CT5171 CA1 Report:

Creating a Continuous Deployment Pipeline for a Web Application

Student Name: Erin Naughton

Student ID: 24105286

Email: e.naughton20@universityofgalway.ie

Module Code: CT5171 – Cloud DevOps

GitHub Repository: https://github.com/enaux/erinspetitions

Web Application: http://13.60.55.0:9090/

Application Description and Organisation

Overview

Erin's Petitions is a Spring Boot web application that enables users to create, view, search and sign petitions. The application implements a Continuous Integration/Continuous Delivery approach using Jenkins, Docker, and AWS EC2.

Core Functionality

The application provides five main features as specified in the requirements:

- **Create Petition**: Users can create new petitions with titles and descriptions
- **View All Petitions**: A homepage displays all available petitions with supporter counts
- **Search Functionality**: Dedicated search page with results page for finding specific petitions
- **View Petition**: Users can view individual petition pages including title and description
- **Sign Petition**: Individual petition pages allow users to add their support with name and email

Application Architecture

- **Backend**: Spring Boot 3.1.4 (Java 17)
- **Build Tool**: Maven
- **Template Engine**: Thymeleaf
- **Frontend**: Bootstrap 5.1.3 + Font Awesome
- **Data Storage**: In-memory (Java data structures)
- **Packaging**: WAR file for external Tomcat deployment
- **Container**: Tomcat 10.1 (Docker)

CI/CD Architecture

GitHub is the version control system. When code changes are pushed to the origin GitHub repository, a GitHub webhook triggers a build in Jenkins running on a live AWS EC2 instance. Jenkins accesses the Jenkinsfile within the specified GitHub repository and commences a build according to the defined pipeline. The pipeline incorporates an approval stage after which the application is deployed to a Tomcat server running in a Docker container on the same EC2 instance, making it publicly accessible at port 9090.

The Jenkins pipeline includes the following stages:

1. **CleanProject**: Clean workspace at start of pipeline
2. **GetProject**: Clone from GitHub
3. **Build**: Compile the application
4. **Test**: Run unit tests
5. **Package**: Create WAR file (erinspetitions.war)
6. **VerifyWAR**: Check the WAR file is structured correctly for deployment
7. **Archive**: Archive the WAR artifact
8. **Approval**: Manual approval before deployment
9. **Deploy**: Build Docker image and deploy Tomcat container

Project Structure

```
erinspetitions/
├── src/main/
│   ├── java/com/example/erinspetitions/
│   │   ├── ErinspetitionsApplication.java   # Main application class
│   │   ├── Petition.java                    # Petition data model
│   │   ├── PetitionController.java          # Web controller
│   │   ├── PetitionService.java             # Business logic
│   │   └── ServletInitializer.java          # WAR deployment support
│   └── resources/
│       ├── application.properties           # App configuration
│       └── templates/                       # Thymeleaf HTML templates
│           ├── index.html                   # List all petitions
│           ├── create-petition.html         # Create new petition
│           ├── view-petition.html           # View and sign petition
│           ├── search.html                  # Search form
│           └── search-results.html          # Search results
├── pom.xml                                  # Maven dependencies
├── Dockerfile                               # Tomcat container config
└── Jenkinsfile                              # CI/CD pipeline
```

Deployment Configuration

- **WAR File**: target/erinspetitions.war
- **Container**: Tomcat 10.1 with JDK 17
- **Context Path**: ROOT (deployed as ROOT.war for root access)
- **Port Mapping**: 9090 (host) -> 8080 (container)
- **Access URL**: http://13.60.55.0:9090/

Reflection on Challenges

EC2 Resource Limitations

The most significant operational challenge was dealing with extremely slow build times on AWS EC2. My initial t3.micro instance became completely unresponsive during the initial Maven build, with SSH connections freezing entirely. This was particularly frustrating when trying to debug pipeline issues. I learned that Spring Boot Maven builds are surprisingly resource-intensive, and the t3.micro instance type doesn't have enough CPU capacity for efficient workflow. Eventually, I switched to testing my application on a local virtual machine as this was able to proceed without freezes.

Spring Boot 3 + Tomcat 10.1 Deployment Issues

Getting the application to deploy correctly was the most challenging part of this assignment. I encountered many different errors at this stage, which lead to a persistent '404 – Not Found' error in the browser when trying to access the web application at http://13.60.55.0:9090/. My solution involved:

- Creating WebConfig.java to disable ErrorPageFilter (to prevent double-registration conflict with Tomcat 10 that was causing the application to crash)
- Adding spring.main.allow-bean-definition-overriding=true to application.properties (to allow WebConfig to override default ErrorPageFilter bean as this could not be successfully done using other structures)
- Configuring spring-boot-maven-plugin with <skip>true</skip> to produce traditional WAR structure (WEB-INF/) instead of executable WAR (BOOT-INF/) that could not be launched by Tomcat
- Adding comprehensive debugging and cleanup steps to the Jenkinsfile (to systematically identify the errors causing the ' 404- Not Found' error

Spring Boot and Thymeleaf Integration

One of the initial challenges was getting the Spring Boot application to properly serve Thymeleaf templates. I encountered issues with template resolution where the navigation between pages wasn't working as expected. The problem turned out to be incorrect controller mappings which took me a long time to identify. It was an easy fix when discovered!

Data Model Design

Implementing the petition signing functionality with both name and email required careful consideration of the data structure. I initially used a List<String> for signatories, but this didn't accommodate the requirement for both name and email collection. Instead, I created a nested Signatory class within the Petition model, which provided a cleaner object-oriented approach while maintaining the in-memory storage requirement.

Jenkins Configuration and Webhooks

Setting up the GitHub webhook to trigger Jenkins builds on the EC2 instance automatically worked very well. I had to reconfigure the webhook several times to match IP address changes that occurred after restarting the EC2 instance due to freezing. I had to test this several times before the automation worked reliably.

Docker and Tomcat Deployment

Integrating Docker into the deployment pipeline presented unexpected challenges, particularly around container networking and caching. Mounting the .war file to the Tomcat container required precise path configurations, and I had to experiment with different approaches to ensure the application deployed correctly and was accessible on port 9090 as specified.

Partially Implemented/Areas for Improvement:

While the core requirements of the assignment are met, with more time I would improve:

- **Testing Implementation**: The project currently has minimal unit test coverage. Given more time, I would implement test-driven development.
- **Database Integration**: While using in-memory storage met the requirements, a proper database like PostgreSQL would provide data persistence between application restarts and better scalability.

Lessons Learned

A key takeaway has been an understanding of the resource requirements for operating CI/CD, i.e. what works on a development machine doesn't necessarily work in cloud environments without proper resource planning. I also gained appreciation for the importance of incremental testing and the value of detailed logging when troubleshooting deployment issues.

Conclusion

This project has required me to apply theoretical DevOps concepts to practical experience. The challenges of environment setup, resource management, and pipeline design have provided invaluable real-world context. I have found it incredibly satisfying to see the complete system work as intended - from code commit to deployment. This has highlighted the power of modern DevOps practices. Building this application has developed my confidence in tackling similar challenges as I progress to professional environments and ultimately has highlighted the importance of systematic planning, persistent troubleshooting, and continuous learning in software development.