Expt No. 08 Advanced DevOps Lab

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.

Theory:

What is SAST?

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

What problems does SAST solve?

SAST takes place very early in the software development life cycle (SDLC) as it does not require a working application and can take place without code being executed. It helps developers identify vulnerabilities in the initial stages of development and quickly resolve issues without breaking builds or passing on vulnerabilities to the final release of the application.

SAST tools give developers real-time feedback as they code, helping them fix issues before they pass the code to the next phase of the SDLC. This prevents security-related issues from being considered an afterthought. SAST tools also provide graphical representations of the issues found, from source to sink. These help you navigate the code easier. Some tools point out the exact location of vulnerabilities and highlight the risky code. Tools can also provide in-depth guidance on how to fix issues and the best place in the code to fix them, without requiring deep security domain expertise.

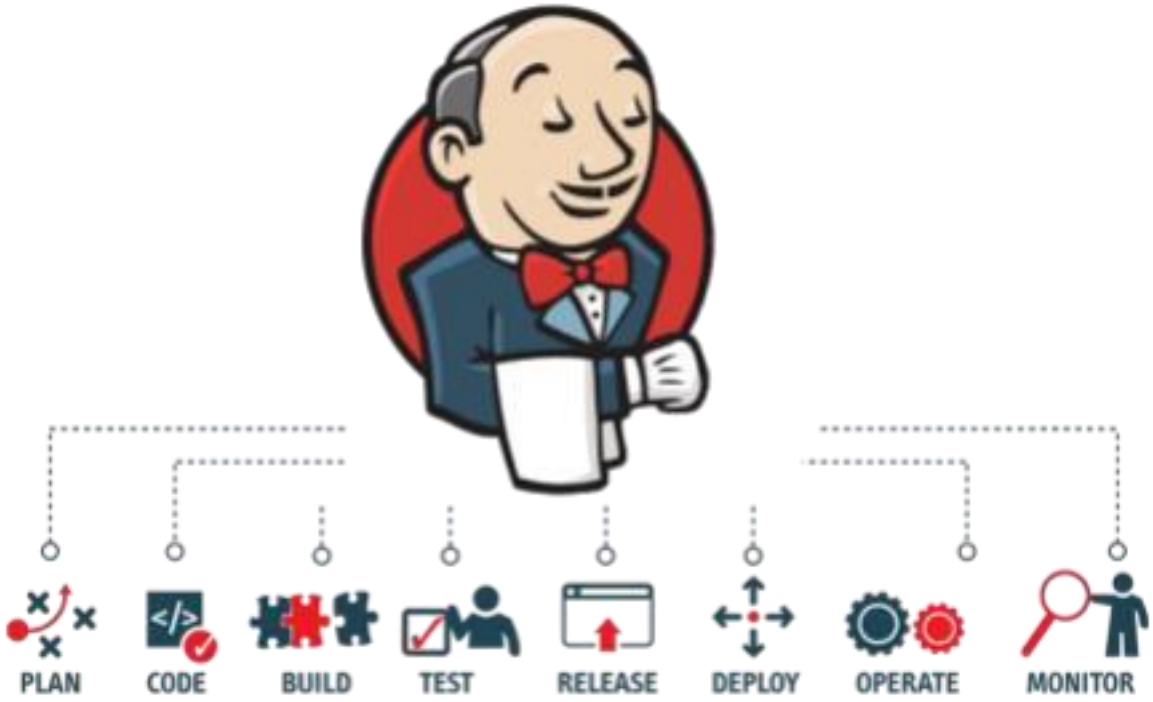
It's important to note that SAST tools must be run on the application on a regular basis, such as during daily/monthly builds, every time code is checked in, or during a code release.

Why is SAST important?

Developers dramatically outnumber security staff. It can be challenging for an organization to find the resources to perform code reviews on even a fraction of its applications. A key strength of SAST tools is the ability to analyze 100% of the codebase. Additionally, they are much faster than manual secure code reviews performed by humans. These tools can scan millions of lines of code in a matter of minutes. SAST tools automatically identify critical vulnerabilities—such as buffer overflows, SQL injection, cross-site scripting, and others—with high confidence. **What is a CI/CD Pipeline?**

CI/CD pipeline refers to the Continuous Integration/Continuous Delivery pipeline. Before we dive deep into this segment, let's first understand what is meant by the term ‘pipeline’?

