Docker

Dokcer 入门

一、Docker的简介

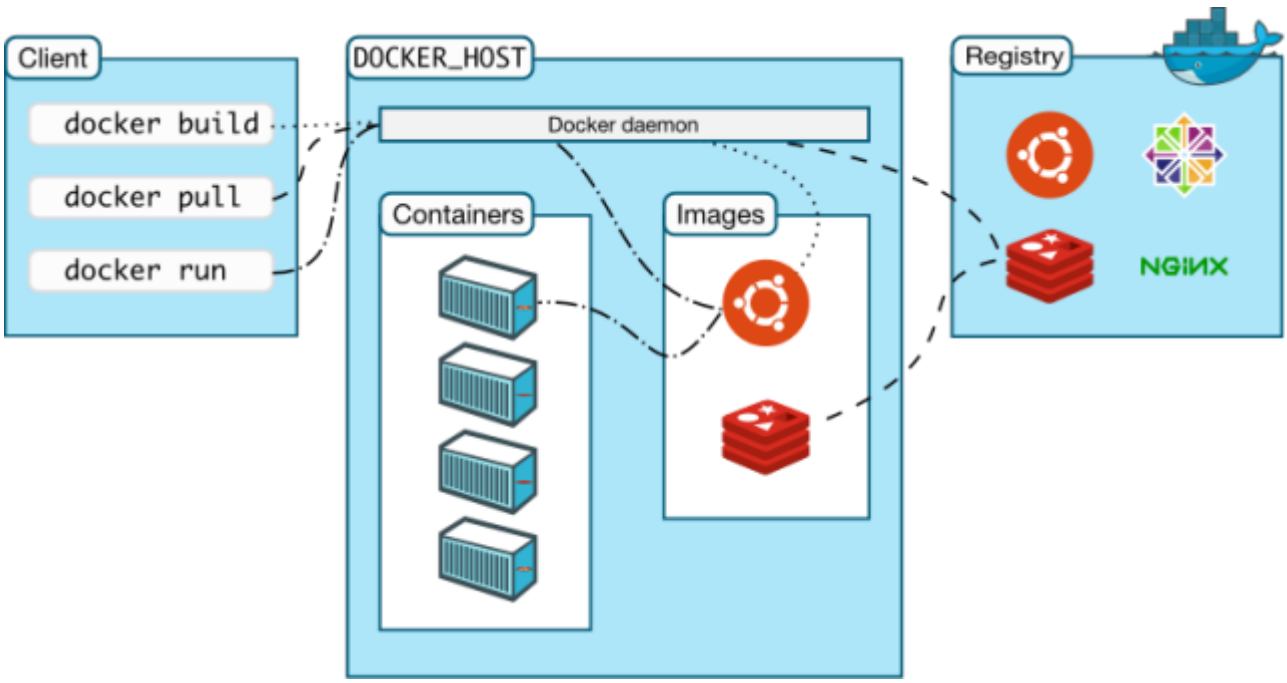
1、三个基本的概念

- **镜像（Image）**：Docker 镜像（Image），就相当于是一个 root 文件系统。比如官方镜像 ubuntu:16.04 就包含了完整的一套 Ubuntu16.04 最小系统的 root 文件系统。
- **容器（Container）**：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。
- **仓库（Repository）**：仓库可看成一个代码控制中心，用来保存镜像。

2、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



名词解释：

Docker镜像（images）：用于创建Docker容器的模板，镜像本身是一个静态的，也是只读的

Docker容器（Container）：是镜像运行的一个实体，具有独立性，是一个或多个运行应用的集合

Docker客户端（Client）：命令行工具，用于管理容器、镜像，实现与Docker的守护进程进行通信

Docker主机（Docker_Host）：俗称宿主机，用于运行Docker守护进程可容器的载体

Docker Registry：docker镜像的仓库 一个仓库包含多个版本的镜像，通常使用tag来区分版本，仓库名:标签的格式来下载不同版本 不写标签 默认情况使用latest为默认标签

二、安装Docker

官方的参考文档：https://docs.docker.com/engine/install/centos/

环境要求：CentOS 7 以上 内核 3.10以上

1、卸载旧版本

之前未安装过分略过

```
yum remove docker \
           docker-client \
           docker-client-latest \
           docker-common \
           docker-latest \
           docker-latest-logrotate \
           docker-logrotate \
           docker-engine
```

2、设置存储库

```
# 下载yum-utils包
yum install -y yum-utils
# 设置镜像仓库
yum-config-manager \
    --add-repo \
    https://download.docker.com/linux/centos/docker-ce.repo     # 默认国外的 推荐换成下面国内的

# 阿里云的Docker镜像地址
yum-config-manager \
    --add-repo \
    https://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo
```

3、直接yum安装

```
# 更新软件包的索引
yum makecache fast
# 安装 docker-ce 社区版
yum install docker-ce docker-ce-cli containerd.io

# 默认安装的是最新版 使用下面方法安装指定版本
yum list docker-ce --showduplicates | sort -r
# 然后安装指定版本
yum install docker-ce-<VERSION_STRING> docker-ce-cli-<VERSION_STRING> containerd.io
```

4、启动测试

```
# 启动
systemctl start docker
# 查看安装信息
docker version
# 查看更详细的信息
docker info
```

