Nagumi Yemarde
70
D15B

PAGE NO:
DATE / /

EXPERIMENT-8

AIM: Create a Jenkins CICD Pipeline with SonarQube / GitLab Integration to perform a static analysis of code to detect bugs, code smells, and security vulnerabilities on a sample code/Java/Python application.

THEORY:

Static application security testing or static analysis, is a testing methodology that analyzes source code to find security vulnerabilities that make your organizations applications susceptible to attack. SAST scans an application before the code is compiled. It's also known as white box testing.

SAST tools give developers real time feedback as they code, helping them fix issues before they pass the code to next phase of SDLC.

SAST tools automatically identify critical vulnerabilities - such as buffer overflows, SQL injection, cross site scripting and others - with high confidence. Thus integrating static analysis into SDLC can yield dramatic results in overall quality of code developed.

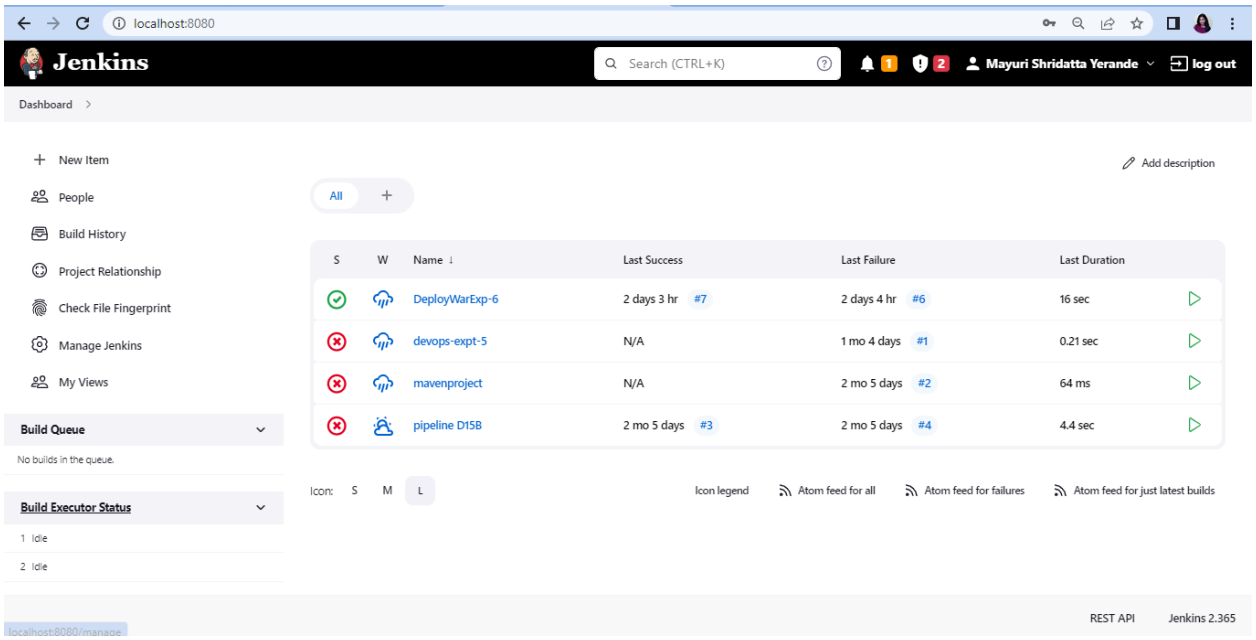
There are six simple steps needed to perform SAST efficiently in organizations that have a very large number of applications built with different languages, frameworks and platforms.

1. Finalize the tool
2. creating scanning interface and deploy the tool
3. customize the tool
4. Prioritize and onboard applications
5. analyze scan results
6. provide governance and training

It is important to note that SAST tools must be run on application on regular basis. such as daily / monthly builds, everytime code is checked in or during a code release.

