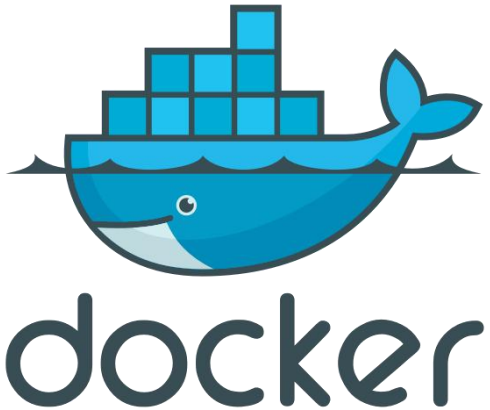
哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

Docker基础教程



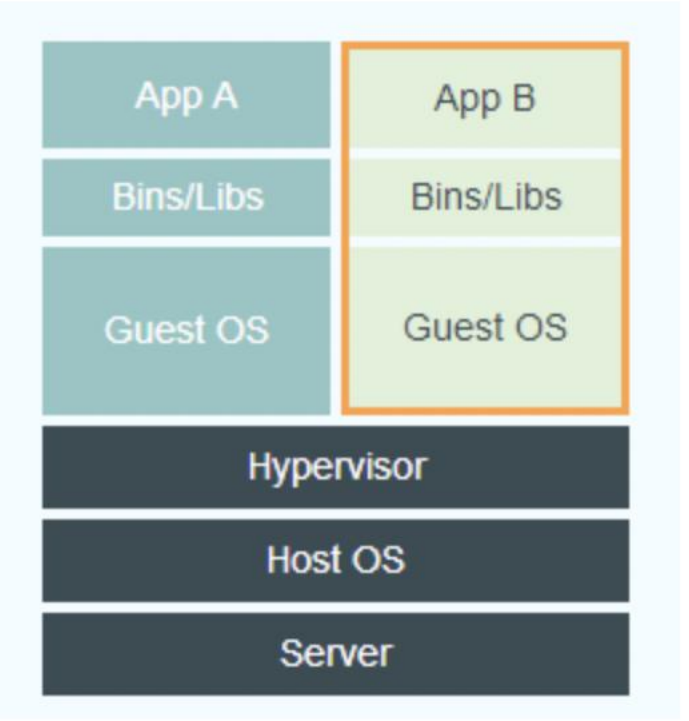
目 录

CONTENTS

01 概　述

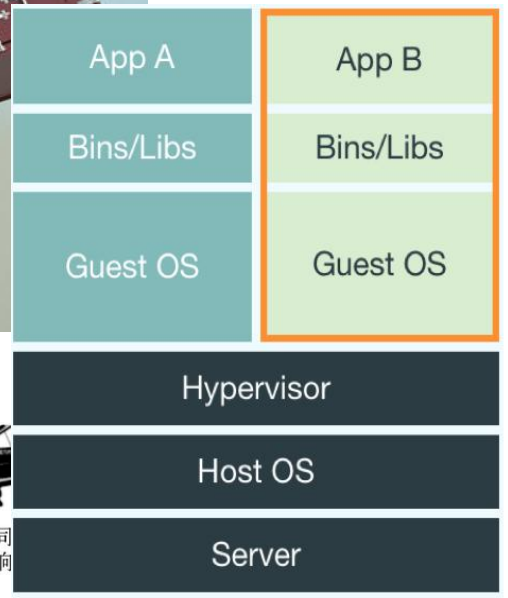
出现背景

- **环境不一致**：开发、测试和生产环境可能存在差异，使得应用在不同环境中表现不一致，导致“在我的机器上可以运行”的现象。
- **慢启动时间**：虚拟机需要较长的启动时间，影响了开发和测试的迭代速度。
- **资源开销大**：虚拟机通常需要较多的系统资源（如CPU和内存），导致资源利用效率低下。
- **部署过程繁琐**：手动部署过程中容易出错，配置和步骤繁琐，增加了上线过程的不确定性。
- **缺乏隔离性能**：在传统部署模式下，多个应用可能会共享相同的系统资源，造成相互影响和干扰。
- **跨平台兼容性问题**：应用在不同的操作系统或硬件配置上运行时，可能面临兼容性问题，导致迁移或扩展变得困难。

