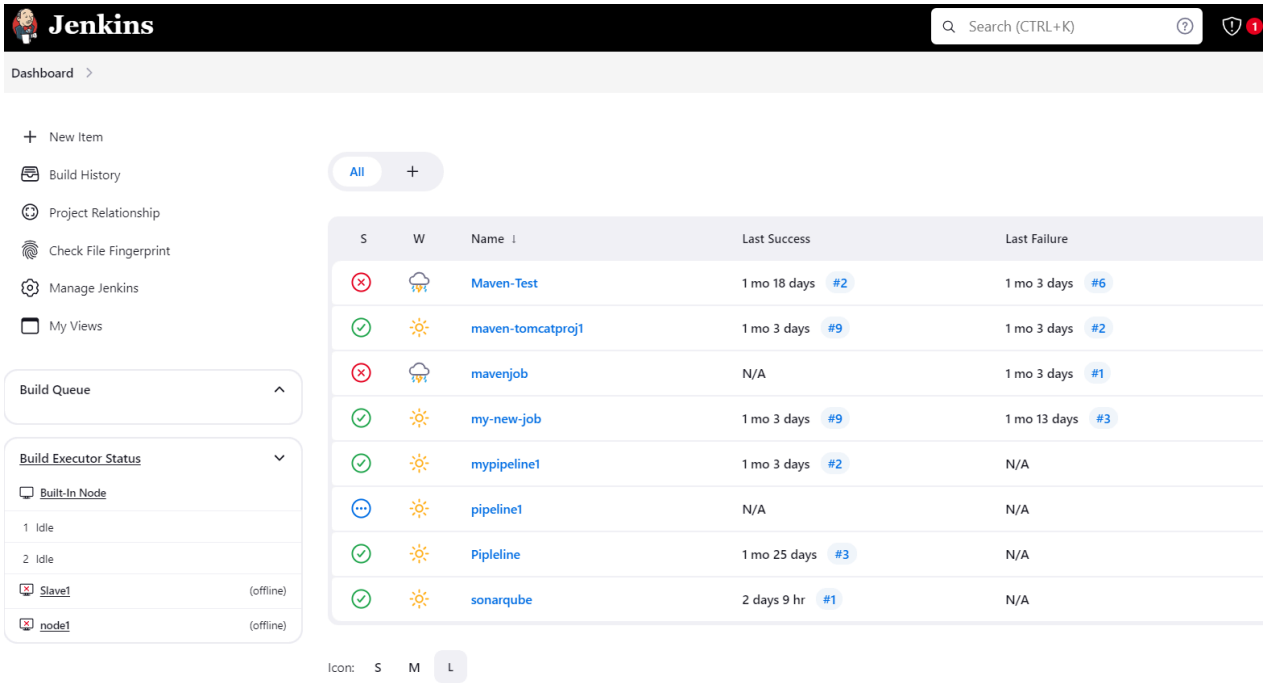
NAME: VEDANT DHOKE CLASS/ ROLL NO: D15C/ 9

AdvanceDevops Experiment 8

Aim: Create a Jenkins CICD Pipeline with SonarQube / GitLab Integration to perform a static analysis of the code to detect bugs, code smells, and security vulnerabilities on a sample Web Java / Python application. dive deep into this segment, let's first understand what is meant by the term ‘pipeline’?