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



Now, consider the CI/CD pipeline as the backbone of the DevOps approach. This Pipeline is responsible for building codes, running tests, and deploying new software versions. The Pipeline executes the job in a defined manner by first coding it and then structuring it inside several blocks that may include several steps or tasks.

What is SonarQube?

SonarQube is an open-source platform developed by SonarSource for continuous inspection of code quality. Sonar does static code analysis, which provides a detailed report of bugs, code smells, vulnerabilities, code duplications.
It supports 25+ major programming languages through built-in rulesets and can also be extended with various plugins.

Benefits of SonarQube

- **Sustainability -** Reduces complexity, possible vulnerabilities, and code duplications, optimising the life of applications.
- **Increase productivity -** Reduces the scale, cost of maintenance, and risk of the application; as such, it removes the need to spend more time changing the code
- **Quality code -** Code quality control is an inseparable part of the process of software development.
- **Detect Errors -** Detects errors in the code and alerts developers to fix them automatically before submitting them for output.
- **Increase consistency -** Determines where the code criteria are breached and enhances the quality
- **Business scaling -** No restriction on the number of projects to be evaluated
- **Enhance developer skills -** Regular feedback on quality problems helps developers to improve their coding skills

Integrating Jenkins with SonarQube:

Prerequisites:

- Jenkins installed
- Docker Installed (for SonarQube)

- SonarQube Docker Image

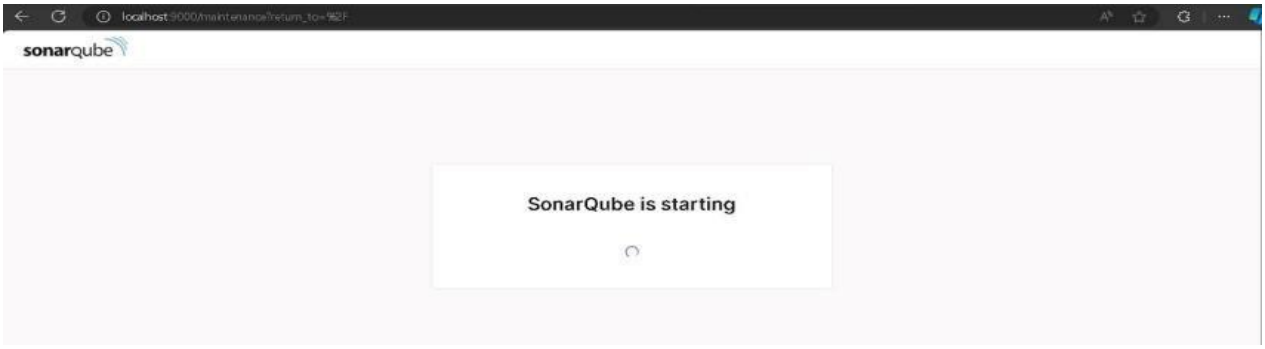
Steps to create a Jenkins CI/CD Pipeline and use SonarQube to perform SAST

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

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

```
PS C:\Users\91773\Desktop\College Resources\Advdevops Exp8> docker run -d --name sonarqube2 -e SONAR_ES_BOOTSTRAP_CHECKS
_DISABLE=true -p 9000:9000 sonarqube:latest
71fc67f0b15baa5be5bdcdd66966938e18682683d020beadcbc909dd027cfe7a
PS C:\Users\91773\Desktop\College Resources\Advdevops Exp8>
```

