Jenkins CI/CD Pipeline Documentation

Overview

This document provides the steps and configuration for setting up a Jenkins CI/CD pipeline.

The pipeline integrates tools such as SonarQube, JaCoCo, Lizard, and OWASP Dependency-Check to ensure code quality,

security, and maintainability. Notifications are sent to madhurdevops30@gmail.com upon success or failure.

Step 1: Jenkins Installation and Setup

Jenkins Installation

Follow the detailed steps in this guide: [Jenkins Installation and Setup on Linux] (https://medium.com/tech-insider/jenkins-installation-and-setup-from-scratch-on-linux-5d9746b11fce).

1. Update the system packages:

  ```bash

  sudo apt update && sudo apt upgrade

  ```

2. Install Java (required for Jenkins):

  ```bash

  sudo apt install openjdk-11-jdk

  ```

3. Add Jenkins repository and import the GPG key:

  ```bash

  curl -fsSL https://pkg.jenkins.io/debian/jenkins.io.key | sudo tee "/usr/share/keyrings/jenkins-keyring.asc" > /dev/null

   echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] https://pkg.jenkins.io/debian binary/ | sudo tee /etc/apt/sources.list.d/jenkins.list > /dev/null
  ```

```

4. Install Jenkins:

```bash
sudo apt update
sudo apt install jenkins
```

5. Start Jenkins:

```bash
sudo systemctl start jenkins
```

6. Enable Jenkins to start on boot:

```bash
sudo systemctl enable jenkins
```

Access Jenkins

Access Jenkins via `http://<server-ip>:8080`. Use the initial admin password from:

```bash
sudo cat /var/lib/jenkins/secrets/initialAdminPassword
```

Step 2: Install Plugins

Essential Plugins

Navigate to ***Manage Jenkins > Plugins*** and install the following:

- "Git" (Source Control Management)
- "Pipeline" (Jenkinsfile support)
- "SonarQube Scanner"
- "Email Extension"
- "JaCoCo Plugin"
- "OWASP Dependency-Check Plugin"
```

Step 3: Source Code Management

Set Up a Git Repository

1. Push your source code to a GitHub/GitLab repository.

2. Include the following in the repository:

   - Source code

   - Unit tests

   - A `Jenkinsfile` for pipeline configuration.

Connect Git to Jenkins

1. In Jenkins, configure a job to pull the repository.

2. Select "Pipeline Script From SCM".

3. Add your Git repository URL and credentials (if private).

Configure Webhook in GitHub/GitLab/Bitbucket:

- **GitHub**:
    1. Go to your repository settings.
    2. Navigate to Webhooks > Add webhook.

       `Set the Payload URL to: http://<jenkins-server>/github-webhook/`
    3. In Content type, choose application/json.
    4. Under Which events would you like to trigger this webhook?, select Just the push event.
    5. Save the webhook.
- **GitLab**:
    1. Go to your repository settings.
    2. Navigate to Webhooks > Add webhook.

       `Set the URL to: http://<jenkins-server>/gitlab-webhook/`

3. Select Push events.
4. Save the webhook.

- **Bitbucket**:

1. Go to your repository settings.
2. Navigate to Webhooks > Add webhook.

   ```
   Set the URL to: http://<jenkins-server>/bitbucket-webhook/
   ```

3. Choose the Repository push event.
4. Save the webhook.

Testing the Webhook:

After configuring the webhook in your Git repository, when a commit is pushed to the main branch, Git will send a request to Jenkins, triggering the pipeline automatically. Jenkins will only proceed if the branch is main, as defined in the Checkout stage.

