DevOps Pipeline for Java Spring Boot using Jenkins, Docker, and AWS EC2

Overview

This project demonstrates setting up a Continuous Integration/Continuous Deployment (CI/CD) pipeline using Jenkins to automate building, testing, and deploying a Java Spring Boot application. Docker is used for containerization, and the final application is deployed on an AWS EC2 instance.

Tools and Technologies Used

- Java Spring Boot: Web application framework.
- Jenkins: For automating the CI/CD process.
- Docker: Containerization tool.
- GitHub: For source code management.
- **AWS EC2**: Deployment platform for hosting the application.
- Maven: Build tool for the Java project.
- YAML: Optional for Jenkins pipeline configuration.
- Git: Version control system.

Step-by-Step Implementation

Step 1: Create a Simple Spring Boot Application

1. Generate the Project

a. Use <u>Spring Initializr</u> with the following configuration:

i. **Project**: Mavenii. **Language**: Java

iii. **Spring Boot**: 2.7.x or latest

iv. Packaging: Jar

v. **Dependency**: Spring Web

2. Create the Basic Application Code

```
Tile Edit Selection View Go Run
                                                                                                                                                  J DemoApplication.java X ♀ application.properties ●
     ∨ DEMO
                                             package com.example.demo;

∨ .mvn\wrapper

       maven-wrapper.properties
                                           3 import org.springframework.boot.SpringApplication;
                                            import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;
        J DemoApplication.java
> resources
                                           8 @SpringBootApplication
9 @RestController
10 public class DemoApplication {
         J DemoApplicationTests.java
                                                     public static void main(String[] args) {
    SpringApplication.run(DemoApplication.class, args);
                                                     @GetMapping("/")
public String hello() {
   return "Hello, DevOps World from Spring Boot!";
       # Jenkinsfile
    > OUTLINE
553
```

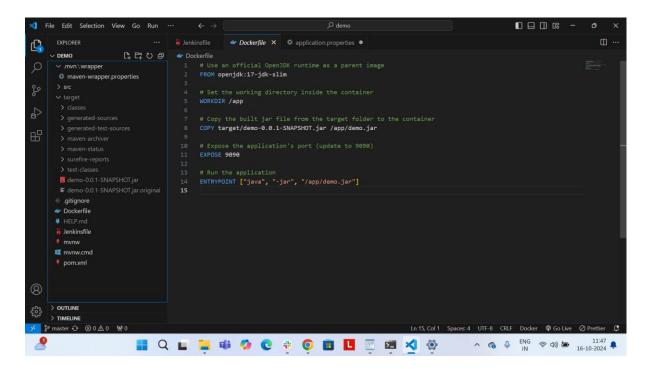
3. Build the Project

In the project root, run:

./mvnw clean package

Step 2: Containerize the Spring Boot Application with Docker

1. Create a Dockerfile



2. Build and Test Docker Image

docker build -t spring-boot-demo . docker run -p 8080:9090 spring-boot-demo

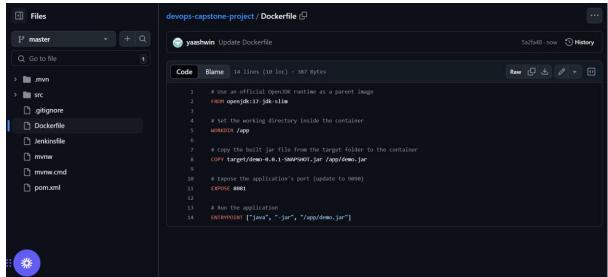
3. Verify Application

Visit http://localhost:9090 to see: "Hello DevOps World!"

Step 3: Push Code to GitHub

1. Initialize Git and Push Code

git init git add . git commit -m "Initial commit" git remote add origin GuviProject git push -u origin master



Step 4: Set Up Jenkins for CI/CD

1. Install Jenkins

a. Install Jenkins on your local machine instance.

2. Configure Jenkins

- a. Install Docker, Maven, and the following Jenkins plugins:
 - i. Git Plugin
 - ii. Maven Integration Plugin
 - iii. Docker Pipeline Plugin
- 3. Create a Jenkins Pipeline Job

```
₽ demo
                                                                                                                                             application.properties
      ∨ DEMO
                         ច្ជេញ

∨ .mvn\wrapper

                                                pipeline {
        maven-wrapper.properties
                                                     tools {
                                                    stages {
                                                          stage('Clone Repository') {
                                                                 ips {
    // Clone the Git repository
    git 'https://github.com/Naadira/devops-capstone-project.git'
                                                         stage('Build with Maven') {
       gitignore
                                                              steps {
       Dockerfile
                                                                 // Grant execute permission to mvnw (if you're using it, otherwise you can omit this line) sh 'chmod +x mvnw'
      mvnw.cmd
                                                         stage('Build Docker Image') {
                                                                  script {
    // Build the Docker image and tag it as 'latest'
    docker.build("naadira/spring-boot-demo:latest")
> OUTLINE > TIMELINE
                                                                                                      Ln 48, Col 41 Spaces: 4 UTF-8 CRLF Groovy 🏟 Go Live 🔗 Prett

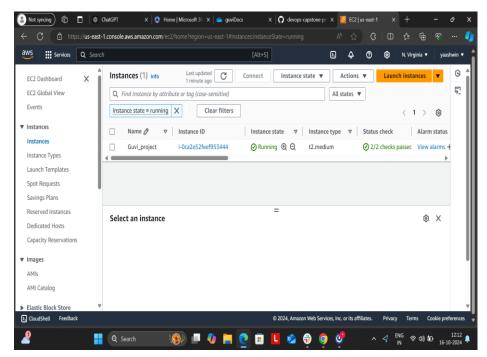
$\mathbb{Y}$ master \(\foat\) \(\otimes 0 \(\Delta\) 0 \(\otimes 0\)

                              11:47 A G D ENG S 40 D 11:47 A G D ENG S 40 D 16:10-2024
```

Step 5: Deploy the Application on AWS EC2

1. Create an EC2 Instance

a. Use Amazon Linux or Ubuntu for the instance.



2. Install Docker on EC2

SSH into the EC2 instance and run:

```
sudo yum update -y
sudo yum install docker -y
```

sudo service docker start
sudo usermod -aG docker ec2-user

3. Configure SSH Access in Jenkins

• Add your EC2 SSH key to Jenkins using the SSH Agent plugin.

Step 6: Monitor the Application

- 1. Access the Deployed Application
 - a. Visit:
 - b. http://<ec2-instance-ip>:9090



Step 7: Build the jenkins pipeline and check the container in EC2:



Project Recap

- Java Spring Boot: Developed a simple RESTful API.
- **Docker**: Used for containerization.
- **GitHub**: Managed source code.
- **Jenkins**: Automated build, test, and deployment.
- **AWS EC2**: Hosted the application in the cloud.