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

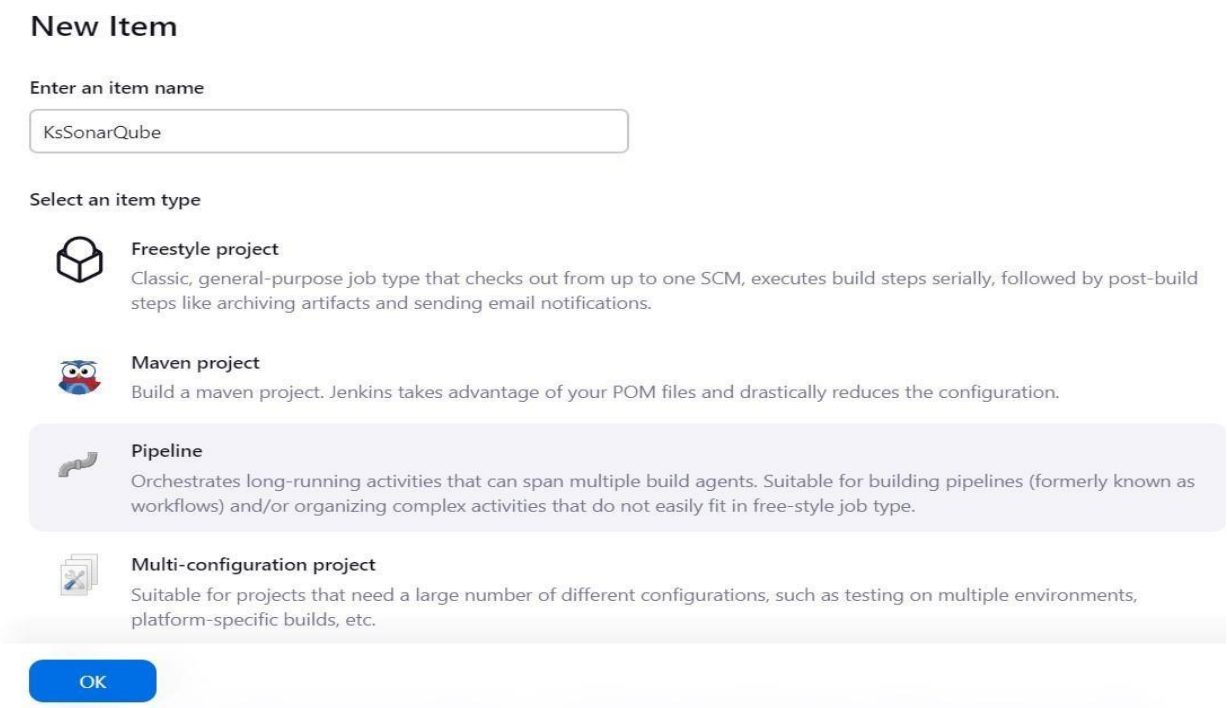


4. Login to SonarQube using username *admin* and password *admin*.

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

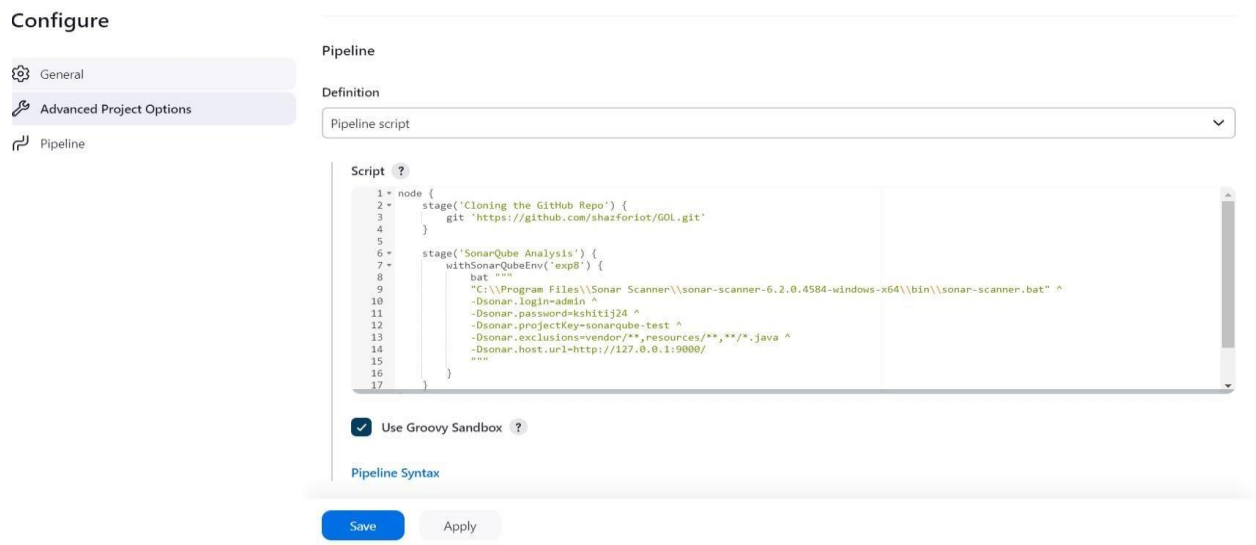
Setup the project and come back to Jenkins Dashboard.

