

# Docker

Na minha máquina funciona

Fernando Teixeira Alves de Araujo

# Agenda

1. Introdução
2. Arquitetura
3. Runtimes de containers
4. Dockerfile
5. Comandos docker
6. Docker Compose
7. Case: Rodando Serviço Local

# Introdução

O Docker é um software de código aberto usado para **implantar aplicativos** dentro de **containers** virtuais.

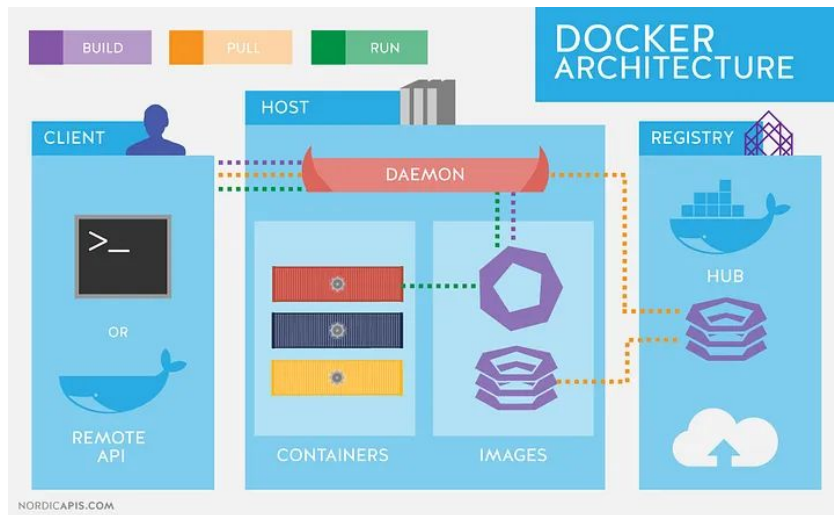
A containerização permite que vários aplicativos funcionem em **diferentes ambientes** complexos.

Por exemplo: o Docker permite executar o WordPress em sistemas Windows, Linux e macOS, sem problemas.

# Arquitetura

**Client:** Permite aos usuários **interagir** com o Docker e acessar os container via **linha de comando** ou **API Remota**.

**Host:** Fornece um ambiente completo para **executar aplicativos**, sendo composto pelo Daemon, imagens, containers, rede e volumes.



Para armazenar imagens customizadas de forma públicas ou privadas, é possível utilizar a própria conta do Docker Hub ou serviços de cloud como ECR.

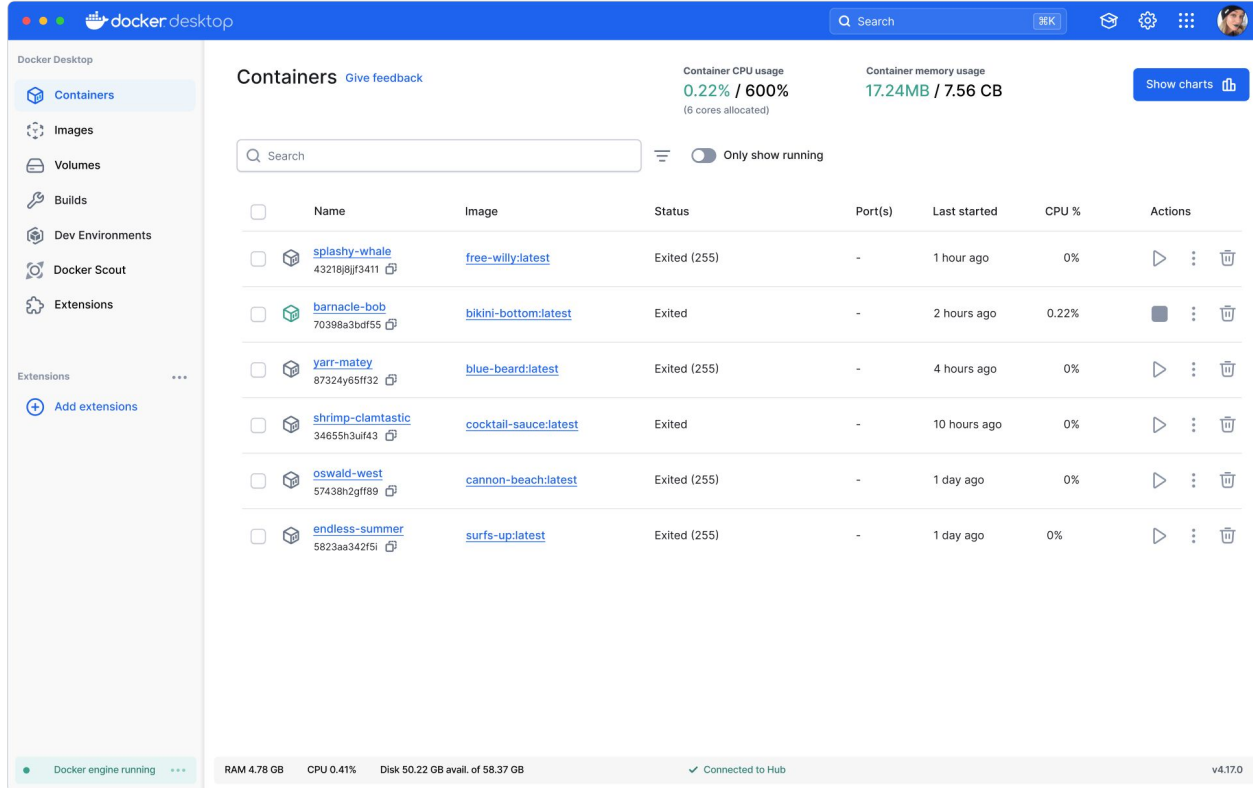
O **daemon** é responsável por todas as **ações** relacionadas aos containers e recebe comandos por meio do Client.

