# Jenkins Pipeline Setup and Configuration

## 1. Jenkins Setup

### Objective

Install and configure Jenkins on a virtual machine or cloud-based service to automate your CI/CD pipeline.

### Steps

1. **Install Jenkins**:
    - On a virtual machine: Follow installation guides for your OS (Ubuntu, Windows).
    - Cloud-based: Use services like Jenkins on AWS.
2. **Install Plugins**:
    - Required Plugins:
        - Git Plugin
        - Pipeline Plugin
        - SonarQube Scanner Plugin
        - JaCoCo Plugin
        - OWASP Dependency-Check Plugin
        - Slack Notification Plugin
        - Email Extension Plugin
3. **Initial Setup**:
    - Access Jenkins via `http://<your-server-ip>:8080`.
    - Unlock Jenkins using the administrator password.
    - Configure the basic setup and install recommended plugins.

---

## 2. Source Code Management

### Objective

Use Git as the source code management tool and configure Jenkins to pull the latest code from your repository.

### Steps

1. **Set Up Git**:
    - Install Git on the Jenkins server.
    - Add Git credentials in Jenkins:
        - Go to **Manage Jenkins > Credentials**.
        - Add SSH or HTTPS credentials for your Git repository.
2. **Configure Git in Pipeline**:

Use the following snippet in your Jenkinsfile:

```
 stage('Checkout Code') {
    steps {
        git branch: 'main', url: 'https://github.com/<your-repo>.git'
    }
}
```

    -

---

## 3. Pipeline Creation

### Objective

Define pipeline stages in a Jenkinsfile to automate builds triggered on every commit.

### Steps

1. **Jenkinsfile Setup**:
    - Create a `Jenkinsfile` in the root of your repository.
    - Define pipeline stages (e.g., Build, Test, Code Quality, Notifications).
2. **Trigger on Commit**:
    - Configure the pipeline to trigger automatically:
        - Go to **Project Configuration > Build Triggers**.
        - Select **Poll SCM** or **GitHub hook trigger for GITScm polling**.

---

## 4. Code Quality Checks

### Objective

Integrate SonarQube to analyze code quality and enforce quality gates.

### Steps

1. **Install SonarQube**:
    - Install SonarQube server locally or use a hosted instance.
    - Install the SonarQube Scanner plugin in Jenkins.
2. **Configure SonarQube in Jenkins**:
    - Go to **Manage Jenkins > Configure System**.
    - Add SonarQube server details.
3. **Add SonarQube Stage**:

Include this stage in your Jenkinsfile:

```
 stage('SonarQube Analysis') {
    steps {
        withSonarQubeEnv('Sonar-Server') {
            sh 'mvn sonar:sar'
        }
    }
}
```

    -
4. **Break Pipeline on Quality Gate Failure**:
    - Enable the "Wait for Quality Gate" step in SonarQube.

---

# 5. Code Coverage

### Objective

Generate and publish code coverage reports using JaCoCo.

### Steps

**Add JaCoCo Plugin to pom.xml**:

```
 <plugin>
    <groupId>org.jacoco</groupId>
```

```
    <artifactId>jacoco-maven-plugin</artifactId>
    <version>0.8.8</version>
    <executions>
      <execution>
        <goals>
          <goal>prepare-agent</goal>
        </goals>
      </execution>
      <execution>
        <id>report</id>
        <phase>verify</phase>
        <goals>
          <goal>report</goal>
        </goals>
      </execution>
    </executions>
</plugin>
```

1.

**Include in Jenkinsfile**:

```
 stage('Code Coverage') {
    steps {
      sh 'mvn jacoco:report'
    }
}
```

2.
3. **Publish Coverage Report**:
    - Use Jenkins JaCoCo plugin to visualize the report.

---

# 6. Cyclomatic Complexity

## Objective

Calculate cyclomatic complexity using Lizard or a similar tool.

## Steps

1. **Install Lizard**:

Install Lizard via pip:
```
pip install lizard
```

- 

**Add Lizard Stage**:

```
stage('Cyclomatic Complexity') {
    steps {
        sh 'lizard . > complexity-report.txt'
        archiveArtifacts artifacts: 'complexity-report.txt'
    }
}
```

2. 
3. **Review Complexity Reports**:

    - Access the `complexity-report.txt` artifact in Jenkins.

---

# 7. Security Vulnerability Scan

## Objective

Scan dependencies for known vulnerabilities using OWASP Dependency-Check.

## Steps

**Add Dependency-Check Plugin to pom.xml**:

```
<plugin>
    <groupId>org.owasp</groupId>
    <artifactId>dependency-check-maven</artifactId>
    <version>8.4.0</version>
    <executions>
        <execution>
            <goals>
                <goal>check</goal>
            </goals>
        </execution>
    </executions>
```

</plugin>

1.

**Add Vulnerability Scan Stage**:

```
 stage('Vulnerability Scan') {
    steps {
       sh 'mvn dependency-check:check'
       archiveArtifacts artifacts: 'target/dependency-check-report.*', allowEmptyArchive: true
    }
}
```

2.

---

# 8. Notifications

## Objective

Send notifications (email and Slack) on build success or failure.

## Steps

1. **Slack Configuration**:
    - Install Slack Notification Plugin.
    - Configure Slack in **Manage Jenkins > Configure System**.
2. **Email Configuration**:
    - Install Email Extension Plugin.
    - Configure SMTP in **Manage Jenkins > Configure System**.

**Add Notifications in Jenkinsfile**:

```
 post {
    always {
       echo "Sending notifications..."
    }
    success {
       slackSend(channel: '#general', message: "SUCCESS: Build ${env.BUILD_NUMBER}
completed successfully!")
       emailext(subject: "SUCCESS: Build ${env.JOB_NAME} #${env.BUILD_NUMBER}",
```

```
            body: "Build succeeded: ${env.BUILD_URL}",
            recipientProviders: [[$class: 'DevelopersRecipientProvider']])
    }
    failure {
        slackSend(channel: '#general', message: "FAILURE: Build ${env.BUILD_NUMBER} failed.
Please check logs.")
        emailext(subject: "FAILURE: Build ${env.JOB_NAME} #${env.BUILD_NUMBER}",
            body: "Build failed: ${env.BUILD_URL}",
            recipientProviders: [[$class: 'CulpritsRecipientProvider']])
    }
}
```