IMPLEMENTATION:

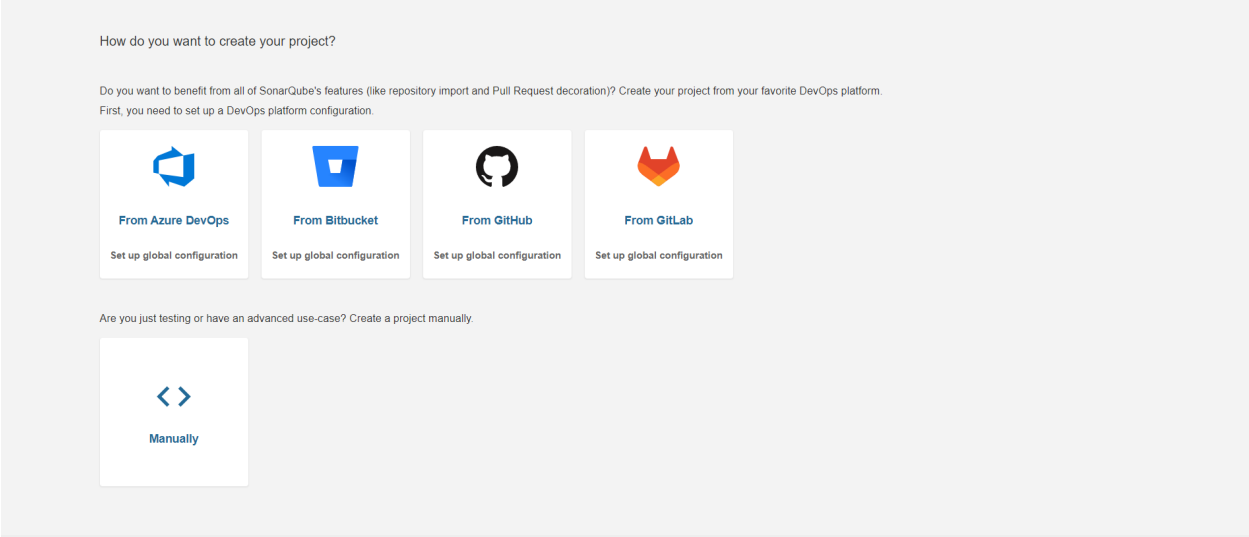
Step 1: Open up Jenkins Dashboard on localhost, port 8080 or whichever port it is at for you.



Step 2: Run SonarQube in a Docker container using this command -
docker run -d --name sonarqube -e SONAR_ES_BOOTSTRAP_CHECKS_DISABLE=true -p 9000:9000 sonar:latest

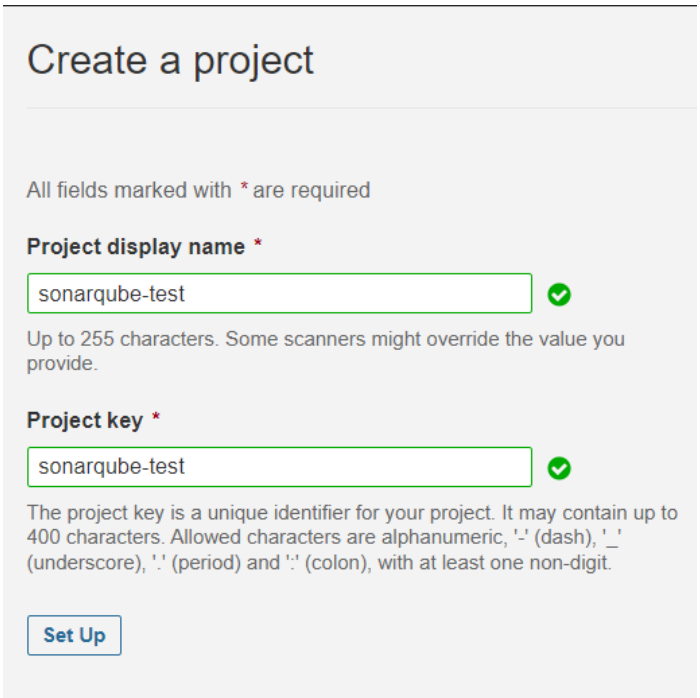
```
C:\Users\mayuri>docker run -d --name sonarqube -e SONAR_ES_BOOTSTRAP_CHECKS_DISABLE=true -p 9000:9000 sonar:latest

Unable to find image 'sonarqube:latest' locally
latest: Pulling from library/sonarqube
9621f1afde84: Pull complete
0da9106727c7: Pull complete
129c5a3f9c32: Pull complete
Digest: sha256:3fa9a76948fab6fafa41950bee256afea943773744723b5e4f38b340643516b9
Status: Downloaded newer image for sonarqube:latest
895ac0b29a9a624f8a4828b0cce1649d1ab0d32ea65874e34a24eff068a9bc93
```

Step 3: Once the container is up and running, you can check the status of SonarQube at localhost port 9000.

