FACI wyden

Containers com Docker para Aplicações Web

Profº Msc. Jeffson Celeiro Sousa
Doutorando em Ciência da Computação - UFPa

Belém, 29 de Janeiro de 2026

Contato

FACI wyden

Ementa

- **Introdução aos containers e ao Docker.**
- **Conceitos fundamentais de virtualização leve, imagens e containers.**
- **Criação de imagens Docker para aplicações web.**
- **Execução e gerenciamento de containers.**
- Persistência de dados com volumes.
- Comunicação entre containers.
- Orquestração básica com Docker Compose.
- Boas práticas para desenvolvimento e preparação de aplicações web em ambiente conteinerizado.

FACI wyden

Objetivos de Aprendizagem

Ao final do curso, o estudante será capaz de:

- Compreender o conceito de containers e sua aplicação no desenvolvimento web
- Criar e executar containers Docker para aplicações web
- Construir imagens Docker utilizando Dockerfile
- Persistir dados e configurar aplicações com variáveis de ambiente
- Orquestrar aplicações multi-serviço com Docker Compose
- Preparar um ambiente padronizado e reprodutível de desenvolvimento.

RESUMO – Primeiro Contato com Docker

Passo 1 – Instalar o Docker

- Windows: https://www.docker.com/get-started/
- Linux:
  - *sudo apt-get install curl*
  - *curl -fsSL https://get.docker.com | sudo bash*
  - Para executar o Docker sem utilizar o sudo, criaremos um grupo de usuário docker e adicionaremos o usuário atual nele:
    - *sudo groupadd docker*
    - *sudo usermod -aG docker $USER*
  - Atualize as mudanças realizadas no grupo:
    - *newgrp docker*

Passo 2 – Verificar Docker

- *docker --version*

Passo 3 – Primeiro Container

- *docker run hello-world*

FACI wyden

RESUMO – Primeiro Contato com Docker

Criando Nossa Primeira Imagem

Estrutura

- *meu-site/*
- *├── index.html*
- *└── Dockerfile*

Dockerfile

- *FROM nginx:alpine*
- *COPY . /usr/share/nginx/html*

RESUMO – Primeiro Contato com Docker

Criando Nossa Primeira Imagem

Dockerfile

- *FROM nginx:alpine*
  - *FROM → imagem base*
- *COPY . /usr/share/nginx/html*
  - *COPY → arquivos do projeto*

Build da imagem

- *docker build -t site-docker .*

Executar

- *docker run -p 8081:80 site-docker*

FACI wyden

Dockerizando uma API Backend

Agora vamos rodar uma API web dentro de um container.

Dockerfile da API

- *FROM node:18*
- *WORKDIR /app*
- *COPY package*.json ./*
- *RUN npm install*
- *COPY . .*
- *EXPOSE 3000*
- *CMD ["node", "index.js"]*

WORKDIR *organiza o projeto*
EXPOSE *documenta a porta*
RUN *executa uma linha de comando no container*
CMD *inicia a aplicação*

Dockerizando uma API Backend

Mas o que é o package.json ?

O package.json é um arquivo de configuração utilizado para estipular e configurar dependências do seu projeto (quais outros pacotes ele vai precisar para ser executado) e scripts automatizados. Através dele conseguimos deixar claro uma "receita" para executar um projeto.

https://www.luiztools.com.br/post/o-guia-completo-do-package-json-do-node-js/
https://gabrieluizramos.com.br/entendendo-o-package-json

Dockerizando uma API Backend

Criando um package.json

Existem algumas formas de criar um arquivo package.json. A maneira mais simples é executar ***npm init*** via terminal na pasta do seu projeto. Algumas perguntas aparecerão para você responder (como nome, descrição, versão) e após confirmar tudo isso, um arquivo básico será gerado para você.

OBS:. Para executar usando o comando você precisa de uma versão do Node + npm.

Docker para Aplicações Web Reais

Problema da Persistência

- Containers não guardam dados
- Ao recriar, tudo é perdido
- Bancos de dados precisam persistir

Docker para Aplicações Web Reais

Volumes no Docker

- Armazenam dados fora do container
- Sobrevivem a reinicializações
- Essenciais para bancos e uploads

Dockerizando uma API Backend

Persistência com Volumes

Criar volume

- *docker volume create pgdata*

Listar volume

- *docker volume ls*

Inspecionar volume

- *docker volume inspect pgdata*

Dockerizando uma API Backend

Rodar Database com volume

Criar Banco de Dados

- *docker run --name pg-test *
- *-e POSTGRES_PASSWORD=postgres *
- *-e POSTGRES_DB=appdb *
- *-p 5432:5432 *
- *-v pgdata:/var/lib/postgresql/data *
- *-d postgres:16*

Dockerizando uma API Backend

Rodar Database com volume

Criar Banco de Dados

- *docker run --name pg-test *
- *-e POSTGRES_PASSWORD=postgres *
- *-e POSTGRES_DB=appdb *
- *-p 5432:5432 *
- *-v pgdata:/var/lib/postgresql/data *
- *-d postgres:16*

