International Journal for Innovative Engineering and Management Research
A Peer Revieved Open Access International Journal
www.ijiemr.org

COPY RIGHT

ELSEVIER SSRN

IJIEMR Transactions, online available on 28th Oct 2022. Link :http://www.ijiemr.org/downloads.php?vol=Volume-11&issue=Issue 10

DOI: 10.48047/IJIEMR/V11/ISSUE 10/13

Title **CI/CD Pipeline using Jenkins to Implement Quality Gates for Static Code Analysis, Static Unit Testing, Quality Gates for Code Coverage**

Volume 11, ISSUE 10, Pages: 108-117

Paper Authors
Dr.A.Radhika, M. Anith

USE THIS BARCODE TO ACCESS YOUR ONLINE PAPER

To Secure Your Paper As Per UGC Guidelines We Are Providing A Electronic Bar Code

International Journal for Innovative Engineering and Management Research
A Peer Revieved Open Access International Journal
www.ijiemr.org

CI/CD Pipeline using Jenkins to Implement Quality Gates for Static Code Analysis, Static Unit Testing, Quality Gates for Code Coverage

Dr.A.Radhika, Associate Professor, Dept of Computer Science and Engineering
SRK Institute of Technology
M. Anitha, Assistant Professor, Dept of Master of Computer Applications,
SRK Institute of Technology

ABSTRACT
Agile methodology was playing a major role a few years back when the software was deployed in monthly, quarterly or annual basis which was time consuming. But now a days software can be deployed multiple times a day using Devops. In current era, delivering creative ideas in a steady and rapid manner is eminently significant for all organizations. In addition to that, organizations need to delivery the products fastly as per the market requirements, As there are frequent delivery of products there is more interaction with the customer and there will be decrease in failure rate. All these can be achieved with the help of Devops methodology. To quickly produce and deploy the software product across multiple platforms and environment, DevOps methodology extends the agile to gain high performance and quality assurance products. Continuous integration/Continuous deployment (CI/CD) is the backbone of DevOps environment. By automating the validation, build, testing, installing and deployment of software, CI/CD bridges the gap between development and operation teams.CI/CD pipeline is implemented using Jenkins not only for building the product,but also to perform static code analysis, static unit testing and quality gates to test the quality of code in the form of reliability, security, maintainability.

Keywords: Git, Maven, Jenkins, Sonar Qube, CI/CD deployment pipeline

I. Introduction

A pipeline is a concept that introduces a series of events or tasks that are connected in a sequence to make quick software releases. If there exist a task and that task has got five different stages, and each stage has got several steps. All the steps in phase one must be completed, to mark the .

later stage to be complete, consider the CI/CD pipeline [1] as the backbone of the DevOps approach. This Pipeline is responsible for building codes, running tests, and deploying new software versions. Continuous Integration (CI) [2]is a practice that integrates code into a shared repository. It uses automated verifications for the early detection of problems. Continuous Integration doesn't eliminate bugs but helps in finding and removing them quickly.

Continuous Delivery (CD): It is the phase where the changes are made in the code before deploying. The team in this phase decides what is to be deployed to the customers and when. The final goal of the pipeline is to make deployments.When both

these practices come together, all the steps are considered automated, resulting in the process we know as CI/CD.

Implementation of CI/CD enables the team to deploy codes quickly and efficiently. The Process makes the team more agile, productive and confident. Jenkins is an open source automation tool written in Java with plugins built for Continuous Integration purpose.Jenkins is used to build and test your software projects continuously making it easier for developers to integrate changes to the project, and making it easier for users to obtain a fresh build. It also allows you to continuously deliver your software by integrating with a large number of testing and deployment technologies. Jenkins is the DevOps tool that is most used for CI/CD pipelines. Jenkins is a tool to build the pipeline.

II. Building CI/CD Pipeline for Jenkins

There are six steps to building a pipeline with Jenkins[3]

1) Download Jenkins from the Jenkins downloads page 'https://www.jenkins.io/download/'.

2) Download the file 'Generic Java package(.war)'.

3)Open the terminal window and enter cd <yourpath>. Use the command java –jar ./Jenkins. war to run the WAR file.

4) Open the web browser and open localhost:8080.

