr/tomcat \

PATH=$PATH:/opt/jdk1.8.0\_212/bin:/usr/tomcat/bin

#暴露容器的8080端口

EXPOSE 8080

#设置启动容器时自动运行tomcat

ENTRYPOINT /usr/tomcat/bin/startup.sh && tail -F /usr/tomcat/logs/catalina.out

FROM centos:7

ADD jdk-8u212-linux-x64.tar.gz /opt/

ADD apache-tomcat-7.0.96.tar.gz /usr/

RUN mv /usr/apache-tomcat-7.0.96 /usr/tomcat

ENV JAVA\_HOME=/opt/jdk1.8.0\_212 \

CATALINA\_HOME=/usr/tomcat \

PATH=$PATH:/opt/jdk1.8.0\_212/bin:/usr/tomcat/bin

EXPOSE 8080

ENTRYPOINT /usr/tomcat/bin/startup.sh && tail -F /usr/tomcat/logs/catalina.out

## **使用 Dockerfile 构建镜像**

docker build -t tomcat:7 .

注意末尾的点,表示构建过程中从当前目录寻找文件

## **启动容器**

### **准备存储目录**

* webapps目录,例如 /opt/webapps
* logs目录,例如 /var/lib/tomcat-logs

mkdir /opt/webapps

mkdir /var/lib/tomcat-logs

### **启动容器,挂载目录**

docker run -d \

-p 8080:8080 \

-v /opt/webapps:/usr/tomcat/webapps \

-v /var/lib/tomcat-logs:/usr/tomcat/logs \

tomcat:7