Projeto Final Devops Engineer

Alisson Melo

Este projeto foi desenvolvido como trabalho final do curso de DevOps. A ideia principal foi criar uma arquitetura simples de microserviços em Python com pipeline CI/CD, Docker, testes automatizados e monitorização distribuída.

Durante o desenvolvimento foram configurados containers Docker, automação com GitHub Actions e integração com Jaeger para observação dos pedidos entre serviços.

Objetivo da Aplicação:

Este projeto simula uma aplicação web baseada em microserviços desenvolvida em Python. O objetivo é demonstrar um ambiente completo de entrega contínua (CI/CD), onde alterações ao código são automaticamente testadas, construídas e disponibilizadas em ambientes DEV, STG e PRD.

A aplicação é composta por dois microserviços principais:

- Gateway Service – responsável por receber pedidos HTTP e comunicar com outros serviços.
- Data Service – fornece dados através de uma API REST simples.

Para observabilidade, foi integrado o Jaeger com OpenTelemetry, permitindo monitorizar chamadas entre serviços e analisar desempenho.

High Level Design (HLD):

O objetivo desta solução foi criar um ambiente simples de entrega contínua para uma aplicação web baseada em microserviços em Python.
A aplicação simula um sistema escalável onde alterações ao código podem ser integradas, testadas e colocadas em produção de forma automatizada.

A solução foi desenhada com os seguintes princípios:

• Automação completa do ciclo CI/CD.
• Isolamento dos serviços com Docker.
• Deploy automático em ambientes DEV, STG e PRD.
• Monitorização distribuída com OpenTelemetry e Jaeger.
• Testes automáticos antes de qualquer deploy.

A aplicação é composta por dois microserviços principais:

- Gateway Service: recebe pedidos HTTP e comunica com outros serviços.
- Data Service: disponibiliza dados através de uma API REST.

Toda a infraestrutura é gerida por Docker Compose e a pipeline CI/CD é executada através de GitHub Actions.

Arquitetura:

A aplicação está dividida em microserviços Python com Flask, todos executados em containers Docker e ligados pela mesma rede Docker Compose.

A comunicação entre serviços ocorre via HTTP REST com resposta em JSON.

Principais componentes:

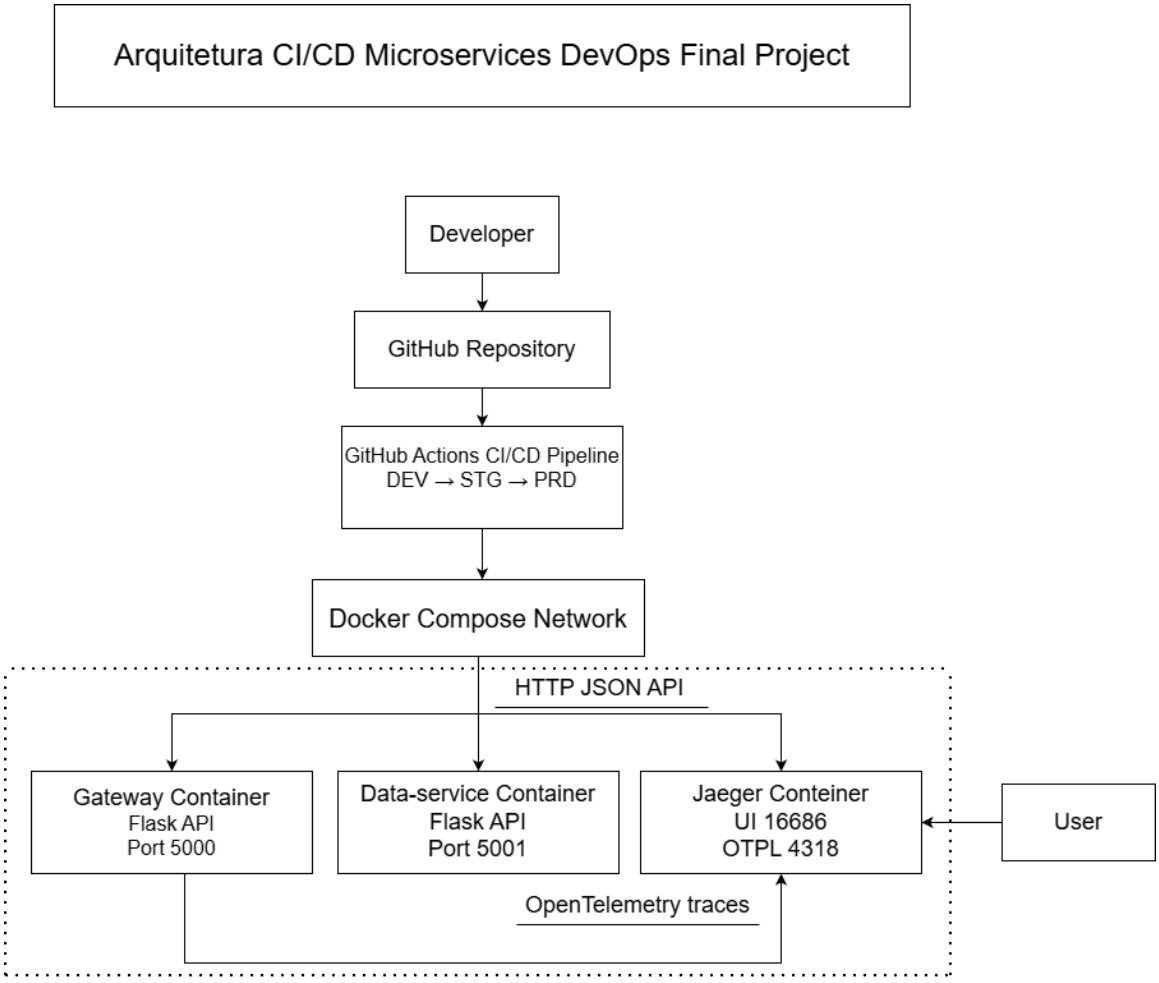
Gateway Container – recebe os pedidos HTTP e encaminha para o data-service.

