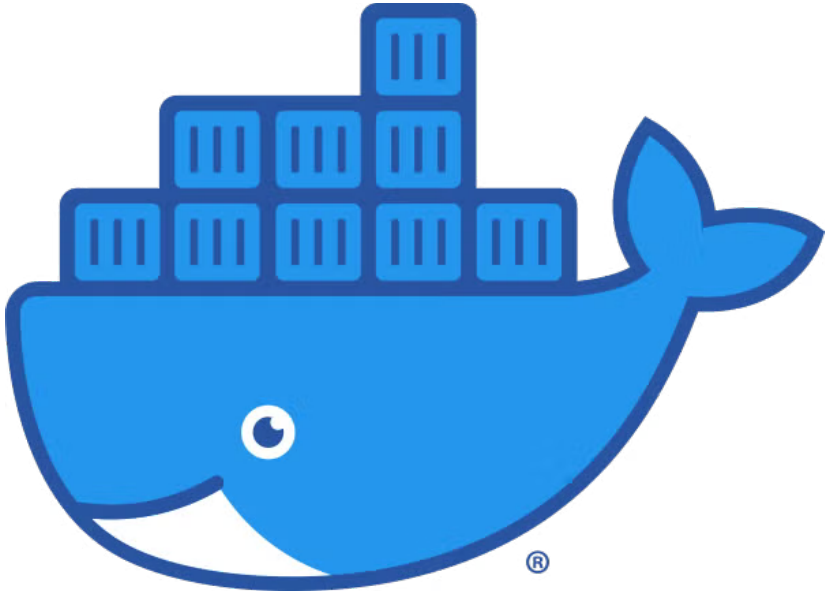
Ambientes Reprodutíveis com Docker

Minicurso Introdutório - Dia 2

Instrutor: Sérgio Fontes
E-mail: fontes.sergio@graduacao.uerj.br

> **Objetivo:** compreender como construir imagens personalizadas e gerenciar persistência de dados com volumes e bind mounts.



O que são Imagens Docker?

- Uma **imagem Docker** é um **modelo imutável** contendo tudo que um container precisa: sistema de arquivos, dependências e configurações.
- É construída a partir de um **Dockerfile**, onde cada instrução (`FROM`, `RUN`, `COPY`, `CMD` etc.) gera uma **camada**.
- As camadas são **armazenadas em cache**, **reutilizáveis** e **compartilháveis**.
- As imagens são **versionadas** e armazenadas em **repositórios** como o Docker Hub.
- Toda imagem deriva de uma **imagem base**, e o ponto inicial é `scratch`, uma imagem vazia.

> O Dockerfile é a *receita*, e a imagem é o *bolo pronto*.

Hierarquia das Imagens Docker

```
scratch
 ├── debian
 │    └── python:3.11-slim
 │         └── sua-imagem:latest
 └── alpine
      └── node:alpine
           └── sua-imagem:latest
```

- `scratch` → imagem vazia (0 bytes), base mínima.
- **Distribuições base**: `debian`, `alpine`, `ubuntu`.
- **Imagens de linguagem**: `python`, `node`, `golang`.
- **Imagens personalizadas**: criadas via Dockerfile.

> Tudo no Docker parte de `scratch`. Cada camada adiciona componentes até formar um ambiente completo.

O que é um Dockerfile?

- Um **Dockerfile** é um **arquivo de texto** que descreve **como construir uma imagem Docker**.
- Cada linha é uma **instrução declarativa**, executada sequencialmente.
- Cada instrução gera uma **nova camada cacheável**.
- O build é **determinístico** — qualquer pessoa pode reproduzir a mesma imagem.
- A primeira linha (`FROM`) define a **imagem base**; a última (`CMD`) define o **comando padrão**.

> O Dockerfile transforma configurações de sistema em **código versionável**, promovendo automação e reprodutibilidade.

Instruções Comuns do Dockerfile

Instrução	Função	Exemplo
FROM	Define a imagem base	`FROM python:3.11-slim`
RUN	Executa comandos e instala pacotes	`RUN pip install -r requirements.txt`
COPY / ADD	Copia arquivos para a imagem	`COPY . /app`
WORKDIR	Define o diretório de trabalho	`WORKDIR /app`
ENV	Define variáveis de ambiente	`ENV PORT=8080`
EXPOSE	Documenta a porta usada pela aplicação	`EXPOSE 8080`
CMD / ENTRYPOINT	Define o comando padrão	`CMD ["python", "main.py"]`

> Cada instrução cria uma camada independente e reaproveitável.

Bind Mounts vs Volumes

Tipo	Definição	Onde é configurado	Persistência	Controle
Bind mount	Conecta diretório do host ao container	`docker run -v /host:/cont`	Persistente	Controlado pelo usuário
Volume	Área de armazenamento gerenciada pelo Docker	`VOLUME /dados` no Dockerfile	Persistente	Gerenciado pelo Docker

> **Bind mounts** pertencem à fase de **execução** (`docker run`), enquanto **volumes** podem ser **sinalizados no Dockerfile**, mas sua **criação e montagem ocorrem durante a execução** (`docker run`).

Trabalhando com Volumes

- Volumes são áreas de **armazenamento persistente** gerenciadas automaticamente pelo Docker.
- São independentes do ciclo de vida do container: os dados permanecem mesmo após a remoção do container.
- Indicados para armazenar dados que precisam ser **mantidos entre execuções**, como bancos de dados, arquivos de log e resultados de aplicações.

Como declarar volumes?

Via Dockerfile

```
VOLUME /dados
```

Essa instrução **marca** o diretório `/dados` como um volume.
Se nenhum volume for especificado ao iniciar o container, o Docker cria um **volume anônimo** automaticamente.

> Volumes anônimos são úteis para testes rápidos, mas não são reutilizados automaticamente entre containers.

Via linha de comando

```
docker run -v meu_volume:/dados minha-imagem
```

Esse comando cria (ou reutiliza) um volume **nomeado** chamado `meu_volume` e o monta no caminho `/dados` dentro do container.

> Volumes nomeados são ideais para ambientes controlados e reutilização de dados entre containers e builds.

Onde os volumes são armazenados?

```
/var/lib/docker/volumes/<nome-do-volume>/_data
```

Esse é o caminho padrão no host Linux onde os dados dos volumes são armazenados.
Em Windows e macOS, o local pode variar dependendo do back-end utilizado (Docker Desktop, WSL2, etc).

> Use `docker volume inspect <nome>` para verificar o caminho exato de qualquer volume criado.