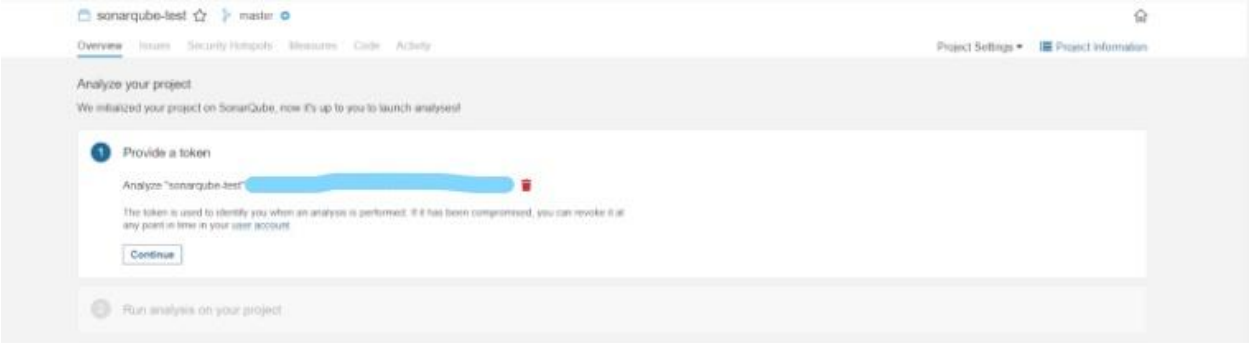


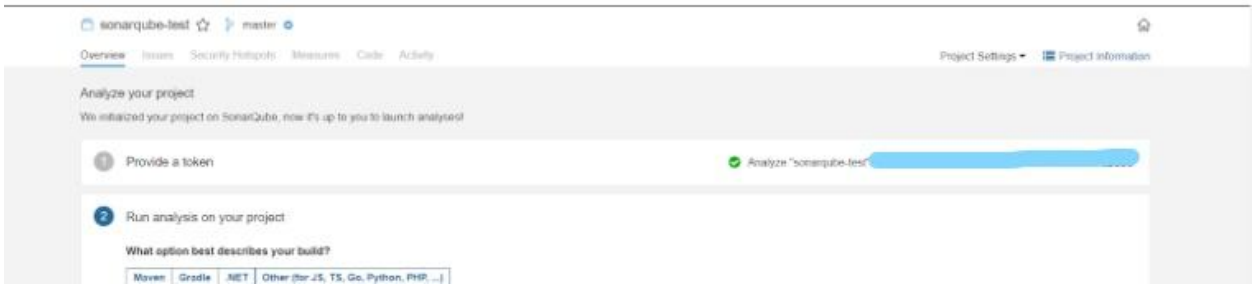
- Login to SonarQube using username admin and password admin.

Step 4: Create a manual project in SonarQube with the name sonarqube-test



Step 5: Set a token name and generate it.
(Copy the token for the integration with jenkins)





Step 6: Setup the project and come back to Jenkins Dashboard.

- Add Credentials of SonarQube Project in Jenkins for your current user.

- Go to Global Credentials

Global credentials (unrestricted)

Credentials that should be available irrespective of domain specification to requirements matching.

ID	Name ↑	Kind	Description
This credential domain is empty. How about adding some credentials?			

Step 7: Add SonarQube Project Token

New credentials

Kind

Secret text

Scope ?

Global (Jenkins, nodes, items, all child items, etc)

Secret

••

ID ?

sonarqube-token

Description ?

sonarqube-token

Create

- As you can see the credentials have been added for the user.