1. Open Jenkin dashboard.

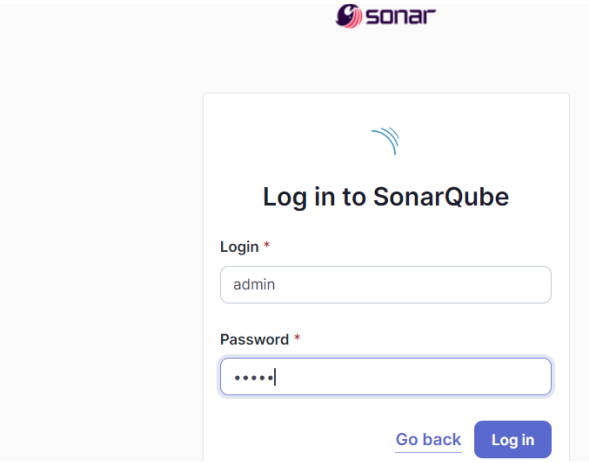


2. Run SonarQube in a Docker container using this command -

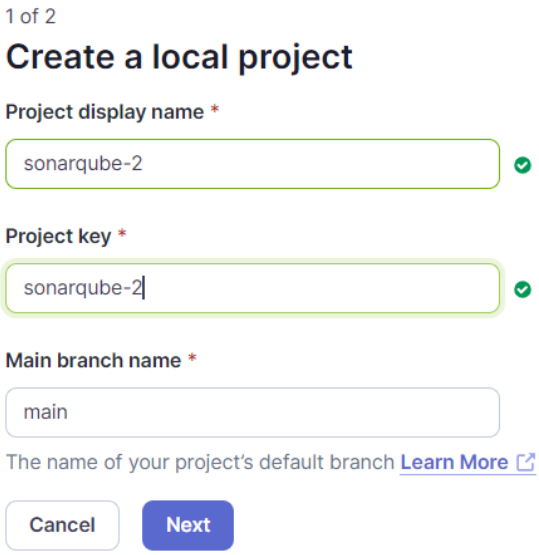
```
C:\Users\91900>docker run -d --name sonarqube -e SONAR_ES_BOOTSTRAP_CHECKS_DISABLE=true -p 9000:9000 sonarqube:latest
d23acccacd96c274f5f87912674ecf2d9adffff185a940c24740f44b29534485
```

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

4. Login to SonarQube using username *admin* and password *admin*.

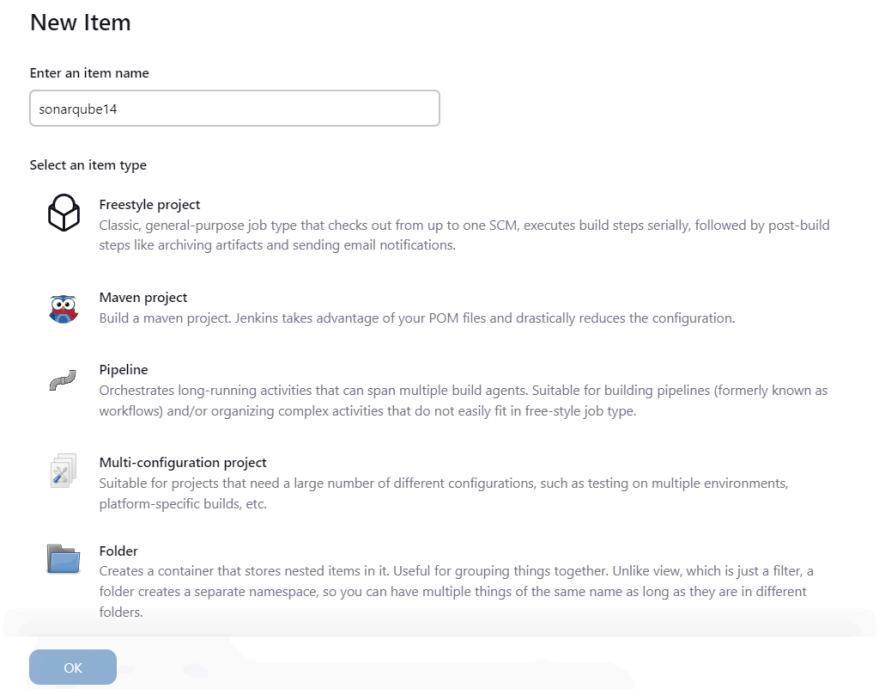


5. Create a manual project in SonarQube with the name **sonarqube-test**



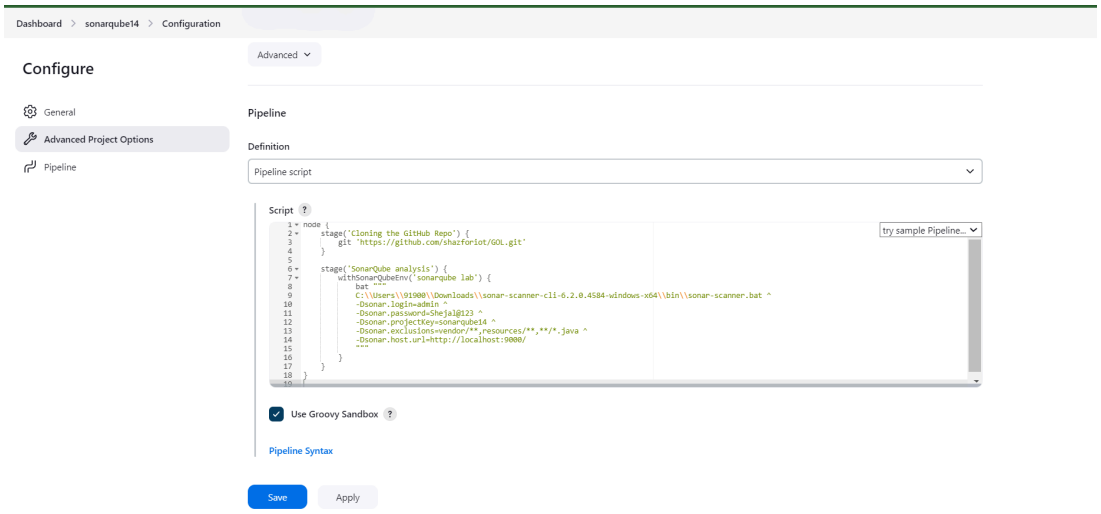
Setup the project and come back to Jenkins Dashboard.

