Integrating sonarqube with Jenkins

1. Install required plugin **Manage Jenkins 🡪 Manage Plugins**

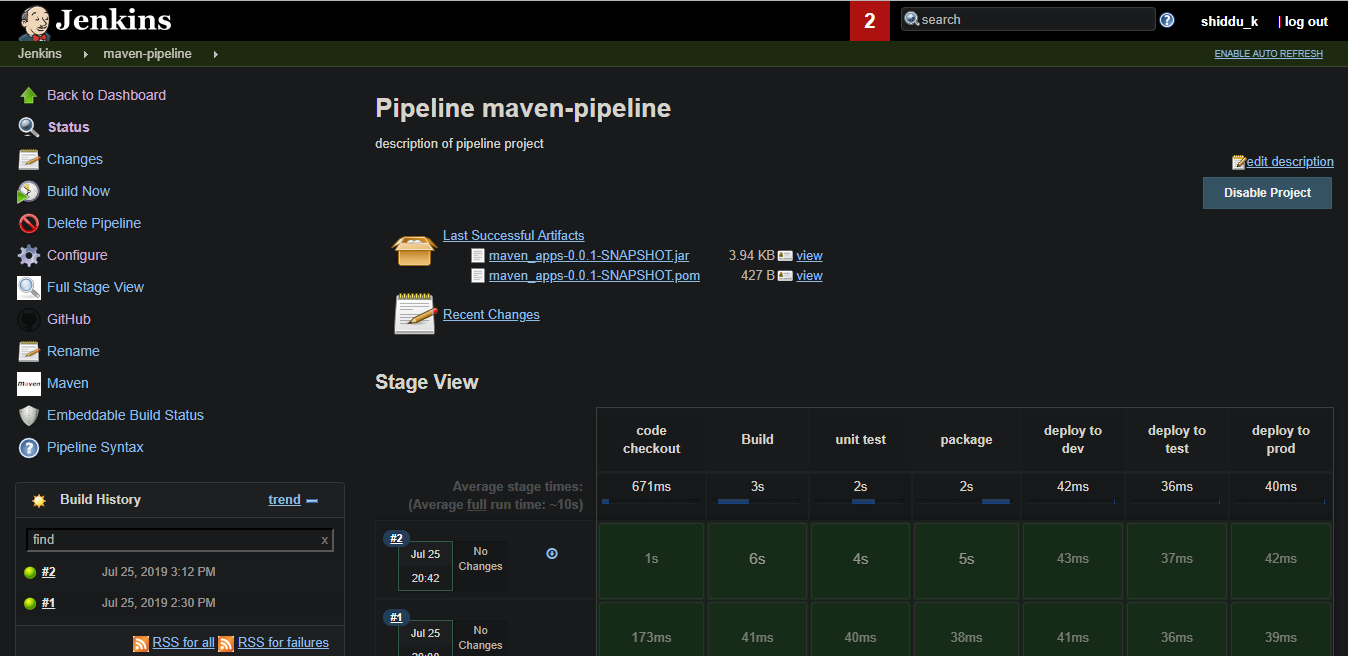
* SonarQube Scanner for Jenkins
* Jacoco
* TaskScanner plugin
* quality gates
* Junit Attachment

1. Create a pipeline project in Jenkins before checkingout the code from github add “Jenkinsfile” in the repository.

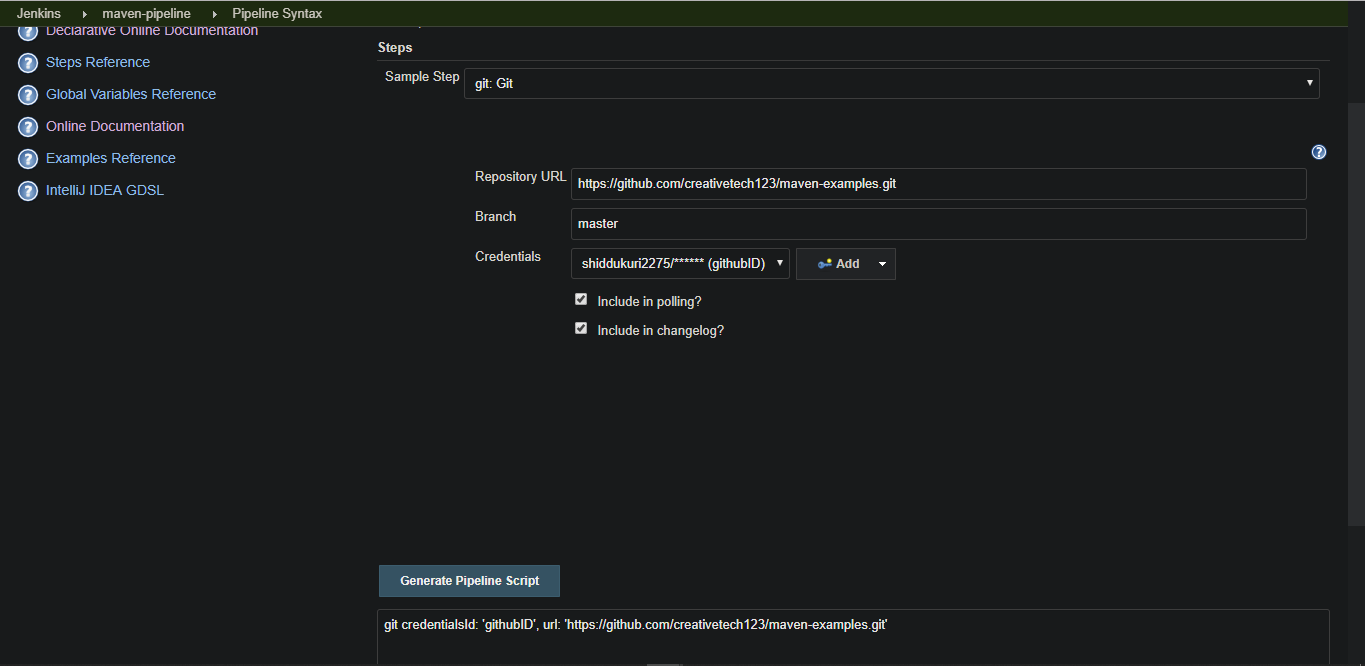
Create the Jenkinsfile using below mentioned stage.