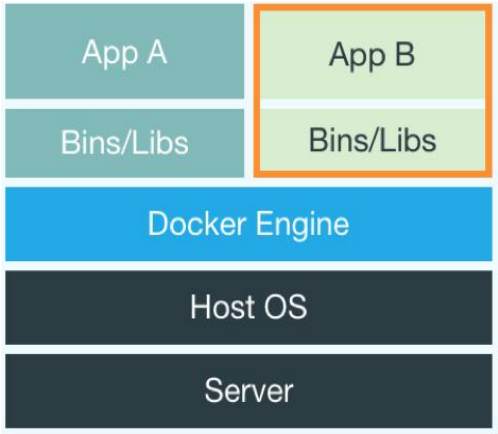


Docker的优势

- 避免环境差异
- 提高开发效率
- 提高资源利用率
- 方便管理和部署
- 支持跨平台



VM



Docker

02

基 本 组 成

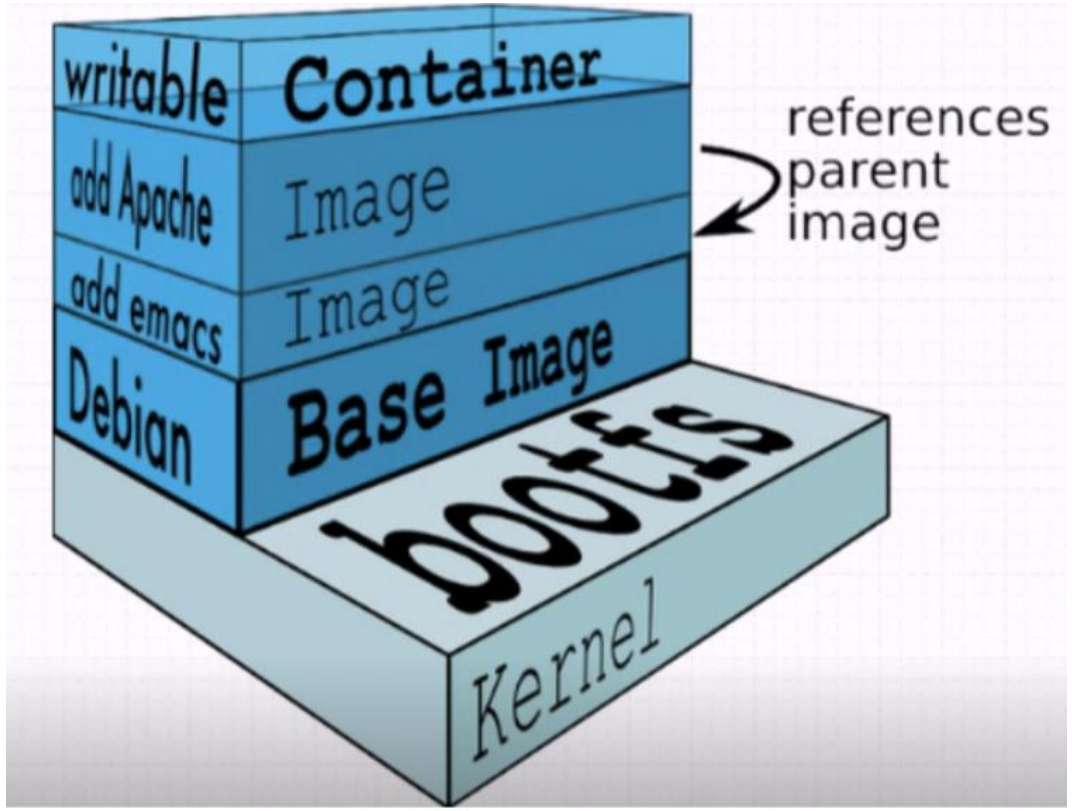
镜像（Image）

- **定义**：镜像是一个只读的模板，包含了应用及其所有依赖、运行环境以及配置等，通常用于创建容器。

特性：

- **只读**：镜像的内容是只读的，不能直接修改。
- **层叠式**：镜像是由多个层组成的，每一层都代表一个文件系统的变更。这种分层结构使得镜像共享底层内容，节省存储空间。
- **可复用**：镜像可以被多次使用来创建多个容器，且每个容器可以拥有自己的运行时状态。
- **版本管理**：镜像可以打标签（tag），不同标签代表不同版本的应用或配置。

镜像（Image）



层次结构

dockerfile

```
# 使用 OpenJDK 作为基础镜像
FROM openjdk:17-jdk-slim

# 设置工作目录
WORKDIR /app

# 将 JAR 文件复制到工作目录
COPY target/my-java-app.jar app.jar

# 指定容器运行时使用的命令
CMD ["java", "-jar", "app.jar"]
```

