CI/CD Pipeline Implementation with Student Management System

Reza Asar (23501688)

Eastern Mediterranean University

CMSE 520: Software Evolution and Maintenance

Prof. Dr. Alexander Chefranov

Oct 31, 2024

Contents

Problem Definition

Modern software development faces challenges in consistently and reliably delivering updates from development to production environments. Manual deployment processes are error-prone, time-consuming, and can lead to environment inconsistencies. While implementing Continuous Integration and Continuous Deployment (CI/CD) pipelines is a solution to these challenges, setting up such pipelines involves complex integration of multiple tools and practices.

This project addresses these challenges by demonstrating a complete CI/CD pipeline implementation using a simple CRUD application. The project will showcase how various DevOps tools and practices integrate to create an automated software delivery pipeline, from code commit to deployment. By using a basic web application (managing student records) as the demonstration platform, the focus remains on the CI/CD implementation rather than application complexity.

The solution will demonstrate:

- Automated build and test processes triggered by code changes
- Containerization for consistent environments
- Automated deployment to a local Kubernetes cluster
- Database integration within the CI/CD pipeline
- Basic monitoring and logging capabilities

This should give future developers and maintainers a practical reference for implementing similar pipelines in their own projects.

List of Functional Requirements

- Source Control Management and Code Integration

    - The system must automatically trigger pipeline processes when code is committed to GitHub
    - All code changes must be version controlled and tracked
    - The system must maintain separate branches for development and production code
    - Database migration scripts must be version controlled alongside code changes
- Automated Build and Test Execution
    - The system must automatically build the application when new code is committed
    - The system must execute unit tests and integration tests automatically
    - The system must fail the build if any tests fail
    - Test results must be logged and accessible
- Automated Deployment Process
    - The system must automatically create Docker containers for both frontend and backend
    - The system must deploy the containerized application to local Kubernetes cluster
    - The system must execute database migrations during the deployment process
    - The system must support rollback capabilities if deployment fails
- Student Record Management (CRUD Operations)
    - The system must store and manage student records with the following properties: StudentId (unique identifier), FirstName, LastName, DateOfBirth, and Email
    - The system must allow creating, retrieving, updating, and deleting student records

List of Non-Functional Requirements

- Deployment Performance

    - The complete CI/CD pipeline execution (from commit to deployment) must complete within 10 minutes
    - Container images must not exceed 500MB in size
    - Database migration scripts must execute within 30 seconds
- Reliability and Stability
    - The system must maintain data consistency during deployments
    - The pipeline must provide clear error messages when builds or deployments fail
    - Failed deployments must automatically rollback to the last stable version
    - Pipeline status and logs must be retained for at least 30 days
- Maintainability
    - All Docker configurations must be defined in Dockerfile and docker-compose files
    - Kubernetes configurations must be defined as YAML manifests
    - Pipeline configurations must be version controlled using GitHub Actions YAML files
    - Code must follow standard naming conventions and include comments
- Monitoring and Logging
    - Pipeline execution status must be visible through GitHub Actions dashboard
    - Deployment logs must be accessible through Kubernetes dashboard

List of Actors and Their Services

Developer Services:

- Commit code changes to GitHub repository
- Create and commit database migration scripts
- Review pipeline execution status and logs

- Execute tests locally
- Access deployment logs and monitoring metrics
- Rollback deployments if needed

CI/CD Pipeline (System Actor) Services:

- Automatically trigger builds on code commits
- Execute automated tests
- Build Docker containers
- Apply database migrations
- Deploy applications to Kubernetes
- Provide status updates and logs
- Perform automatic rollbacks on failure