6. Create a New Item in Jenkins, choose **Pipeline**.



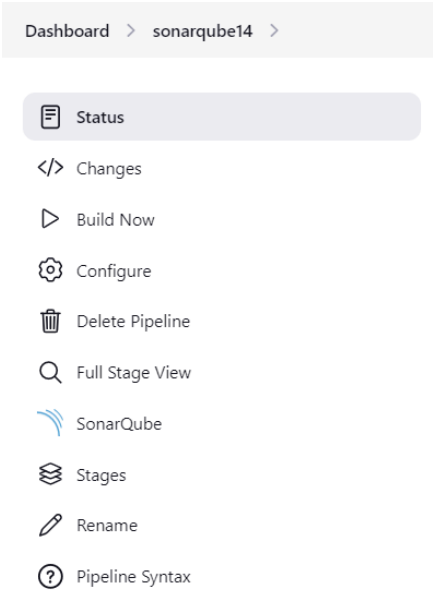
7. Under Pipeline Script, enter the following -

```
node {
    stage('Cloning the GitHub Repo') {
        git 'https://github.com/shazforiot/GOL.git'
    }
    stage('SonarQube analysis') {
        withSonarQubeEnv('sonarqube') {
            sh "<PATH_TO_SONARQUBE_FOLDER>//bin//sonar-scanner \
            -D sonar.login=<SonarQube_USERNAME> \
            -D sonar.password=<SonarQube_PASSWORD> \
            -D sonar.projectKey=<Project_KEY> \
-D sonar.exclusions=vendor/**,resources/**,**/*.java \
            -D sonar.host.url=http://127.0.0.1:9000/"
        }
    }
}
```

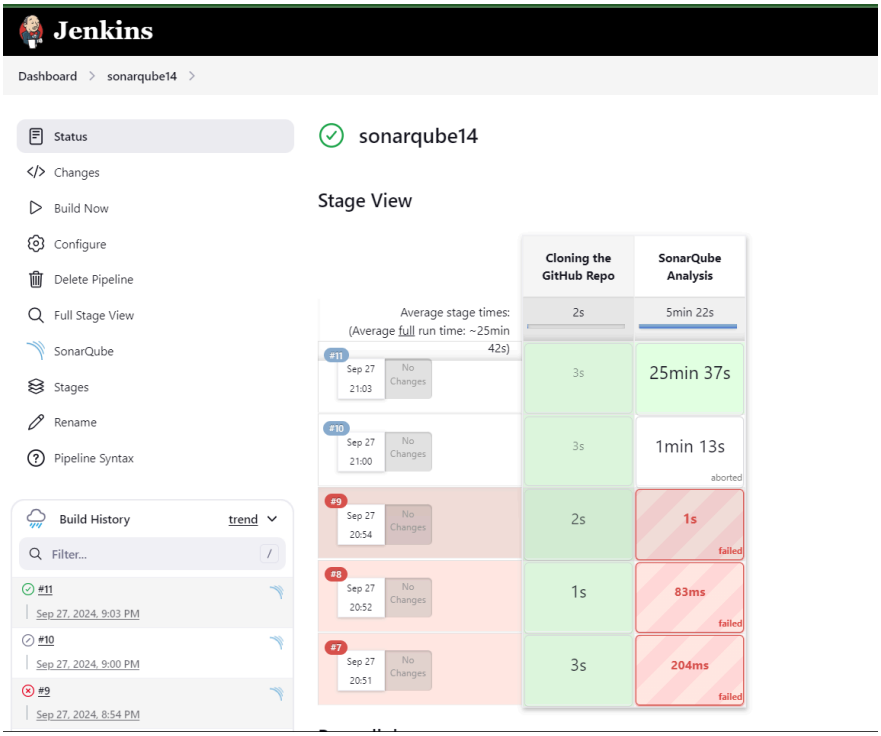


It is a java sample project which has a lot of repetitions and issues that will be detected by SonarQube.