Configure and Execute a Pipeline Job With a Direct Script

- Choose Pipeline script as the Destination and paste the Jenkins file content in the Script from the GitHub.To keep the changes save the file.Click on the Build Now to process the build.
- To check the output, click on any stage and click Log; a message will appear on the screen.

5)The Jenkins dashboard opens creates new jobs there.

Create FreeStyle Projects in Jenkins[2]

1)Set Global Tool Configuration (Java,Maven)

2)Select Free Style Project

3)Check 'use Custom workspace' and pass the project path

4)Build select ' Invoke top-level Maven Tarhet" option

5)pass goal-clean test option to test the maven code application

6)Apply and save and the build the maven application using Jenkins

Create Pipeline Job in Jenkins

- Create a Pipeline Job
- Select and define what Jenkins job that is to be created.
- Select Pipeline, give it a name and click OK.
- Scroll down and find the pipeline section.
- Retrieve the Jenkins file from SCM (Source Code Management) Either or directly write a pipeline script

International Journal for Innovative Engineering and Management Research
A Peer Revieved Open Access International Journal
www.ijiemr.org

Configure and Execute a Pipeline With SCM

- Copy the GitHub RepositoryURL by clicking on Clone or download and then click on Configure to modify the existing job.
- Scroll to the Advanced Project Options setting and select Pipeline script from the SCM option.Paste the GitHub repository URL.
- Type Jenkins file in the Script, and then click on the Save button.
- Next, click on Build Now to execute the job again.
- There will be an additional stage, in this case, i.e., Declaration: Checkout SCM.
- Click on any stage and click on Log.

III.SonarQube

SonarQube [4]checks code quality and code security to enable the writing of cleaner and safer code. It currently supports code analysis in 27 programming languages using different plugins available for the default standard rule set. SonarQube is an automatic code analysis tool to find bugs, vulnerabilities and code smells in your source code. It can be integrated with the existing development workflow to enable continuous code analysis across project branches and pull requests. It can also be integrated directly into IDEs (Eclipse, VS Code, Visual Studio and Intellij) to find code related issues while developing code.

The workflow which should be followed for continuous static code analysis is

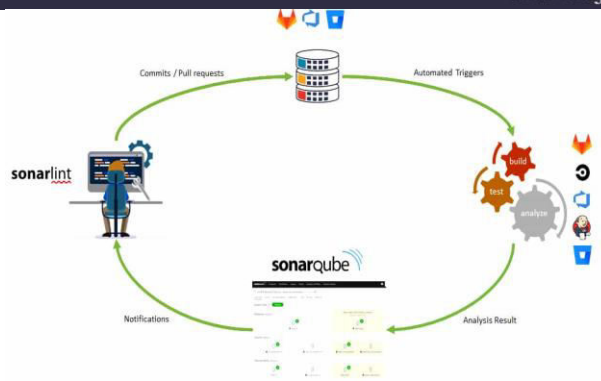


Figure 2: Standard development process with SonarQube

In the figure, the standard development process is shown through the following steps:

1. The developer should use SonarLint to receive immediate feedback in the IDE while coding, and then commit the code to the source code repository (GitHub, GitLab, Azure DevOps, Bit bucket).
2. Continuous integration (CI) pipeline should be triggered to produce builds, run unit tests and to analyse the source code with the help of the SonarQube scanner. CI tools that can be easily integrated with SonarQube analysis are Jenkins, GitLab, Azure DevOps, Bitbucket, and others.
3. Once the pipeline executes the analysis, the scanner publishes the results to the SonarQube server. Then the latter provides feedback to developers via the SonarQube interface, email, in-IDE notifications (via SonarLint), and decoration on pull requests.

Integrating SonarQube analysis with the Jenkins pipeline

The following Jenkins plugins are required to be installed:

1. Blue Ocean
2. GitHub
3. Maven Integration
4. Pipeline Maven Integration
5. Cobertura

6. SonarQube Scanner for Jenkin's.To do so, *Jenkins >Manage Plugins >Available >Search* for each plugin, and click Install without clicking the Restart button at the bottom of the page.

- Creating a token in SonarQube:
  1. Go to the SonarQube dashboard. Click on *My Account.*
  2. Then go to *Security tab >Token section >Generate Tokens section,* give an appropriate name to the token, and click on the *Generate* button to its right.
- Creating Webhook in SonarQube for connection with Jenkins:
  1. Go to *SonarQube dashboard >Administration >Configuration >Webhooks* and click the Create button on the right side to *create* a webhook.
  2. Provide name, URL as 'JENKINS-URL/sonarqube-webhook/' and a secret for connection with Jenkins.
  3. Click on the *Create* button to save.