**Registry:** São serviços que fornecem locais de onde irá **armazenar e baixar as imagens**. Em outras palavras, o Registry, contém os repositórios Docker que hospedam as imagens. Como **Docker Hub**.

# Runtimes de containers



# Runtimes de containers - Docker Desktop

































The screenshot shows the Docker Desktop application window. The left sidebar contains navigation options: Containers (selected), Images, Volumes, Builds, Dev Environments, Docker Scout, and Extensions. Below these are 'Extensions' and 'Add extensions'. The main panel is titled 'Containers' and includes a search bar, a filter toggle for 'Only show running', and summary statistics for container CPU and memory usage. A table lists several containers, including 'splashy-whale', 'barnacle-bob', 'yarr-matey', 'shrimp-clamstastic', 'oswald-west', and 'endless-summer'. The bottom status bar shows system metrics like RAM, CPU, and disk space, along with the Docker engine status and connection to the Hub.

**Containers** [Give feedback](#)

Container CPU usage: 0.22% / 600% (6 cores allocated)  
Container memory usage: 17.24MB / 7.56 CB [Show charts](#)

Search

☐ Only show running

| <input type="checkbox"/> | Name   | Image                                 | Status       | Port(s) | Last started | CPU % | Actions   |
|--------------------------|--|---------------------------------------|--------------|---------|--------------|-------|---|
| <input type="checkbox"/> |  <a href="#">splashy-whale</a><br>432188ijf3411      | <a href="#">free-willy:latest</a>     | Exited (255) | -       | 1 hour ago   | 0%    |    |
| <input type="checkbox"/> |  <a href="#">barnacle-bob</a><br>70398a3bdf55        | <a href="#">bikini-bottom:latest</a>  | Exited       | -       | 2 hours ago  | 0.22% |    |
| <input type="checkbox"/> |  <a href="#">yarr-matey</a><br>87324y65ff32          | <a href="#">blue-beard:latest</a>     | Exited (255) | -       | 4 hours ago  | 0%    |    |
| <input type="checkbox"/> |  <a href="#">shrimp-clamstastic</a><br>34655h3uff43  | <a href="#">cocktail-sauce:latest</a> | Exited       | -       | 10 hours ago | 0%    |    |
| <input type="checkbox"/> |  <a href="#">oswald-west</a><br>57438h2gff89         | <a href="#">cannon-beach:latest</a>   | Exited (255) | -       | 1 day ago    | 0%    |    |
| <input type="checkbox"/> |  <a href="#">endless-summer</a><br>5823aa342f5i      | <a href="#">surfs-up:latest</a>       | Exited (255) | -       | 1 day ago    | 0%    |    |

Docker engine running ...

RAM 4.78 GB CPU 0.41% Disk 50.22 GB avail. of 58.37 GB Connected to Hub v4.17.0

# Runtimes de containers - Docker Desktop

## Personal

For new developers and/or students getting started with containers.

**\$0**

- ✓ Docker Desktop ⓘ
- ✓ Unlimited public repositories
- ✓ 200 image pulls per 6 hours
- ✓ Docker Engine + Kubernetes ⓘ
- ✓ 3 Scout enabled repos
- ✓ Local Scout analysis

Get Started

## Docker Desktop license agreement

Docker Desktop is licensed under the Docker Subscription Service Agreement. When you download and install Docker Desktop, you will be asked to agree to the updated terms.