5、验证是否正确安装

```
# 下载测试镜像，并在容器中运行
docker run hello-world
# 查看是否下载成功
docker imasges
# /var/lib/docker docker的默认工作路径
```

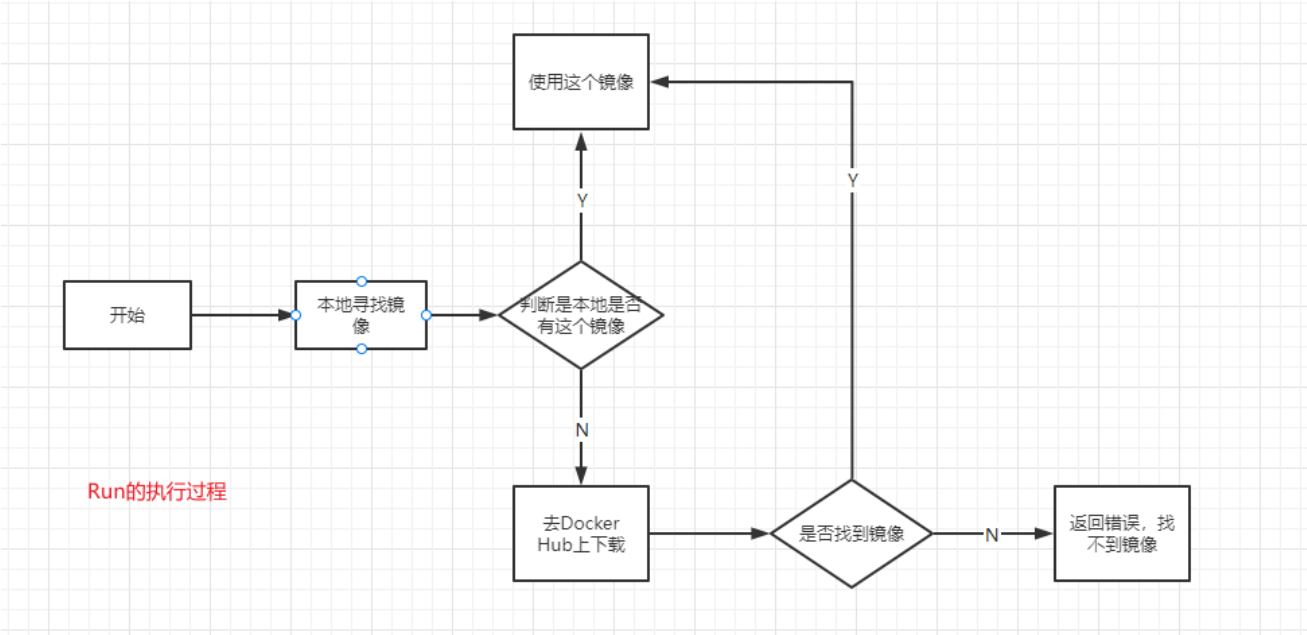
```
[root@docker ~]# docker images
REPOSITORY    TAG       IMAGE ID       CREATED         SIZE
hello-world   latest    feb5d9fea6a5   3 months ago    13.3kB
[root@docker ~]#
```

6、卸载方法

```
# 卸载软件包
yum remove docker-ce docker-ce-cli containerd.io
# 删除资源
rm -rf /var/lib/containerd
rm -rf /var/lib/docker
```