Connecting SonarQube with Jenkins:
1. Go to *Manage Jenkins >Configure System >SonarQube servers. Click on Add* go to *Jenkins home page >Manage SonarQube.*
2. Enter the name and the URL of the SonarQube server; for example, *https://test.sonarqube-url.com/.*
3. Server authentication token: Add the created token to Jenkins.
4. Select the *Secret* text from the drop down of *Kind.*
5. Copy and paste the generated token from SonarQube and provide an ID to a token. Click the *Add* button to save the token in Jenkins.
6. Select *Secret* from the drop down in SonarQube authentication token using the same ID.
7. Now click on the *Advanced* button to provide advanced settings.
8. Add the SonarQube webhook token to Jenkins just like we added the SonarQube authentication token as secret text.
9. Select the webhook secret from the drop-down in 'Webhook secret' for advanced configuration of *SonarQube Server* in Jenkins.
10. Save the configuration. It should look like what is shown in Figure 3.

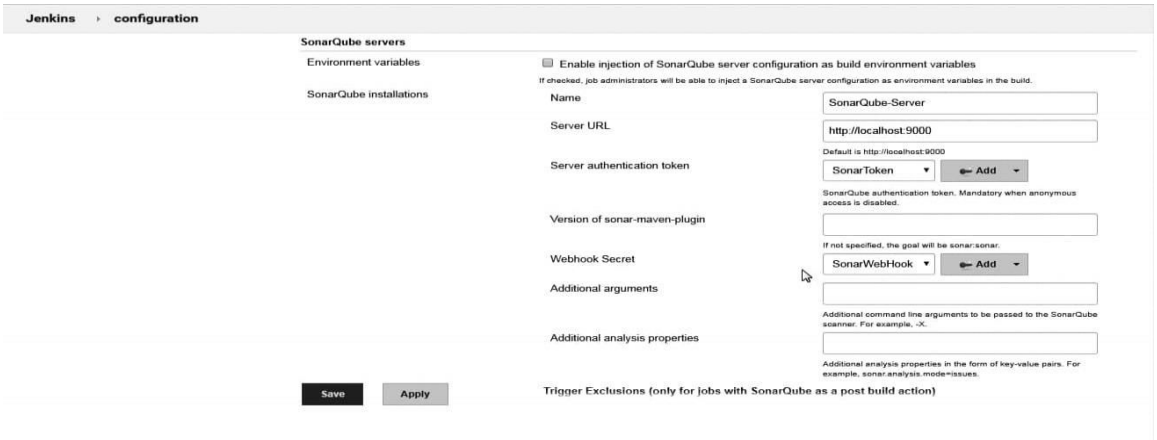


Figure 3: SonarQube server configuration to manage Jenkins

IV. Unit test, Maven Java code and publishing the code coverage report inJenkins:

SonarQube is not capable of generating the units and code coverage results by itself .It usually imports the reports executed by the test framework used in the application.Junit is the test framework.Jacoco [5] and Cobertura are the plugins for the generation of code coverage.

Add the plugins in Maven pom.xml file. Below is the pom.xml code for unit test and code coverage of Java applications.

```
<build>
<plugins>
<plugin>
<groupId>org.jacoco</groupId>
<artifactId>jacoco-maven-plugin</artifactId>
<version>0.7.5.201505241946</version>
<executions>
<execution>
<goals>
<goal>prepare-agent</goal>
</goals>
</execution>
<execution>
<id>report</id>
<phase>prepare-package</phase>
<goals>
<goal>report</goal>
</goals>
</execution>
<execution>
<id>jacoco-check</id>
<goals>
<goal>check</goal>
</goals>
</execution>
</executions>
</plugin>
<plugin>
<groupId>org.codehaus.mojo</groupId>
<artifactId>cobertura-maven-plugin</artifactId>
<version>${cobertura.version}</version>
<configuration>
<formats>
<format>xml</format>
</formats>
</configuration>
<executions>
<execution>
<phase>package</phase>
<goals>
<goal>cobertura</goal>
</goals>
</execution>
</executions>
</plugin>
<plugin>
<groupId>org.sonarsource.scanner.maven</groupId>
<artifactId>sonar-maven-plugin</artifactId>
<version>3.6.0.1398</version>
</plugin>
</plugins>
</build>
```