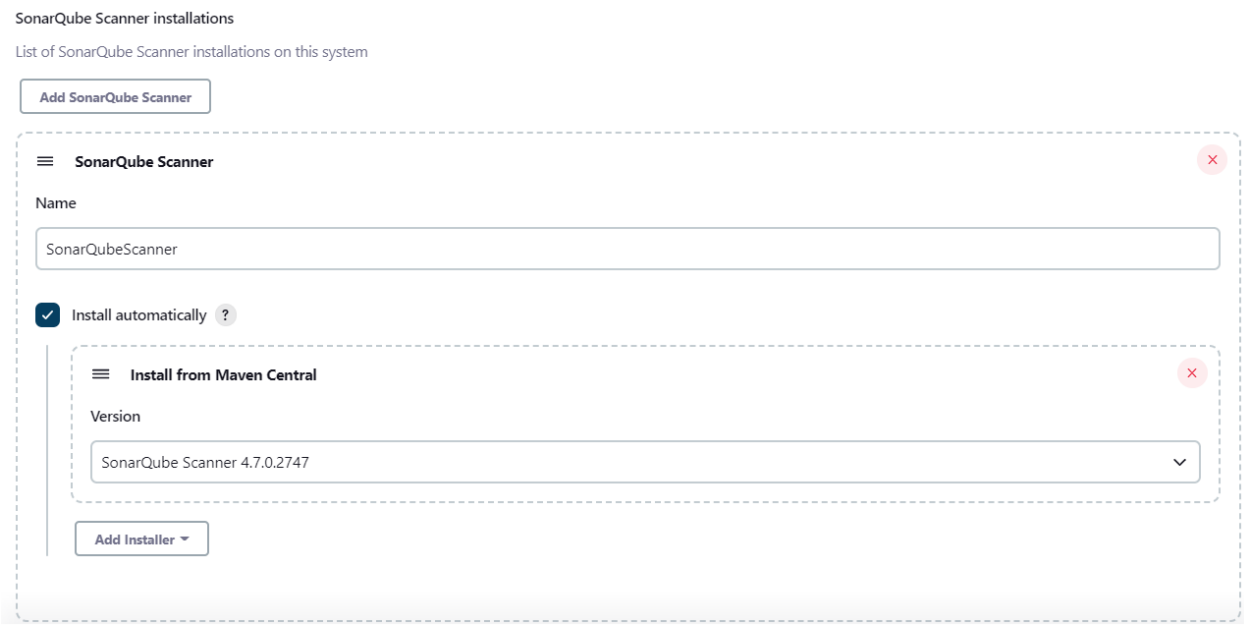
Global credentials (unrestricted)

Credentials that should be available irrespective of domain specification to requirements matching.

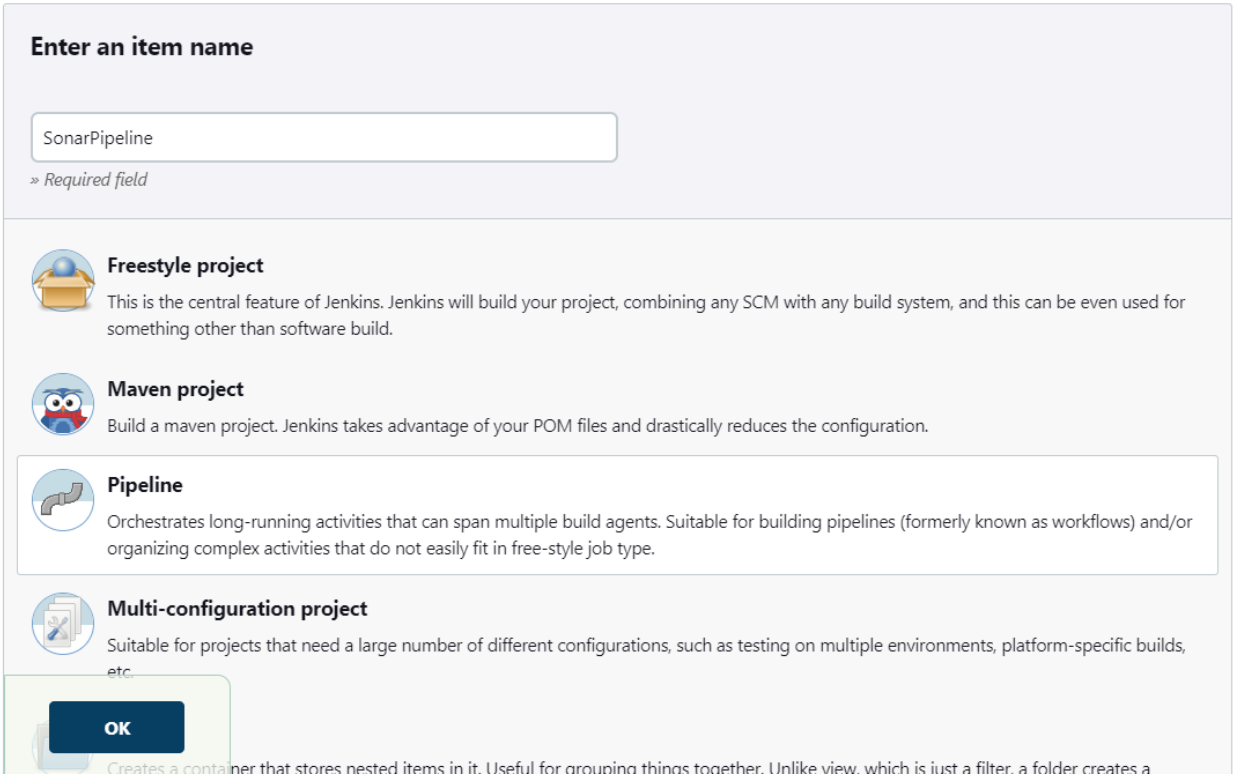
ID	Name ↑	Kind	Description
sonarqube-token	sonarqube-token	Secret text	sonarqube-token