三、Run的流程和原理

1、Run的执行过程

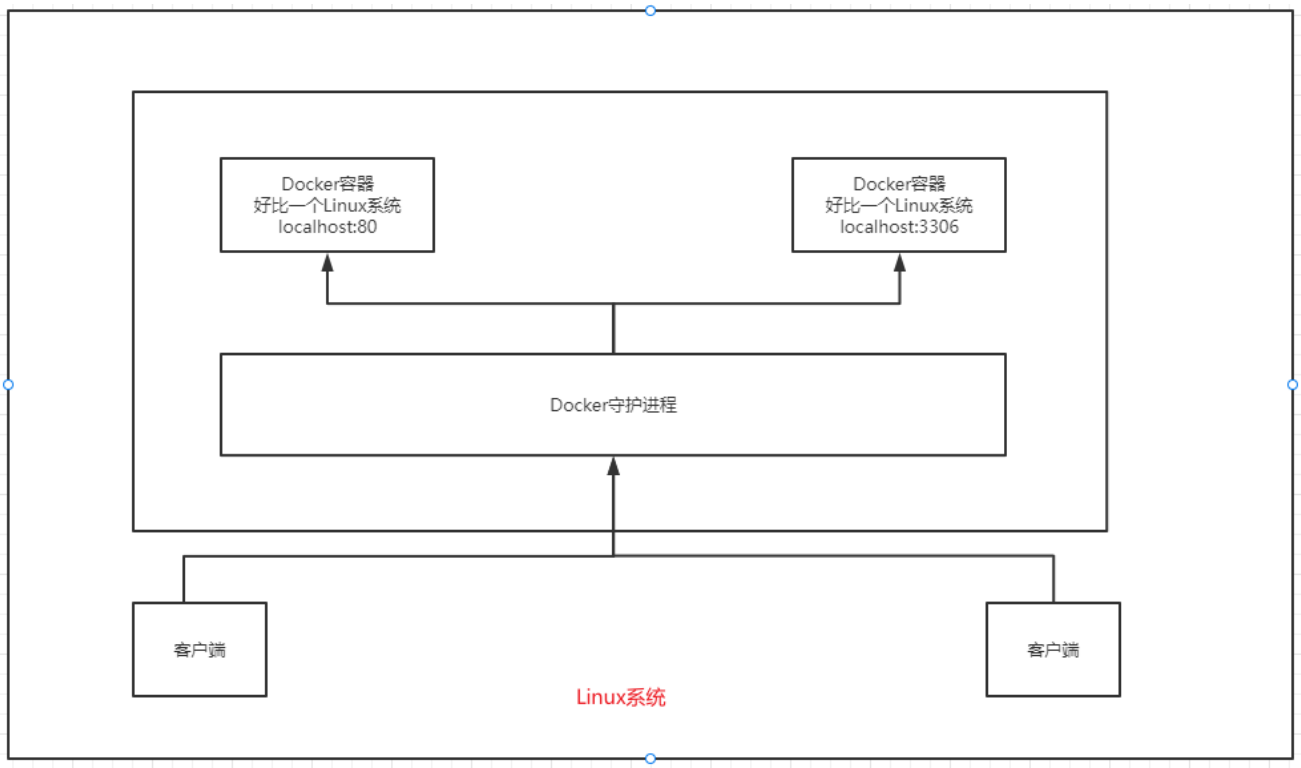


2、原理

Docker是如何工作的?

首先Docker是一个Client-Server（C/S）架构的，Docker的守护进程运行在主上（宿主机），通过Socket从客户端访问。

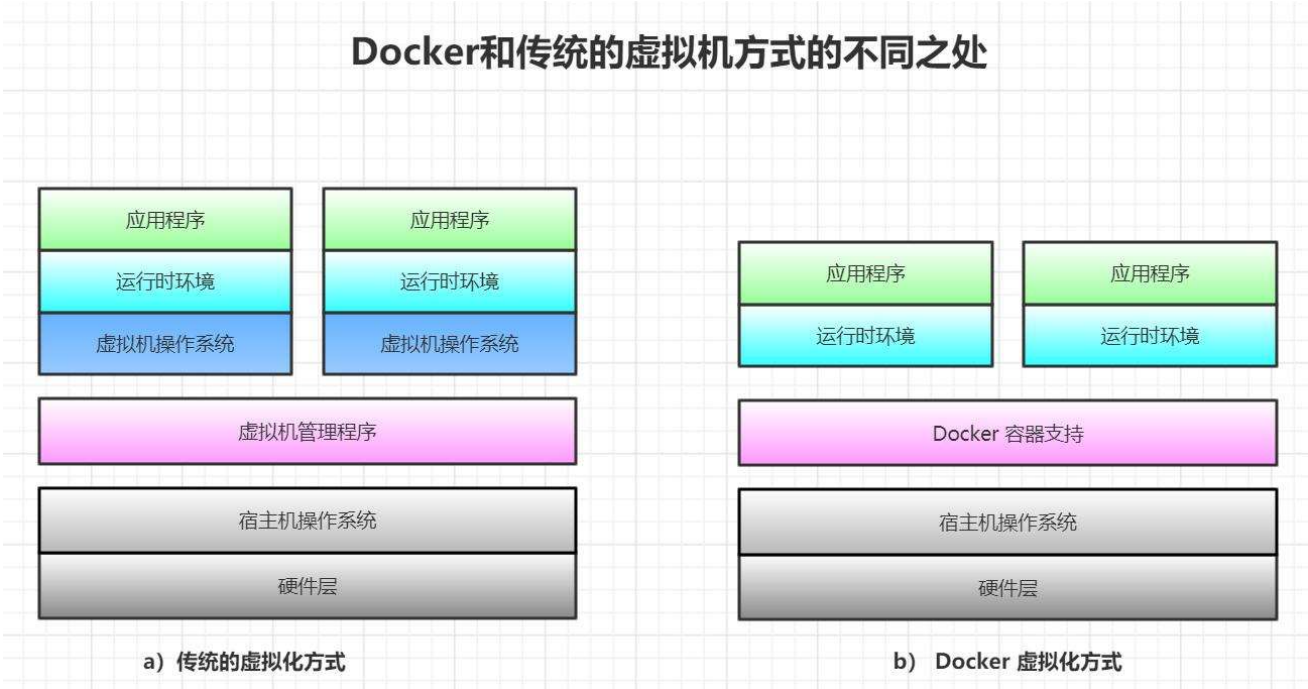
当DockerServer接收到Docker-Client指令就会执行这个命令!



3、Docker为什么比虚拟化（KVM，VM）快?