制作示例

容器（Container）

- **定义**：容器是镜像的一个可运行实例，是一个*轻量级*、*独立*的进程环境，能在其中运行**一个或一组**应用程序。
- 特性：
- **可读写**：容器是可读写的，运行时会在镜像的基础上创建一个可修改的文件系统层，用于存储运行时生成的数据或应用状态。
- **隔离性**：容器与主机系统和其他容器在进程、网络、文件系统等方面是隔离的，提供了良好的安全性和资源管理。
- **轻量级**：容器的启动速度快，资源占用少，相比虚拟机更加*高效*。
- **瞬态性**：容器可以随时创建、启动、停止和删除，能够快速适应应用负载的变化。

仓库（Reposity）

- **定义**：仓库是一个集中存储的地方，用于*保存和管理* Docker 镜像。它通常与 Docker Hub 或私有 仓库（Harbor）相关联，允许用户上传、下载和共享自己的 Docker 镜像。

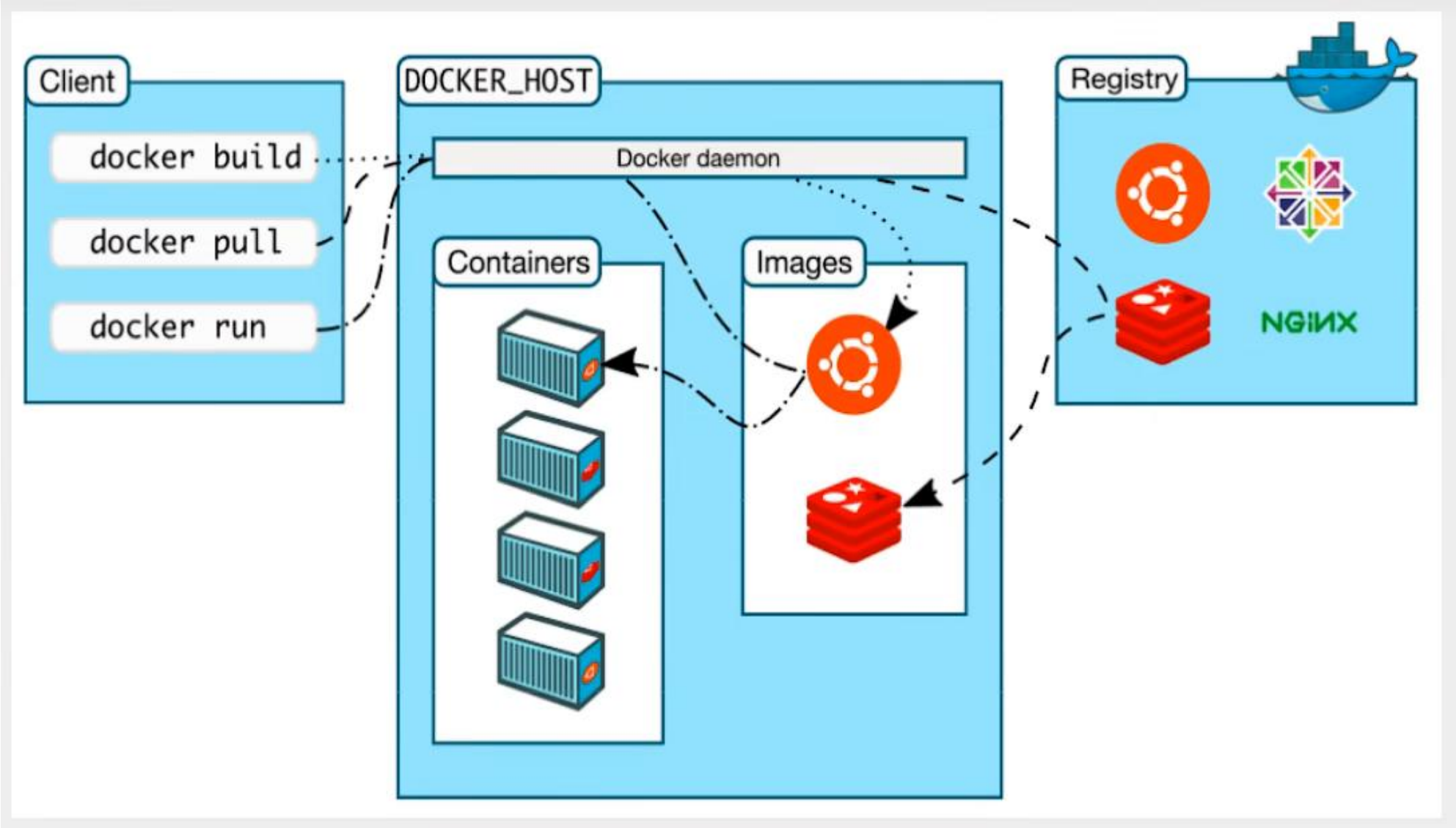
特性：

- **标签**：每个镜像可以有一个或多个标签，用于版本管理。例如，myapp:1.0 和 myapp:latest 可以指向同一个镜像的不同版本。
- **版本控制**：仓库允许用户管理和跟踪不同版本的镜像，便于在不同环境中进行持续集成和持续交付（CI/CD）的管理。
- **搜索与发现**：用户可以通过仓库*搜索*所需的镜像，查找第三方的应用镜像或使用官方的基础镜像。