Step 8: Under configuration system, tab mention the SonarQube installations

Step 9: Add SonarQube Scanner installation in Global Tool Configuration



Step 10: Create a New Item in Jenkins, choose Pipeline.



Step 11: Under Pipeline Script, enter the following -

```
node {
  stage('Cloning the GitHub Repo') {
    git 'https://github.com/shazforiot/GOL.git'
  }
  stage('SonarQube analysis') {
    withSonarQubeEnv('sonarqube') {
      sh
"/c/ProgramData//Jenkins/.jenkins/tools/hudson.plugins.sonar.Son
arRunnerInstallation/sonarqube/bin//sonar-scanner \
      -D sonar.login=admin \
      -D sonar.password=yourpassword \
      -D sonar.projectKey=sonarqube-test \
      -D sonar.exclusions=vendor/**,resources/**,**/*.java \
      -D sonar.host.url=http://127.0.0.1:9000/"
    }
  }
}
```

Pipeline

Definition

Pipeline script

Script ?

```
node {
    stage('Cloning the GitHub Repo') {
        git 'https://github.com/shazforiot/GOL.git'
    }
    stage('SonarQube analysis') {
        withSonarQubeEnv('sonarqube') {
            sh "/c/ProgramData//Jenkins/.jenkins/tools/hudson.plugins.sonar.SonarRunnerInstallation/sonarqube/bin//sonar-scanner \
            -D sonar.login=admin \
            -D sonar.password=[redacted]\
            -D sonar.projectKey=sonarqube-test \
            -D sonar.exclusions=vendor/**,resources/**,**/*.java \
            -D sonar.host.url=http://127.0.0.1:9000/"
        }
    }
}
```

- It is a java sample project which has a lot of repetitions and issues that will be detected by SonarQube.

Step 12: Run The Build.

Dashboard > SonarPipeline >

- Status
- Changes
- Build Now
- Configure
- Delete Pipeline
- Full Stage View
- SonarQube
- Rename
- Pipeline Syntax