Jenkinsfile

```
pipeline {
    agent any

    tools {
        maven 'Maven'   // Specify tools (if needed)
        jdk 'JDK11'
    }

    stages {
        stage('Checkout') {
            steps {
                checkout scm
                // Ensure that this is the 'main' branch before proceeding
                script {
                    if (env.GIT_BRANCH != 'origin/main') {
                        currentBuild.result = 'SUCCESS'  // No further steps if not on
main
                        return
                    }
                }
            }
        }

        stage('Code Quality Analysis') {
            steps {
                script {
                    // Run SonarQube Analysis
                    withSonarQubeEnv('SonarQube') {
                        sh 'mvn sonar:sonar'
                    }
                }
            }
        }

        stage('Code Coverage') {
            steps {
```

```
                sh 'mvn clean test'
                jacoco execPattern: '**/jacoco.exec'
            }
        }

        stage('Cyclomatic Complexity') {
            steps {
                sh 'lizard . > lizard-report.txt'
                archiveArtifacts artifacts: 'lizard-report.txt', allowEmptyArchive: true
            }
        }

        stage('Dependency Vulnerability Check') {
            steps {
                sh 'dependency-check.sh --project MyProject --scan ./ --out dependency-
check-report'
                archiveArtifacts artifacts: 'dependency-check-report/*',
allowEmptyArchive: true
            }
        }

        stage('Build') {
            steps {
                sh 'mvn clean package'
            }
        }
    }

    post {
        success {
            emailext subject: 'Build Successful: ${env.JOB_NAME} #${env.BUILD_NUMBER}',
                     body: "The Jenkins build for job '${env.JOB_NAME}' (Build
#${env.BUILD_NUMBER}) was successful.\n\nDetails:\nBuild URL: ${env.BUILD_URL}\nConsole
Output: ${env.BUILD_URL}console",
                     to: 'madhurdevops30@gmail.com'
        }
        failure {
            emailext subject: 'Build Failed: ${env.JOB_NAME} #${env.BUILD_NUMBER}',
                     body: "The Jenkins build for job '${env.JOB_NAME}' (Build
#${env.BUILD_NUMBER}) has failed.\n\nDetails:\nBuild URL: ${env.BUILD_URL}\nConsole
Output: ${env.BUILD_URL}console",
                     to: 'madhurdevops30@gmail.com'
        }
    }
}
```

Tools Integration

- **SonarQube**:
  - Install SonarQube on a separate server or use SonarCloud.
  - Configure **SonarQube Scanner** in Jenkins **(Manage Jenkins > Configure System > SonarQube Servers).**
  - Update the `pom.xml` or build script to include SonarQube plugin.
- **Code Coverage (JaCoCo)**:
  - Add the JaCoCo Maven plugin to the pom.xml.

```
xml

<plugin>
    <groupId>org.jacoco</groupId>
    <artifactId>jacoco-maven-plugin</artifactId>
    <version>0.8.7</version>
    <executions>
        <execution>
            <goals>
                <goal>prepare-agent</goal>
            </goals>
        </execution>
```

```
    </executions>
</plugin>
```

Cyclomatic Complexity (Lizard):

- Install **Lizard** on the build agent.

```
bash

pip install lizard
```

Security Scanning (OWASP Dependency-Check):

- Download the OWASP Dependency-Check CLI and configure it in Jenkins.

Notifications

- Configure email notifications:
  - Go to **Manage Jenkins > Configure System > Extended E-mail Notification**.
  - Add SMTP server details (e.g., Gmail, SES).

Each tool is integrated as a stage in the Jenkinsfile, ensuring automated quality checks during the build process.

Summary

- "Stages":

1. **"Checkout"**: Pulls code from SCM.
2. **"Code Quality Analysis"**: Runs SonarQube scan.
3. **"Code Coverage"**: Generates JaCoCo coverage reports.
4. **"Cyclomatic Complexity"**: Measures complexity using Lizard.
5. **"Dependency Scan"**: Checks vulnerabilities with OWASP Dependency-Check.
6. **"Build"**: Compiles and packages the application using Maven.

Notifications

Email notifications are configured to send build results to madhurdevops30@gmail.com.

- Success: A summary of the successful build with links to the build and console logs.

- Failure: Details of the failed build, including a link to the console output for debugging.