Verificar container

- *docker ps*
- *docker logs pg-test --tail 20*

Dockerizando uma API Backend

Testar persistência

Criar tabela e inserir dados no banco de dados (*no container*)

- *docker exec -it pg-test psql -U postgres -d appdb*

Rode

- *CREATE TABLE notes (*
- * id SERIAL PRIMARY KEY,*
- * title TEXT NOT NULL*
- *);*
-
- *INSERT INTO notes (title) VALUES ('primeira nota');*
- *SELECT * FROM notes;*

Dockerizando uma API Backend

Testar persistência

Criar tabela e inserir dados no banco de dados (*no container*)

- *docker exec -it pg-test psql -U postgres -d appdb*

Saia do psql (Banco de Dados)

- *\q*

Dockerizando uma API Backend

Testar persistência

Remover container e recriar

- *docker stop pg-test*
- *docker rm pg-test*

Recrie apontando para o mesmo volume

- *docker run --name pg-test *
- *-e POSTGRES_PASSWORD=postgres *
- *-e POSTGRES_DB=appdb *
- *-p 5432:5432 *
- *-v pgdata:/var/lib/postgresql/data *
- *-d postgres:16*

Dockerizando uma API Backend

Testar persistência

Consulte novamente

- *docker exec -it pg-test psql -U postgres -d appdb -c "SELECT * FROM notes;"*

Docker para Aplicações Web Reais

Comunicação entre Containers

- Containers não usam *localhost*
- Comunicação via rede Docker
- Serviços se comunicam pelo nome

Em Docker, serviços se chamam pelo nome do serviço.

Dockerizando uma API Backend

Criar uma rede e colocar os containers nela

Criar rede

- *docker network create app-net*
- *docker network ls*

Remova e suba o container na rede criada

- *docker stop pg-test && docker rm pg-test*
- *docker run --name db *
- *--network app-net *
- *-e POSTGRES_PASSWORD=postgres *
- *-e POSTGRES_DB=appdb *
- *-v pgdata:/var/lib/postgresql/data *
- *-d postgres:16*

FACI wyden

Dockerizando uma API Backend

API Node/Express Local

Criar a API

- *mkdir api && cd api*
- *npm init -y*
- *npm i express pg*

Crie o index.js

- *github da aula!*

Teste sem docker

- *node index.js // Abrir no Chrome GET http://localhost:3000/health*

FACI wyden

Dockerizando uma API Backend

Build da API com Dockerfile

- *docker build -t api-app .*

Rodar a API em container na mesma rede do DB

- *docker run --name api *
- *--network app-net *
- *-p 3000:3000 *
- *-e DB_HOST=db *
- *-e DB_PORT=5432 *
- *-e DB_USER=postgres *
- *-e DB_PASS=postgres *
- *-e DB_NAME=appdb *
- *-d api-app*

Dockerizando uma API Backend

Testar escrita e persistência via API

Criar nota (via curl)

- *curl -X POST http://localhost:3000/notes *
- *-H "Content-Type: application/json" *
- *-d '{"title":"nota via API"}'*

Listar notas

- *curl http://localhost:3000/notes*

Derrube o container

- *docker rm -f api*

Dockerizando uma API Backend

Testar escrita e persistência via API
Crie novamente o container

- *docker run --name api *
- *--network app-net *
- *-p 3000:3000 *
- *-e DB_HOST=db *
- *-e DB_PORT=5432 *
- *-e DB_USER=postgres *
- *-e DB_PASS=postgres *
- *-e DB_NAME=appdb *
- *-d api-app*

Listar novamente

- *curl http://localhost:3000/notes*

Docker para Aplicações Web Reais

Variáveis de Ambiente

- Configurações fora do código
- Senhas, portas, URLs
- Evitam hardcode

Código é código. Configuração é configuração.

Docker para Aplicações Web Reais

O Que é Docker Compose ?

- Orquestra múltiplos containers
- Um arquivo controla tudo
- Subir aplicação inteira com um comando

Docker Compose e kubernetes é o que mais se usa no dia a dia.

Docker para Aplicações Web Reais

docker-compose.yml

- Define serviços
- Define portas
- Define volumes
- Define redes

Introdução ao Docker Compose

Problema sem Compose:

- Muitos comandos
- Difícil repetir
- Erros frequentes

Dockerizando uma API Backend

DOCKER COMPOSE

Criar .env.example e .env

- *DB_HOST=db*
- *DB_PORT=5432*
- *DB_USER=postgres*
- *DB_PASS=postgres*
- *DB_NAME=appdb*
- *POSTGRES_PASSWORD=postgres*
- *POSTGRES_DB=appdb*

Copie para .env:

- *cp .env.example .env*

Dockerizando uma API Backend

DOCKER COMPOSE

Criar docker-compose.yml (dev)

- *github da aula!*

Subir

- *docker compose up -d --build*

Verificar

- *docker compose ps*
- *docker compose logs api --tail 30*

Dockerizando uma API Backend

DOCKER COMPOSE

Atenção!!!