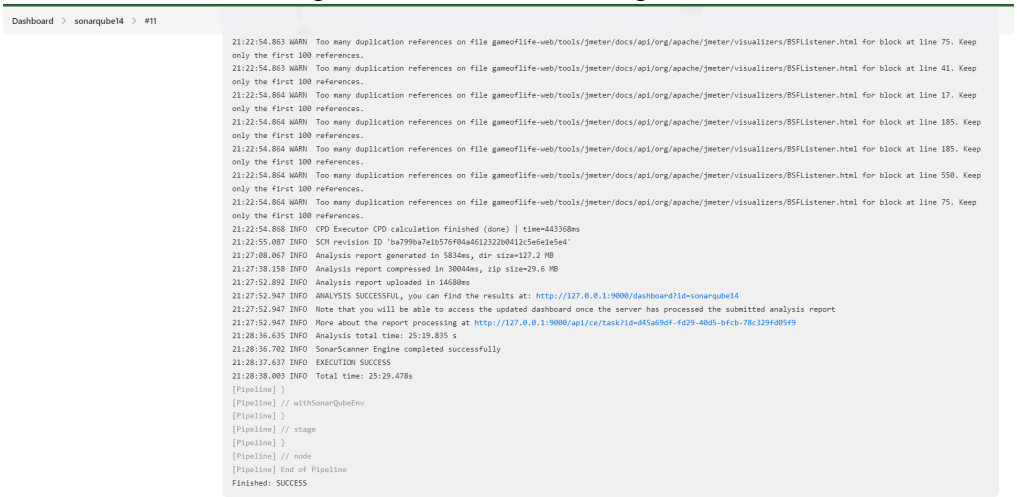
8. Run The Build.



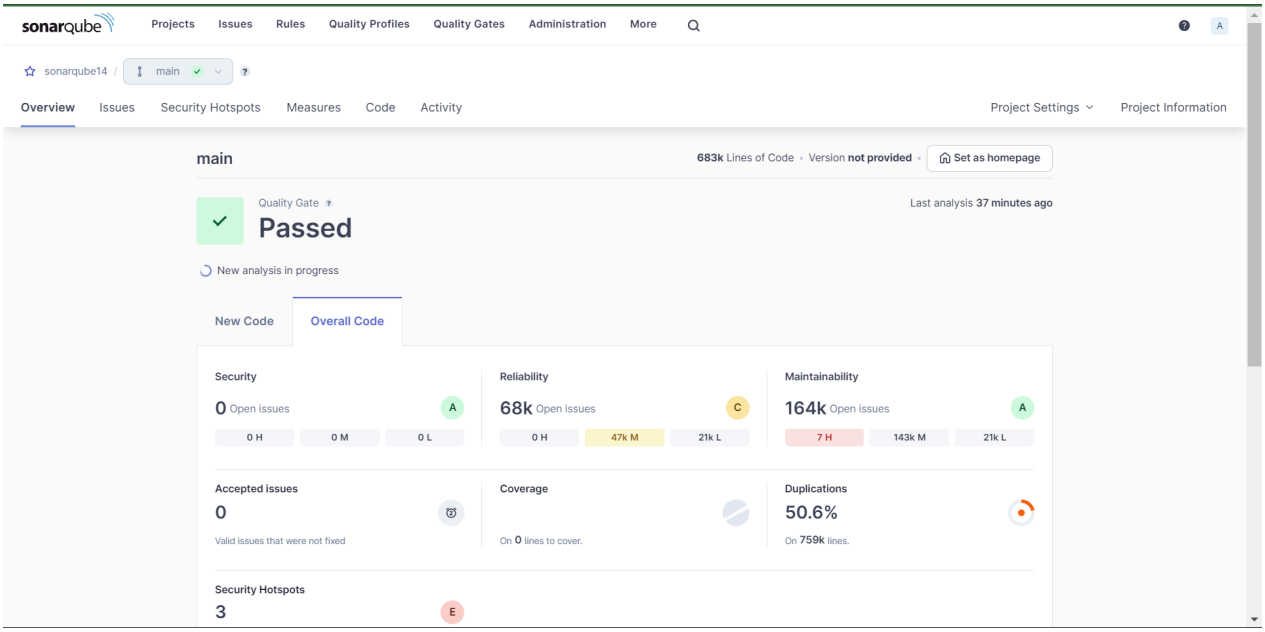
NAME: VEDANT DHOKE CLASS/ ROLL NO: D15C/ 9



