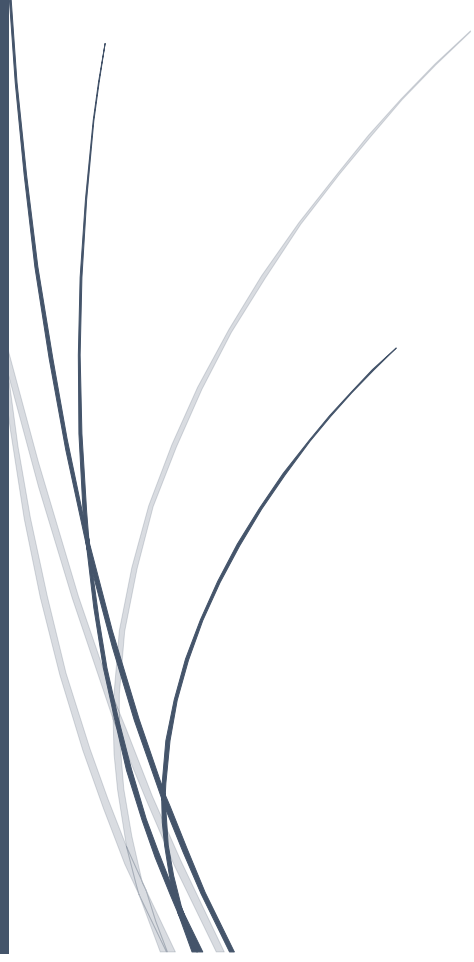
Engenharia de Software

11/10/2025

TP2

Microsserviços e DevOps com Spring Boot e Spring Cloud

Professor(a): Flávio da Silva Neves



Samuel Hermany
INSTITUTO INFNET

LINK GITHUB

https://github.com/faculdade-infnet/V-2-Microsservicos-e-Spring-Cloud/tree/main/TP2

Parte 1 - MÁQUINAS VIRTUAIS X CONTAINER DOCKER

1. Descreva as principais diferenças entre a máquina virtual e um container docker.

Máquina Virtual (VM):

- É Baseada em hypervisor, emulando a máquina física completa com definições de quantidade de memória. núcleos de processador, armazenamento em disco e outro recurso.
- São mais pesadas, pois precisam inicializar um sistema operacional inteiro.
- Possuem isolamento total, com recursos dedicados (CPU, memória, disco).
- O tempo de inicialização é mais lento.
- Usadas quando é necessário ambiente completamente isolado ou sistemas operacionais diferentes no mesmo host.

Container Docker:

- São um pacote com um sistema operacional virtual (SO), de forma isolada e independente e podem ser executados em qualquer local.
- Compartilha o mesmo sistema operacional do host, sendo assim muito mais leve e rápido, pois não há necessidade de carregar um sistema operacional completo.
- O isolamento é lógico, não físico, o que reduz consumo de recursos.
- São iniciados e interrompidos rapidamente, facilitando escalabilidade e implantação contínua, sendo ideal para microsserviços e serviços em nuvem.
- Ambientes previsíveis e isolados de outros aplicativos.

2. Descreva os principais benefícios de usar Docker.

- **Portabilidade**: o container Docker leva junto todas as dependências da aplicação, garantindo que ela funcione da mesma forma em qualquer ambiente.
- **Leveza e desempenho**: containers Docker compartilham o mesmo sistema operacional do host, sendo muito mais leves e rápidos que máquinas virtuais.
- **Inicialização rápida**: um container Docker inicia em segundos, facilitando a escalabilidade e o deploy contínuo.
- **Isolamento**: cada container Docker executa de forma independente, evitando conflitos entre aplicações e dependências.
- **Escalabilidade**: permite criar, replicar e remover containers Docker facilmente, integrando-se bem com orquestradores como Kubernetes ou Docker Swarm.
- **Reprodutibilidade**: o uso de imagens Docker garante que o ambiente seja idêntico em todas as máquinas, reduzindo problemas de “funciona na minha máquina”.
- **Automação e integração contínua**: simplifica pipelines CI/CD, permitindo construir, testar e implantar aplicações automaticamente.

Parte 2 - CONTÊINERES E A ARQUITETURA DE MICROSSERVIÇOS

3. Descreva a arquitetura do Docker, explique como funciona seus componentes.

A arquitetura do Docker segue o modelo cliente-servidor, composta por três principais partes:

1. Docker Client (Cliente)
- É a interface de comunicação entre o usuário e o Docker.
- É o que você usa quando executa comandos como `docker build`, `docker run`, `docker pull` etc.

2. Docker Daemon (Servidor)
- Também chamado de dockerd, é o coração do Docker.
- Ele executa e gerencia containers, imagens, volumes e redes.
- Recebe os comandos do cliente e realiza as ações no sistema operacional.

3. Docker Registry (Repositório de Imagens)
- É o local onde as imagens Docker são armazenadas e distribuídas.
- O Docker Hub é o registro público mais conhecido, mas empresas também podem ter registries privados.

Seu funcionamento se da da seguinte forma:
O Docker Client envia comandos, o Docker Daemon executa e gerencia containers, e o Docker Registry armazena as imagens. Juntos, esses componentes tornam o Docker uma plataforma leve, portátil e eficiente para criar e executar aplicações isoladas

4. O que é um Dockerfile e como podemos usar ele?

É como uma receita que o Docker segue para gerar uma imagem pronta para rodar sua aplicação.

Como usar ele:

1. Crie o arquivo Dockerfile na raiz do seu projeto.
2. No terminal, execute:
```
docker build -t minha-aplicacao .
```
3. Após a imagem ser criada, rode o container:
```
docker run -p 3000:3000 minha-aplicacao
```
4. Sua aplicação estará rodando dentro de um container.

