Docker

<LAB365>

SENAI

AGENDA

- Aprender a gerenciar imagens Docker e containers.
- Realizar práticas com o Docker

Componentes do Docker

- **Docker Engine:** O núcleo do Docker, responsável pela criação e execução dos containers.
- **Docker Images:** Pacotes leves e independentes que contêm tudo necessário para executar uma aplicação.
- **Docker Containers:** Instâncias em execução de imagens Docker.
- **Docker Registry:** Serviço para armazenar e distribuir imagens Docker (ex.: Docker Hub).
- **Docker Compose:** Ferramenta para definir e executar aplicativos com vários containers.

Docker e Sua Arquitetura

O Docker é composto por vários componentes que trabalham juntos para criar, executar e gerenciar containers:

- **Docker Engine:** O coração do Docker, responsável por criar e executar containers.
- **Docker Client:** A interface de linha de comando (CLI) que os usuários usam para interagir com o Docker Engine.

Docker e Sua Arquitetura

- **Docker Registry (Docker Hub):** Um serviço que armazena e distribui imagens Docker.

- **Docker Images:** Pacotes leves e independentes que contêm tudo o que é necessário para executar uma aplicação.

Docker e Sua Arquitetura

- **Docker Containers:** Instâncias em execução de imagens Docker.
- **Docker Volumes:** Mecanismo para persistência de dados fora do ciclo de vida do container.
- **Docker Networks:** Mecanismo que conecta containers a redes virtuais isoladas.

Demonstração de Comandos Básicos do Docker

Vamos explorar alguns dos principais comandos do Docker:

1. **‘docker pull’:** Baixar imagens do Docker Hub.
2. **‘docker run’:** Criar e executar containers.
3. **‘docker ps’:** Listar containers em execução.
4. **‘docker stop’** e **‘docker rm’:** Parar e remover containers.
5. **‘docker images’:** Listar imagens locais.
6. **‘docker rmi’:** Remover imagens locais.
7. **‘docker exec’:** Executar comandos dentro de um container em execução.

Docker CLI - Subir uma instância do postgree com docker

```
docker run -d --name meu-postgres -e POSTGRES_PASSWORD=P@ssword -p
5432:5432 postgres
```

Este comando cria e executa um contêiner chamado "meu-postgres"

a partir da imagem oficial do PostgreSQL.

-d executa o contêiner em segundo plano

-e define a variável de ambiente POSTGRES_PASSWORD como "p@ssword" para definir a senha do banco de dados

-p mapeia a porta 5432 do contêiner para a porta 5432 do host, permitindo acessar o banco de dados PostgreSQL fora do contêiner.

Referência : https://hub.docker.com/_/postgres

INTERVALO DE AULA

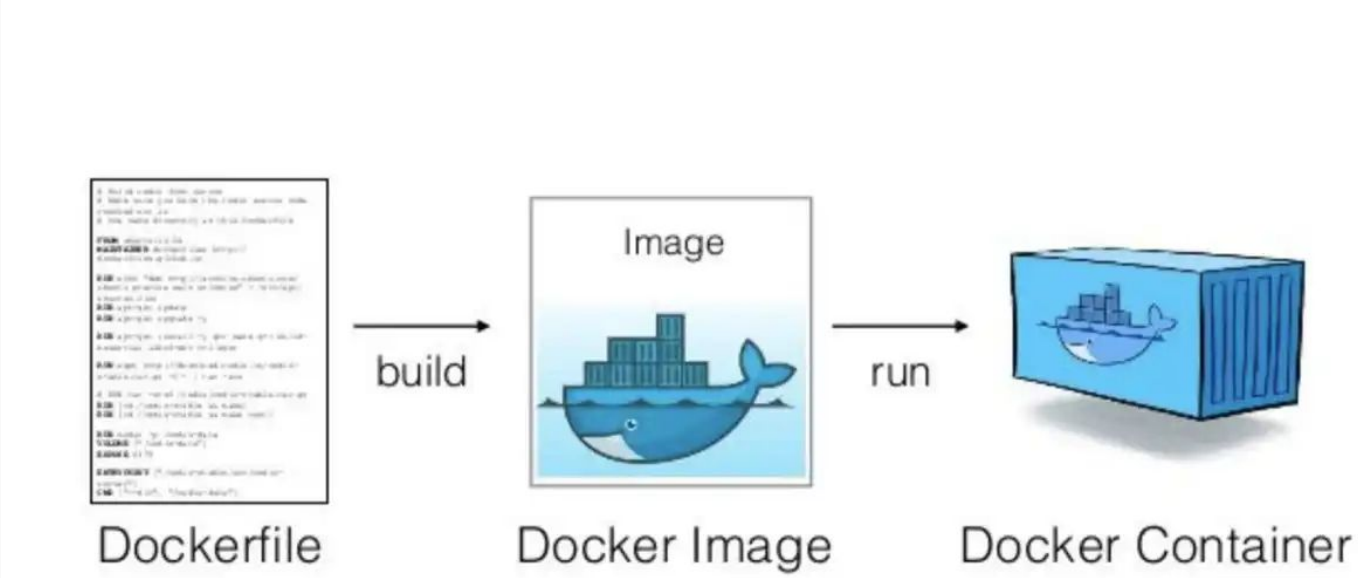
DEV!