|  |
| --- |
| node { |
|  |  |
|  | stage('code checkout') { |
|  |  |
|  | git credentialsId: 'githubID', url: 'https://github.com/creativetech123/maven-examples.git' |
|  |  |
|  | } |
|  |  |
|  | stage('Build') { |
|  | withMaven(jdk: 'JDK', maven: 'maven') { |
|  | sh 'mvn clean compile' |
|  | } |
|  |  |
|  |  |
|  | } |
|  |  |
|  | stage('unit test') { |
|  | withMaven(jdk: 'JDK', maven: 'maven') { |
|  | sh 'mvn test' |
|  | } |
|  |  |
|  | } |
|  | stage('sonarqube analysis'){ |
|  | withSonarQubeEnv(credentialsId: 'sonarqubeid') { |
|  | withMaven(jdk: 'JDK', maven: 'maven') { |
|  | sh 'mvn sonar:sonar' |
|  | } |
|  | } |
|  | } |
|  |  |
|  | stage("Quality Gate"){ |
|  | timeout(time: 1, unit: 'HOURS') { |
|  | def qg = waitForQualityGate() |
|  | if (qg.status != 'OK') { |
|  | error "Pipeline aborted due to quality gate failure: ${qg.status}" |
|  | } |
|  | } |
|  | } |
|  |  |
|  |  |
|  | stage('package') { |
|  | withMaven(jdk: 'JDK', maven: 'maven') { |
|  | sh 'mvn package' |
|  | } |
|  |  |
|  | } |
|  |  |
|  | stage('deploy to dev') { |
|  |  |
|  | } |
|  |  |
|  | stage('deploy to test') { |
|  |  |
|  | } |
|  |  |
|  | stage('deploy to prod') { |
|  |  |
|  | } |
|  | } |
|  |  |

**First stage** of the code is **“code checkout”** you can generate the code with help of pipeline syntax available above the Build History.

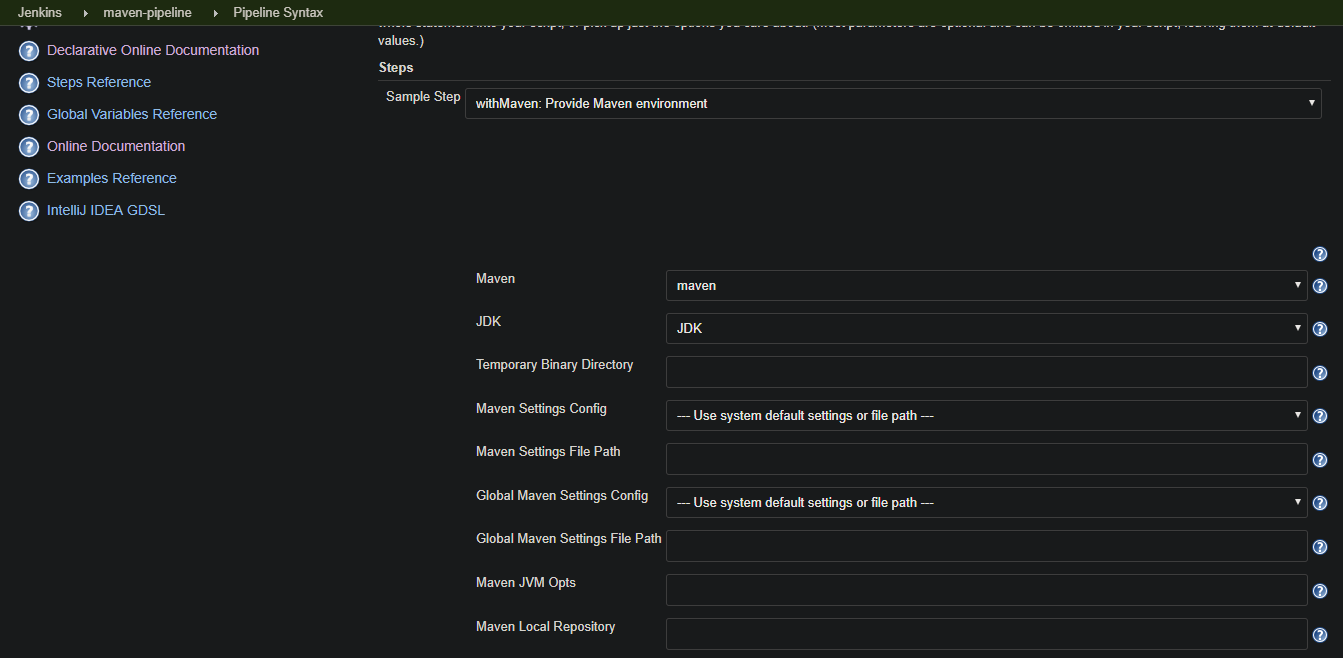


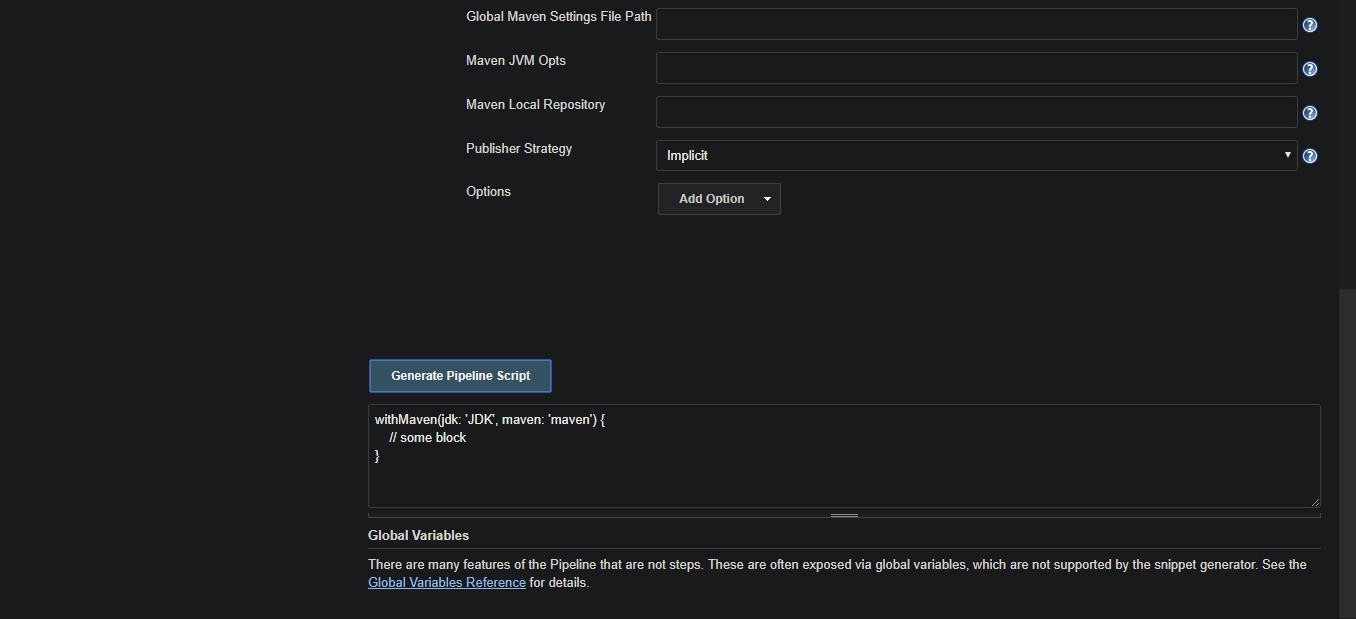
Select “**git: Git**” in sample steps and add the git hub Repo URL and credentials and click on generate pipeline script, it will generate the script of github copy and paste the script in the first stage of Jenkinsfile.



