# ADVANCED DEVOPS EXP 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.

## THEORY:

**Static Application Security Testing (SAST)**
SAST is a methodology for testing an application's source code to identify security vulnerabilities before the code is compiled. This type of testing, also referred to as white-box testing, helps improve application security by finding weaknesses early in development.

**Problems SAST Solves**

- **Early Detection**: SAST finds vulnerabilities early in the Software Development Life Cycle (SDLC), allowing developers to fix issues without affecting builds or passing vulnerabilities to the final release.
- **Real-Time Feedback**: Developers receive immediate feedback during coding, helping them address security issues before moving to the next stage of development.
- **Graphical Representations**: SAST tools often provide visual aids to help developers navigate the code and identify the exact location of vulnerabilities, offering suggestions for fixes.
- **Regular Scanning**: SAST tools can be configured to scan code regularly, such as during daily builds, code check-ins, or before releases.

**Importance of SAST**

- **Resource Efficiency**: With a larger number of developers than security experts, SAST allows full codebase analysis quickly and efficiently, without relying on manual code reviews.
- **Speed**: SAST tools can analyze millions of lines of code within minutes, detecting critical vulnerabilities such as buffer overflows, SQL injection, and cross-site scripting (XSS) with high accuracy.

**CI/CD Pipeline**
A Continuous Integration/Continuous Delivery (CI/CD) pipeline is a sequence of automated tasks designed to build, test, and deploy new software versions rapidly and consistently. It plays a crucial role in DevOps practices, ensuring fast and reliable software releases.

**SonarQube**
SonarQube is an open-source platform from SonarSource that performs continuous code quality inspections through static code analysis. It identifies bugs, code smells, security vulnerabilities, and code

duplications in a wide range of programming languages. SonarQube is extendable with plugins and integrates seamlessly into CI/CD pipelines.

**Benefits of SonarQube**

- **Sustainability**: By reducing complexity and vulnerabilities, SonarQube extends the lifespan of applications and helps maintain cleaner code.
- **Increased Productivity**: SonarQube minimizes maintenance costs and risks, resulting in fewer code changes and a more stable codebase.
- **Quality Code**: Ensures code quality checks are integrated into the development process.
- **Error Detection**: Automatically identifies coding errors and alerts developers to resolve them before moving to production.
- **Consistency**: Helps maintain consistent code quality by detecting and reporting violations of coding standards.
- **Business Scaling**: SonarQube supports scaling as the business grows without any restrictions.

**Implementation:**

**Prerequisites**

1. Jenkins installed on your machine.
2. Docker installed to run SonarQube.
3. SonarQube installed via Docker

**1. Set Up Jenkins**

- Open Jenkins Dashboard on localhost:8080 or your configured port
- Install the necessary plugins:
  - SonarQube Scanner Plugin

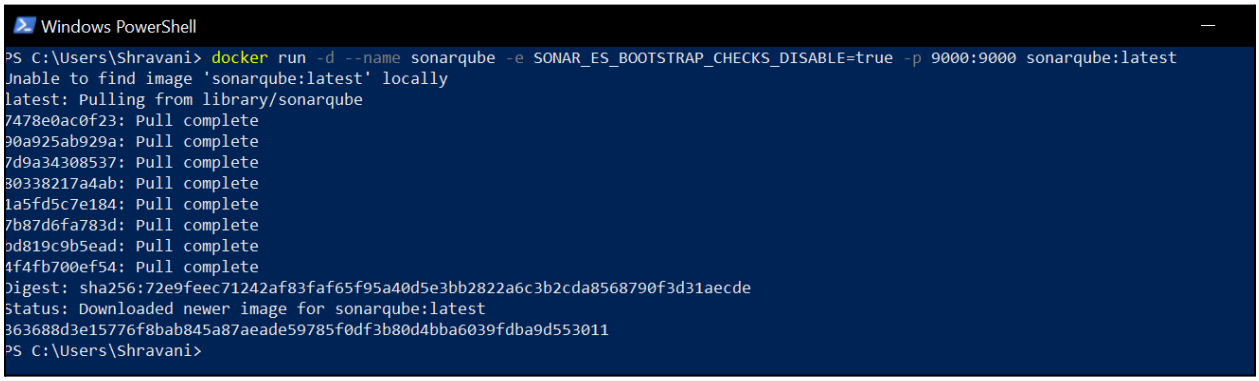
**.2. Run SonarQube in Docker**

Run the following command to start SonarQube in a Docker container:

command :

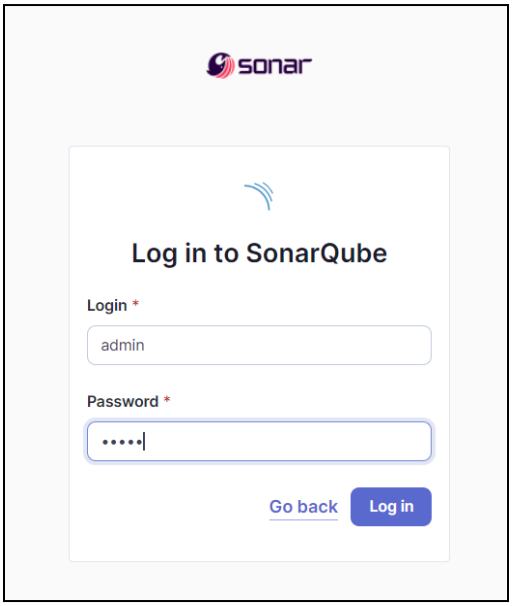
docker run -d --name sonarqube -e SONAR_ES_BOOTSTRAP_CHECKS_DISABLE=true - p 9000:9000 sonarqube:latest

- Check SonarQube status at http://localhost:9000.
- Login with your credentials:




