DevOps API Project Report

I. Executive Summary

This project demonstrates enterprise-grade DevOps practices through a production-ready REST API. The implementation features automated CI/CD pipelines, containerized deployment, Kubernetes orchestration, comprehensive observability, and multi-layered security scanning. With 90%+ test coverage and fully automated workflows, the project exemplifies modern software engineering best practices.

II. Project Architecture

The application is a Flask-based REST API providing CRUD operations for item management. Built with Python 3.11 and deployed using Gunicorn, the application maintains a minimal footprint while delivering robust functionality through structured logging, health monitoring, and Prometheus metrics integration.

The infrastructure leverages Docker for containerization with multi-stage builds that optimize image size and security. Kubernetes orchestration provides high availability through multiple replicas, automated health checks, and load balancing. The deployment supports both local development and production environments with seamless transitions between stages.

III. CI/CD Pipeline

A fully automated 6-stage pipeline executes on every code commit, ensuring quality and security throughout the development lifecycle. The pipeline progresses from unit testing with coverage analysis, through static and dynamic security scanning, to automated Docker image builds and deployments. Each stage gates the next, preventing vulnerable or broken code from reaching production.

Security scanning occurs at four distinct layers: static code analysis identifies vulnerabilities in source code, dependency scanning detects known CVEs in packages, container scanning analyzes built images, and dynamic testing validates the running application. This defense-in-depth approach ensures comprehensive security coverage.

IV. Observability

The project implements complete observability through three integrated pillars. Prometheus collects metrics including request rates, response times, and error rates, providing quantitative performance data. Structured JSON logging with unique trace IDs enables request correlation and debugging across distributed systems. A Grafana dashboard visualizes key metrics in real-time, offering insights into application health and performance trends.

This observability stack enables proactive monitoring, rapid troubleshooting, and data-driven optimization decisions. The integration of metrics, logs, and traces provides comprehensive visibility into application behavior in production environments.

V. Key Achievements

- Automated 6-stage CI/CD pipeline with GitHub Actions
- Multi-layered security scanning (SAST, dependency, container, DAST)
- Kubernetes deployment with high availability and auto-healing
- Complete observability with Prometheus and Grafana
- 90%+ test coverage with comprehensive unit tests
- Production-ready containerization with optimized builds
- Structured logging with distributed tracing support

VI. Technology Stack

Application: Python 3.11, Flask, Gunicorn
CI/CD: GitHub Actions
Infrastructure: Docker, Kubernetes
Observability: Prometheus, Grafana
Security: Bandit, Trivy, OWASP ZAP
Testing: pytest with coverage reporting

VII. Conclusion

This project successfully demonstrates modern DevOps principles from development through production deployment. The automated pipeline ensures code quality and security, while the orchestrated infrastructure provides reliability and scalability. Comprehensive observability enables operational excellence, and the clean architecture supports maintainability and extension.

The implementation serves as a practical reference for building cloud-native applications with enterprise-grade DevOps practices, balancing automation, security, and operational excellence.

Repository: https://github.com/ghalia52/devops-api-project
Container Registry: https://hub.docker.com/r/ghaliaba1/devops-api