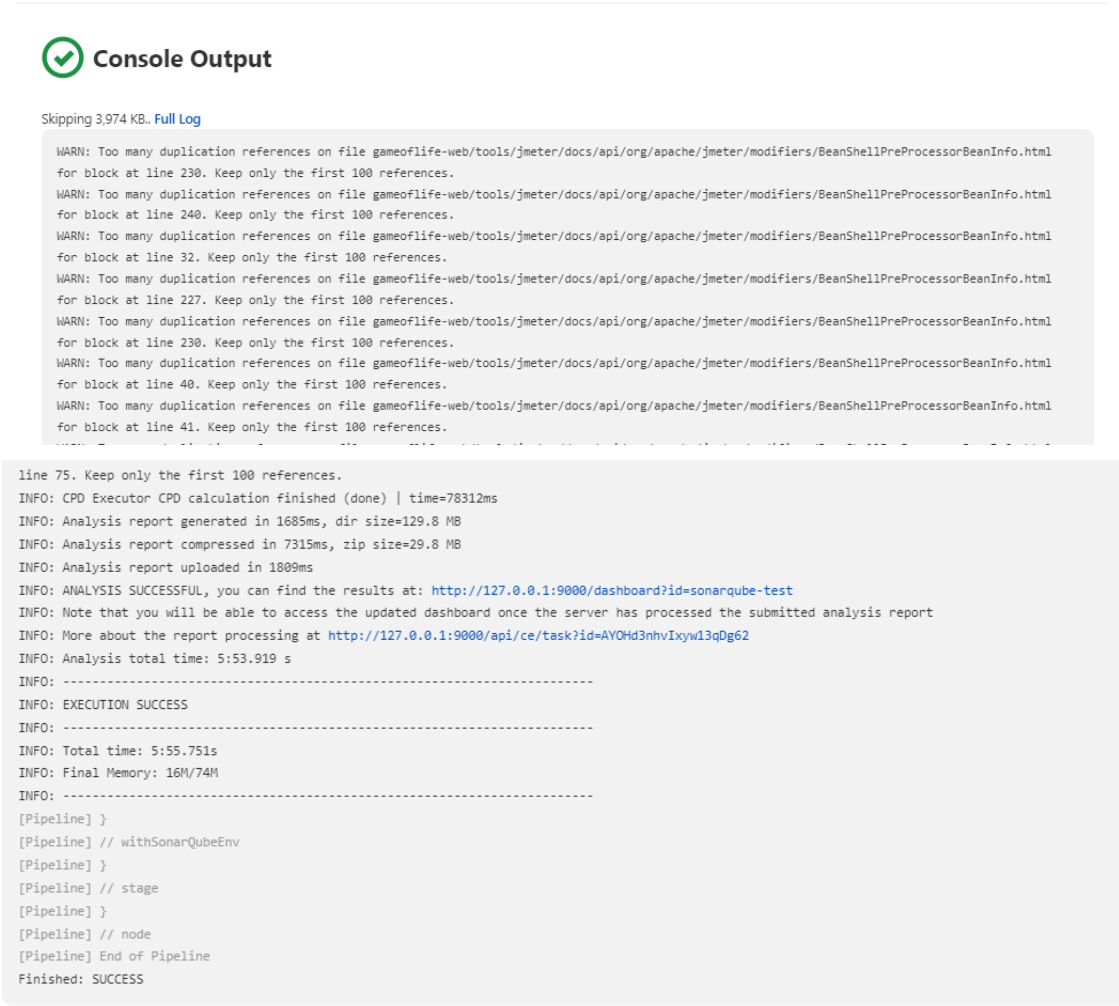
Build History trend

Recent Changes

Stage View

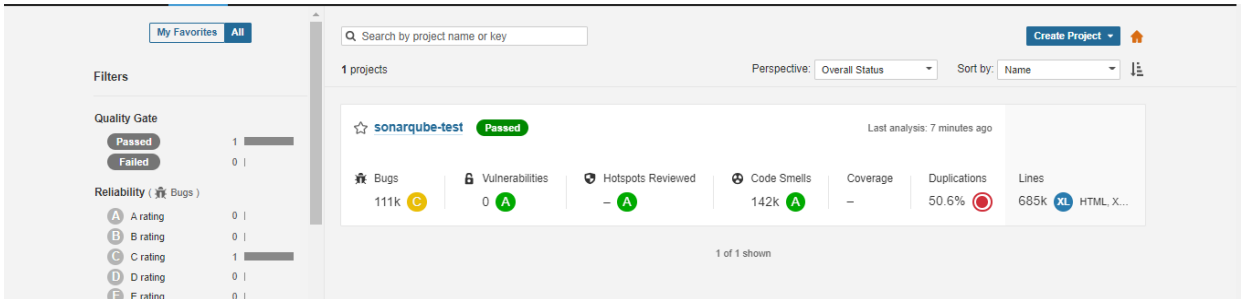
	Cloning the GitHub Repo	SonarQube analysis
Average stage times: (Average full run time: ~5min 59s)	50s	3min 14s
#4 Sep 29 09:42 No Changes	1s	5min 57s

Step 13: Check the console output once the build is complete.