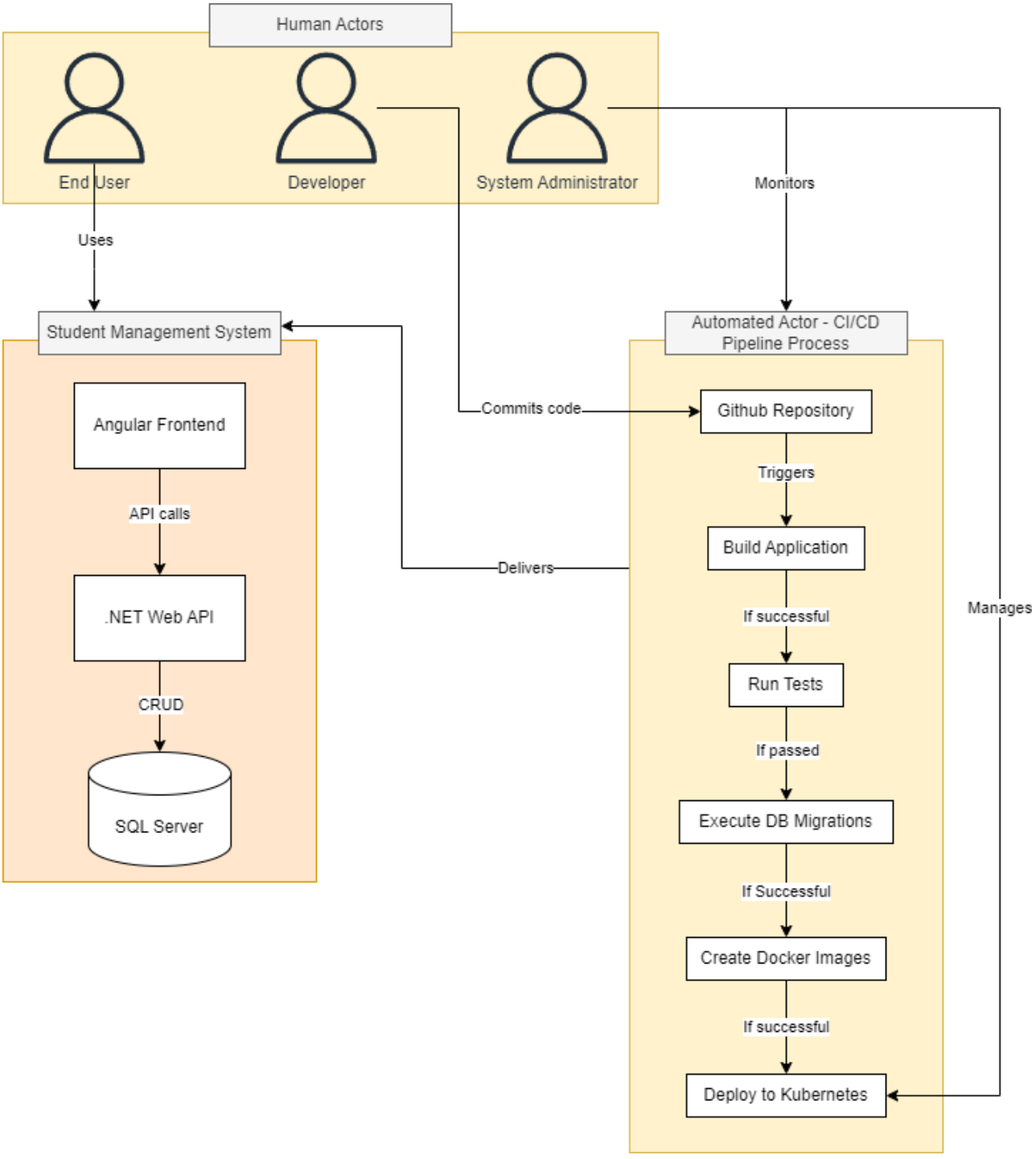
End User (Student Management System User) Services:

- View student records
- Add new student records
- Update existing student records
- Delete student records

System Administrator Services:

- Monitor system health and performance
- Access Kubernetes dashboard
- View application logs
- Manage container registry
- Configure pipeline settings
- Maintain infrastructure components

Context diagram of the system



Tools to be used

Development Tools

- Visual Studio 2022 (Backend development)
- Visual Studio Code (Frontend development)
- Git (Version control client)

Framework & Languages

- .NET 8 (Backend framework)
- Angular (Frontend framework)
- SQL Server (Database)

Pipeline & Containerization Tools

- GitHub (Source code repository & version control)
- GitHub Actions (CI/CD pipeline automation)
- Docker Desktop (Containerization)
- Kubernetes (Container orchestration)
- Minikube/Docker Desktop's Kubernetes (Local Kubernetes cluster)

Testing Tools

- XUnit (Unit testing framework)
- Moq (Mocking framework for testing)

Monitoring & Management Tools

- Kubernetes Dashboard (Container management)
- Docker Desktop Dashboard (Container monitoring)

Feasibility Analysis

Technical Feasibility

- All required tools are freely available and well-documented
- Development team has experience with .NET and SQL Server
- Local deployment using Docker Desktop and Kubernetes eliminates cloud service costs
- The technology stack is modern and widely supported by the developer community
- Project complexity is manageable as it focuses on basic CRUD operations

Resource Feasibility

- No additional hardware required beyond development machine
- All tools have community/free editions available
- Local deployment eliminates ongoing infrastructure costs
- Development can be completed using existing development machines

Operational Feasibility

- Local deployment simplifies operations and maintenance
- Docker ensures consistent environments across development and deployment
- Automated pipeline reduces manual intervention and potential errors
- Basic CRUD operations make the system easy to maintain
- Rollback capabilities ensure system can recover from failures

Timeline Feasibility

- Project scope is limited to essential CRUD operations
- Using well-known frameworks reduces development time
- CI/CD automation will speed up deployment processes