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



7. Under Pipeline Script, enter the following - node {

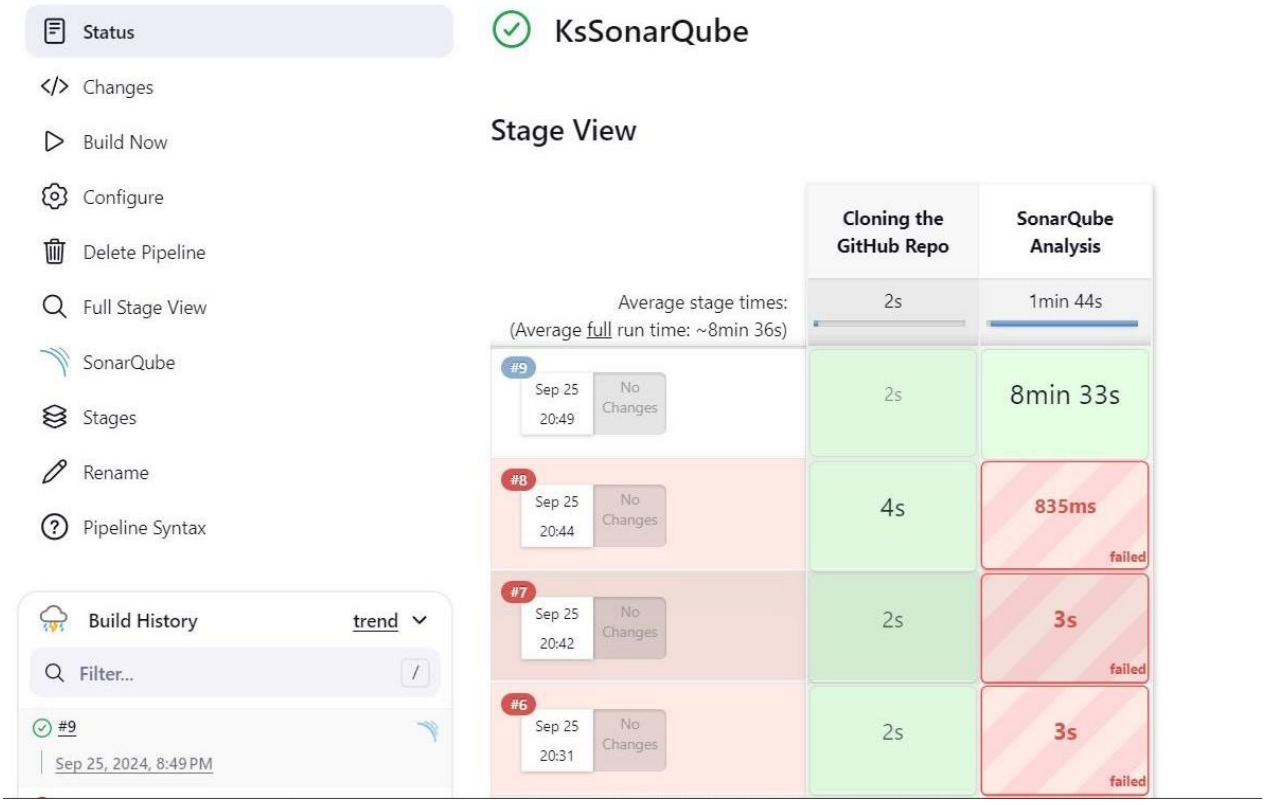
```
    stage('Cloning the GitHub Repo') {
      git 'https://github.com/shazforiot/GOL.git'
    } stage('SonarQube analysis')
    {
      withSonarQubeEnv('sonarqube') { sh