**Second Stage** is the **“build stage”** in this stage you should set the Environment Maven and Java you can generate the script same as above step by selecting **“withMaven: Provide Maven environment”** in Sample step and selecting the Maven and JDK versions set in your tool and click on generate the script of Environment copy and paste the script in the second stage of Jenkinsfile and give the shell command what script it has to be execute like

sh ‘mvn clean compile’





**Third stage** is unit test in this stage code is tested which uses JUnit.

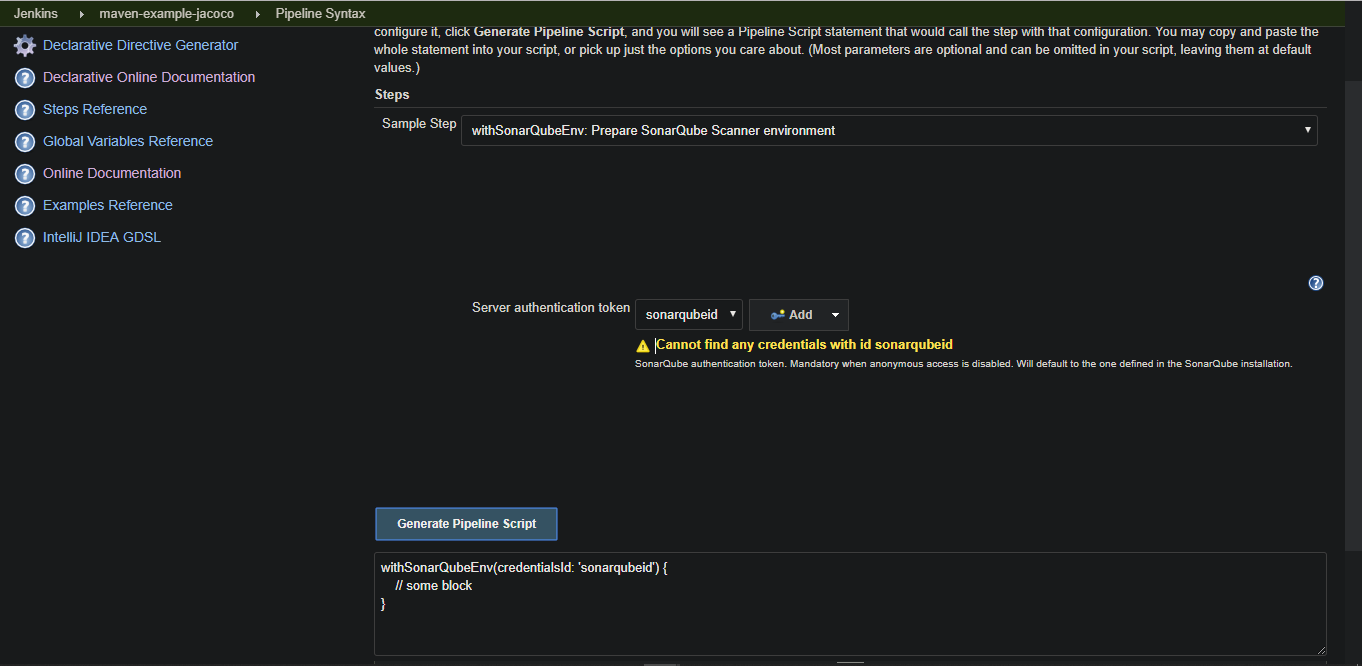
**Fourth stage** is integrating the sonarqube with Jenkins using credentials you can generate the script same as above step by selecting **“withSonarQubeEnv: Prepare SonarQube Scanner environment”** in Sample step and selecting the server authentication token of Sonarqube and

click on generate the script of Environment copy and paste the script in the Fourth stage of Jenkinsfile and give the shell command what script it has to execute.

To know detail about SonarQube please visit below link

<https://www.baeldung.com/sonar-qube>

<https://docs.sonarqube.org/latest/analysis/scan/sonarscanner-for-jenkins/>



If you get  this error please go to your sonarcloud account 🡪organization🡪project🡪click on administration🡪permissions

And change the permission of the user.

**Fifth stage** Quality gate analysis The waitForQualityGate step will pause the pipeline until SonarQube analysis is completed and returns quality gate status. A Quality Gate is a set of conditions the project must meet before it can qualify for production release.

For this you should create the webhook for sonarqube by visiting sonarcloude🡪 your organization🡪Administration 🡪webhook