1）Docker有着比虚拟机更少的抽象层

2）Docker利用的宿主机的内核，虚拟机是需要虚拟出硬件，再装一个完整的系统，拥有自己的内核



总计：新建一个容器的时候，Docker不需要像虚拟机一样重新加载一个操作系统，避免了引导。虚拟机器加载的是一个系统，是分钟级的，而Docker就相当于在系统中启动一个服务，是秒级的。

四、Docker的常用命令

1、帮助命令

官方命令帮助文档：https://docs.docker.com/reference/

```
# 版本信息
docker version
# 显示docker的系统信息 更加详细
docker info
# 获取帮助
docker 命令 --help
```

2、镜像命令

```
# 查看本机上说有的镜像
docker images

# 结果解释
镜像的仓源    镜像的标签    镜像的ID        创建时间        镜像大小
REPOSITORY    TAG         IMAGE ID       CREATED         SIZE
hello-world   latest      feb5d9fea6a5   3 months ago    13.3kB

# 常用参数
-a --all    # 显示所有镜像
-q --quiet  # 只显示镜像ID
```

```
# 搜索仓库中的镜像 -- docker search 镜像名
docker search mysql # mysql 可以是任何镜像的名字

# 结果解释
镜像的名字                          描述信息                                          被收藏次数
NAME                                DESCRIPTION                                     STARS
OFFICIAL   AUTOMATED
mysql                               MySQL is a widely used, open-source relation…   11931     [OK]
mariadb                             MariaDB Server is a high performing open sou…   4561      [OK]


# 上面的命令也就等同于在 下面的Dokcer Hub官方镜像仓库中搜索
# https://registry.hub.docker.com/

# 常用参数
-f --filter # 过滤

[root@docker ~]# docker search mysql -f STARS=10000 # = -- >= 并不是一定要相等而是最接近
NAME      DESCRIPTION                                     STARS     OFFICIAL   AUTOMATED
mysql     MySQL is a widely used, open-source relation…   11931     [OK]
```

```
# 下载镜像 -- docker pull 镜像名[:tag] tag就是选择版本 默认最新版
docker pull mysql # mysql 可以是任何镜像的名字

[root@docker ~]# docker pull mysql
Using default tag: latest   # 不指定tag 默认就是 latest
latest: Pulling from library/mysql
72a69066d2fe: Pull complete     # 分层下载, docker image 的核心 联合文件系统
93619dbc5b36: Pull complete
99da31dd6142: Pull complete
626033c43d70: Pull complete
37d5d7efb64e: Pull complete
ac563158d721: Pull complete
d2ba16033dad: Pull complete
688ba7d5c01a: Pull complete
00e060b6d11d: Pull complete
```

```
1c04857f594f: Pull complete
4d7cfa90e6ea: Pull complete
e0431212d27d: Pull complete
Digest: sha256:e9027fe4d91c0153429607251656806cc784e914937271037f7738bd5b8e7709 # 签名信息
Status: Downloaded newer image for mysql:latest
docker.io/library/mysql:latest # 真是地址

# 等价写法
docker pull mysql === docker pull docker.io/library/mysql:latest

# 指定版本下载
docker pull mysql:5.7   # 指定版本必须在仓库中存在

# 分层下载 联合文件系统 简单解释
# 当我们下载同一个镜像是 公用的西可以不用再次下载
```

```
#删除镜像 -- docker rmi -f <id...>
docker rmi -f feb5d9fea6a5

# 删除所有
docker rmi -f $(docker images -aq)
```

3、容器命令

前提：有了镜像才可以创建容器

```
# 下载一个Centos7 的镜像
docker pull centos:7
```

创建容器

```
# 创建一个容器但是不启动它
docker create 选项 镜像名

# 新建容器并启动 -- docker run [可选参数] 镜像名 [启动bash]
docker run [可选参数] 镜像名
# 参数
-i  # 允许交互
-t  # 启动一个伪终端
-d  # 后台的方式运行
-h  # 指定容器的主机名
-P              # 大P     指定随机端口
--rm            # 容器停止后删除掉 默认不删除
--name="Name"    # 容器的名字 用来区分容器
--network        # 容器的网络链接方式 默认 NAT
-p              # 小p 指定容器端口
    -p  ip:主机端口:容器端口
    -p  主机端口:容器端口
    -p 容器端口
    容器端口
```

```
# 启动并进入容器
docker run -it  centos /bin/bash
```

启动和停止容器

```
# 启动
dcoker start 容器ID
# 重启
dcoker restart 容器ID
# 停止
dcoker stop 容器ID    # 温柔的停止
dcoker kill 容器ID    # 强制停止
# 暂停
dcoker pause 容器ID
# 恢复暂停
dcoker unpause 容器ID
```

查看容器

```
# 列出所有运行的容器
docker ps    # 会列出正在运行的
-a # 会列出正在运行的 和 历史运行过的容器
-n=number    # 会显示最近运行的几个 1就是显示1个 2及时2个 以此类推
-q  # 只显示容器的ID
```