- **API 接口**：大多数仓库提供 API 接口，方便集成自动化工具，如 CI/CD 系统，自动*推动*和*提取*镜像。


Harbor
搜索 Harbor...
中文简体
admin
项目
日志
系统管理
用户管理
机器人账户
仓库管理
复制管理
分布式分发
标签
项目定额
审查服务
垃圾清理
配置管理
项目
项目
私有 0
公开 5
总计 5
镜像仓库
私有 0
公开 44
总计 44
已使用的存储空间
15.40 GiB
新建项目
删除
所有项目
项目名称 访问级别 角色 类型 镜像仓库数 创建时间
flannel 公开 项目管理员 项目 2 2024/5/5 下午9:05
framework 公开 项目管理员 项目 1 2023/10/20 上午9:54
infrastructure 公开 项目管理员 项目 6 2024/6/1 下午2:54
library 公开 项目管理员 项目 27 2023/4/21 下午10:41
testreposity 公开 项目管理员 项目 8 2024/9/27 下午11:56
页面大小 15
1 - 5 共计 5 条记录

Harbor仓库示例

三者关系

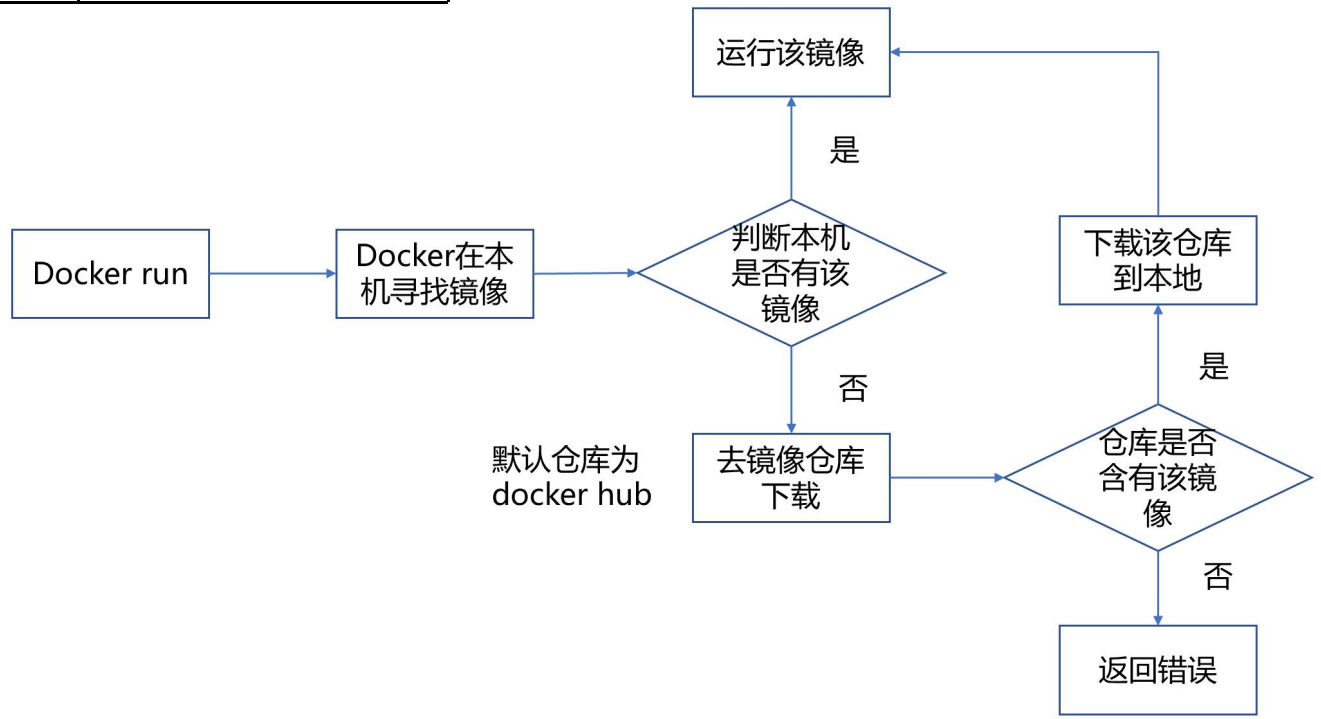

Client
docker build
docker pull
docker run
DOCKER_HOST
Docker daemon
Containers
Images
Registry
NGINX