Now, to create a Jenkins pipeline, create a *Jenkins file* in the source code repository. Add the script below to implement unit test cases and code coverage:

```
pipeline {
agent {
node {
label 'master'
}
}
stages {
stage('Continuous Integration') {
steps {
withMaven(jdk: 'JAVA_HOME', maven:
 'MAVEN_HOME') {
bat 'mvn clean'
bat(script: 'mvn test cobertura:cobertura install',
label: 'Unit Testing and Code Coverage')
cobertura(autoUpdateHealth: true,
 autoUpdateStability: true, classCoverageTargets:
'target/site/cobertura/', coberturaReportFile:
```

```
'target/site/cobertura/*.xml', failUnstable: true, zoo
}
}
}
}
}
```

The *Jenkinsfile* in the code above includes the steps given below in the pipeline:
1. Execute unit test result and Cobertura code coverage. Get the reports in a SonarQube acceptable format (SonarQube supports JUnit, Cobertura and Jacoco reports).

2. Publish the unit test and coverage reports to the Jenkins dashboard. Save and run the pipeline in Jenkins again. On successful execution of unit testing, it will publish the results shown in Figure 4.

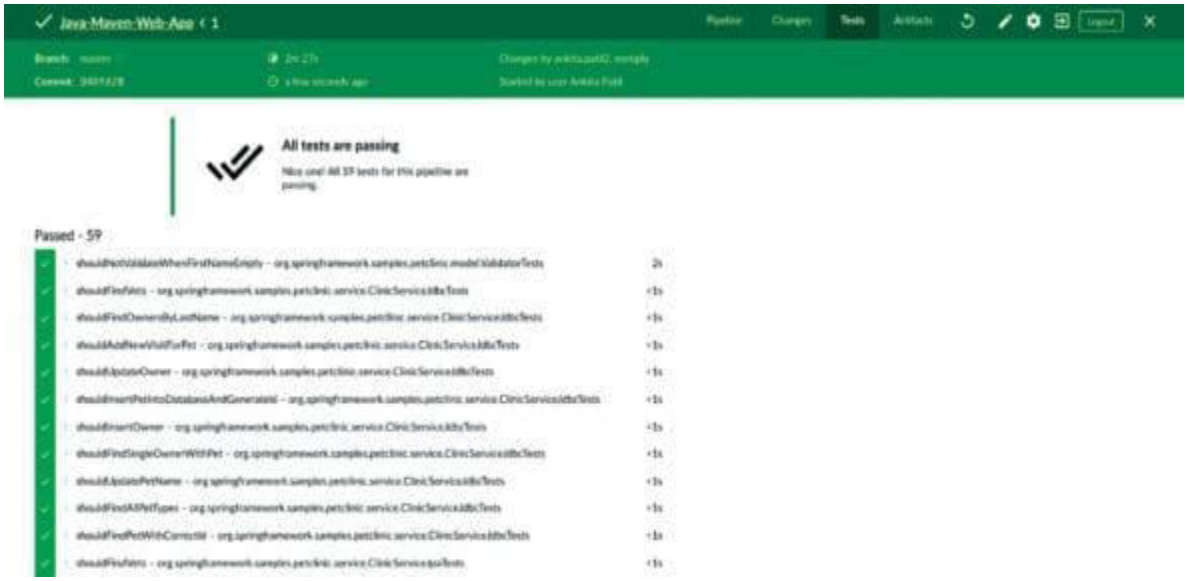


Figure 4: Unit test report on Jenkins dashboard

The code coverage report looks like what is shown in Figure 5.

