CI/CD Pipeline with GitHub Actions & Docker

Project Report

Project Repository Github:https://github.com/rugved02/cicd-pipeline-demo

Docker Image:rugved2005/cicd-demo-app

Abstract

This project demonstrates the implementation of a comprehensive CI/CD (Continuous Integration/Continuous Deployment) pipeline using modern DevOps tools and practices. The solution automates the entire software delivery process from code commit to production deployment, utilizing GitHub Actions for automation, Docker for containerization, and Minikube for local Kubernetes orchestration on AWS EC2. The pipeline ensures code quality through automated testing, maintains deployment consistency through containerization, and provides scalable infrastructure management through Kubernetes.

Introduction

In today's fast-paced software development environment, manual deployment processes are error-prone and time-consuming. This project addresses these challenges by implementing an automated CI/CD pipeline that streamlines the development workflow from code changes to production deployment.

The primary objective was to create a complete DevOps solution that:
- Automatically tests code changes upon commit
- Builds consistent Docker images for deployment
- Deploys applications to a Kubernetes cluster
- Provides monitoring and rollback capabilities
- Runs entirely on cloud infrastructure (AWS EC2)

This implementation serves as a foundation for modern software delivery practices, demonstrating how small teams can leverage enterprise-grade DevOps tools without significant infrastructure overhead.

Tools Used

Core Technologies

- AWS EC2: Cloud computing platform providing the infrastructure foundation
- Ubuntu 22.04 LTS: Operating system for the EC2 instance

• GitHub: Version control and source code repository
• GitHub Actions: CI/CD automation and workflow orchestration

Containerization & Orchestration

• Docker: Application containerization platform
• Docker Compose: Multi-container application orchestration
• Docker Hub: Container image registry and storage
• Minikube: Local Kubernetes cluster for development and testing
• kubectl: Kubernetes command-line interface

Development Stack

• Python 3.9: Programming language for the sample application
• Flask: Lightweight web framework for the demo application
• Git: Version control system for source code management

Security & Configuration

• SSH: Secure remote access to EC2 instance
• AWS Security Groups: Network access control and firewall rules
• GitHub Secrets: Secure storage of sensitive credentials
• Docker Hub Access Tokens: Secure container registry authentication

Conclusion

The successful implementation of this CI/CD pipeline demonstrates the power of modern DevOps practices in automating software delivery. The project achieved all primary objectives:

Key Accomplishments:
• Automated Testing: Every code commit triggers automated validation, ensuring code quality
• Consistent Deployments: Docker containerization eliminates environment-specific issues
• Scalable Infrastructure: Kubernetes provides robust orchestration and scaling capabilities
• Security Best Practices: Implemented secure credential management and network access controls

Technical Benefits:
• Reduced Deployment Time: Manual deployment process reduced from hours to minutes
• Error Reduction: Automation eliminated human errors in deployment processes
• Improved Reliability: Consistent containerized deployments across environments
• Enhanced Monitoring: Real-time visibility into application health and deployment status

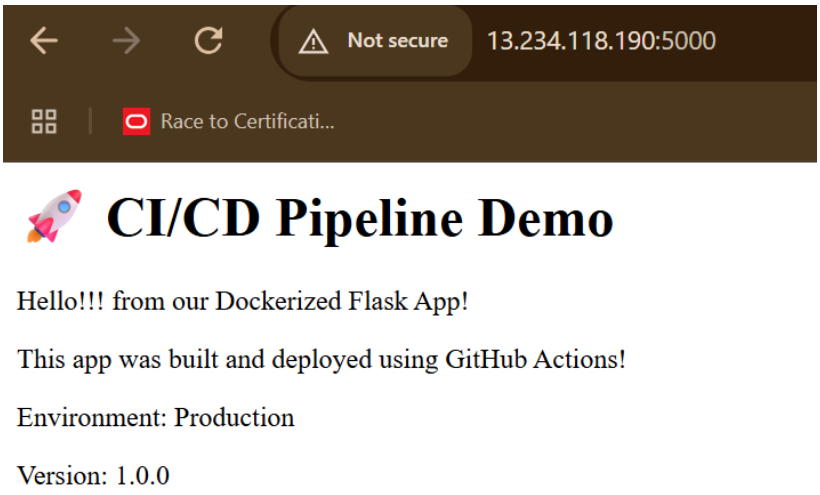
Learning Outcomes:
This project provided hands-on experience with industry-standard DevOps tools and practices, including cloud infrastructure management, containerization strategies, and Kubernetes orchestration. The implementation serves as a foundation for scaling to production environments with additional features such as monitoring, logging, and advanced security configurations.

Future Enhancements:
The pipeline can be extended with advanced features including automated rollback mechanisms,

comprehensive monitoring with Prometheus and Grafana, multi-environment deployments (staging, production), and integration with additional security scanning tools.

This CI/CD pipeline represents a significant step toward modern software delivery practices, demonstrating how individual developers and small teams can leverage enterprise-grade automation tools to improve their development workflows and deployment reliability.

Running App:



CI-CD Pipeline: