# Final Java DevOps CI/CD Project Documentation

## 1. Introduction

This project is a simple Java web application that requires an automated CI/CD pipeline using DevOps tools. The application runs as a container on a virtual machine (VM) on port 8080.

## 2. DevOps Workflow

CI/CD Pipeline Steps  
 Continuous Integration (CI):  
1. Clone the source code from GitHub.  
2. Build the application using Maven (mvnw).  
3. Run unit tests using mvnw.  
4. Dockerize the application and push the image to Docker Hub.  
  
 Continuous Deployment (CD):  
1. Deploy the application using Ansible by running a container from the built image.  
2. Monitor the application using Prometheus.

## 3. Technology Stack

GitHub Source Code Management  
 Maven Build Tool  
 Docker Containerization  
 Jenkins CI/CD Automation  
 Ansible Configuration Management & Deployment  
 Terraform Infrastructure as Code (IaC) for AWS  
 AWS EC2 Hosting Environment  
 Prometheus Monitoring

## 4. Architecture Overview

Infrastructure  
 Local VM : Hosts the application and Jenkins.  
 Dockerized Java Web Application: Packaged and deployed using Docker.  
 Terraform: Used for provisioning AWS infrastructure.  
 Jenkins Pipeline: Manages the CI/CD process.

## 5. Implementation Steps

Step 1: Setting Up the GitHub Repository  
 Create a GitHub repository and push the source code.  
 Add necessary files: Dockerfile, Jenkinsfile, and Ansible Playbook.  
  
 Step 2: Creating the CI/CD Pipeline  
 Define the pipeline stages in Jenkinsfile:  
 1. Checkout: Pull the latest code.  
 2. Build: Compile and package using mvnw.  
 3. Test: Run unit tests.  
 4. Dockerize: Create and push the Docker image.  
 5. Deploy: Use Ansible to run the container.  
  
 Step 3: Infrastructure Deployment Using Terraform  
 Define EC2 instance setup in Terraform.  
 Deploy Jenkins and Docker environment on the instance.  
  
 Step 4: Automating Deployment with Ansible  
 Configure Ansible Playbook to pull and run the Docker image.

## 6. Deployment & Monitoring

Use Prometheus for realtime application monitoring.  
 Ensure logging and alerting mechanisms are in place.

## 7. Expected Deliverables

The GitHub repository contains:  
 Source Code (src/ directory)  
 Dockerfile  
 Jenkinsfile  
 Ansible Playbook

## 8. Running the Application

To start the application, use:  
  
./mvnw cargo:run P tomcat90

## 9. Challenges & Solutions

Challenges:  
 Setting up an efficient CI/CD pipeline.  
 Managing cloud resources dynamically.  
 Ensuring reliable deployment automation.  
  
 Solutions:  
 Used Jenkins pipeline for automation.  
 Applied Terraform for consistent infrastructure provisioning.  
 Implemented Ansible for streamlined deployment.

## 10. Conclusion & Future Improvements

This project demonstrates a complete CI/CD implementation for a Java web application using DevOps best practices. Future improvements may include integrating Kubernetes for orchestration and enhancing security measures.  
  
Author: Ahmed Sameh  
Project Type: Java DevOps CI/CD  
Tools: GitHub, Docker, Jenkins, Ansible, Terraform, AWS, Prometheus

# 3. DevOps Methodology

DevOps combines software development (Dev) and IT operations (Ops) to shorten the systems development life cycle while delivering highquality software. It focuses on continuous integration, continuous deployment, automation, and monitoring.

# 4. Technology Stack

GitHub: Version control and collaboration.  
 Maven: Java build automation tool.  
 Docker: Containerization for scalable deployment.  
 Jenkins: CI/CD automation tool.  
 Ansible: Configuration management and automation.  
 Prometheus: Monitoring and alerting tool.

# 5. System Architecture

The architecture consists of:  
1. Source Control Layer: GitHub for versioning.  
2. CI/CD Pipeline: Jenkins automates building, testing, and deployment.  
3. Containerization Layer: Docker for packaging applications.  
4. Infrastructure Management: Terraform provisions AWS EC2 instances.  
5. Configuration Management: Ansible ensures system consistency.  
6. Monitoring & Logging: Prometheus tracks system health.

# 6. Environment Setup

To set up the environment:  
1. Install Java, Maven, and Docker.  
2. Set up Jenkins.  
3. Deploy Ansible playbooks for automation.

# 7. CI/CD Pipeline Design

The CI/CD pipeline consists of:  
 Code Commit: Developers push changes to GitHub.  
 Automated Build & Test: Jenkins triggers a Maven build and runs unit tests.  
 Containerization: Docker creates an image and pushes it to Docker Hub.  
Automated Deployment: Ansible pulls the latest image and runs it on AWS.

# 8. Implementation Details

Each stage in the CI/CD pipeline is automated using Jenkins pipelines. Configuration files like Dockerfile, Jenkinsfile.

# 9. Testing and Quality Assurance

Testing strategies include:  
 Unit Testing: Validates individual components.  
 Integration Testing: Ensures interoperability.  
 Load Testing: Tests performance under stress.

# 10. Deployment Strategies

Various deployment strategies are considered:  
 BlueGreen Deployment: Reduces downtime.  
 Rolling Updates: Gradual release of new features.  
 Canary Deployments: Testing new versions with a small subset of users.

# 11. Infrastructure as Code (Terraform)

Terraform automates AWS provisioning using declarative configurations, ensuring consistent deployments across environments.

# 12. Configuration Management (Ansible)

Ansible manages configurations, automates setup, and ensures system consistency through YAML playbooks.

# 13. Monitoring and Logging

Prometheus monitors system health and generates alerts for anomalies, ensuring proactive issue resolution.

# 14. Challenges and Lessons Learned

Challenges faced and their solutions include:  
 Managing dependencies: Used Docker to containerize applications.  
 Scaling CI/CD pipeline: Optimized Jenkins agents for parallel execution.

# 15. Future Enhancements

Potential improvements include:  
 Kubernetes integration for orchestration.  
 Serverless deployment for cost optimization.

# 16. References

Official documentation links for tools like Jenkins, Docker, Terraform, and Ansible.