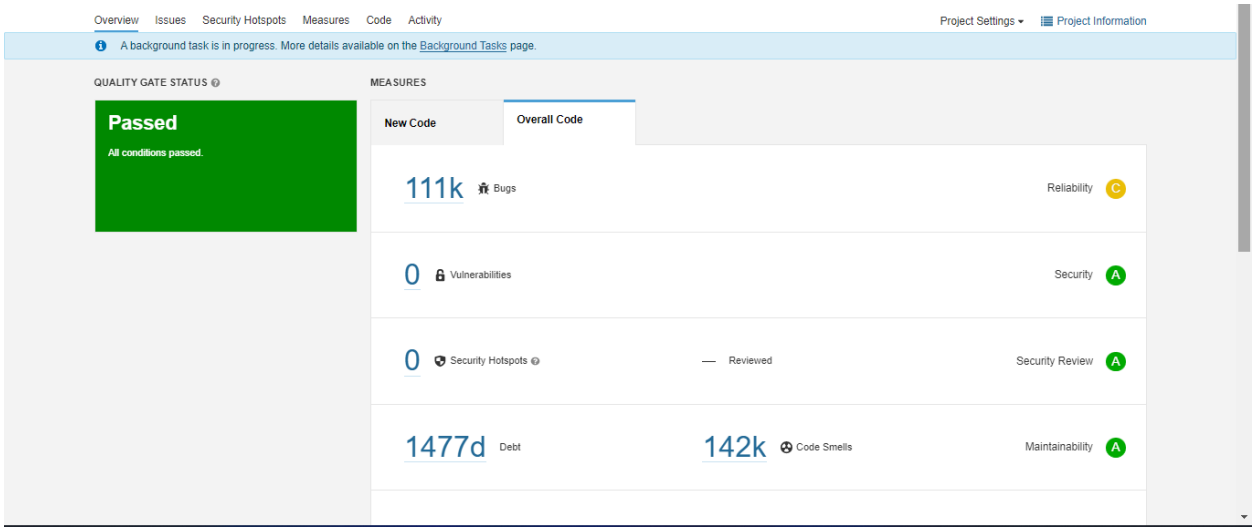


Step 14: After that, check the project in SonarQube.

- Wait till the SAST process gets completed on SonarQube.

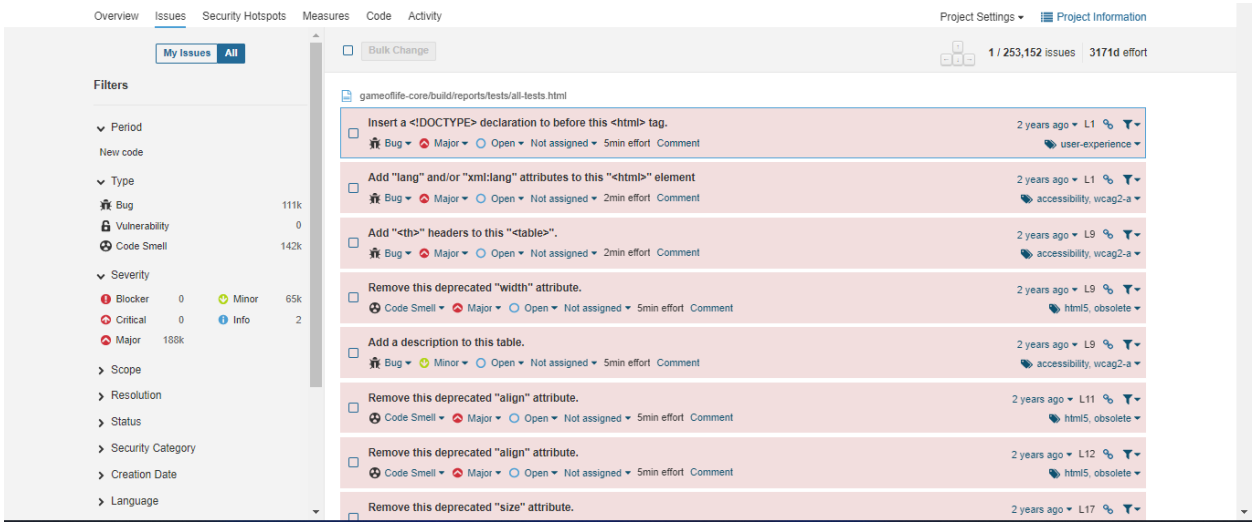


- Under different tabs, check all different issues with the code.