        "<PATH_TO_SONARQUBE_FOLDER>//bin//sonar-scanner \
        -D sonar.login=<SonarQube_USERNAME> \
        -D sonar.password=<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.

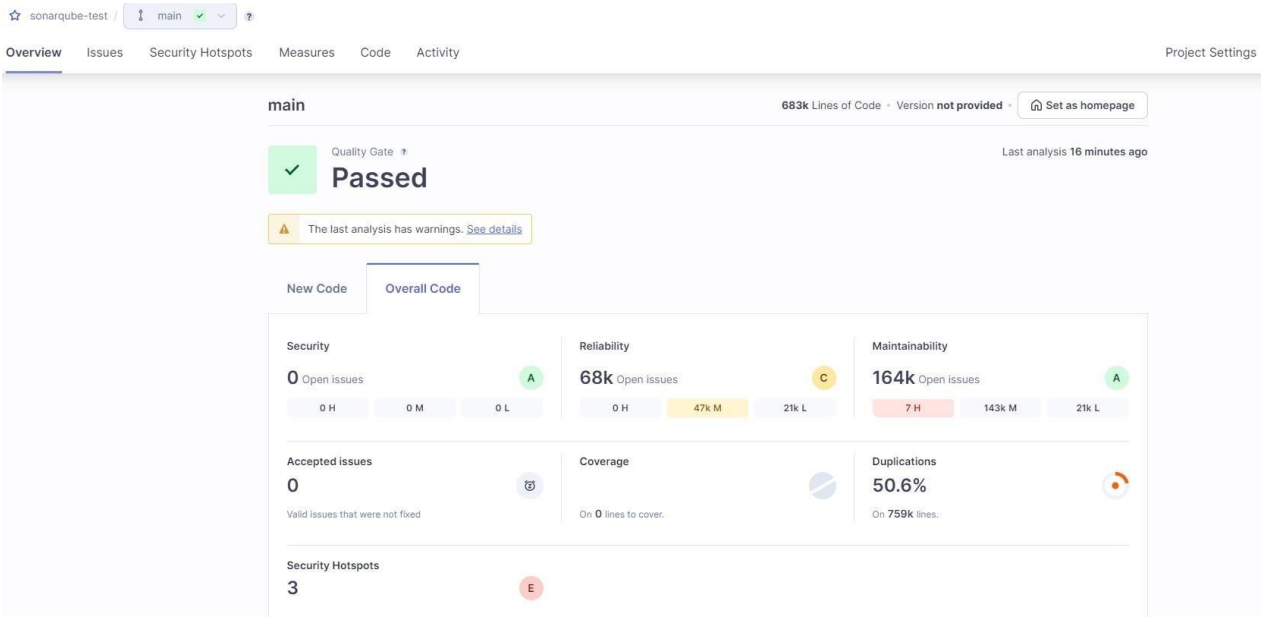
8. Run The Build.

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



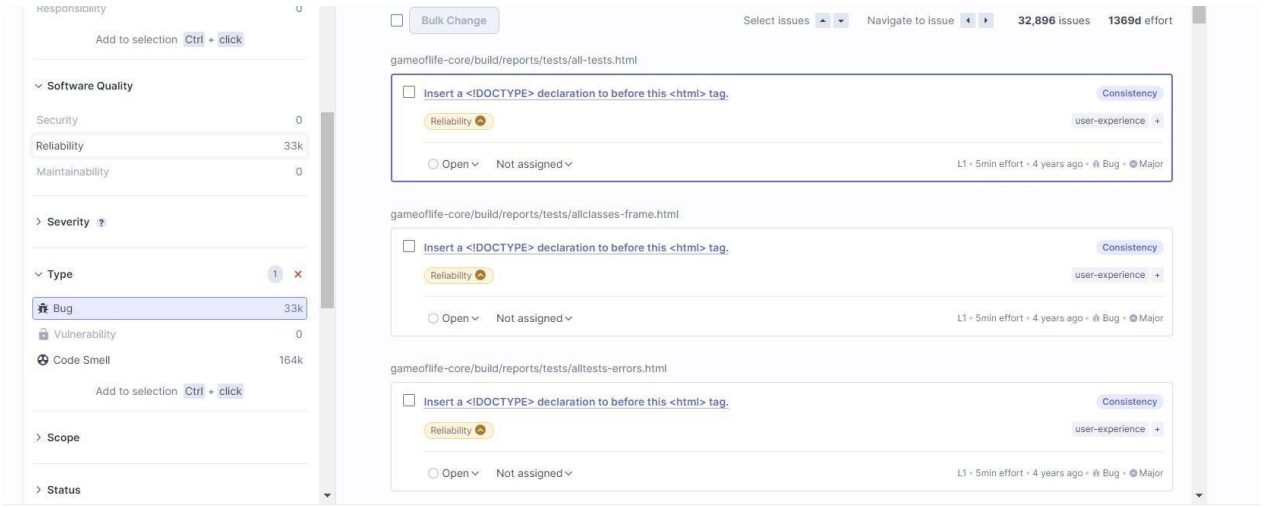

