

Docker

xbZhong

2025-10-01

[本页PDF](#)

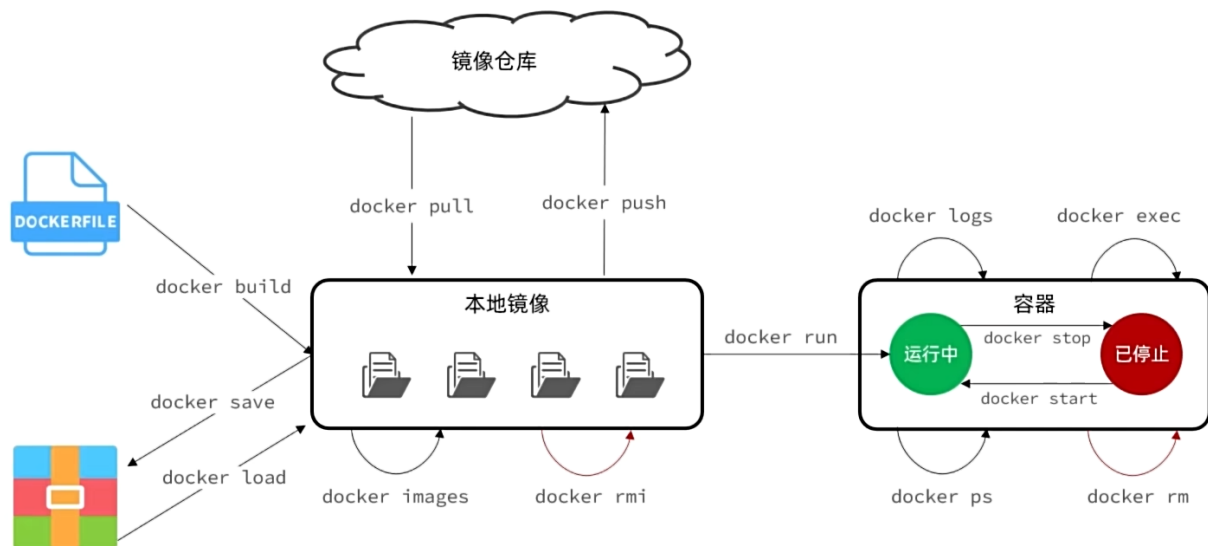
Docker

当我们利用Docker安装应用时，Docker会自动搜索并下载应用**镜像**。镜像不仅包含应用本身，还包含应用运行所需要的环境、配置、系统函数库。Docker会在运行镜像时创建一个**隔离环境**，称为**容器**

```
docker run -d \  
--name mysql \  
-p 3306:3306 \  
-e TZ=Asia/Shanghai \  
-e MYSQL_ROOT_PASSWORD=123 \  
mysql
```

- `docker run`：创建并运行一个容器，`-d` 是让容器在后台运行
- `--name mysql`：给容器起个名字，必须唯一
- `-p <宿主机端口>:<容器端口>`：设置端口映射
 - docker的进程是对外隔离的
 - 外部想要访问docker进程可以通过访问宿主机的端口从而访问到docker应用
- `-e KEY = VALUE`：设置环境变量
- `mysql`：运行的镜像名
 - 一般由两部分组成：`[repository]:[tag]`
 - 其中 `repository` 是镜像名，`tag` 是镜像版本

常见命令



高级软件工程师

image-20250929204331485

- `docker pull` : 从远程仓库拉取镜像
- `docker images` : 查看本地镜像
- `docker rmi` : 删除本地镜像
- `docker push` : 将镜像推送到远程仓库
- `docker build` : 根据 Dockerfile 构建镜像
 - `-t` : 给镜像起名称, 格式是 `repository:tag`
 - `.` : 指定Dockerfile所在的目录, 如果在当前目录, 指定为 `.`
- `docker save` : 将镜像保存为 `tar` 文件
 - `-o` : 后面跟文件名称
- `docker load` : 从 `tar` 文件加载镜像
 - `docker load -i <镜像文件.tar>`
- `docker run` : 创建并启动容器
 - `-d` : 后台运行
 - `--name` : 后面跟容器名字
 - `-p` : 跟端口映射, `-p [宿主端口]:[容器端口]`
 - `--network` : 后面跟网络名称
 - `-e` : 设置环境变量
 - 可以设置容器的环境变量
 - `--privileged` : 特权模式
 - 设置为 `true` 则允许容器访问宿主机设备
 - `--hostname` : 指定容器的主机名
- `docker stop` : 停止运行中的容器
- `docker start` : 启动已停止的容器
- `docker ps` : 查看运行中的容器
 - `-a` : 显示所有容器
- `docker rm` : 删除容器
 - `-f` : 强制删除
- `docker logs` : 查看容器日志
 - `-f` : 持续跟踪日志
- `docker exec` : 进入容器或执行命令
 - `-it` : 交互式终端

- `docker update` : 对容器信息进行更新

命令别名

将别名写入 `Shell` 配置文件, 如 `~/.bashrc` 或者 `~/.zshrc` , 如下

```
alias dps = 'docker ps -a'
```