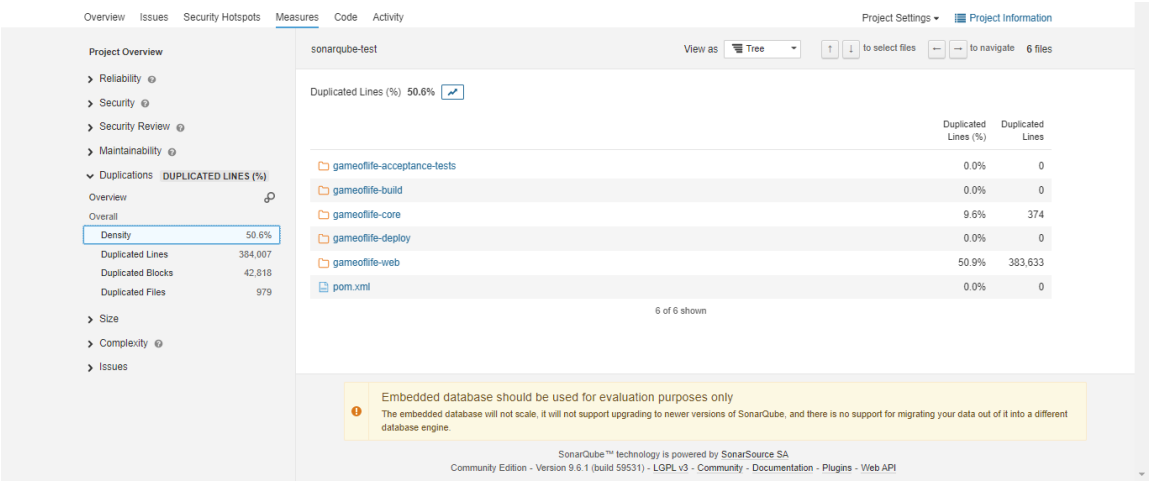


Code Problems -

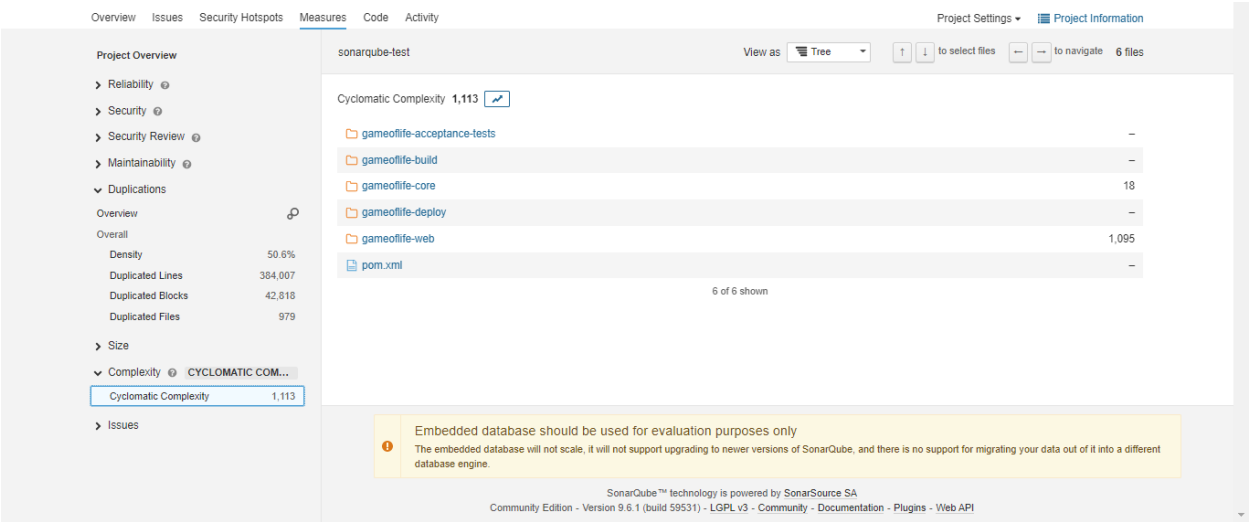
Bugs and Code Smells



Duplications -



Cyclomatic Complexity -



PAGE NO:
DATE / /

CONCLUSIONS

We successfully created Jenkins CICD pipeline with sonarcube to perform static analysis of code to detect bugs, code smells and security vulnerabilities on sample application.