```
for block at line 17. Keep only the first 100 references.
20:56:18.455 WARN  Too many duplication references on file gameoflife-web/tools/jmeter/docs/api/org/apache/jmeter/gui/util/TextAreaCellRenderer.html
for block at line 296. Keep only the first 100 references.
20:56:18.455 WARN  Too many duplication references on file gameoflife-web/tools/jmeter/docs/api/org/apache/jmeter/gui/util/TextAreaCellRenderer.html
for block at line 75. Keep only the first 100 references.
20:56:18.456 INFO  CPD Executor CPD calculation finished (done) | time=107093ms
20:56:18.490 INFO  SCM revision ID 'ba799ba7e1b576f04a4612322b0412c5e6e1e5e4'
20:57:50.106 INFO  Analysis report generated in 3149ms, dir size=127.2 MB
20:57:56.943 INFO  Analysis report compressed in 6828ms, zip size=29.6 MB
20:57:58.685 INFO  Analysis report uploaded in 1732ms
20:57:58.688 INFO  ANALYSIS SUCCESSFUL, you can find the results at: http://127.0.0.1:9000/dashboard?id=sonarqube-test
20:57:58.688 INFO  Note that you will be able to access the updated dashboard once the server has processed the submitted analysis report
20:57:58.688 INFO  More about the report processing at http://127.0.0.1:9000/api/ce/task?id=18847db4-4f06-4766-9ad4-ee006448353c
20:58:06.225 INFO  Analysis total time: 8:22.672 s
20:58:06.231 INFO  SonarScanner Engine completed successfully
20:58:06.824 INFO  EXECUTION SUCCESS
20:58:06.857 INFO  Total time: 8:31.713s
[Pipeline] }
[Pipeline] // withSonarQubeEnv
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