安装

系统		工具	方式
Linux	CentOS	包管理工具：yum	指令
	Ubuntu	包管理工具：apt	
Windows	-	Docker Desktop	安装包
MacOS	-	Docker Desktop for Mac	

测试

```
[root@iZbp13qr3mm4ucsjumrlgqZ ~]# docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
2db29710123e: Pull complete
Digest: sha256:53f1bbee2f52c39e41682ee1d388285290c5c8a76cc92b42687eecf38e0af3f0
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.
```



镜像指令

命令	描述
docker pull <image>	从 Docker Hub 下载镜像
docker images	列出所有下载的镜像
docker rmi <image>	删除指定的镜像
docker build <options> <path>	使用 Dockerfile 构建镜像
docker tag <image> <new_image>	给镜像打标签
docker search <term>	在 Docker Hub 中搜索镜像

其他指令

命令	描述
docker --version	显示 Docker 的版本信息
docker info	显示 Docker 的系统信息
docker network ls	列出所有 Docker 网络

容器指令

命令	描述
docker run <options> <image>	创建并启动一个新容器
docker ps	列出当前运行的容器
docker ps -a	列出所有容器（包括停止的容器）
docker stop <container>	停止正在运行的容器
docker start <container>	启动一个已停止的容器
docker restart <container>	重启指定的容器
docker exec -it <container> <command>	在运行中的容器内执行命令
docker logs <container>	查看指定容器的日志
docker rm <container>	删除已停止的容器
docker inspect <container>	查看容器的详细信息

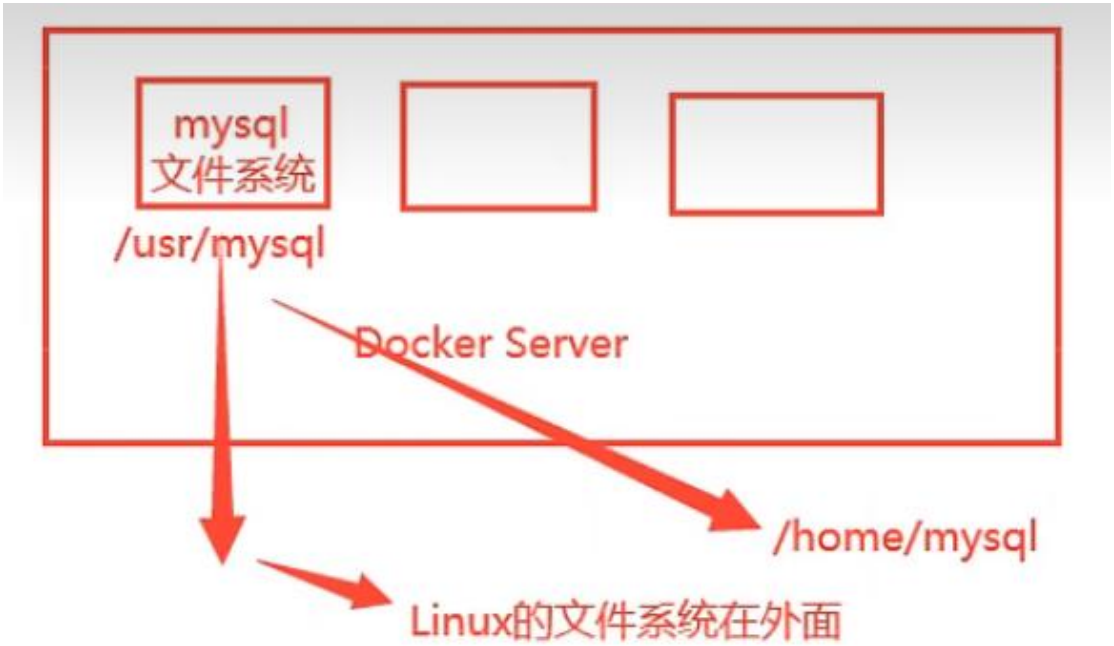
03

数据卷

数据卷（Volume）

- **出现原因**：为了解决容器化应用中数据管理和持久性的需求。在容器的生命周期中，容器内部的文件系统是*短暂*的，不具备数据持久化的特性。为此引入了数据卷的概念，以提供一个*持久*、*可共享*的数据存储解决方案。