退出容器

```
# 退出容器
exit    # 直接退出 并停止容器
Ctrl + P + Q # 退出不停止
```

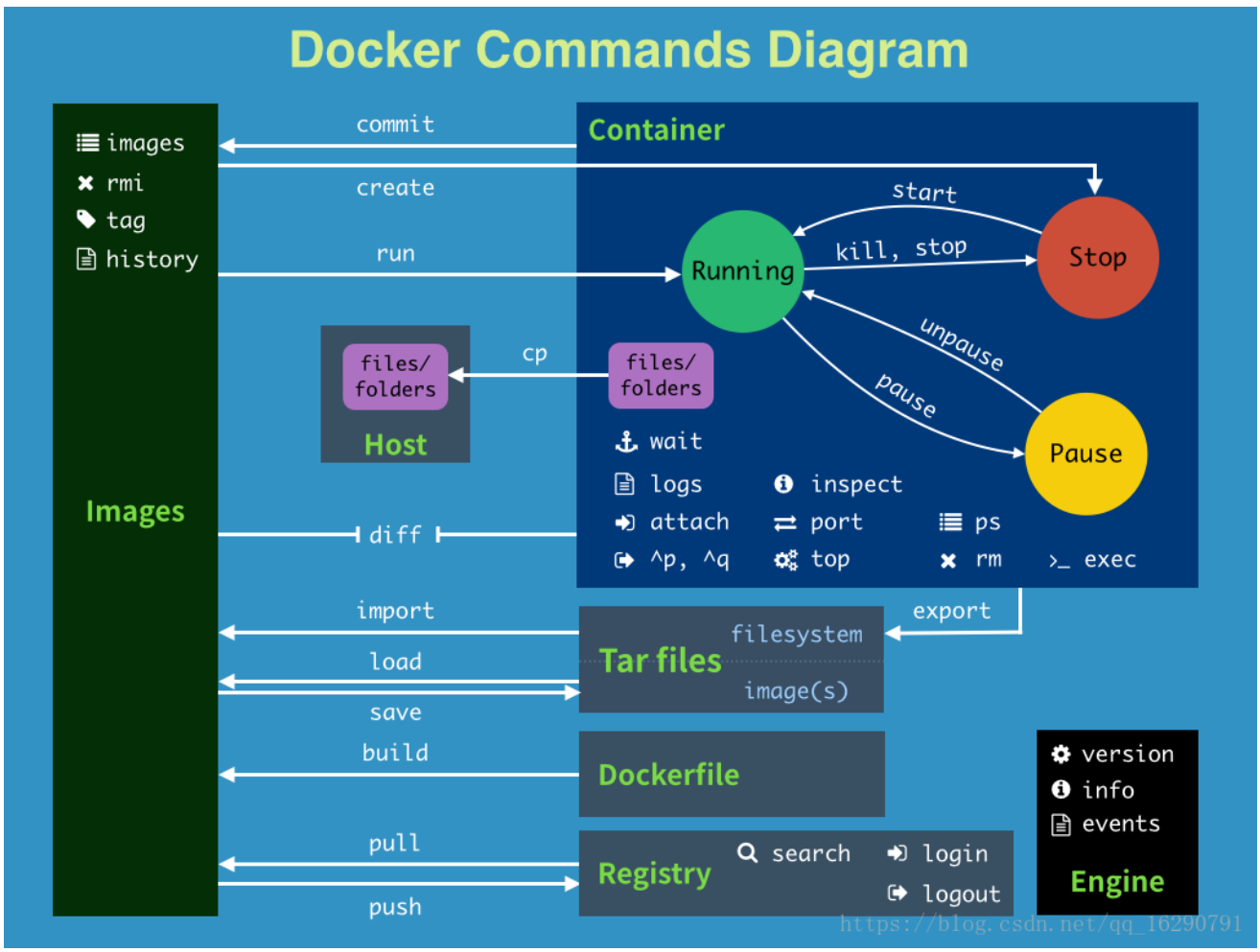
删除容器

```
# 删除容器 -- docker rm 容器ID
docker rm 容器ID  # 不能删除正在运行的容器
-f # 强制删除

# 删除所有
docker rm -f $(docker ps -aq)
```

常用的其他命令

```
# 后台启动一个容器
docker run centos
# 问题:  docker ps -a 查看没有启动
# 原因: docker 使用后台启动，就必须要有一个前台进程，不然doncker会认为没有应用，就会自动停止
```

```
# 查看日志
docker logs -ft  容器ID
--tail 10 # 显示的条数

# 查看容器中的进程
docker top 容器ID

# 查看镜像的元数据
docker inspect 容器ID

# 进入正在运行的容器
docker exec -it 容器ID  /bin/bash # 方式一 进入后开启一个新的终端
docker attach 容器ID # 方式二 进入正在运行的终端

# 从容器内拷贝文件到主机上
docker cp 容器ID:/home/a.txt ./

# 将容器打包成镜像
docker commit -m "描述" 容器ID 镜像名:tag

# 查看CPU 内存信息
docker stats
```



五、docker镜像

1、什么是docker的镜像

docker镜像是一个轻量级的、可执行的独立的软件包，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要的所有内容（代码、运行环境、库、环境变量、配置文件）

