

**UniFECAF**

**Analise e Desenvolvimento de Sistemas**

**Projeto Cloud DevOps**

**Lucas Ribeiro Soares**

**RA: 105437**

**Implementação DevOps com Docker, Kubernetes e CI/CD**

## 1. Visão Geral da Arquitetura

O projeto “Pedidos Veloz” foi desenvolvido utilizando arquitetura de microserviços. Ele é composto por:

**Gateway**: ponto único de entrada da aplicação

**Orders**: serviço de pedidos

**Stock**: serviço de estoque

**PostgreSQL**: banco de dados

Os serviços se comunicam internamente por rede Docker/Kubernetes. Apenas o Gateway é exposto externamente.

## 2. Ambiente Local com Docker Compose

O ambiente local é padronizado através do Docker Compose. Com um único comando:

```
docker compose up -d
```

Todos os serviços sobem automaticamente, incluindo rede, volumes e variáveis de ambiente.

O arquivo docker-compose.yaml define:

Serviços

Redes internas

Volume persistente do banco

Variáveis de ambiente

O README documenta claramente os comandos de execução.

## 3. Conteinerização e Versionamento

Cada serviço possui seu próprio Dockerfile, utilizando:

Imagens leves (node:18-alpine)

Usuário não-root

Instalação apenas de dependências de produção

Exposição de portas necessárias

As imagens são versionadas com tag v1, permitindo controle de versão e reproduzibilidade.

## 4. Kubernetes

Na pasta /k8s estão os manifests:

- Deployments (gateway, orders, stock, postgres)
- Services (acesso interno e externo)
- ConfigMap (configurações)
- Secret (credenciais)
- HPA (autoescala)

Readiness e liveness probes garantem que o Kubernetes apenas direcione tráfego para pods saudáveis.

A estratégia de segurança mantém serviços internos não expostos externamente.

## 5. CI/CD

O pipeline GitHub Actions automatiza:

- Checkout do código
- Build das imagens Docker
- Validações básicas (lint e scan simulados)
- Publicação simulada das imagens

O pipeline é acionado a cada push na branch main, garantindo integração contínua.

Secrets do GitHub são utilizados para futuras integrações com registry real

## 6. Estratégia de Deploy

O Kubernetes utiliza **Rolling Update**, permitindo atualização dos serviços sem indisponibilidade. Novas versões dos pods são criadas enquanto versões antigas são removidas gradualmente.

Essa estratégia reduz riscos de deploy em produção.

## 7. Escalabilidade

O Kubernetes utiliza **Rolling Update**, permitindo atualização dos serviços sem indisponibilidade. Novas versões dos pods são criadas enquanto versões antigas são removidas gradualmente.

Essa estratégia reduz riscos de deploy em produção.

## 8. Observabilidade

A proposta de observabilidade inclui:

- **Prometheus** para métricas
- **Loki** para logs
- **Jaeger** para tracing distribuído

Os containers emitem logs via stdout, que podem ser coletados pelo Kubernetes

## 9. Infraestrutura como Código

A pasta /terraform contém um esqueleto representando como a infraestrutura Kubernetes poderia ser provisionada em nuvem. Essa abordagem garante reprodutibilidade da infraestrutura

## 10. Testes de funcionamento

### Docker:

```
docker compose up -d  
http://localhost:8080
```

### Kubernetes:

```
kubectl apply -f k8s/  
kubectl get pods
```

Todos os pods em estado **Running** confirmam funcionamento correto.

## Conclusão

O projeto implementa um pipeline DevOps completo, desde desenvolvimento local padronizado até simulação de produção em Kubernetes, com CI/CD, escalabilidade e observabilidade propostas.