挂载方式	示例命令
具名数据卷	docker run -v my_volume:/data <image>
匿名数据卷	docker run -v /data <image>
主机目录挂载	docker run -v /host/path:/container/path <image>



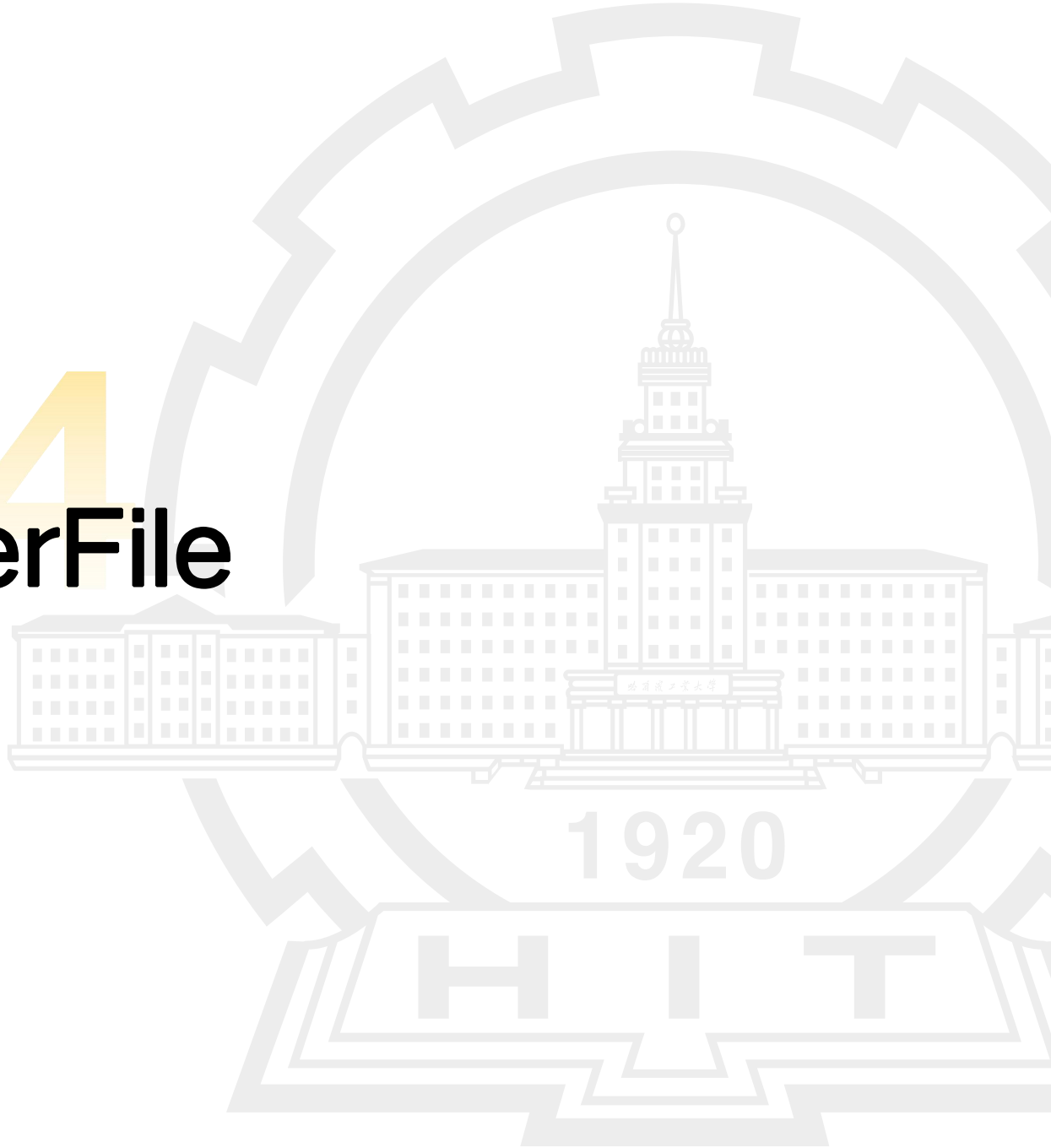
具名数据卷：my_volume 是用户定义的命名数据卷，/data 是容器内的挂载路径。