Finalizamos o nosso primeiro período de hoje. Que tal descansar um pouco?!

Nos vemos em 20 minutos.

Início: 20:22
Retorno: 20:42

Dockerfile



Dockerfile

Dockerfile é um arquivo de texto que contém uma série de instruções que o Docker usa para criar uma imagem Docker. Essas instruções especificam os passos necessários para configurar e construir o ambiente dentro de um container Docker.

O Dockerfile é usado como entrada para o comando docker build, que cria a imagem Docker com base nas instruções fornecidas.

Dockerfile

FROM: Especifica a imagem base a ser usada. Todas as instruções subsequentes são executadas em cima dessa imagem. Por exemplo:

```
# Usar a imagem oficial do Node.js como a imagem base
FROM node:alpine
```

Dockerfile

WORKDIR: Define o diretório de trabalho dentro do contêiner onde os comandos subsequentes serão executados. Por exemplo:

```
# Definir o diretório de trabalho dentro do container
WORKDIR /app
```

Dockerfile

COPY ou ADD: Copia arquivos do sistema de arquivos do host para dentro do contêiner. Por exemplo:

```
# Copiar os arquivos 'package.json' e 'package-lock.json' (ou 'yarn.lock')
para o diretório de trabalho
COPY package*.json ./
```

Dockerfile

RUN: Executa comandos no contêiner durante o processo de construção da imagem. Por exemplo:

```
# Instalar as dependências do projeto
RUN npm install
```

Dockerfile

EXPOSE: Expõe uma porta para quando a aplicação for iniciada.

```
# Expor a porta que o servidor de desenvolvimento utiliza
EXPOSE 3000
```

Dockerfile

CMD: Executa scripts, podendo ser os scripts presentes no package.json.

```
# Comando para iniciar a aplicação usando o servidor de desenvolvimento
CMD ["npm", "run", "dev"]
```

Dockerfile

```dockerfile
# Usar a imagem oficial do Node.js como a imagem base
FROM node:alpine

# Definir o diretório de trabalho dentro do container
WORKDIR /app

# Copiar os arquivos 'package.json' e 'package-lock.json' para o diretório de trabalho
COPY package*.json ./

# Instalar as dependências do projeto
RUN npm install

# Copiar os arquivos restantes do projeto para o diretório de trabalho
COPY . .

# Expor a porta que o servidor de desenvolvimento utiliza
EXPOSE 3000

# Comando para iniciar a aplicação usando o servidor de desenvolvimento
CMD ["npm", "run", "dev"]
```

Dockerfile

Comando para buildar a imagem:

```
docker build . -t nome_imagem
```

Comando para criar o container a partir da nova imagem:

```
docker run -d -p 3000:3000 nome_imagem
```

Lembre-se de expor a porta e o host no vite : `"dev": "vite --host 0.0.0.0 --port 3000"`

Dockerfile

Comando para criar uma imagem com outro nome

```
docker tag  nome_imagem_jaExistente novoNomeImagem
```

Comando subir imagens no docker hub

```
docker push seu_user/Imagem
```

lembre-se que a imagem já precisa existir com o nome da sua conta/imagem ex: vitorlassen/timer

AVALIAÇÃO DOCENTE

O que você está achando das minhas aulas neste conteúdo?

Clique aqui ou escaneie o QRCode ao lado para avaliar minha aula.

Sinta-se à vontade para fornecer uma avaliação sempre que achar necessário.

OBRIGADO!

<LAB365>