9. Check the console output once the build is complete.



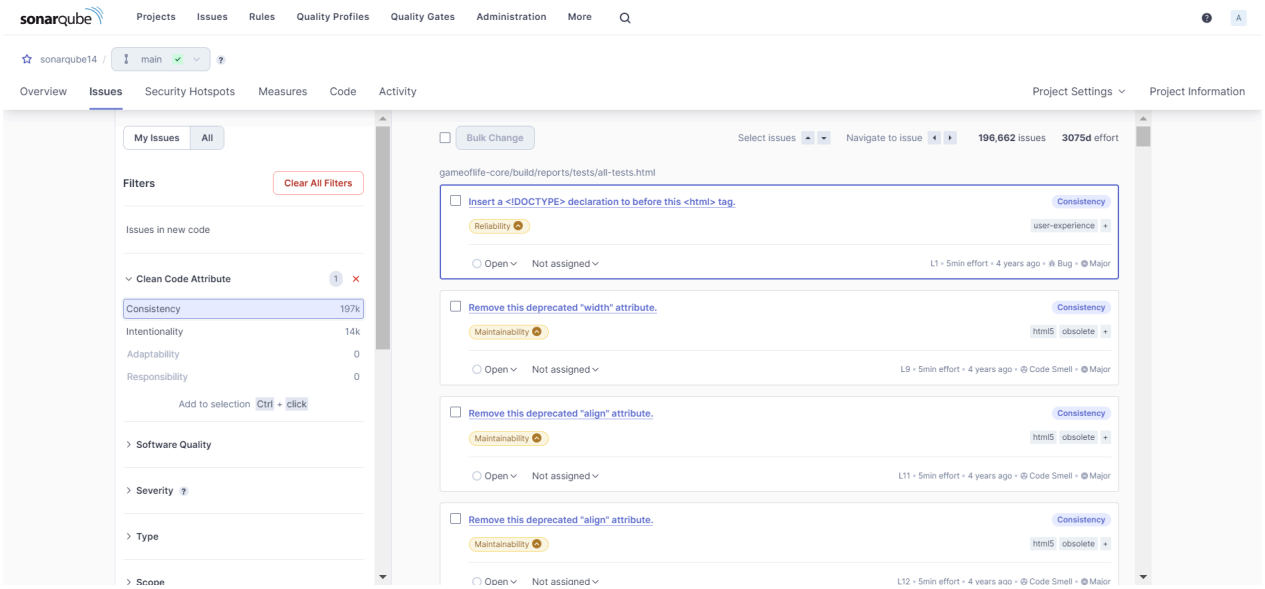
10. After that, check the project in SonarQube.