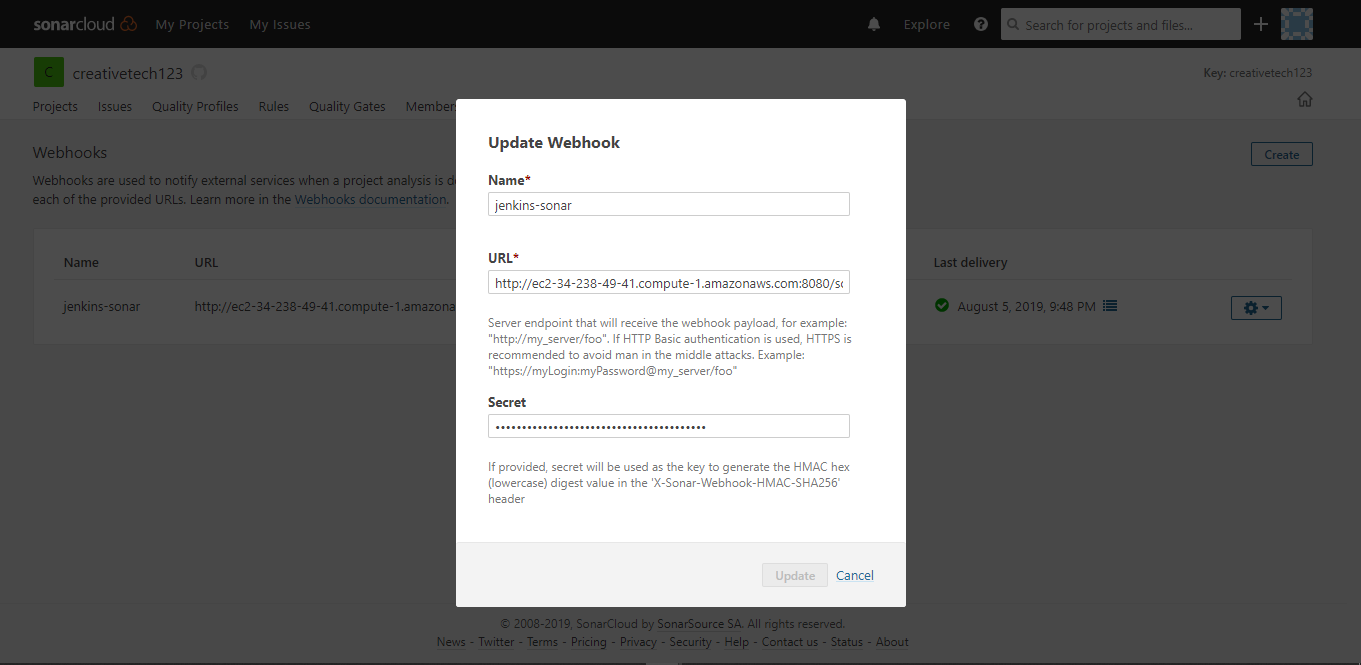
Click on create, give an appropriate name and your Jenkins URL enter secrete key of sonarqube and click on create.

Jenkins url eg. <http://ec2-34-238-49-41.compute-1.amazonaws.com:8080/sonarqube-webhook/>

To know more about quality gate visit

<https://www.baeldung.com/sonar-qube>

<https://medium.com/@tarunprakash/quality-gates-a-must-have-thing-for-the-code-analysis-process-75b33d6b49dc>



**Sixth stage** is package the output of the package stage is war, jar, zip files that are Stored in the Artifactory repository like JFrog Artifactory and from there war, jar, zip files are deploy in the server.

Rest all stages are related to deployment to staging, testing, production environment.

# JaCoCo code coverage

**Code coverage** is a software metric used to measure how many lines of our code are executed during automated tests.

You can know more about the jacoco visit the below link

<https://www.baeldung.com/jacoco>

jacoco plugin used in maven project which is written in the pom.xml is

|  |
| --- |
| plugin> |
|  | <groupId>org.jacoco</groupId> |
|  | <artifactId>jacoco-maven-plugin</artifactId> |
|  | <version>${jacoco.version}</version> |
|  | <executions> |
|  | <execution> |
|  | <goals> |
|  | <goal>prepare-agent</goal> |
|  | </goals> |
|  | </execution> |
|  | <execution> |
|  | <id>jacoco-report</id> |
|  | <phase>test</phase> |
|  | <goals> |
|  | <goal>report</goal> |
|  | </goals> |
|  | </execution> |
|  | <execution> |
|  | <id>jacoco-check</id> |
|  | <goals> |
|  | <goal>check</goal> |
|  | </goals> |
|  | <configuration> |
|  | <rules> |
|  | <rule> |
|  | <element>PACKAGE</element> |
|  | <limits> |
|  | <limit> |
|  | <counter>LINE</counter> |
|  | <value>COVEREDRATIO</value> |
|  | <minimum>0.9</minimum> |
|  | </limit> |
|  | </limits> |
|  | </rule> |
|  | </rules> |
|  | </configuration> |
|  | </execution> |
|  | </executions> |
|  | </plugin> |
|  |  |
|  | </plugins> |
|  | </build> |
|  |  |
|  | </project> |

Other Junit and code coverage plugins code is

|  |
| --- |
| <dependency> |
|  | <groupId>org.junit.jupiter</groupId> |
|  | <artifactId>junit-jupiter-engine</artifactId> |
|  | <version>${junit.version}</version> |
|  | <scope>test</scope> |
|  | </dependency> |
|  |  |
|  | </dependencies> |
|  | <build> |
|  | <finalName>maven-code-coverage</finalName> |
|  | <plugins> |
|  | <plugin> |
|  | <groupId>org.apache.maven.plugins</groupId> |
|  | <artifactId>maven-surefire-plugin</artifactId> |
|  | <version>3.0.0-M1</version> |
|  | </plugin> |

**embeddable-build-status-plugin**

You can also add the embeddable bui ld status plugin in github in the README.md file

This plugin allows to add customizable [shields.io](https://shields.io/) like badges to any website.

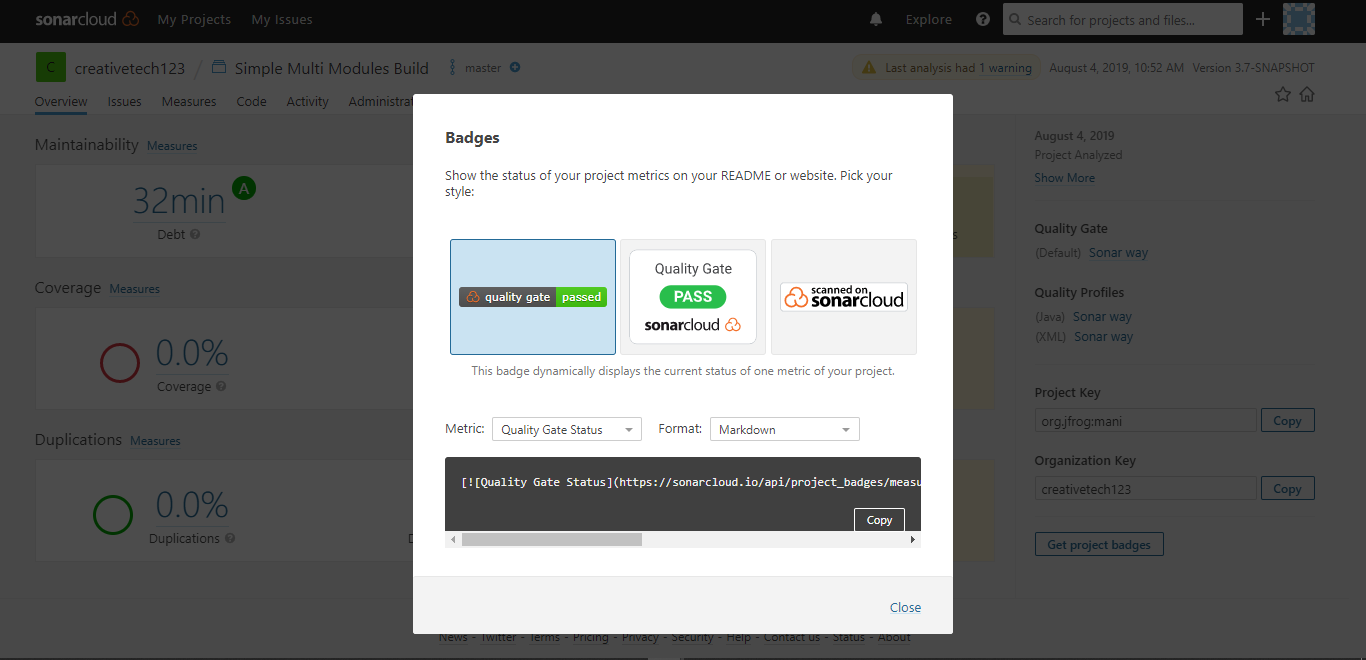
For eg:

[![BuildStatus]([http://ec2-34-238-49-41.compute-1.amazonaws.com:8080/buildStatus/icon?job=maven-example-jacoco)](http://ec2-34-238-49-41.compute-1.amazonaws.com:8080/job/maven-example-jacoco/)](http://ec2-34-238-49-41.compute-1.amazonaws.com:8080/buildStatus/icon?job=maven-example-jacoco)%5d(http://ec2-34-238-49-41.compute-1.amazonaws.com:8080/job/maven-example-jacoco/))

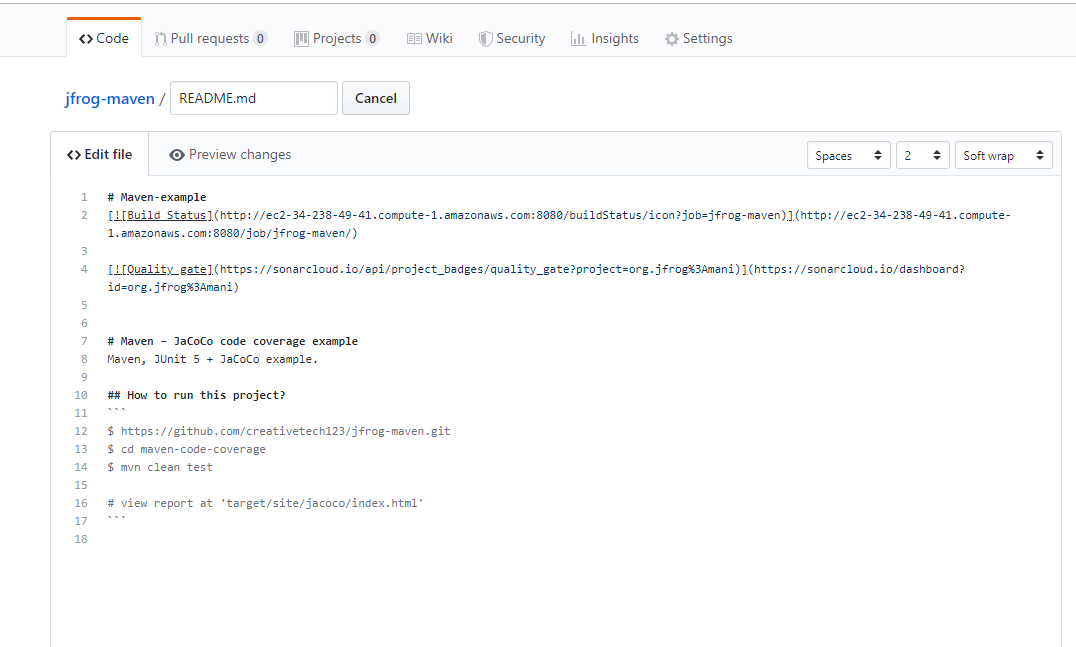
Add the following line in your README.md file with your Jenkins URL and project name.

And you can also add project badges from your sonarcloud server.

Got to your project – >overview – on the right bottem corner you will see a button **get project badges** click on that



Select the badge and copy the given link and paste it in the README.md file.