Our [Docker Subscription Service Agreement](#) ⓘ states:

- Docker Desktop is free for small businesses (fewer than 250 employees AND less than \$10 million in annual revenue), personal use, education, and non-commercial open source projects.
- Otherwise, it requires a paid subscription for professional use.
- Paid subscriptions are also required for government entities.
- The Docker Pro, Team, and Business subscriptions include commercial use of Docker Desktop.

Read the [Blog](#) ⓘ and [Docker subscription FAQs](#) ⓘ to learn how this may affect companies using Docker Desktop.



## Runtimes de containers - Rancher Desktop

File Edit View Help

General

Containers

Port Forwarding

Images

Snapshots

Troubleshooting

Diagnostics

Extensions

Epinio

Logs Explorer

Cluster Dashboard

Preferences

Rancher Desktop

Containers

Start Delete

Namespace: default Filter

| State                                      | Name                         | Image                   | Port(s)   | Uptime  |
|--|------------------------------|-------------------------|-----------|---------|
| <input type="checkbox"/> up about 22 mi... | desktop-penpot-postgres-1    | postgres:15             |           | running |
| <input type="checkbox"/> up about a min... | desktop-penpot-redis-1       | redis:7                 |           | running |
| <input type="checkbox"/> up about 2 hou... | desktop-penpot-mailcatcher-1 | sj26/mailcatcher:latest | 1080:1080 | running |
| <input type="checkbox"/> created           | sad_lovelace                 | busybox                 |           | created |
| <input type="checkbox"/> exited            | affectionate_mcnulty         | redis                   |           | exited  |

Version: 1.16.0 Network status: online Kubernetes: 1.21.14 CE: containerd



# Dockerfile

O Dockerfile é um arquivo no qual fica contido as **configurações** para a **criação** de uma nova imagem.

Já tendo o Dockerfile construído, com um `docker build` é possível **gerar as imagens** para rodar local e com o `docker push` é possível **enviar para o Registry**.

Pensando em criar uma nova imagem, temos os seguintes **comandos**: `FROM`, `RUN`, `CMD`, `ENTRYPOINT`, `COPY`, `ADD`, `WORKDIR`, `EXPOSE`, `ENV`, `ARG`, `LABEL`, `VOLUME`, `USER`, `ONBUILD`, `SHELL` e `HEALTHCHECK`

# Dockerfile - Comandos

**FROM:** define a imagem base o estágio do multistage

```
FROM maven:3.9.9-eclipse-temurin-17 as builder
```

**RUN:** Executa um comando em tempo de criação da imagem

```
RUN mvn clean install
```

**CMD e ENTRYPOINT:** Executa um comando em tempo de inicialização da imagem

```
CMD java -jar /app.jar
```

**COPY e ADD:** Copia arquivos da máquina local para a imagem em tempo de criação da imagem

```
COPY . /app/
```

```
COPY --from=builder /app/target/*.jar /app.jar
```

# Dockerfile - Comandos

**WORKDIR:** Define a localização do diretório no qual será executado os comandos posteriores.

```
WORKDIR /app/
```

**ENV:** Define as variáveis de ambiente da imagem

```
ENV APP_NAME $APP_NAME
```

**ARG:** Define os argumentos que são enviados no momento de criação da imagem no comando de build do docker

```
ARG APP_NAME
```

**LABEL:** Define labels da imagem como vendor, version e etc

```
LABEL application=$APP_NAME
```

# Dockerfile - Comandos

**EXPOSE:** define a porta que vai ser exposta pelo container

```
EXPOSE 8080
```

**VOLUME:** define o volume para persistência de dados

```
VOLUME /data
```

**USER:** define o usuário será usado para executar os comandos

```
USER appuser
```

# Dockerfile - Comandos

**ONBUILD:** Define os comandos a serem executados em outras bases Dockerfile

```
ONBUILD RUN apt-get update
```

**SHELL:** Define o shell que será executado para executar os comandos (/bin/sh, /bin/bash)

```
SHELL ["/bin/sh", "-c"]
```

**HEALTHCHECK:** Usado para validar a saúde do container

```
HEALTHCHECK --interval=30s --timeout=5s --retries=3 CMD  
curl -f http://localhost:8080/actuator/health || exit 1
```

# Dockerfile - Multistage

Um Dockerfile pode ser configurado com multistage quando se pretende fazer múltiplas operações sem deixar vestígios e sujeiras.