如何得到镜像：

- 从远程仓库下载
- 自己制作

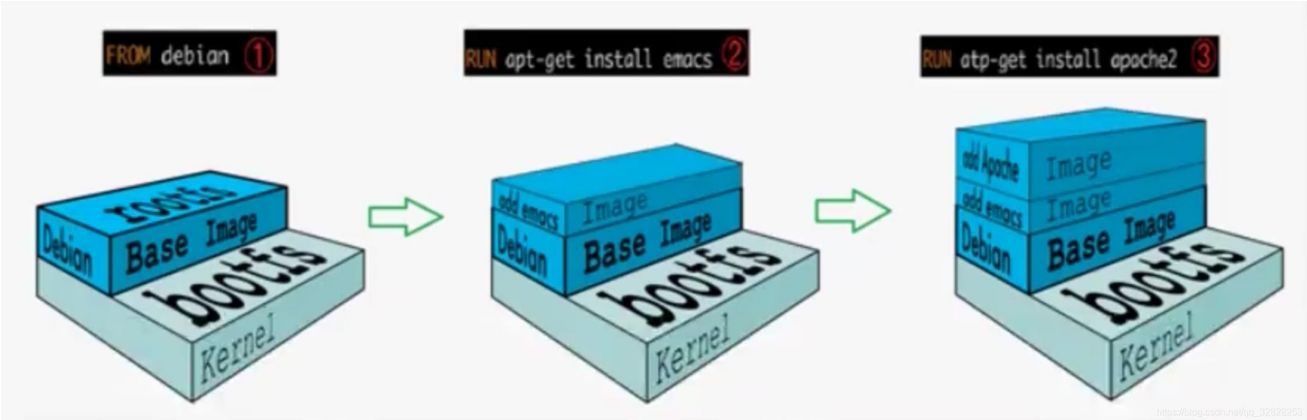
2、镜像加载原理

`UnionFS（UFS）联合文件系统`：是一种分层、轻量级并且高新能的文件系统，其最主要的功能是将多个不同位置的目录联合挂载到同一个目录下。

Docker的镜像实际上就是由一层一层的文件系统组成，这种层级的文件系统就是UFS

bootfs（doot file system）主要包含bootlocader和kernel，bootlocader主要是引导加载kernel，Linux刚启动时会加载bootfs文件系统，Docker的最底层就是bootfs，当boot加载完成后整个内核就在内存中了，这个时候内存的使用权已经由bootfs转交给内核，同时系统也会自动卸载bootfs。

rootfs（foot file system）在boootfs之上就是，典型的Linux文件系统（/etc、/bin）等标准目录。rootfs就是各种不同的发行版。



3、分层原理

```
[root@docker ~]# docker pull redis
Using default tag: latest
latest: Pulling from library/redis
a2abf6c4d29d: Already exists 已经存在的层将不会再重新下载
c7a4e4382001: Pull complete
4044b9ba67c9: Pull complete
c8388a79482f: Pull complete
413c8bb60be2: Pull complete
1abfd3011519: Pull complete
Digest: sha256:db485f2e245b5b3329fdc7eff4eb00f913e09d8feb9ca720788059f
dc2ed8339
Status: Downloaded newer image for redis:latest
docker.io/library/redis:latest
[root@docker ~]#
```

可以通过docker inspect redis查看每一个层

```
"RootFS": {
    "Type": "layers",
    "Layers": [
        "sha256:2edcec3590a4ec7f40cf0743c15d78fb39d8326bc029073b41ef9727da6c851f",
        "sha256:9b24afeb7c2f21e50a686ead025823cd2c6e9730c013ca77ad5f115c079b57cb",
        "sha256:4b8e2801e0f956a4220c32e2c8b0a590e6f9bd2420ec65453685246b82766ea1",
        "sha256:529cdb636f61e95ab91a62a51526a84fd7314d6aab0d414040796150b4522372",
        "sha256:9975392591f2777d6bf4d9919ad1b2c9afa12f9a9b4d260f45025ec3cc9b18ed",
        "sha256:8e5669d8329116b8444b9bbb1663dda568ede12d3dbcce950199b582f6e94952"
    ]
},
"Metadata": {
    "LastTagTime": "0001-01-01T00:00:00Z"
}
```

每一个都是一个层

4、commit 生成新的镜像

```
# 生成新的镜像
docker commit -m="描述信息" -a="作者" 容器id 生成的镜像名:[tag]
```