5. Cite exemplos de comandos úteis no dia a dia de um desenvolvedor que pretende usar o Dockerfile em seus projetos com Spring Boot.

```
# Cria uma imagem chamada minha-api-springboot com base nas instruções do Dockerfile na pasta atual (.)
docker build -t minha-api

# Executa o container em segundo plano (-d), mapeando a porta 8080 do host para o container, e dá o nome api-container.
docker run -d -p 8080:8080 --name api-container minha-api

# Parar container
docker stop api-container

# Remover container
docker rm api-container

# Remover imagens antigas (para liberar espaço)
docker rmi minha-api

# Mostra containers em execução
docker ps

# Mostra todos os containers (ativos e parados)
docker ps -a

# Lista todas as imagens disponíveis
docker images

# Gera uma nova versão da imagem com as mudanças realizadas no projeto (por exemplo, v2).
docker build -t minha-api-springboot:v2 .
```

6. O que é o Docker Compose e como podemos usar ele

O Docker Compose é uma ferramenta que permite definir e gerenciar múltiplos containers Docker como um único sistema, usando um arquivo de configuração YAML (docker-compose.yml).

Utilização:

1. Você cria um arquivo chamado `docker-compose.yml`, onde define:
   - As imagens a serem usadas ou construídas.
   - As portas expostas.
   - As variáveis de ambiente.
   - As dependências entre serviços.
2. O Compose lê esse arquivo e cria todos os containers automaticamente, configurando a rede e a comunicação entre eles.

7. Cite exemplos de como usar o Docker Compose.

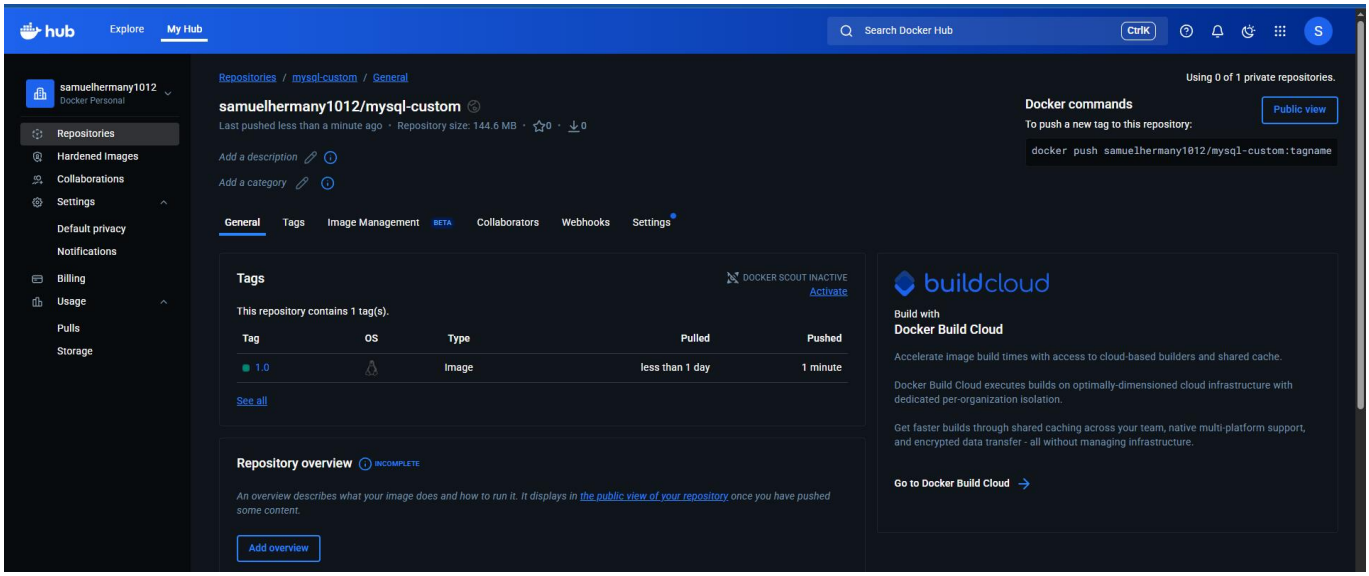
```
# O serviço app (Spring Boot) depende do serviço db (MySQL).
# Ambos são executados em rede interna, o que permite que a aplicação acesse o banco usando o nome do serviço (db).
version: '3.8'

services:
  app:
    build: .
    container_name: minha-api
    ports:
      - "8080:8080"
    depends_on:
      - db
    environment:
      - SPRING_DATASOURCE_URL=jdbc:mysql://db:3306/meubanco
      - SPRING_DATASOURCE_USERNAME=root
      - SPRING_DATASOURCE_PASSWORD=123456

  db:
    image: mysql:8
    container_name: mysql-db
    environment:
      - MYSQL_ROOT_PASSWORD=123456
      - MYSQL_DATABASE=meubanco
    ports:
      - "3306:3306"
```

Parte 3 - MIGRAÇÃO DE DOCKER PARA O KUBERNETES

1. Você deverá baixar e configurar o MySQL no Docker usando o Docker CLI e publicar no Docker Hub (Não usar a imagem do MySQL pronta).

Dentro da pasta do github tem uma pasta “Parte 3” com um arquivo “REDME.md” e dentro dele esta o passo a passo que eue utilizei para publicar a imagem no **DockerHhub**.

https://hub.docker.com/repository/docker/samuelhermany1012/mysql-custom



2. Faça a migração de um Docker para o Kubernetes ou Minikube.

Parte 4 - Implantação de aplicações Spring Boot no Kubernetes ou Minikube

1. Desenvolva uma aplicação usando o Spring Boot com pelo menos 5 endpoints e implante ela em um container Docker. Para fazer esse processo você pode usar o Dockerfile.