## Por exemplo:

Quando temos que **gerar um JAR** de um programa e depois pegar essa imagem e **disponibilizar a sua execução**.

Para gerar um JAR, precisamos de um **Maven** além de ter que baixar **todas as dependências** para gerar um único jar.

Sendo assim, criamos primeiro stage para **gerar o JAR** que conterà o **Maven** e **todas as dependências** baixadas. E, no segundo stage, **manteríamos apenas o JAR gerado** com as configurações para essa aplicação **rodar**.

As imagens de cada stage podem ser **diferentes**, por exemplo, podemos usar a imagem `maven:3.9.9` no primeiro stage e o `openjdk:24-slim`

# Comando Docker - Build

Comando responsável por gerar uma imagem docker que poderá ser usada posteriormente pelos containers:

Exemplo:

```
docker build -t docker-app:1.0.0 --build-arg  
APP_NAME=docker-app-0.0.1-SNAPSHOT .
```

# Comando Docker - Pull

Comando responsável por baixar uma imagem docker para a máquina local.

Exemplo:

```
docker pull ubuntu:24.04
```



# Comando Docker - Images

Comando responsável por mostrar as imagens docker baixadas na máquina local.

Exemplo:

```
docker images
```

# Comando Docker - Remove Image

Comando responsável por apagar uma imagem docker existente na máquina local.

Exemplo:

```
docker rmi docker-app
```

# Comando Docker - Run

Comando responsável por rodar um container baseado em uma imagem docker.

A imagem baseada pode existir na máquina local ou não sendo que se não existir, a imagem é baixada ao rodar o comando

Exemplo:

```
docker run hello-world
```

# Comando Docker - PS

Comando responsável por mostrar os containers.

Se usar o parâmetro `-a`, será mostrado todos os containers que estejam rodando ou parados, se não usar o parâmetro `-a`, mostra apenas os containers que estão rodando.

Exemplo:

```
docker ps
```

```
docker ps -a
```

# Comando Docker - Stop

Comando responsável por parar um containers que esteja rodando.

Exemplo:

```
docker stop docker-app
```

# Comando Docker - Start

Comando responsável por iniciar um containers que já foi criado e esteja parado.

Exemplo:

```
docker start docker-app
```

# Comando Docker - Rm

Comando responsável por pagar um containers que já foi criado e esteja parado.

Exemplo:

```
docker rm docker-app
```

# Comando Docker - Prune

Comando responsável por pagar todos containers que estejam parados.

Exemplo:

```
docker container prune
```



# Comando Docker - Login

Comando responsável por fazer o login em um Registry.

Se não informar o servidor de autenticação, por padrão vai autenticar no Docker Hub

Exemplo:

```
docker login
```

```
docker login registry.example.com
```

# Comando Docker - Push

Comando responsável por jogar uma imagem docker criada localmente para um Registry.

É necessário estar logado em um Registry.

Exemplo:

```
docker push fernandoteixeira/docker-app:1.0.0
```

# Comando Docker - Volume

## Listar volumes

```
docker volume ls
```

## Apagar todos os volumes sem relação com o containers criados

```
docker volume prune
```

## Criar um volume para ser usado por volumes

```
docker volume create mongo
```

## Apagar um volume sem relação com conatiners criados

```
docker volume rm mongo
```

# Comando Docker - Network

## Listar networks

```
docker network ls
```

## Criar um network

```
docker network create my-network
```

## Remover um network

```
docker network rm my-network
```

# Docker Compose - Arquivo YML

- O YML do docker compose é usado para configurar a subida dos containers, volumes e redes para que seja possível subir um ambiente mais complexo como, por exemplo, um ambiente com aplicação e banco de dados.

```
version: "3.8"
services:
  hello-world:
    image: hello-world
    volumes:
      - hello-world-volume:/data
    networks:
      - local-network
networks:
  local-network:
    driver: bridge
volumes:
  hello-world-volume:
```

# Docker Compose - Comando

Considerando que já se tem um arquivo docker-compose.yaml criado, podemos executar as configurações e subir os container, volumes e networks configurados a partir do seguinte comando:

```
docker-compose up
```

E o comando seguinte para apagar as configurações criadas pelo comando anterior:

```
docker-compose down
```

# Case: Rodando Serviço Local

Na aplicação do CASE, temos uma API de pessoa que:

- Persiste os dados em um MongoDB;
- Criar Volume para manter persistido os dados do MongoDB
- Compila o JAR e gera uma imagem do docker-app para rodar o container apontando para o MongoDB;
- Criar uma rede local para permitir que o container do docker-app acesse o MongoDB

Obrigado