10. After that, check the project in SonarQube.

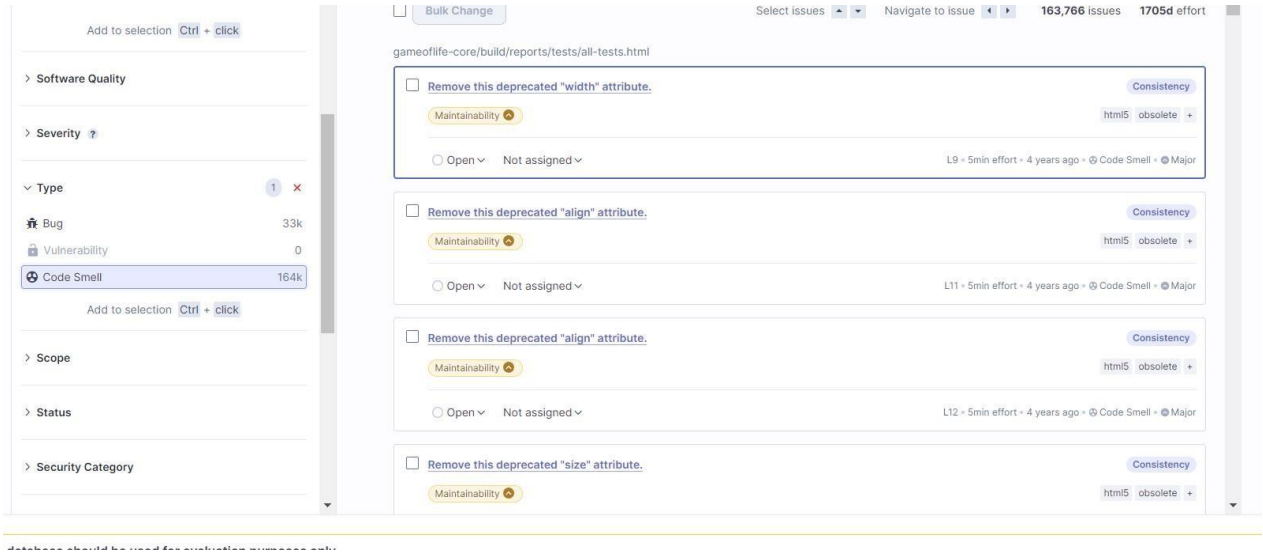


Under different tabs, check all different issues with the code. 11.