匿名数据卷：Docker 会自动创建一个临时数据卷，/data 是容器内的挂载路径。

主机目录挂载：/host/path 是主机上的目录路径，/container/path 是容器内部的挂载点

04

DockerFile



DockerFile

- 用于构建镜像的文件

- 构建步骤：

  1、编写一个dockerfile文件

  2、docker build 构建成为一个镜像

  3、docker run运行镜像

  4、docker push发布镜像到仓库。

常用命令

```
1.  FROM          # 基础镜像，一切从这里开始构建
2.  MAINTAINER    # 镜像是谁写的：姓名+邮箱
3.  RUN           # 镜像构建的时候需要运行的命令
4.  ADD           # 步骤：tomcat 镜像，这个 tomcat 压缩包。添加内容
5.  WORKDIR       # 镜像的工作目录
6.  VOLUME        # 挂载的目录
7.  EXPOSE        # 暴露端口配置
8.  CMD           # 指定这个容器启动的时候要运行的命令，只有最后一个会生效，可被
    替代
9.  ENTRYPOINT    # 指定这个容器启动的时候要运行的命令，可以追加命令
10. ONBUILD       # 当构建一个被继承 DockerFile 这个时候就会运行 ONBUILD 的指令。
    触发指令。
11. COPY          # 类似 ADD，将我们文件拷贝到镜像中
12. ENV           # 构建的时候设置环境变量
```

DockerFile

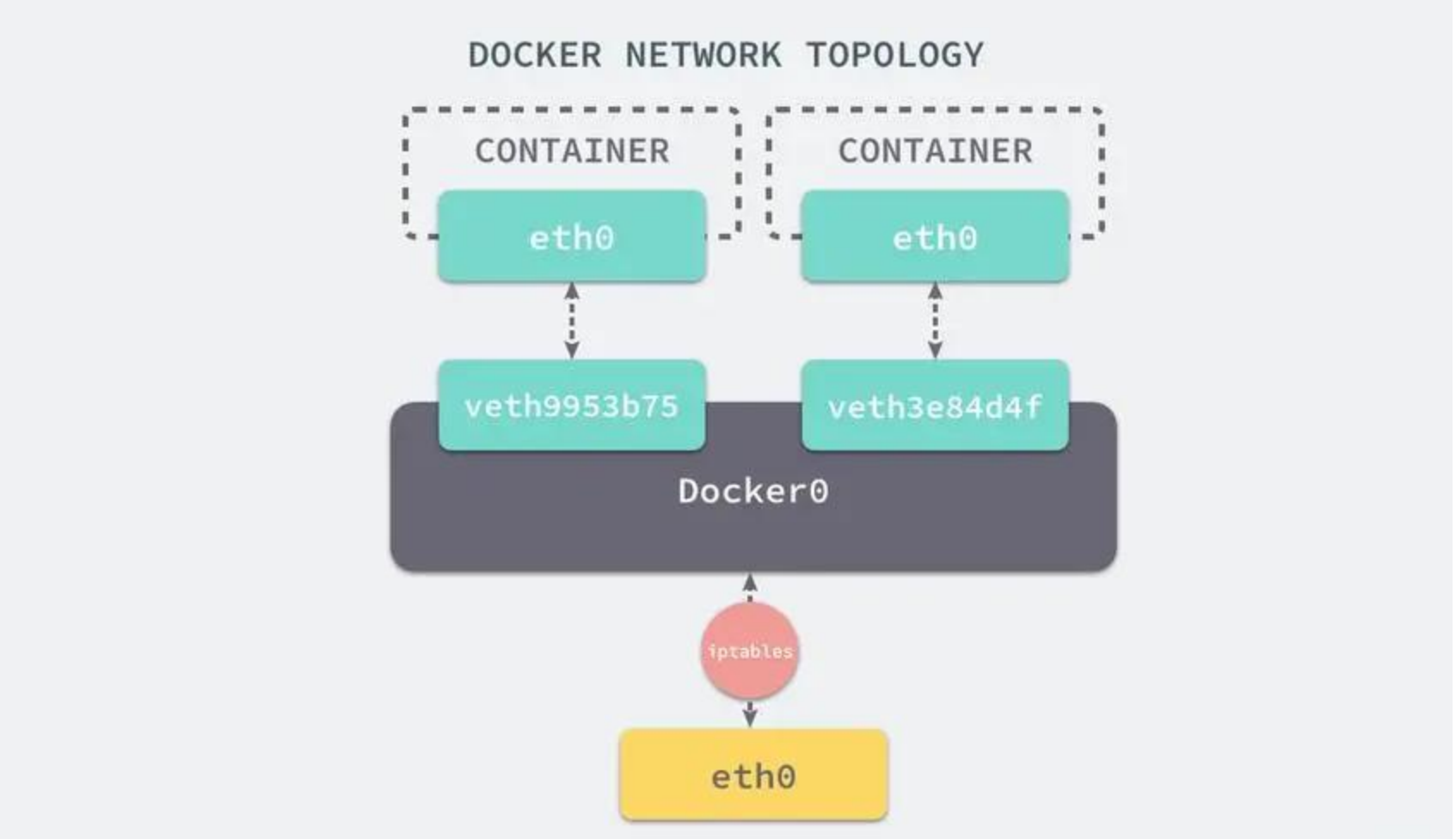
```
# 选择基础镜像，这里使用 OpenJDK 8
FROM openjdk:8

# 设置工作目录
WORKDIR /app

# 将 JAR 文件复制到容器中
COPY ./target/clusterAgent-0.0.1-SNAPSHOT.jar clusterAgent-0.0.1-SNAPSHOT.jar

# 暴露应用程序的端口（假设应用程序运行在 8086 端口）
EXPOSE 8086

# 设置启动命令
CMD ["java", "-jar", "clusterAgent-0.0.1-SNAPSHOT.jar"]
```

