#### **2. Set Up Jenkins Job**

1. Create a New Job:
   * Go to Jenkins dashboard.
   * Click on "New Item" and enter a name for your job.
   * Choose "Freestyle project" and click "OK".
2. Configure Source Code Management:
   * In the job configuration page, under "Source Code Management", select "Git".
   * Enter the repository URL and credentials if necessary.
3. Configure Build Triggers:
   * Under "Build Triggers", you can choose "Poll SCM" or "GitHub hook trigger for GITScm polling" to automate the build process.
4. Add Build Steps:
   * Under "Build", click "Add build step" and select "Invoke Gradle script" or "Invoke top-level Maven targets" depending on your build tool.
   * For Gradle, specify build as the Tasks.
   * For Maven, specify clean install.
5. Save and Apply the configuration.

#### **3. Pipeline as Code with Jenkinsfile**

Instead of a freestyle project, you can define your pipeline in a Jenkinsfile and store it in your repository.

##### **Example Jenkinsfile for a Spring Boot Application:**

pipeline {

agent any

stages {

stage('Checkout') {

steps {

git 'https://github.com/your-repo/spring-boot-app.git'

}

}

stage('Build') {

steps {

sh './gradlew clean build' // Use 'mvn clean install' if using Maven

}

}

stage('Test') {

steps {

sh './gradlew test' // Use 'mvn test' if using Maven

}

}

stage('Package') {

steps {

sh './gradlew bootJar' // Use 'mvn package' if using Maven

}

}

stage('Deploy') {

steps {

// Add your deployment steps here, e.g., using SCP, SSH, Docker, etc.

sh 'scp build/libs/\*.jar user@server:/path/to/deploy'

}

}

}

post {

success {

echo 'Build and Deploy succeeded!'

}

failure {

echo 'Build or Deploy failed!'

}

}

}

#### **4. Run the Pipeline**

* Commit and push your Jenkinsfile to your repository.
* Create a new pipeline job in Jenkins.
* In the job configuration, point to your repository where the Jenkinsfile is located.
* Jenkins will automatically detect and execute the pipeline defined in the Jenkinsfile.