- Quando o docker compose sobe, ele cria um **NOVO** volume de dados padrão.
- No ***docker run*** anterior, usamos um volume chamado pgdata *(ex.: -v pgdata:/var/lib/postgresql/data)*
  - No docker-compose, mesmo que você escreva pgdata: o Compose pode criar um volume com prefixo do projeto, tipo:
    - api_pgdata (nome varia conforme a pasta/projeto)
- Resultado: você tem 2 volumes diferentes → um tem os dados, o outro está vazio.

Dockerizando uma API Backend

DOCKER COMPOSE

Solução:

- *volumes:*
- *pgdata:*
- *external: true*
- *name: pgdata*

Agora o ***Compose*** vai usar o volume pgdata “correto”, sem criar outro.

Dockerizando uma API Backend [PROD]

DOCKER COMPOSE

Objetivo:

- ***healthcheck*** *do DB*
- ***restart policy***
- *API só começa "executar" quando DB estiver saudável (com depends_on + condition)*
- *Ideal: separar arquivo docker-compose.prod.yml*

FACI wyden

Dockerizando uma API Backend [PROD]

DOCKER COMPOSE
Adicione no service do db:

- *restart: unless-stopped*
- *healthcheck:*
- *test: ["CMD-SHELL", "pg_isready -U postgres -d ${POSTGRES_DB}"]*
- *interval: 5s*
- *timeout: 3s*
- *retries: 10*

Adicione no service da api:

- *restart: unless-stopped*
- *depends_on:*
- *db:*
- *condition: service_healthy*

Dockerizando uma API Backend [PROD]

DOCKER COMPOSE

Subir containers

- *docker compose -f docker-compose.prod.yaml --env-file .env up -d --build*

Saúde

- *docker ps*
- *docker inspect --format='{{json .State.Health}}' $(docker ps -qf "name=db")*

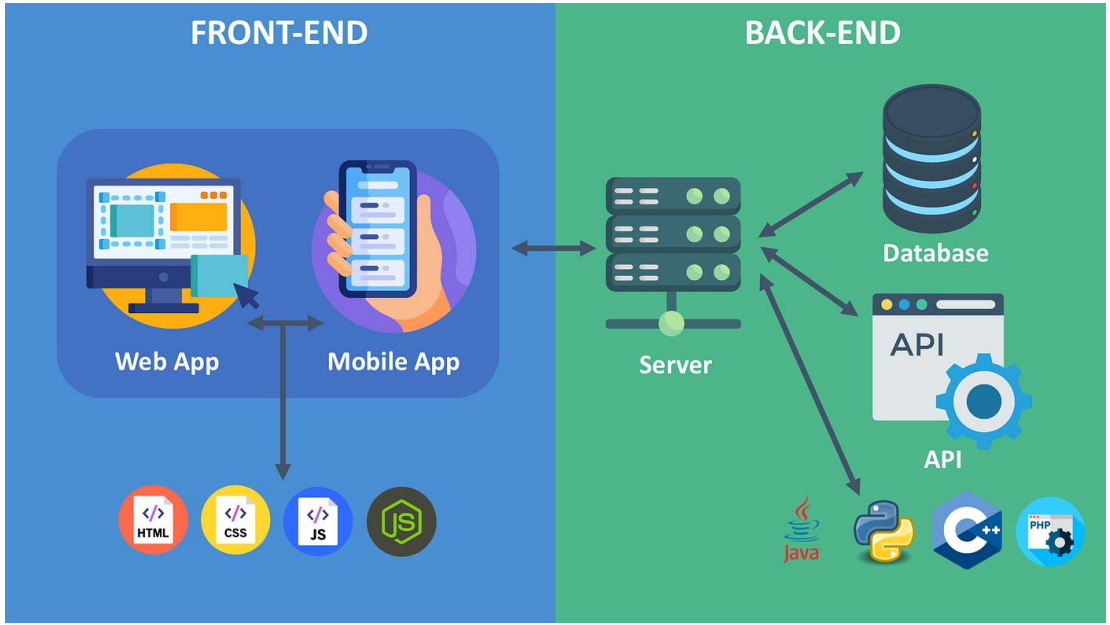
FACI wyden

Docker para Aplicações Web Reais

Arquitetura Típica

- Frontend
- Backend (API)
- Banco de dados

Tudo em containers!



APLICAÇÃO EM PRODUÇÃO (SIMULADA)

Componentes:

- Frontend (opcional)
- Backend (API)
- Banco de dados
- Volumes
- Variáveis de ambiente

Resumo

Agora você é capaz de:

- Rodar aplicações web completas
- Trabalhar como em ambiente profissional
- Preparar base para produção
- Entender DevOps na prática

FACI

wyden