After adding code and committing the code it looks like below image

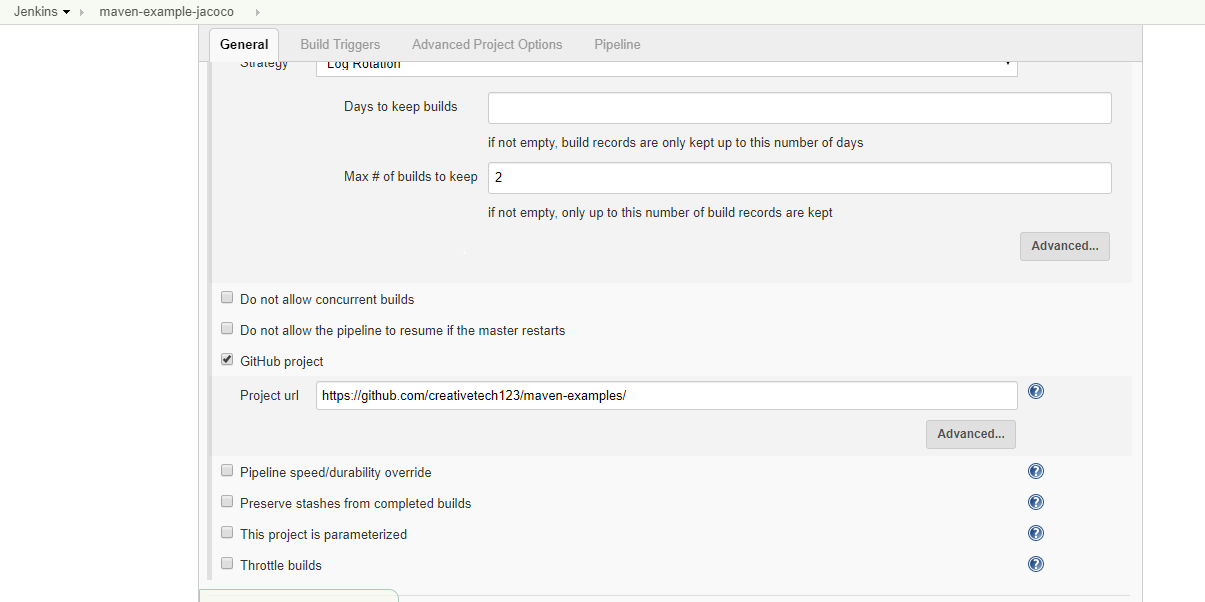


You can know more about embeddable-build-status-plugin by clicking on below link

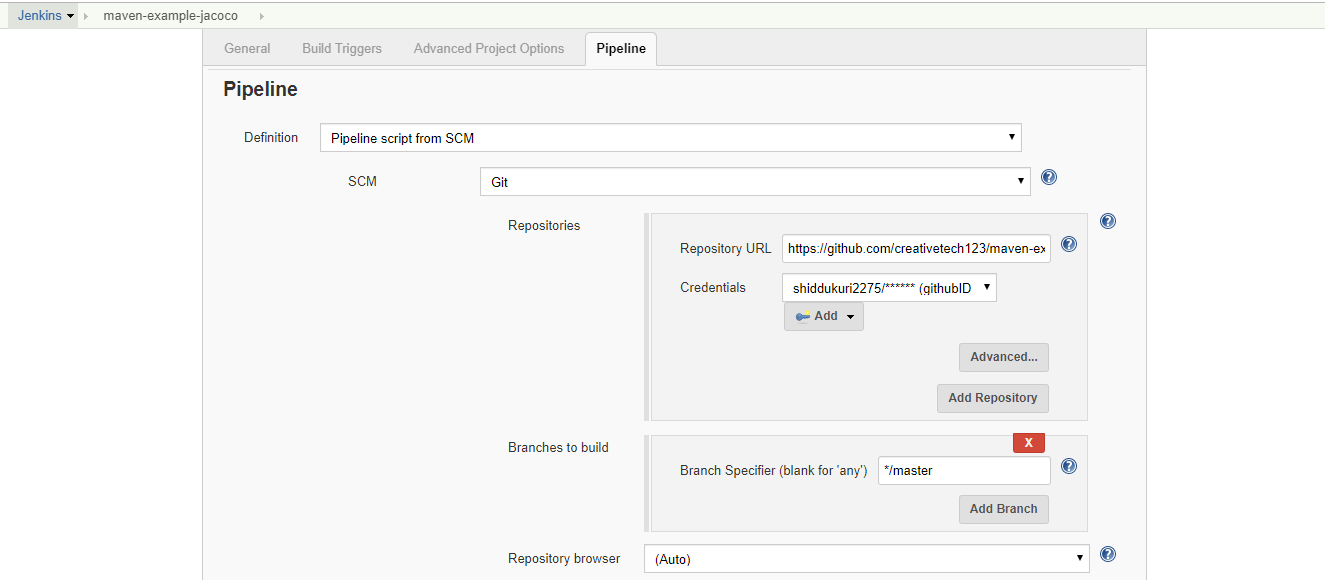
<https://github.com/jenkinsci/embeddable-build-status-plugin>

Save the Jenkinsfile and commit in the github repository

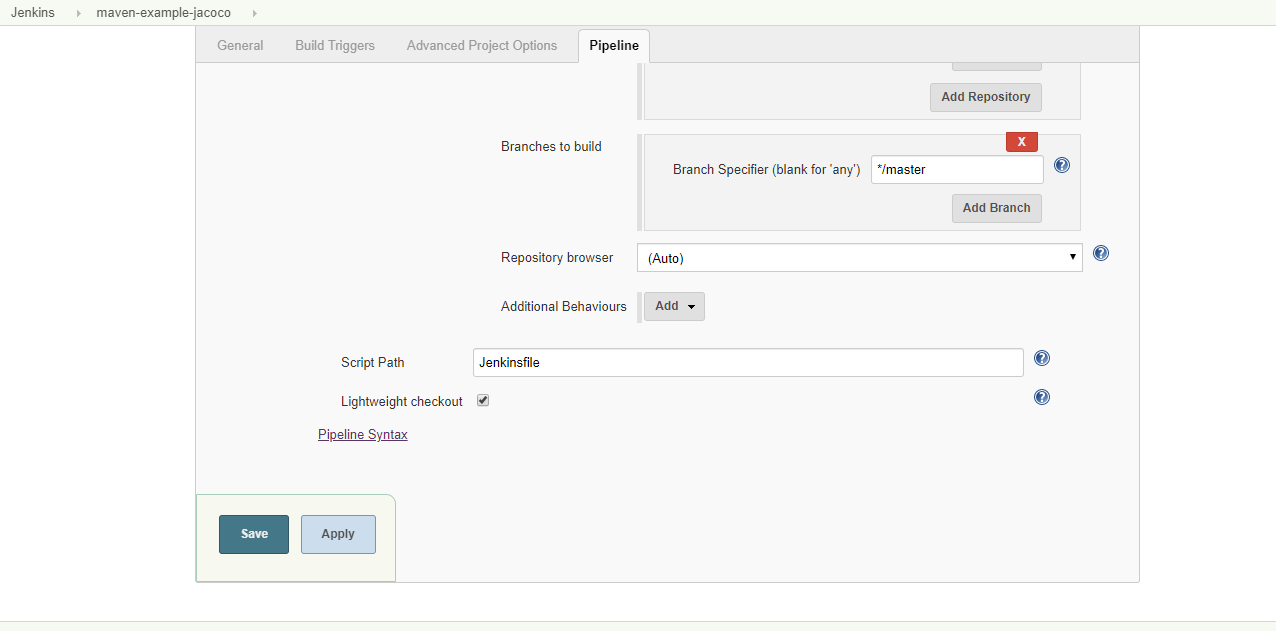
1. Create the maven pipeline job.
2. Give the description of job.
3. Select as github job and give appropriate github project url.



1. In pipeline stage select definition as pipeline script from SCM
2. Select SCM as git and give the correct repository URL and credentials.