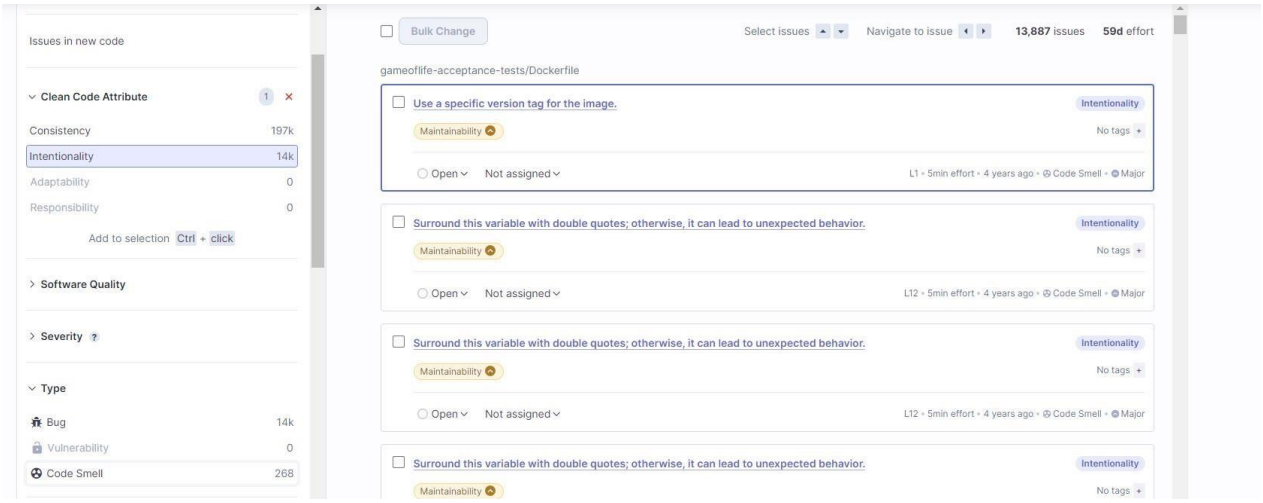
Bugs



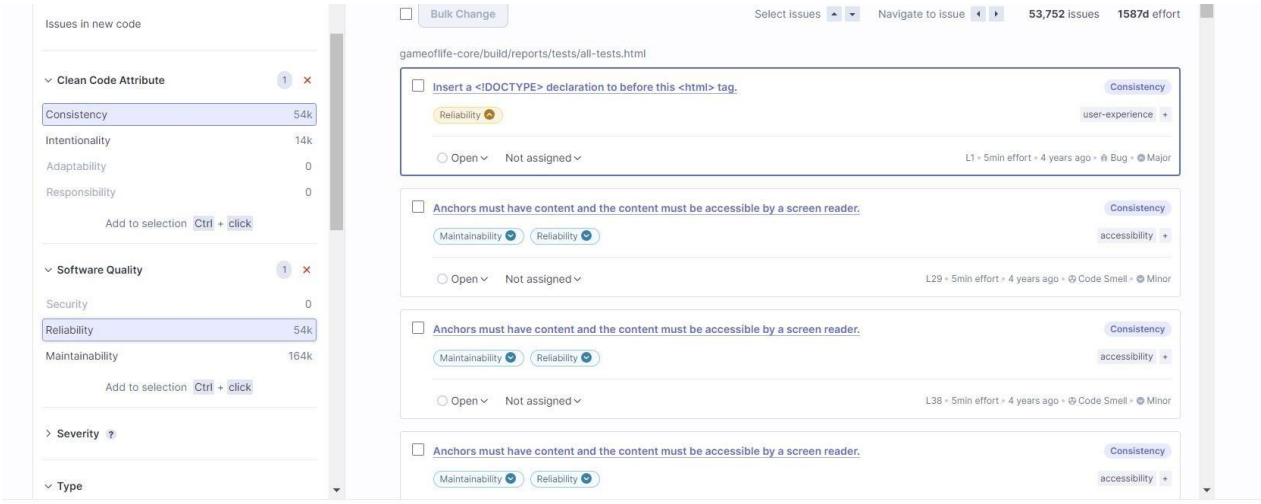
Code Smells



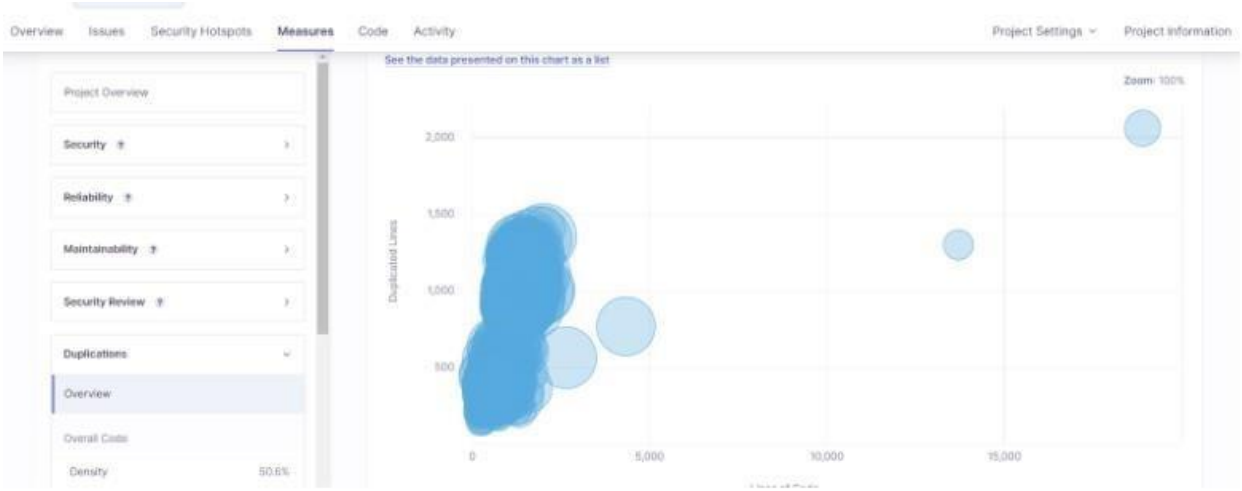
Intentional issues



Reliabilities issue



Duplicates



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 integrated Jenkins with SonarQube to enable automated code quality checks within our CI/CD pipeline. We started by deploying SonarQube using Docker, setting up a project, and configuring it to analyze code quality. Next, we configured Jenkins by installing the SonarQube Scanner plugin, adding SonarQube server details, and setting up the scanner tool. We then developed a Jenkins pipeline to automate the process of cloning a GitHub repository and running SonarQube analysis on the code. This integration helps ensure continuous monitoring of code quality, detecting issues such as bugs, code smells, and security vulnerabilities throughout the development process.