International Journal for Innovative Engineering and Management Research
A Peer Revieved Open Access International Journal
www.ijiemr.org

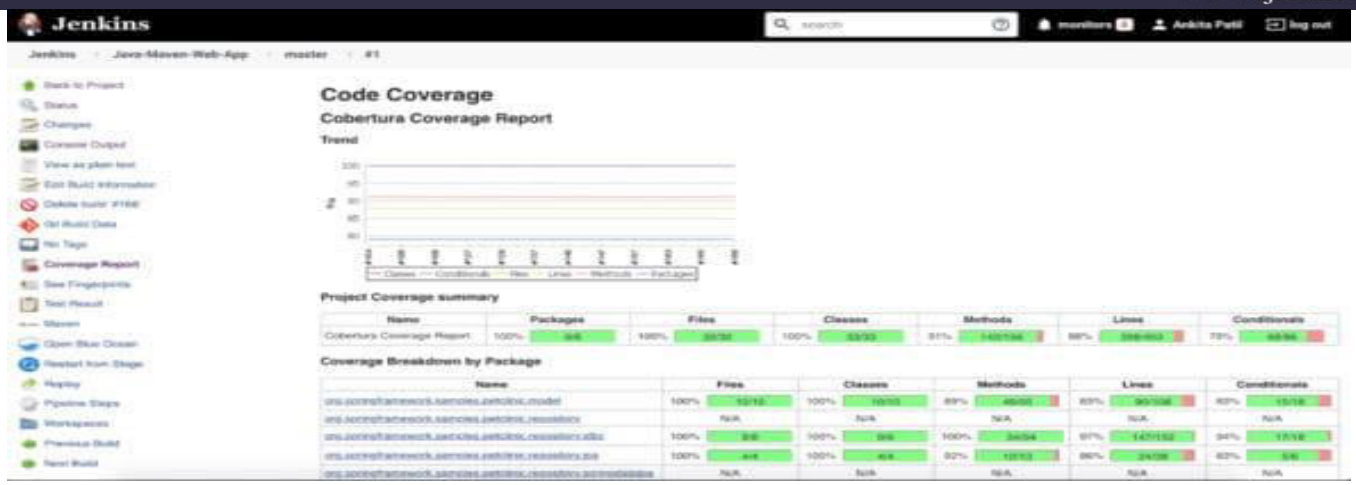


Figure 5: Cobertura code coverage on Jenkins

V. Implementing SonarQube analysis from Jenkins pipeline:

First, create the *sonar-project.properties* file in the root of the repository. This file is used to define the analysis parameter, which we need to provide to the SonarQube scanner during analysis[6].

The sample *sonar-project.properties* file for our Maven application is in the code below:

```
sonar.projectKey=java-sonar-runner-sample
sonar.projectName=Simple Java project analyzed with the SonarQube Runner
sonar.projectVersion=1.0
# Comma-separated paths to directories with sources (required)
sonar.sources=src/main
sonar.tests=src/test
sonar.java.binaries=target/classes
sonar.java.test.binaries=target/test-classes
#Unit Test And Code Coverage
sonar.junit.reportPaths=target/surefire-reports
sonar.java.cobertura.reportPath=target/site/cobertura/coverage.xml
sonar.coverage.jacoco.xmlReportPaths=target/site/jacoco/jacoco.xml
sonar.sourceEncoding=UTF-8
```

Now, referring to this property file we need to perform SonarQube analysis from Jenkins. So update the *Jenkinsfile,* and add in it the code given below for the 'Continuous Integration' stage to perform SonarQube analysis (the code below has the entire *Jenkinsfile* for unit testing and SonarQube analysis):

```
pipeline {
agent {
node {
label 'maven'
}
}
stages {
stage('Continuous Integration') {
steps {

withMaven(jdk: 'JAVA_HOME', maven:
'MAVEN_HOME') {
bat 'mvn clean'
bat(script: 'mvn test cobertura:cobertura install',
 label: 'Unit Testing and Code Coverage')
cobertura(autoUpdateHealth: true,
autoUpdateStability: true, classCoverageTargets:
'target/site/cobertura/', coberturaReportFile:
'target/site/cobertura/*.xml',
failUnstable: true, zoomCoverageChart: true)
}
```

```
withSonarQubeEnv(installationName: 'SonarQube-
bat(script: 'D://Softwares//sonar-scanner-cli//sonar-
}
waitForQualityGate(abortPipeline: true, cr
'SonarWebHook')
}
}
}
}
```

1. *withSonarQubeEnv:* SonarQube analysis using sonar-scanner in batch script and the *sonar-project.properties* files we created.
2. *waitForQualityGate*: Checking for the 'quality gate' status after the SonarQube analysis is executed.

