# 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://16.171.29.22:9090/

## 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 CI/CD pipeline 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

The application is organised as follows:

**Backend Components:**

- PetitionController.java - Handles HTTP requests and page routing
- PetitionService.java - Business logic and in-memory data storage
- Petition.java - Data model
- Spring Boot with Maven for dependency management

**Frontend Components:**

- Thymeleaf templates for server-side rendering
- Bootstrap 5 for responsive styling
- Five main html pages: index (view all), create-petition, view-petition, search, and search-results

**Data Storage:**

- In-memory Java collections (ArrayList) as specified
- Pre-loaded with three petitions
- Signatory data stored with both name and email

## CI/CD Pipeline Architecture

The continuous deployment pipeline connects all of the components from code commit to production deployment. 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 an AWS EC2 instance. Jenkins access 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.

## 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.

### 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 simple 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 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 file permissions and container networking. 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, there are several areas that could be enhanced with more time:

- **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.
- **Enhanced Error Handling**: The application has basic error handling but could benefit from more robust validation and user-friendly error messages.
- **Enhanced Monitoring**: Adding proper logging and monitoring to both the application and pipeline would improve debuggability and operational awareness.

## Lessons Learned

A key takeaway has been an understanding of the resource requirements for CI/CD pipelines, 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 pipeline issues.

If given more time, I would focus on implementing proper monitoring and alerting for the pipeline, adding rollback capabilities to the deployment process, and creating a staging environment for more thorough testing before production deployments.

## 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.