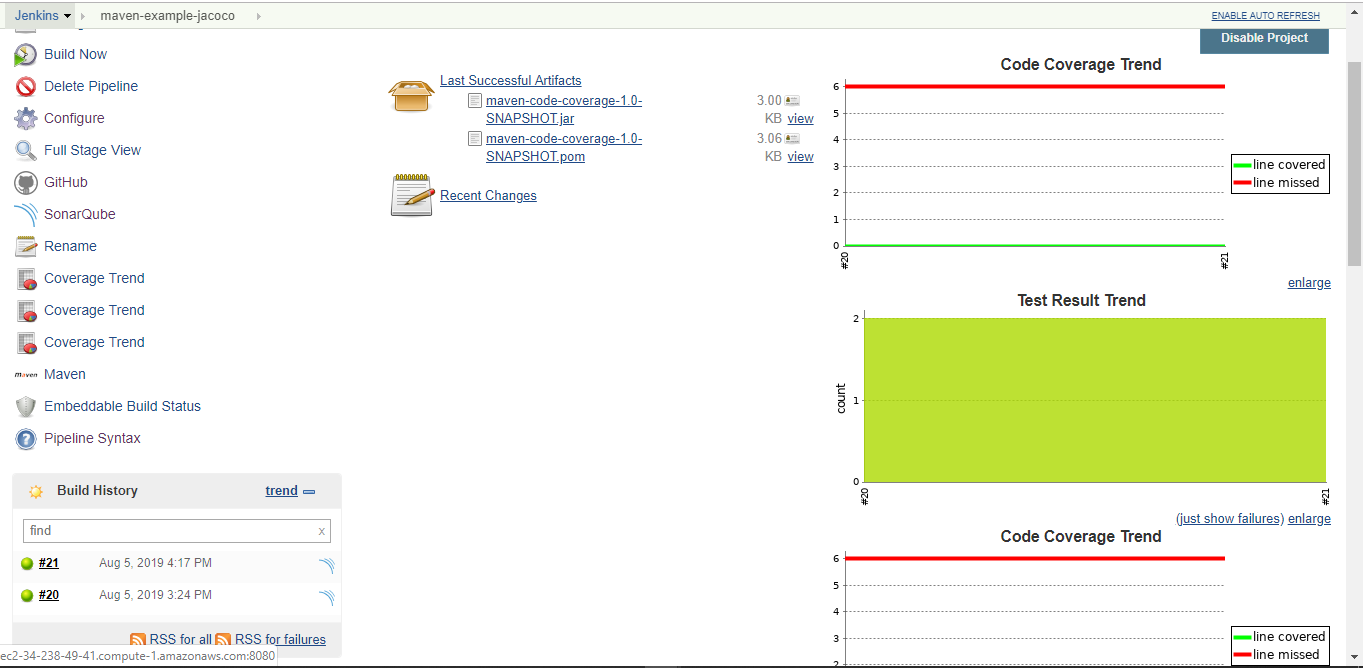


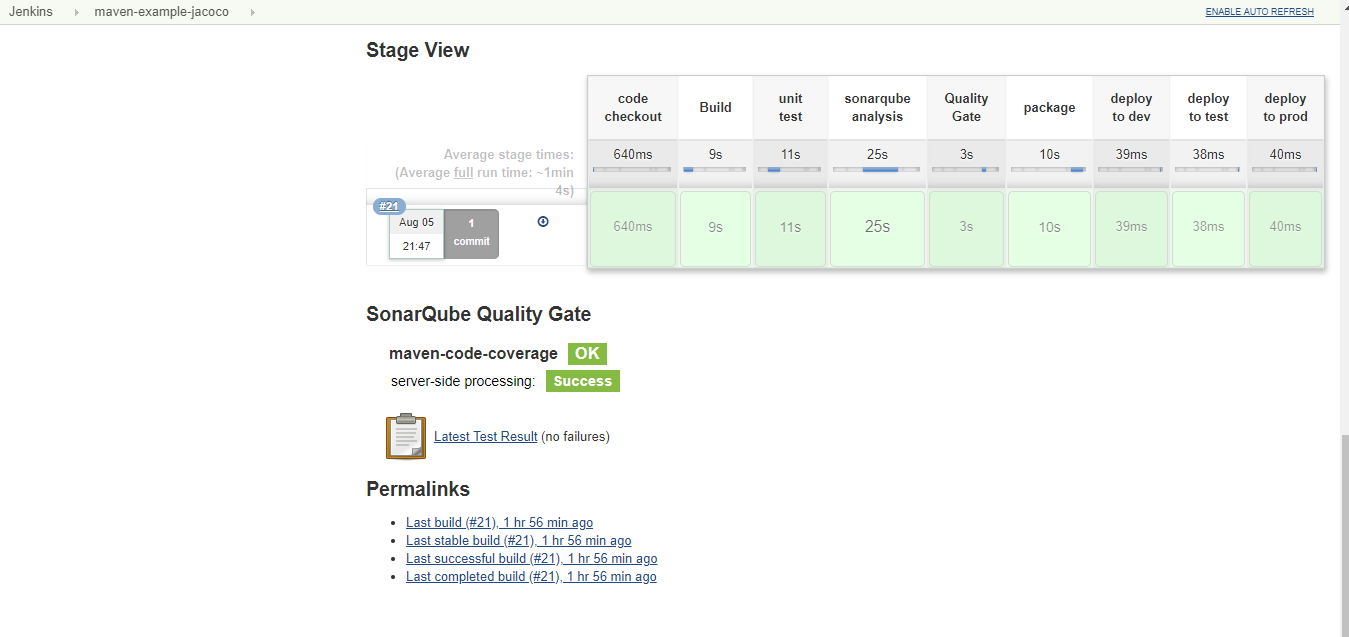
1. Give Script path as Jenkinsfile.