The code has referenced the SonarQube server (SonarQube-Server), SonarToken and ner -Dproject.s SonarWebhook, which we configured in the 'Manage Jenkins with Jenkins and SecretId: SonarQube' setup.

The two steps performing SonarQube analysis in *Jenkinsfile* above are:

Save the pipeline and run it again. On its successful execution, we will be able to find 'quality gate' results in the Jenkins pipeline console, as shown in Figure 6.

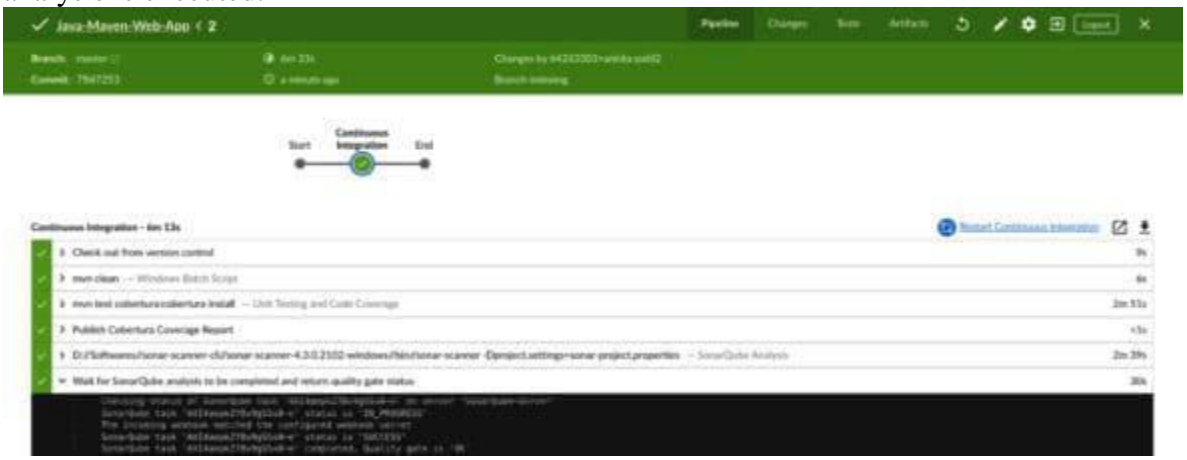


Figure 6: SonarQube 'quality gate' check in Jenkins pipeline

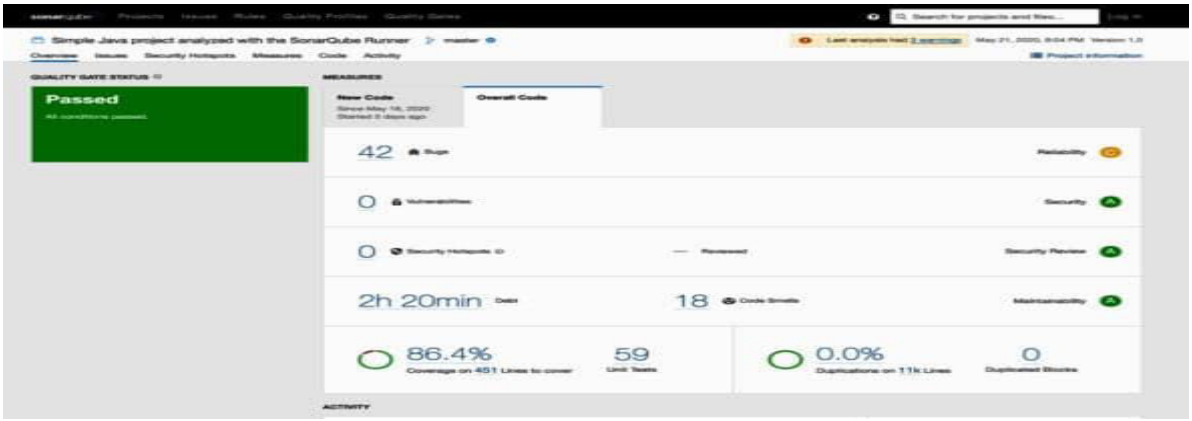


Figure 7: SonarQube project dashboard after analysis from Jenkins

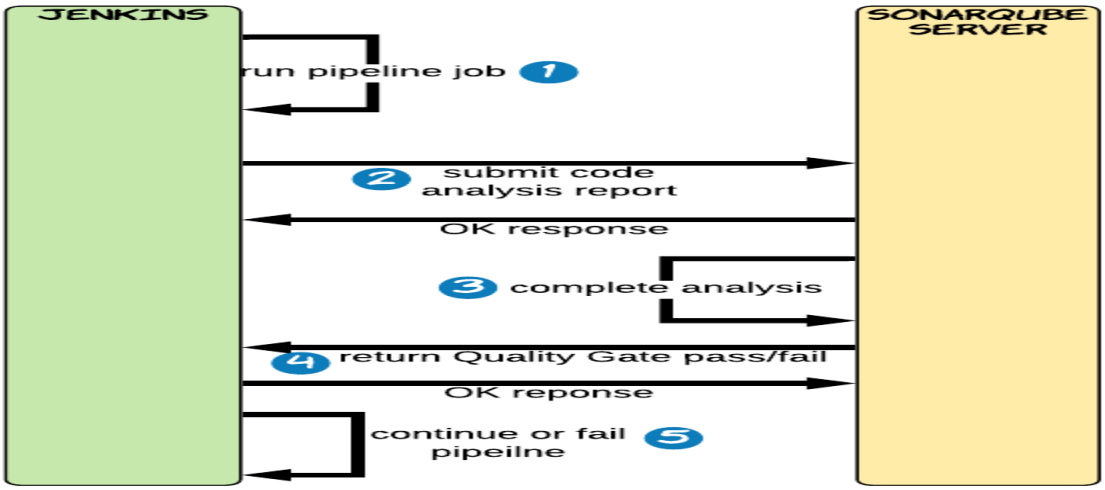
International Journal for Innovative Engineering and Management Research
A Peer Revieved Open Access International Journal
www.ijiemr.org

This is the successfully analysation of Maven Java application using SonarQube analysis in Jenkins CI pipeline.

VI: Quality gates

In SonarQube a quality gate [7]is a set of conditions that must undergo while evaluating a project. *Set the SonarQube Scanner plugin for Jenkins. It includes two features that we're going to make use of today:*

1. SonarQube server configuration – the plugin lets you set your SonarQube server location and credentials. This information is then used in a SonarQube analysis pipeline stage to send code analysis reports to that SonarQube server.