---

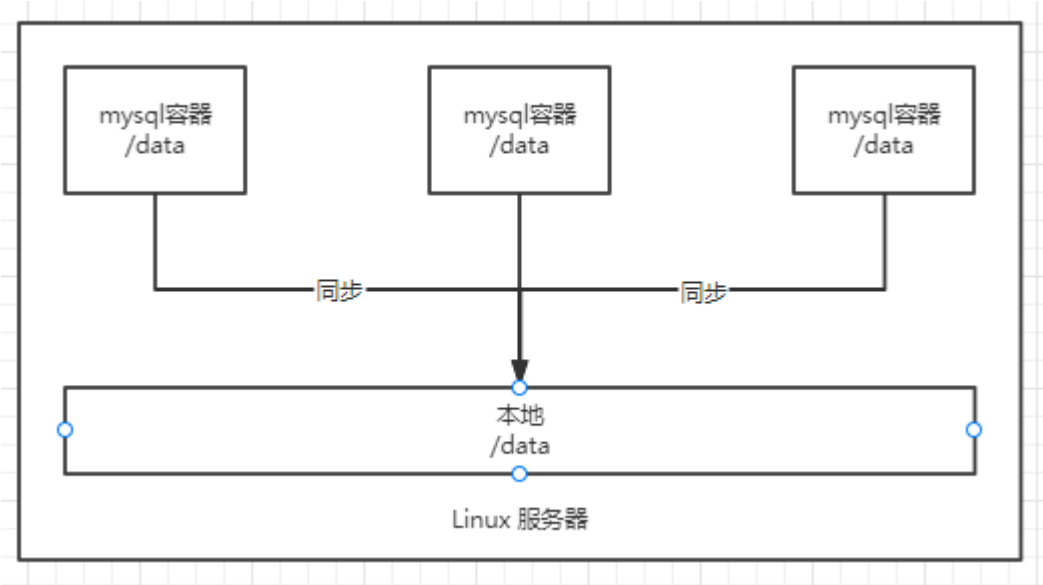
docker 的深入

六、容器数据卷

1、什么是数据镜像卷

如果将数据存储再容器中，那么删除容器就等于删除数据，所有需要将`数据持久化`,也就是将数据存储到本地中。

本质：就是容器之间可以实现数据共享，将Docker中产生的数据同步到本地，说白了就是目录的挂载，将容器里的目录挂载到本地。



总结：容器数据持久化和同步，也可以实现容器之间的数据共享，是一个双向绑定

2、创建使用数据卷

> 方式一：使用 -v 参数实现

```
# 启动即挂载
docker run -it -v 本地挂载目录:容器挂载目录 镜 像名 启动bash

# 实例 将MySQL的/home目录挂载到本地的/home/data目录
docker run -it -v /home/data:/home mysql /bin/bash

# 通过docker inspcet 查看挂载信息
```

```
"Mounts": [
    {
        "Type": "bind",
        "Source": "/home/data",
        "Destination": "/home",
        "Mode": "",
        "RW": true,
        "Propagation": "rprivate"
    },
```

实战：实现MySQL数据的持久化，将MySQL的的数据目录和配置文件挂载到本地

```
# 获取镜像
docker pull mysql:5.7

# 注意点:
# 1）运行容器时需要做数据目录挂载
# 2）启动mysql需要配置密码

# 去启动容器
```

```
docker run -d -p 3306:3306 -v /home/mysql/conf:/etc/mysql/conf.d -v
/home/mysql/data:/var/lib/mysql -e MYSQL_ROOT_PASSWORD=123456 --name mysql mysql:5.7

# 然后通过Navicat链接mysql创建一个test库，再在本地挂载的目录下喜欢查看
[root@docker data]# ls
auto.cnf         client-key.pem  ib_logfile1         private_key.pem  sys
ca-key.pem       ib_buffer_pool  ibtmp1              public_key.pem   test
ca.pem           ibdata1         mysql               server-cert.pem
client-cert.pem  ib_logfile0     performance_schema  server-key.pem
# 已经有了test这个库
# 然后将容器散出掉 再次查看
[root@docker data]# docker ps -a -n 1
CONTAINER ID   IMAGE       COMMAND                  CREATED          STATUS
PORTS     NAMES
d3f2cd4d46a0   mysql:5.7   "docker-entrypoint.s…"   27 minutes ago   Exited (0) 54 seconds ago
          mysql
[root@docker data]# docker rm -f d3f2cd4d46a0
d3f2cd4d46a0
[root@docker data]# ls
auto.cnf    client-cert.pem  ibdata1      mysql               public_key.pem   sys
ca-key.pem  client-key.pem   ib_logfile0  performance_schema  server-cert.pem  test
ca.pem      ib_buffer_pool   ib_logfile1  private_key.pem     server-key.pem
# 再次查看数据 依然存在 这就证明了数据持久成功
```

3、具名挂载和匿名挂载

```
# 匿名挂载 挂载时只写了容器内的路径就是匿名挂载
-v 容器内路径

# 查看所有的 volume（卷） 信息
[root@docker data]# docker volume ls
DRIVER    VOLUME NAME
local     b3917b06acdf135ea74f4dc669998986a4eb7d17104d9fd72e4c0c6c8ea2cf69

# 具名挂载
-v 卷名:容器内路径

# 创建一个具名挂载 推荐使用具名挂载
 docker run -d -P  -v nginx:/etc/nginx nginx
# 使用volume ls查看
[root@docker data]# docker volume ls
DRIVER    VOLUME NAME
local     b3917b06acdf135ea74f4dc669998986a4eb7d17104d9fd72e4c0c6c8ea2cf69
local     nginx
# 可以发现就有了一个 nginx的卷名

# 查看卷的位置
docker colume inspect 具名
[root@docker data]# docker volume inspect nginx
[
    {
```

```
        "CreatedAt": "2022-01-09T12:32:57+08:00",
        "Driver": "local",
        "Labels": null,
        "Mountpoint": "/var/lib/docker/volumes/nginx/_data", # 挂载的具体地址
        "Name": "nginx",
        "Options": null,
        "Scope": "local"
    }
]


# 如何区分是那种挂载方式
-v 容器内路径 # 匿名改在
-v 卷名:容器内路径 # 具名挂载
-v 本地目录:容器内路劲 # 指定路径挂载
```

4、扩展：

```
# 设置对目录的读写权限 限制的是本地的
# 只读 只能通过宿主机来改变
docker run -d -P -v 具名:/etc/nginx:ro nginx
docker run -d -P -v 具名:/etc/nginx:rw nginx
```

七、Dockerfile

就是用来构建Docker镜像的构建文件！命令脚本！

1、指令

指令	作用	示例
FROM	指定基础镜像，一般是官方提供的干净基础镜像	FROM centos
MAINTAINER	指定维护者的信息 作者 （姓名+邮箱）	
RUN	镜像构建的是很好需要执行的命令	
ADD	可以是 一个软件的压缩包 会自动解压 比如要做一个tomcat的镜像 就COPY一个tomcta的压缩包进去 会自动解压	
WORKDIR	设置工作目录	
VOLUME	设置卷 会自动挂载到主机（本地）	
EXPOSE	对外开放的端口 == -p	
CMD	指定容器启动时候 要指定的命令 只有最后一个会生效	
ENTRYPOINT	指定容器启动时要执行的命令 可以追加命令	
ONBUILD	当构建被继承 Dockerfile 这个是时候就会运行 ONBUILD的指令。是一个触发指令	
COPY	将文件拷贝到镜像中	
ENV	构建的时候设置环境变量	