1. Apply and Save
2. Build the project

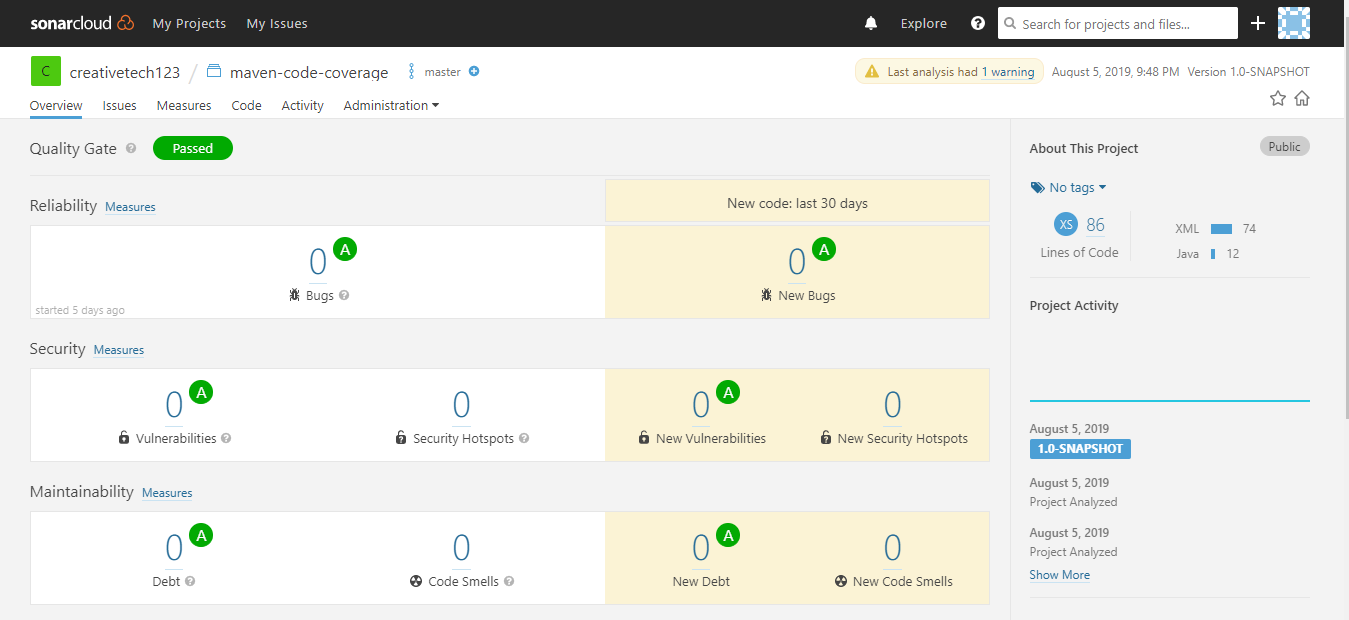
After build complete your pipeline looks like

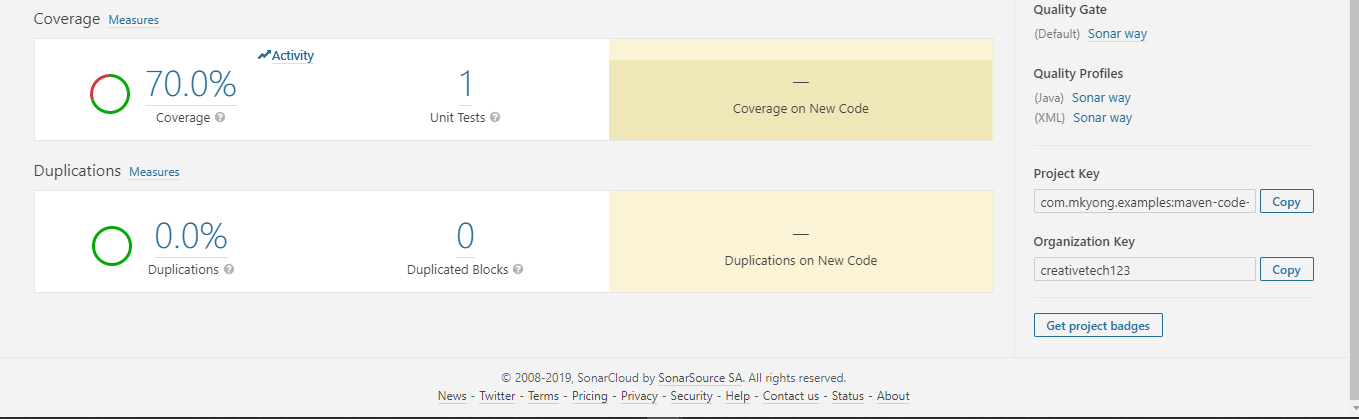




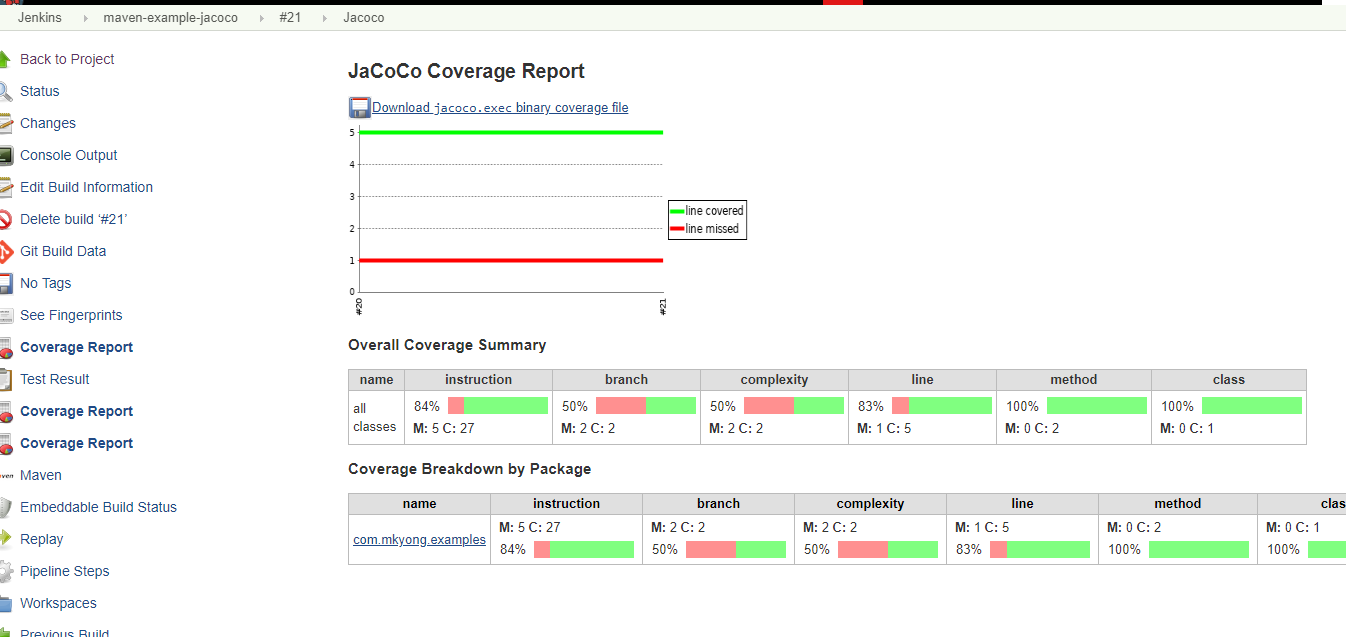
After clicking on OK or 3 lines at the corner it will redirect to sonarcloud server and you will find all the information of project like

Reliability, Security, Maintainability, Coverage, Duplications of code.





When you click on Jacoco code coverage report it gives overall coverage summery and looks like



And the build status plugin will display the status of the build whether success or failure