2. SonarQube Quality Gate webhook – when a code analysis report is submitted to SonarQube, unfortunately it doesn't respond synchronously with the result of whether the report passed the quality gate or not. To do this, a webhook call must be configured in SonarQube to call back into Jenkins to allow our pipeline to continue (or fail). The SonarQube Scanner Jenkins plugin makes this webhook available



Here's a full breakdown of the interaction between Jenkins and SonarQube:

1. a Jenkins pipeline is started
2. the SonarQube scanner is run against a code project, and the analysis report is sent to SonarQube server
3. SonarQube finishes analysis and checking the project meets the configured Quality Gate
4. SonarQube sends a pass or failure result back to the Jenkins webhook exposed by the plugin
5. the Jenkins pipeline will continue if the analysis result is a pass or optionally otherwise fail.

International Journal for Innovative Engineering and Management Research
A Peer Revieved Open Access International Journal
www.ijiemr.org

Adding a quality gate

SonarQube comes with its own *Sonar way* quality gate enabled by default. If you click on Quality Gates .Create a new Quality gate and give any quality gate name as A.Radhika. Click Save then on the next screen click Add Condition. Select On Overall Code. Search for the metric Maintainability Rating and choose worse than A. This means that if existing code is not maintainable then the quality gate will fail. Click Add Condition to save the condition.

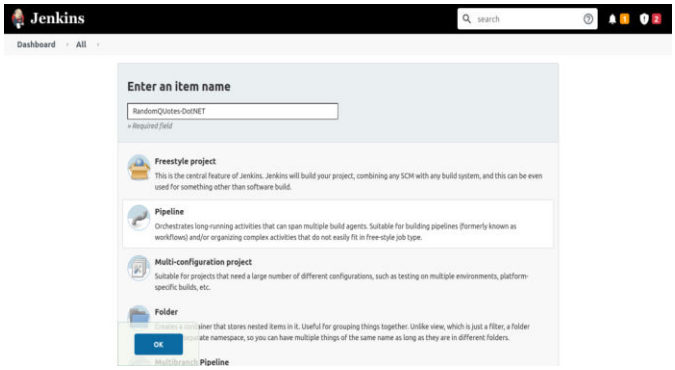
Finally click Set as Default at the top of the page to make sure that this quality gate will apply to any new code analysis.