基础示例

Docker网络

网络模式	描述	使用场景
bridge	默认网络模式，容器之间可以通过 IP 地址通信，且具有隔离性。	适用于大多数容器化应用，尤其是需要互相通信的应用。
host	容器共享主机的网络栈，使用主机的 IP 地址。	性能关键的应用，如高性能服务或需要直接访问主机网络的应用。
none	不配置网络，容器无法与其他容器或外部网络通信。	用于需要完全隔离的场景，例如特定的安全需求或测试。

Docker网络



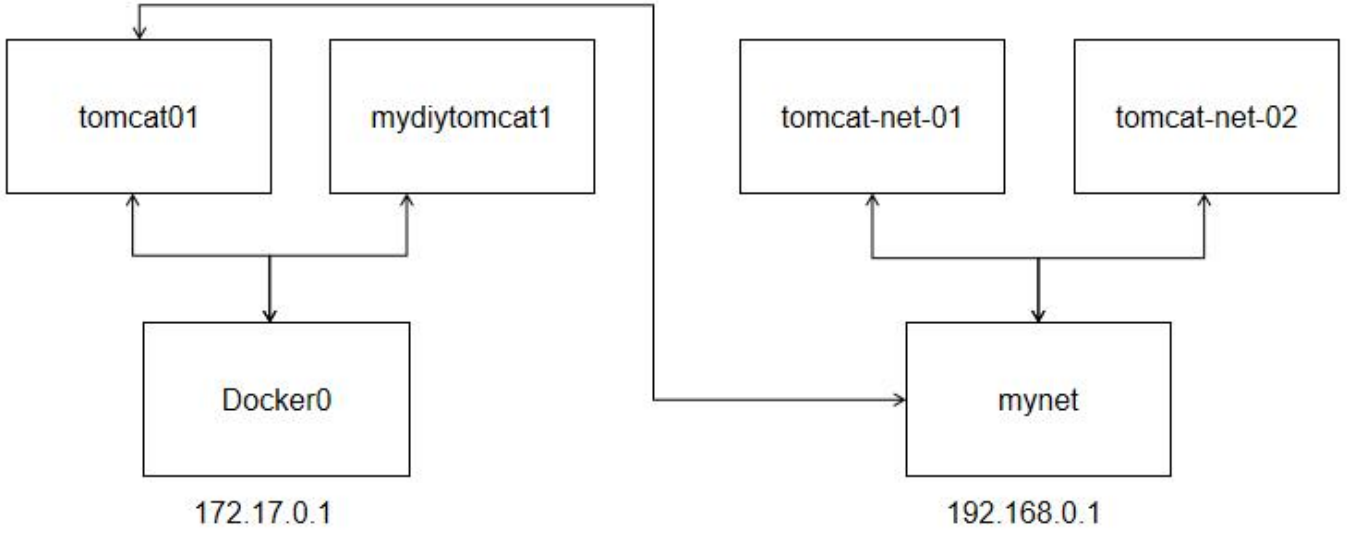
容器间网络模型图

自定义网络

```
[root@hit-17 ~]# docker network create --driver bridge --subnet 192.168.0.0/16 --gateway
192.168.0.1 mynet
31fe92f269859ca77c9701825938b1a5f43030f6e19f55c9949766ee8c62cc2f
[root@hit-17 ~]# docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
4ca979f7fe0d   bridge    bridge    local
947bc5b3781a   host      host      local
31fe92f26985   mynet     bridge    local
7a0494339374   none      null      local
```

自定义网络名称

容器和网络连通示例



感谢聆听！