CI- CD Ecommerce Web Application

This assignment is designed to simulate a real enterprise CI/CD workflow where strict code quality enforcement, automated builds, and automated deployment are required. The complete workflow must be implemented using Jenkins Declarative Pipeline only. Freestyle jobs, manual steps between stages, or partial automation are not allowed. The application to be used is available in the GitHub repository: https://github.com/Msocial123/EcommerceApp.git. The pipeline should automatically trigger whenever code is pushed to this repository.

Two separate EC2 instances must be provisioned in AWS. One instance should host Jenkins with all necessary plugins installed and properly configured, including email notification setup. The second instance should host SonarQube. SonarQube must be installed, running properly, and integrated with Jenkins. A custom Quality Gate should be created in SonarQube with conditions such as minimum 70% code coverage, no critical bugs, and no blocker vulnerabilities. If the Quality Gate fails, the pipeline must automatically fail and stop further execution.

The pipeline should begin by automatically cloning the GitHub repository. Static code analysis must then be performed using Sonar Scanner integrated with the SonarQube server running on the separate EC2 instance. After the analysis is completed, the pipeline must check the custom Quality Gate status. If the project does not meet the defined quality standards, execution should stop immediately.

Once the Quality Gate is passed, the pipeline should build the application using Maven. The build process must run strictly through Jenkins and should generate a .war file. Manual builds are not permitted. If the build fails, the pipeline must stop automatically. After a successful build, the generated WAR file should be deployed automatically to a configured Tomcat server. Tomcat must be properly installed and running, and after deployment, the application should be accessible through a browser to confirm successful deployment.

The SonarQube analysis report should be downloaded and uploaded back into the same GitHub repository so that the report is versioned and traceable. Email notifications must be configured in Jenkins to send alerts on both success and failure. The email should contain details such as build status, job name, build number, and timestamp. Proper error handling must be implemented to ensure stability and reliability of the pipeline.

As part of the class work submission, the Jenkins Declarative Pipeline script (Jenkinsfile) should be provided along with screenshots showing Jenkins job execution, the SonarQube dashboard with Quality Gate status, and proof of successful Tomcat deployment. The SonarQube report should be visible in the GitHub repository, and the repository link should be shared. An architecture diagram can also be included for better clarity.

This exercise helps in understanding real-world CI/CD pipeline design, enforcement of code quality standards, automation of Java application builds and deployments, and integration of multiple DevOps tools in a production-like setup. Additional improvements may include securing Jenkins and SonarQube with authentication, parameterizing the pipeline for

multiple environments, implementing build versioning, and archiving build artifacts within Jenkins.