Running Unit test cases

Install Junit plugin to process the result of JUnit tests.

To install the plugin:

1. Click Manage Jenkins, then Manage Plugins, then Available.
2. Enter junit in the search box.
3. Select the JUnit option, and click Install without restart:
4. To create a new pipeline project, click New Item, enter RandomQuotes-DotNET for the item name, select the Pipeline option, and click the OK button:



5.
6. Paste the following pipeline script into the Pipeline section, and click the Save button:

```
pipeline {
  // This pipeline requires the following plugins:
  // * Git: https://plugins.jenkins.io/git/
  // * Workflow Aggregator: https://plugins.jenkins.io/workflow-aggregator/
  // * MSTest: https://plugins.jenkins.io/mstest/
  agent 'any'
  stages {
```

International Journal for Innovative Engineering and Management Research
A Peer Revieved Open Access International Journal
www.ijiemr.org

```
14.    stage('Environment') {
15.      steps {
16.         echo "PATH = ${PATH}"
17.      }
18.    }
19.    stage('Checkout') {
20.      steps {
21.
22.        script {
```

```
30.        sh(script: 'dotnet restore')
31.      }
32.    }
33.    stage('Build') {
34.      steps {
35.        sh(script: 'dotnet build --configuration Release', returnStdout: true)
36.      }
37.    }
38.    stage('Test') {
39.      steps {
40.        sh(script: 'dotnet test -l:trx || true')
41.      }
42.    }
43.  }
44.  post {
45.    always {
```

```
23.          checkout([$class: 'GitSCM', branches: [[name: '*/master']], userRemoteConfigs: [[url: 'https://github.com/Random
24. amples/RandomQuotes.git']]])
25.        }
26.      }
27.    }
28.    stage('Dependencies') {
29.      steps {
46.      mstest(testResultsFile: '**/*.trx', failOnError: false, keepLongStdio: true)
47.    }
48.  }
49. }
```

50. The Test stage calls dotnet test to run the unit tests, passing the argument -l:trx to write the test results in a Visual Studio Test Results (TRX) file.
51. This command will return a non-zero exit code if any tests failed. To ensure the pipeline continues to be processed in the event of a failed test, you return true if dotnet test indicates a failure:

```
52. sh(script: 'dotnet test -l:trx || true')
```

CONCLUSION

Implementing CI/CD pipeline using jenkins improves the application development process Significantly. To improve the productivity of the system as the release cycle was found to be shorter. The Goals of the CI, which is to provide higher quality code by running Unit test cases against the code and fixing bugs, issues at the pre-release itself which enables ability to release the product within earlier stages of time which in turns compete in the market place. In the Continuous Integration process, we were able to identify the failure pattern and finding the quality of the product by implementing quality gates. Since the release are small they are at low risk which in turn decrease the cognitive load. If any bug is found in the post deployment , the fixes can be provided to the customer within in lesser time. Continuous Delivery can help overcome this barrier to quick deployments. Error rates and infrastructure costs can be quickly and easily measured once the CICD is implemented. in Continuous Integration process gives us the irregular trends found while quality testing which is useful when exceptions and errors

in an application is critical, using these, the errors can be easily fixed.

REFERENCES

[1] Mphasis Stelligent, Continuous Security in Continuous Delivery Pipeline. Available at:https://stelligent.com/2016/04/05/continuous-security/ (2016).

[2] Continuous Integration, Continuous Testing, and Continuous Delivery Mitesh Soni IGATE Gandhinagar, Indiamitesh.soni@igate.com

[3]https://towardsdatascience.com/create-your-first-ci-cd-pipeline-with-jenkins-and-github-6aefe21c9240

[4] https://www.jenkins.io/doc/pipeline/steps/sonar/

[5]https://www.lambdatest.com/blog/reporting-code-coverage-using-maven-and-jacoco-plugin/

[6]https://www.opensourceforu.com/2021/08/static-code-analysis-using-sonarqube-and-jenkins/

[7] https://medium.com/automationmaster/sonarqube-c7df46614012