CI/CD Pipeline Implementation with Student Management System

(Final Report)

Reza Asar (23501688)

Eastern Mediterranean University

CMSE 520: Software Evolution and Maintenance

Prof. Dr. Alexander Chefranov

Dec 23, 2024

Contents

Introduction

Modern software development teams face complex challenges in delivering reliable software quickly. Teams often work on different features simultaneously, requiring robust processes to integrate and deploy code changes efficiently. This complexity has led to the adoption of CI/CD (Continuous Integration/Continuous Delivery) practices, which automate the building, testing, and deployment of applications Kim, Behr, & Spafford (2016).

Source control systems play a crucial role in this process. They allow developers to track changes, maintain different versions of code, and collaborate effectively. GitHub, one of the most widely used platforms, provides source control functionality along with additional features that support modern development workflows, including pull requests and automated pipelines Sommerville (2016).

Containerization through Docker has revolutionized application deployment. By packaging applications with their dependencies into containers, teams ensure consistent behavior across different environments. This approach eliminates the "it works on my machine" problem and simplifies application deployment and scaling.

Kubernetes extends these benefits by orchestrating containerized applications. It automates container deployment, scaling, and management across multiple servers. While complex, Kubernetes provides powerful features for maintaining application availability and managing resources efficiently.

This project demonstrates these concepts through a simple student management system, implementing modern development practices from code commit to deployment. The focus is on establishing a robust CI/CD pipeline rather than building complex application features.

Implementation Details

This section outlines the technical implementation of our development and deployment pipeline, demonstrating how code changes flow from development through to production-ready deployment. The focus is on automating the build, test, and deployment processes to ensure reliable and consistent software delivery.

CI/CD Pipeline with GitHub Actions

Our pipeline starts when code is pushed to a feature branch. Developers create pull requests to merge their changes into the main branch. The pipeline automatically builds the application, runs tests, and creates Docker images upon successful merge. This ensures code quality and maintains deployment readiness.

Docker Image Creation and Registry

Both frontend and backend applications are containerized using Docker. Images are automatically built and pushed to Docker Hub when changes are merged to main. This provides consistent, isolated environments and simplifies deployment. The frontend image uses Nginx for serving the Angular application, while the backend image contains the .NET API.

Local Kubernetes Deployment

Our Kubernetes configuration deploys three main components:

- SQL Server for data persistence
- .NET backend API
- Angular frontend

Each component runs in separate pods, with services managing communication between them. This setup demonstrates production-like deployment while remaining manageable on local development machines.

Pull Request Workflow

Code changes follow a structured review process:

1. Developers create feature branches
2. Pull requests are opened for code review
3. GitHub Actions verifies builds and tests
4. After approval, changes are merged to main
5. The pipeline automatically builds and deploys the updated application

This workflow ensures code quality and maintains a stable main branch.

Testing Documentation

Unit Tests Implementation

The testing strategy focuses on core business logic and data access layers. Unit tests cover:

- Repository pattern implementation using in-memory database
- Validation rules for student data
- Business logic in service layer

Key tested components include:

```
// Example test coverage
public class StudentRepositoryTests
{
    [Fact]
    public async Task Add_ShouldAddNewStudent()
    {
        // Test implementation details...
    }
}

public class StudentValidatorTests
{
    [Fact]
    public async Task Validate_WithValidStudent_ShouldPass()
    {
        // Validation test implementation...
    }
}
```

Integration Tests

Integration tests verify component interaction, particularly focusing on:

- Database operations through Entity Framework
- API endpoint functionality
- Data persistence and retrieval

Manual Testing Results

Manual testing validated:

- Frontend-backend integration
- Kubernetes deployment functionality
- Container orchestration
- Database persistence across deployments

Key test scenarios included:

1. CRUD operations through the UI
2. Container restart behavior
3. Service discovery in Kubernetes
4. Data persistence after pod restarts

Technical Details

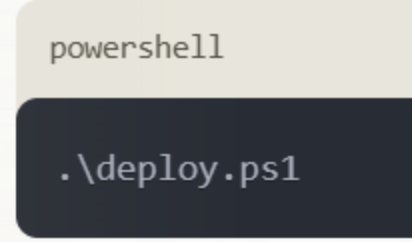
How to Run the Project

Prerequisites:

- Docker Desktop with Kubernetes enabled
- .NET 8 SDK
- Node.js and npm
- Git

Setup Steps:

1. Clone repository
2. Build and deploy using PowerShell script:



The script handles:

- Cleanup of existing deployments
- SQL Server deployment
- Backend API deployment
- Frontend deployment

Environment Setup Requirements

Development Environment:

- Visual Studio 2022 or later
- Visual Studio Code
- Docker Desktop configured with:
    - Kubernetes enabled
    - At least 4GB available memory
    - Virtualization support enabled

Known Issues and Workarounds

- SQL Server persistence requires manual volume configuration
- Frontend service occasionally requires pod restart after initial deployment
- Local Kubernetes storage limitations requiring specific configurations

Workarounds implemented:

- Specific mount paths for SQL Server data
- Container restart handling in deployment script
- Environment-specific configuration files

Future Improvements

Enhanced Monitoring and Observability

- Integration with Prometheus for metrics collection
- Grafana dashboards for visual monitoring
- Advanced logging aggregation system

Production Environment Optimizations

- High-availability database configuration
- Production-grade storage solutions
- Load balancing improvements
- Horizontal scaling configurations

DevOps Enhancements

- Automated environment provisioning
- Blue-green deployment strategy
- Disaster recovery procedures
- Backup automation

Performance Optimizations

- Caching implementation
- Query optimization
- Frontend bundle optimization

Conclusion

This CI/CD pipeline implementation project aims to demonstrate a comprehensive approach to continuous integration and deployment using industry-standard tools and best practices. While the web application functionality will be basic, the focus will be on building a robust, end-to-end automated pipeline.

The project will showcase proficiency in integrating diverse technologies, including source control (GitHub), testing (XUnit/Moq), containerization (Docker), orchestration (Kubernetes), and monitoring. By successfully implementing this pipeline, students will gain hands-on experience with practices highly relevant in the industry today.

Overall, this project provides an excellent learning platform to understand and apply modern CI/CD methodologies. It imparts skills in build automation, environment consistency, testing, deployment, and monitoring - core competencies for effectively developing and maintaining software in real-world scenarios. The exposure to challenges and problem-solving approaches during the project will prepare students well for careers in software engineering and DevOps.

References

Sommerville, I. (2016). *Software engineering* (10th ed.). Pearson Education Limited.

Kim, G., Behr, K., & Spafford, G. (2016). *The phoenix project: A novel about IT, DevOps, and helping your business win*. IT Revolution Press.