然后重新加载配置: `source ~/.bashrc`

可以使用 `alias` 查看已定义的别名

数据卷

数据卷是一个虚拟目录, 是容器内目录和宿主机目录之间映射的桥梁

- 容器删除后数据仍保留
- 直接修改宿主机文件, 容器内实时生效

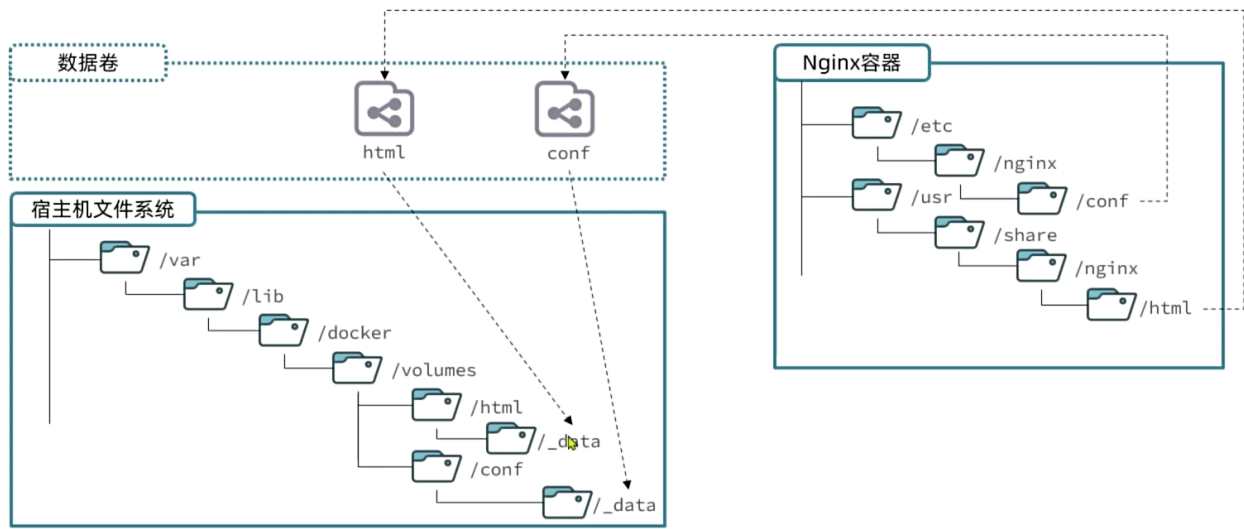


image-20250929210608574

常用命令

- `docker volume create` : 创建数据卷
- `docker volume ls` : 查看所有数据卷
- `docker volume rm` : 删除指定数据卷
- `docker volume inspect` : 查看某个数据卷详情
- `docker volume prune` : 清除数据卷

数据挂载

- 在执行 `docker run` 命令的时候, 使用 `-v 数据卷:容器内目录` 可以完成数据卷挂载
- 使用 `-v 本地目录:容器内目录` 可以完成数据在本地目录的挂载
 - 必须以 `/` 或者 `./` 开头, 否则会被识别成数据卷
- 宿主机默认目录: `/var/lib/docker/volumes/`
 - 数据卷存储位置: `/var/lib/docker/volumes/数据卷名称/_data/`
- 容器默认目录: `/var/lib/docker/`

- 数据卷存储位置: `/var/lib/docker/volumes/`

自定义镜像

镜像结构

- 采用分层存储结构
- 由多个只读层（Layers）堆叠而成
- 可以共享很多基础的层

镜像结构

入口（Entrypoint）

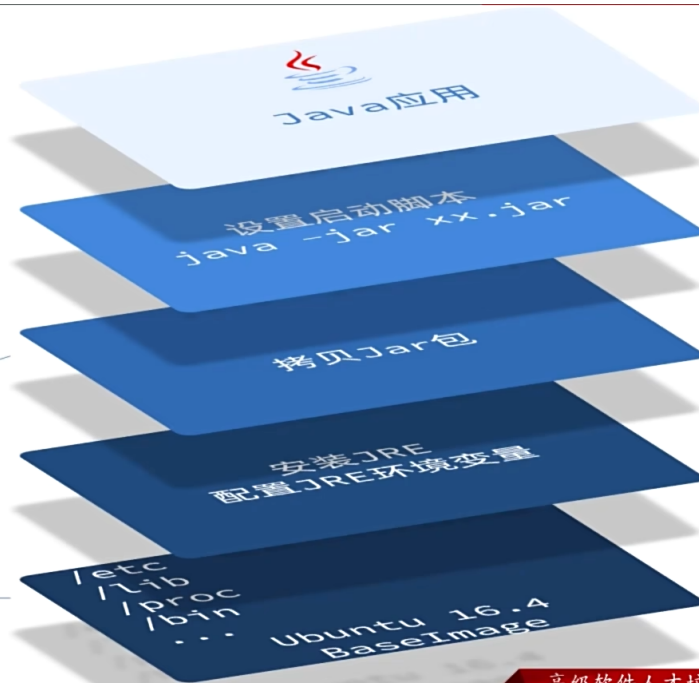
镜像运行入口，一般是程序启动的脚本和参数

层（Layer）

添加安装包、依赖、配置等，每次操作都形成新的一层。

基础镜像（BaseImage）

应用依赖的系统函数库、环境、配置、文件等



高级软件人才培养去家

image-20250929225240951

Dockerfile

Dockerfile（定义Docker镜像的结构和构建逻辑）就是一个文本文件，其中包含一个个的指令，用指令来说明要执行什么操作来构建镜像

常见语法

- **FROM**：指定基础镜像
- **ENV**：设置环境变量
- **COPY**：拷贝本地文件到镜像的指定目录
- **RUN**：执行Linux的shell命令
- **EXPOSE**：指定容器运行时监听的端口
- **ENTRYPOINT**：镜像中应用的启动命令，容器运行时调用

网络

默认情况下，所有容器都是以bridge方式连接到Docker的一个虚拟网桥上：

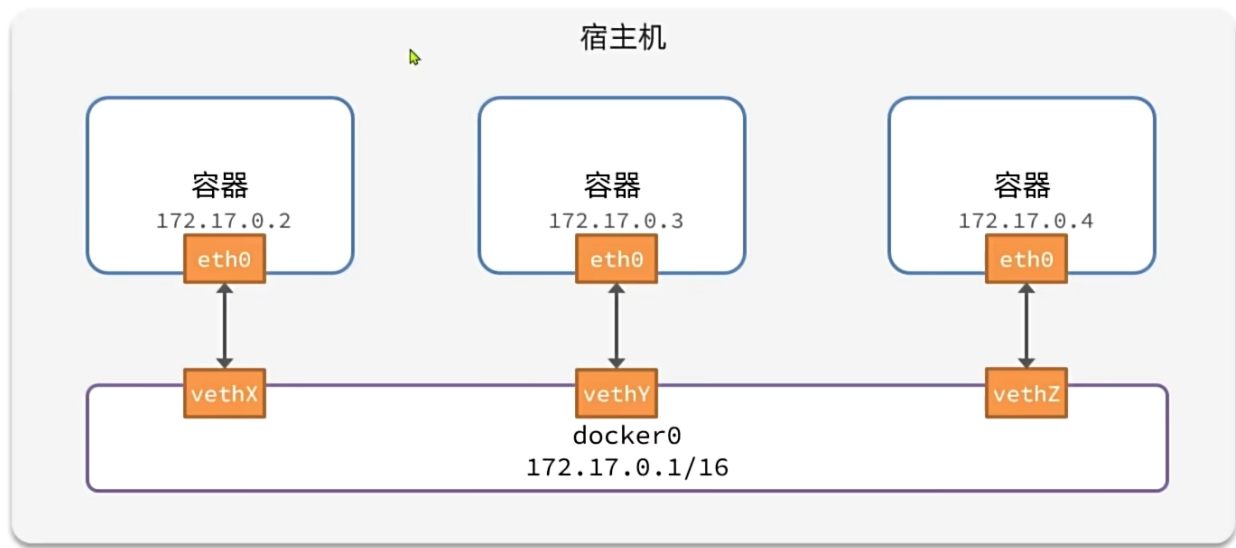


image-20250930114318613

加入自定义网络的容器才可以通过容器名相互访问，常见命令如下：

- `docker network create` : 创建一个网络
- `docker network ls` : 查看所有网络
- `docker network rm` : 删除指定网络
- `docker network prune` : 清楚未使用的网络
- `docker network connect` : 使指定容器加入某网络
 - `docker network connect [网络名] [容器名]`
- `docker network disconnect` : 使指定容器离开某网络
- `docker network inspect` : 查看网络详细信息

DockerCompose

通过一个单独的 `docker-compose.yml` 文件来定义一组相关联的应用容器，帮助我们实现多个相互关联的Docker容器的快速部署

格式

- `version` : 项目版本
- `services` : 容器
 - `image` : 镜像名称
 - `container_name` : 容器名称 (自定义)
 - `port` : 端口映射
 - `environment` : 环境变量
 - `volumes` : 数据卷挂载
 - `networks` : 网络名称
 - `build` : 构建镜像
 - `context` : 目录
 - `dockerfile` :
- `networks` : 网络配置
 - `name` : 网络名称

构建命令格式

- `docker compose [OPTIONS] [COMMAND]`
- Options
 - `-f` : 指定 `compose` 文件的路径和名称
 - `-p` : 指定 `project` 名称
- command
 - `up` : 创建并启动所有 `service` 容器
 - `down` : 停止并移除所有容器、网络
 - `ps` : 列出所有启动的容器
 - `logs` : 查看指定容器日志
 - `stop` : 停止容器
 - `start` : 启动容器
 - `restart` : 重启容器
 - `top` : 查看运行的进程
 - `exec` : 在指定的运行中容器执行命令