Data-service Container – devolve dados JSON através de API REST.

Jaeger Container – monitorização e tracing distribuído com OpenTelemetry.

Pipeline CI/CD – automatiza build, testes e deploy (DEV, STG e PRD).

Diagrama:



Pipeline CI/CD:

Foi configurado um workflow no GitHub Actions para automatizar todo o processo.

Etapas principais:

Checkout do código.

Build dos containers Docker.

Execução de testes com pytest.

Deploy automático nos ambientes DEV, STG e PRD.

Smoke tests após deploy.

Testes:

Foram criados testes básicos com pytest para validar as APIs. O objetivo foi garantir que os serviços respondem corretamente antes de avançar para deploy.

Os testes são executados automaticamente na pipeline antes do deploy.

Monitoramento:

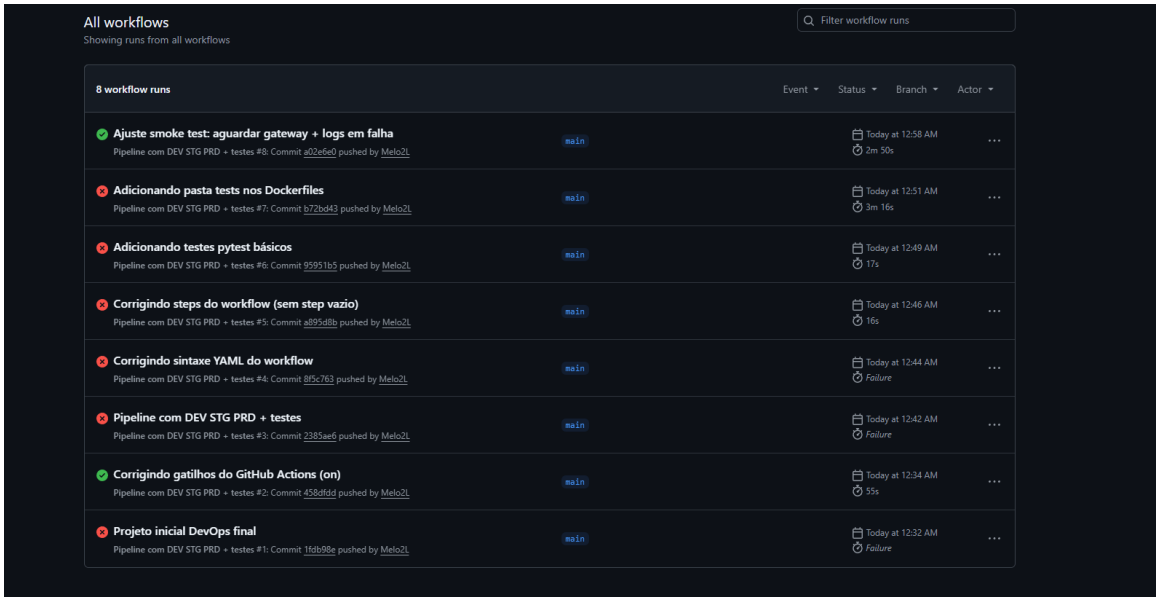
Foi utilizado OpenTelemetry para gerar traces distribuídos e Jaeger para visualização. Isto permite perceber latência, falhas e fluxo entre microserviços.

Interface Jaeger normalmente acessível em: http://localhost:16686

Erros e Ajustes Durante o Projeto:

Durante o desenvolvimento surgiram alguns erros comuns, principalmente relacionados com configuração de CI/CD, permissões do GitHub, sintaxe YAML e sincronização dos containers Docker.

A imagem abaixo mostra alguns desses erros:



Limpeza e Destruição da Infraestrutura:

Após a validação final do projeto, foi realizada a limpeza completa da infraestrutura Docker para evitar consumo desnecessário de recursos.

Foram utilizados os seguintes comandos:

docker compose down <-- Para os Containers.

docker compose down -v <-- Remove dados persistentes.

docker system prune –af <-- Remove Containers parados, imagens não usadas, cache e networks.

Conclusão:

No geral, o projeto permitiu consolidar conceitos importantes de DevOps como automação, containerização, integração contínua e observabilidade.

Fontes:

Documentação oficial Docker – https://docs.docker.com
Documentação GitHub Actions – https://docs.github.com/actions
Documentação OpenTelemetry – https://opentelemetry.io/docs
Documentação Jaeger – https://www.jaegertracing.io/docs

Stack Overflow – apoio na resolução de erros e dúvidas técnicas.
ChatGPT – apoio na estrutura do projeto, debugging.
DeepSeek e Reddit – apoio complementar em dúvidas técnicas e validação de soluções.