2、构建

1641708045060

实战1：制作一个自己的Centos

```
# 创建一个目录
mkdir my-centos
# 创建一个dockerfile文件
vim my-dockerfil-centos
FROM centos     # 指定基础镜像文件

ENV MYPATH /usr/local    # 设置环境变量
WORKDIR $MYPATH     # 指定工作目录

RUN yum -y install vim net-tools     # 预装软件

EXPOSE  80  # 暴露端口

CMD echo "-----end-----"

# 通过这个文件构建
docker build -f my-dockerfil-centos -t my-centos:v1.0 .
```

```
# 可以查看到我们自己构建的镜像
[root@docker my-centos]# docker images
REPOSITORY   TAG        IMAGE ID       CREATED         SIZE
my-centos    v1.0       fa7d17f5f47c   4 minutes ago   326MB

# 可以通过history查看构建过程
docker history 镜像ID
```

CMD 与 ENTRYPOINT的区别

```
# CMD
# 会把我们通过 run 的时传进去的参数 替换掉CMD指定的的命令

# 创建一个dockerfile
vim cmd-contos
FROM centos
CMD ["ls" "-a"]

# 构建
docker build -f cmd-contos -t cmd-contos .

# 启动一下 不加参数
[root@docker my-centos]# docker run cmd-centos
.
..
.dockerenv
bin

# 启动一下 添加参数
[root@docker my-centos]# docker run cmd-centos -l
docker: Error response from daemon: OCI runtime create failed: container_linux.go:380: starting
container process caused: exec: "-l": executable file not found in $PATH: unknown.
ERRO[0000] error waiting for container: context canceled


# ENTRYPOINT
# 会把我们通过 run 的时传进去的参数 追加到原有指定的后面

# 创建一个dockerfile
vim entrypoint-contos
FROM centos
CMD ["ls" "-l"]

# 构建
docker build -f entrypoint-contos -t entrypont-contos .

# 启动一下 不加参数
[root@docker my-centos]# docker run entrypoint-centos
total 0
lrwxrwxrwx   1 root root   7 Nov  3  2020 bin -> usr/bin
drwxr-xr-x   5 root root 340 Jan  9 07:15 dev
drwxr-xr-x   1 root root  66 Jan  9 07:15 etc
```

```
drwxr-xr-x   2 root root   6 Nov  3  2020 home


# 启动一下 添加参数
[root@docker my-centos]# docker run entrypoint-centos -l
total 0
drwxr-xr-x   1 root root   6 Jan  9 07:16 .
drwxr-xr-x   1 root root   6 Jan  9 07:16 ..
-rwxr-xr-x   1 root root   0 Jan  9 07:16 .dockerenv
lrwxrwxrwx   1 root root   7 Nov  3  2020 bin -> usr/bin
drwxr-xr-x   5 root root 340 Jan  9 07:16 dev
drwxr-xr-x   1 root root  66 Jan  9 07:16 etc
```

实战二：创建Tomcat镜像

准备镜像文件 tomcat压缩包，jdk压缩包，一个构建的文件dockerfile！

```
[root@docker tomcat]# ll
total 147772
-rw-r--r-- 1 root root   8938514 Jan  9 15:34 apache-tomcat-7.0.73.tar.gz
-rw-r--r-- 1 root root         0 Jan  9 15:35 dockerfile
-rw-r--r-- 1 root root 142376665 Jan  9 15:34 jdk-7u67-linux-x64.tar.gz
[root@docker tomcat]#
```

编写dockerfile文件 使用Dockerfile 会被自动寻找

```
# vim Dockerfil

FROM centos
MAINTAINER liuhai<308766753@qq.com>

# COPY readme.txt /usr/local/readme.txt

ADD jdk-7u67-linux-x64.tar.gz /usr/local/
ADD apache-tomcat-7.0.73.tar.gz /usr/local/

RUN yum -y install vim net-tools
RUN ln -s /usr/local/jdk1.7.0_67 /usr/local/java
RUN ln -s /usr/local/apache-tomcat-7.0.73 /usr/local/tomcat

ENV MYPATH /usr/local/
WORKDIR $MYPATH

ENV JAVA_HOME /usr/local/java
ENV TOMCAT_HOME /usr/local/tomcat
ENV PATH $PATH:$JAVA_HOME/bin:$TOMCAT_HOME/bin

EXPOSE 8080

CMD /usr/local/tomcat/bin/startup.sh && tail -0f /usr/local/tomcat/logs/catalina.out

# 构建一个镜像
doker build -t diytomcat .
```

```
# 查看是否构建成功
[root@docker tomcat]# docker images
REPOSITORY   TAG        IMAGE ID       CREATED         SIZE
diytomcat    latest     663ce1381c7d   2 minutes ago   604MB

# 启动
docker run -d -p 8080:8080 diytomcat
```