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

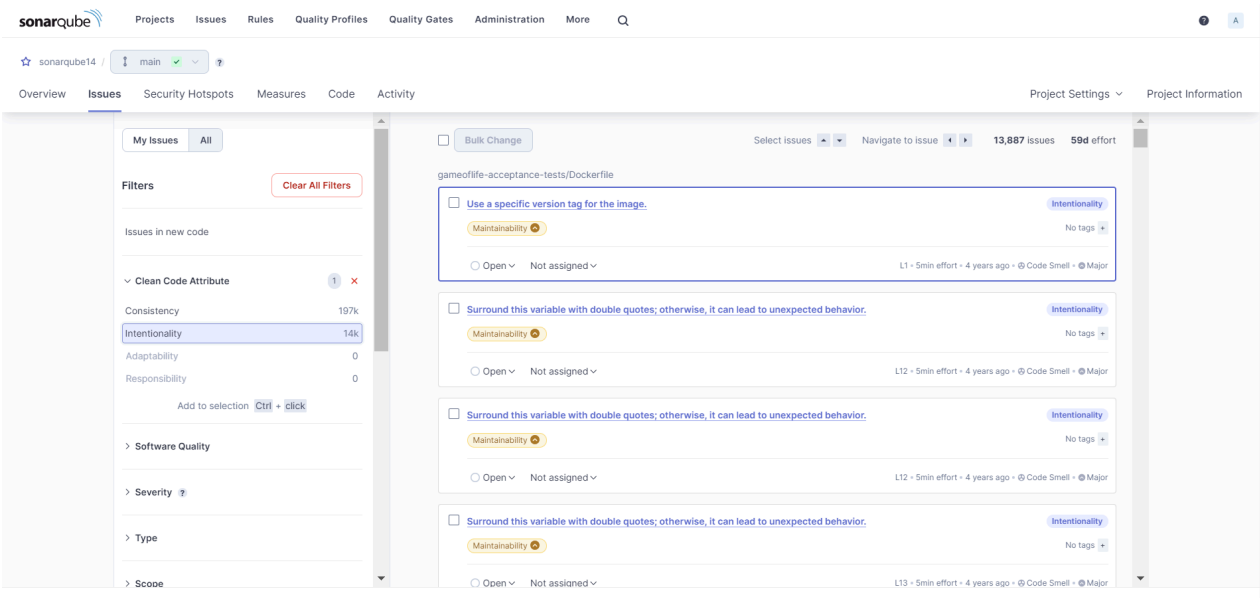
11. **Code Problems -**

Consistency

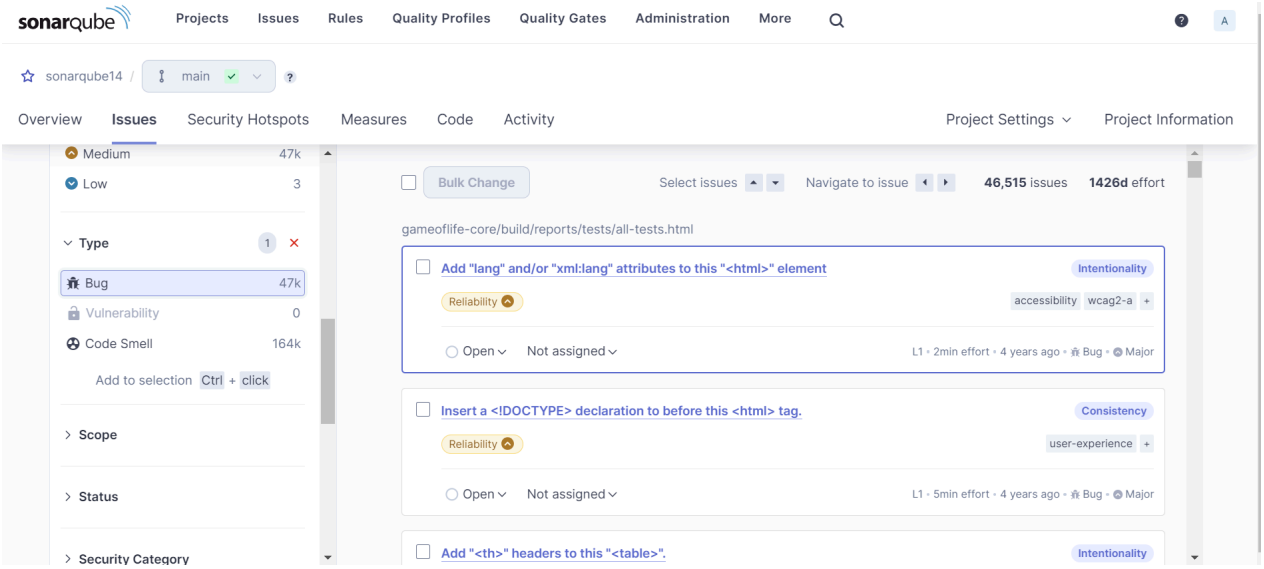


NAME: VEDANT DHOKE CLASS/ ROLL NO: D15C/ 9

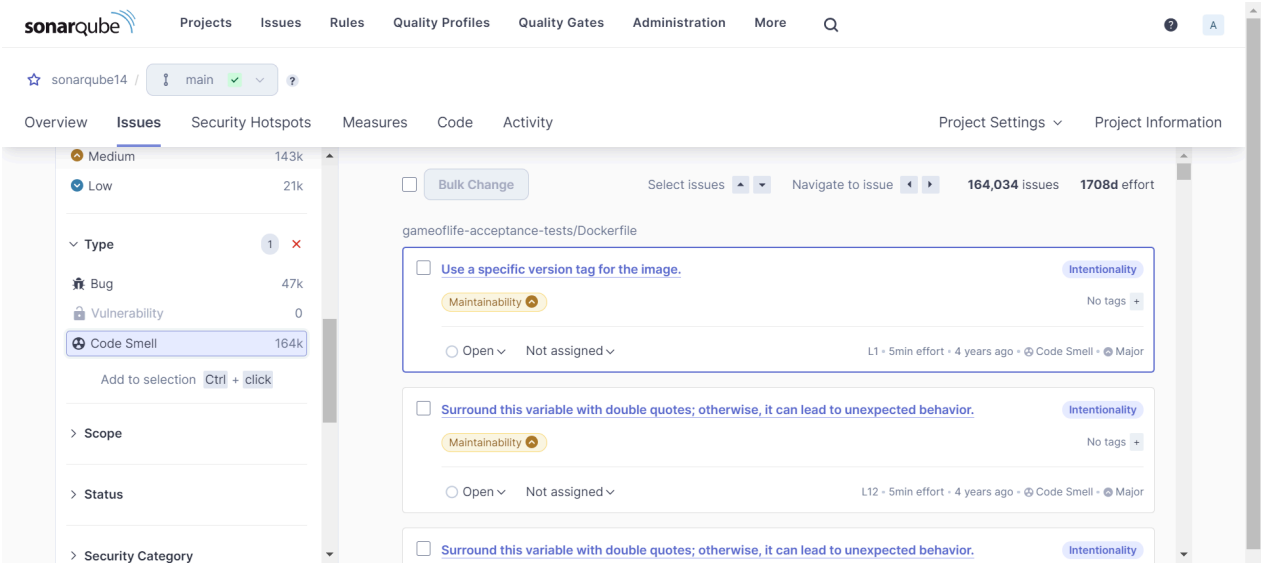
Intentionality



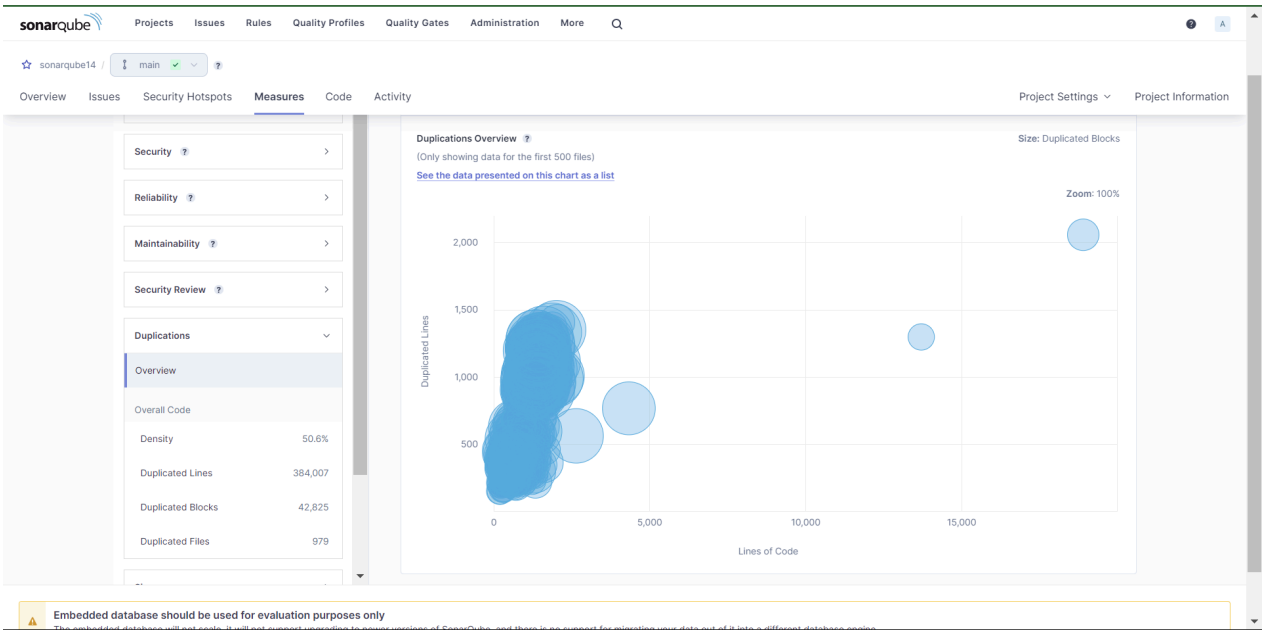
Bugs and Code Smells



NAME: VEDANT DHOKE CLASS/ ROLL NO: D15C/ 9

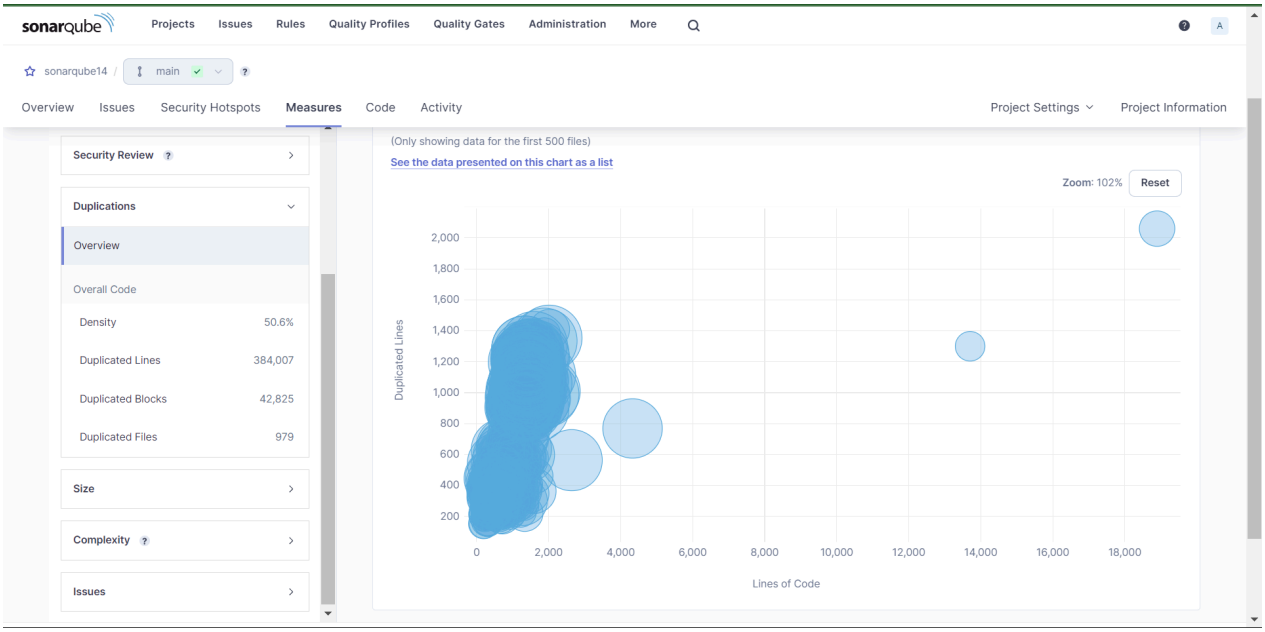


Duplicates



NAME: VEDANT DHOKE CLASS/ ROLL NO: D15C/ 9

Cyclomatic Complexities



In this way, we have created a CI/CD Pipeline with Jenkins and integrated it with SonarQube to find issues in the code like bugs, code smells, duplicates, cyclomatic complexities, etc.

Conclusion:

In this experiment, we successfully cloned a GitHub repository and integrated it with SonarQube for comprehensive code analysis. SonarQube provided valuable insights into various types of program issues, such as:

Consistency: Detected non-adherence to coding standards and formatting rules.

Intentionality: Flagged potential logical or structural errors within the code.

Severity: Classified issues by their level of criticality (e.g., critical, major, minor).

Duplicates: Identified redundant code segments, suggesting potential optimization.

Cyclomatic Complexity: Measured the complexity of the code based on the number of control flow paths, highlighting sections that might be difficult to maintain or prone to errors.

Issues Faced:

SonarQube Scanner Path Error: Initially, Jenkins failed to detect the correct path for the SonarQube Scanner. This required manual intervention to adjust the pipeline script and specify the correct path for the scanner bash file, allowing the pipeline to execute successfully.

SonarQube Login Issues: Default credentials for logging into SonarQube (username: admin, password: admin) did not work due to a configuration reset or password change in a previous session. This necessitated troubleshooting and reconfiguration of access credentials.

Slow Analysis Process: During the scan of larger repositories, the SonarQube analysis process was notably slow.