RELATÓRIO TÉCNICO – PARTE PRÁTICA

Projeto: Plataforma Pedidos Veloz

Aluno: Yan Coloda
Curso: Análise e Desenvolvimento de Sistemas
Disciplina: DevOps e Computação em Nuvem
Ano: 2026

1. Introdução

Este relatório apresenta detalhadamente a implementação prática do projeto Plataforma Pedidos Veloz, desenvolvido com foco em práticas modernas de DevOps, arquitetura cloud-native, conteinerização, automação de pipelines e observabilidade.

O principal objetivo do projeto foi construir uma solução escalável, confiável e automatizada, capaz de atender às demandas de um e-commerce em crescimento, reduzindo falhas operacionais e riscos durante atualizações.

2. Visão Geral do Sistema

A aplicação foi estruturada segundo o modelo de microsserviços, no qual cada componente é responsável por uma funcionalidade específica.

Os principais serviços implementados foram: API Gateway, Serviço de Pedidos, Serviço de Pagamentos, Serviço de Estoque e Banco de Dados PostgreSQL.

A comunicação entre os serviços ocorre por meio de rede interna, garantindo baixo acoplamento e maior flexibilidade de manutenção.

3. Ambiente de Desenvolvimento Local

O ambiente de desenvolvimento foi padronizado utilizando Docker e Docker Compose, permitindo que todos os serviços sejam executados localmente com um único comando.

O arquivo docker-compose.yml define serviços, redes, volumes persistentes e variáveis de ambiente, garantindo reprodutibilidade.

Essa abordagem reduz inconsistências entre ambientes de desenvolvimento, teste e produção.

4. Conteinerização e Versionamento

Cada microsserviço possui um Dockerfile próprio estruturado com multi-stage build, reduzindo o tamanho das imagens finais.

Foram adotadas boas práticas como uso de imagens base mínimas, usuário não-root, remoção de dependências desnecessárias e cache de camadas.

As imagens são versionadas e armazenadas em registry remoto, facilitando rollback e rastreabilidade.

5. Orquestração com Kubernetes

O ambiente de produção utiliza Kubernetes para orquestração dos containers, garantindo alta disponibilidade e tolerância a falhas.

Foram utilizados recursos como Deployments, Services, ConfigMaps, Secrets e Ingress.

As probes de liveness e readiness monitoram a saúde dos pods, permitindo reinicialização automática em caso de falhas.

As configurações sensíveis são armazenadas em Secrets, garantindo maior segurança.

6. Integração Contínua e Entrega Contínua (CI/CD)

O pipeline de CI/CD foi implementado utilizando GitHub Actions, automatizando o processo de integração, testes, build e publicação.

Cada push no repositório dispara a execução do pipeline, garantindo validação contínua.

Os secrets utilizados no pipeline são armazenados de forma segura no repositório.

7. Monitoramento e Observabilidade

A observabilidade foi implementada com base nos três pilares: métricas, logs e traces.

O Prometheus é responsável pela coleta de métricas, enquanto o Grafana fornece dashboards interativos.

O Jaeger, integrado ao OpenTelemetry, permite o rastreamento distribuído das requisições.

Essas ferramentas facilitam o diagnóstico rápido de falhas e gargalos.

8. Estratégias de Deploy e Escalabilidade

Foi adotada a estratégia de Rolling Update, permitindo atualizações graduais sem interrupção do serviço.

O Horizontal Pod Autoscaler ajusta automaticamente o número de réplicas conforme o consumo de CPU e memória.

Essa abordagem garante elasticidade durante picos de acesso.

9. Considerações de Segurança

Foram aplicadas boas práticas de segurança em diferentes camadas da arquitetura.

As imagens Docker utilizam usuários não privilegiados e dependências mínimas.

No Kubernetes, Secrets, namespaces e políticas básicas de acesso foram utilizados para reduzir riscos.

10. Infraestrutura como Código (Terraform)

A infraestrutura foi modelada utilizando Terraform em nível de esqueleto.

Essa abordagem permite versionamento da infraestrutura, reprodutibilidade e facilidade de manutenção.

11. Resultados Obtidos

A solução desenvolvida proporcionou maior estabilidade, redução de falhas e padronização dos processos.

O tempo de deploy foi reduzido, e a rastreabilidade de erros foi ampliada por meio da observabilidade.

O ambiente passou a suportar picos de acesso com maior eficiência.

12. Desafios Enfrentados

Durante o desenvolvimento, foram enfrentados desafios relacionados à configuração do Kubernetes, integração entre serviços e instrumentação de observabilidade.

Esses desafios foram superados por meio de estudos em documentação oficial e testes contínuos.

13. Trabalhos Futuros

Como melhorias futuras, destacam-se a implementação de service mesh, políticas avançadas de segurança, canary deploy e automação completa da infraestrutura.

Também é prevista a expansão do monitoramento com alertas automatizados.

14. Conclusão

O projeto Plataforma Pedidos Veloz atendeu aos requisitos propostos, integrando conteinerização, Kubernetes, CI/CD, observabilidade e infraestrutura como código.

A solução está alinhada aos princípios cloud-native e prepara a empresa para crescimento sustentável.