### 3. Create a Project in SonarQube

- Go to Projects > Create Project.
- Name the project (e.g., sonarqube-test)

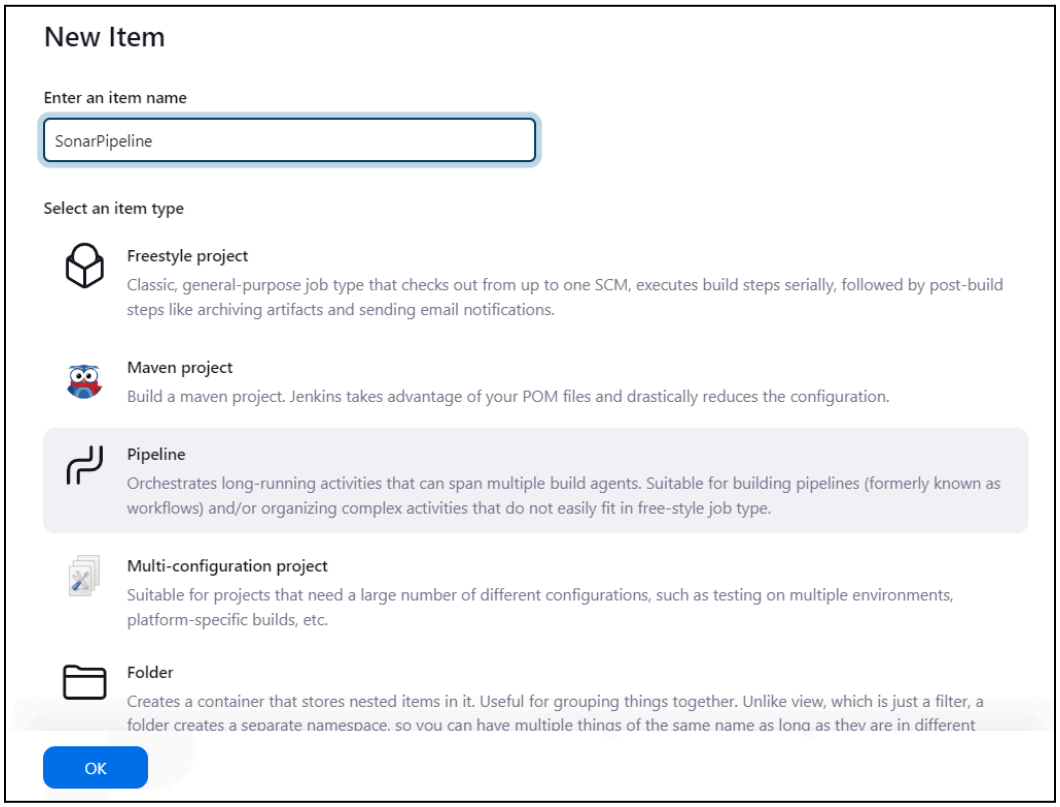



### 4. Generate SonarQube Token

- Go to My Account > Security > Generate Tokens.
- Copy the generated token for later use

### 5. Create a Jenkins Pipeline

- Go to Jenkins Dashboard, click New Item, and select Pipeline.



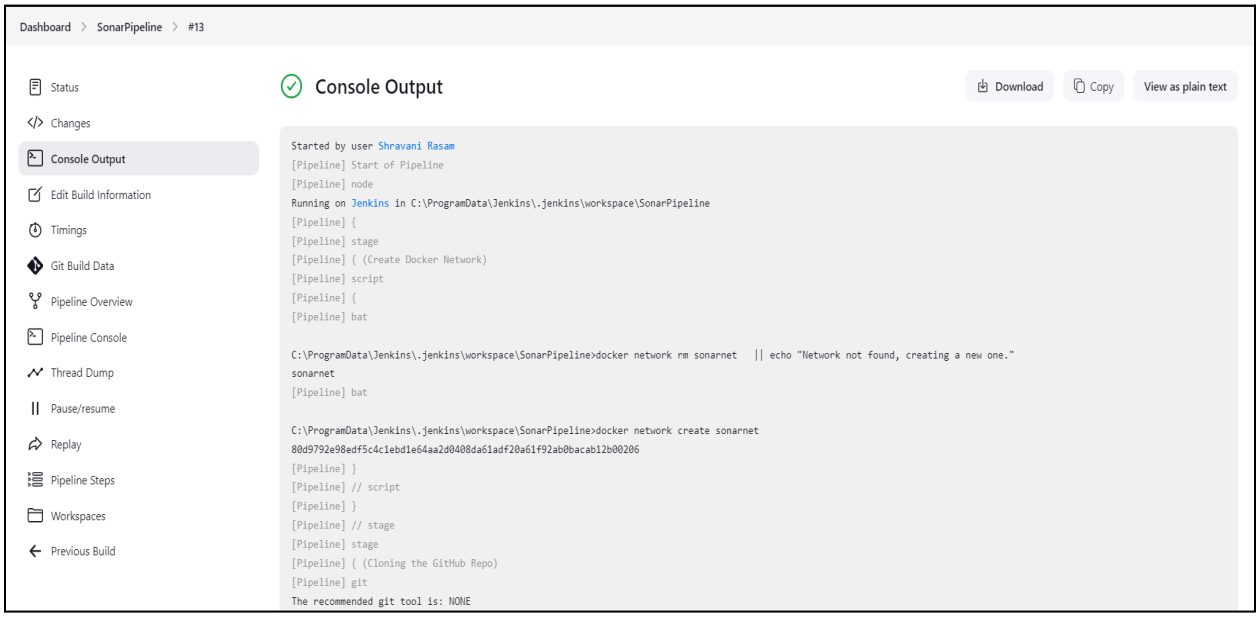

### 6. Under Pipeline Script, enter the following script:

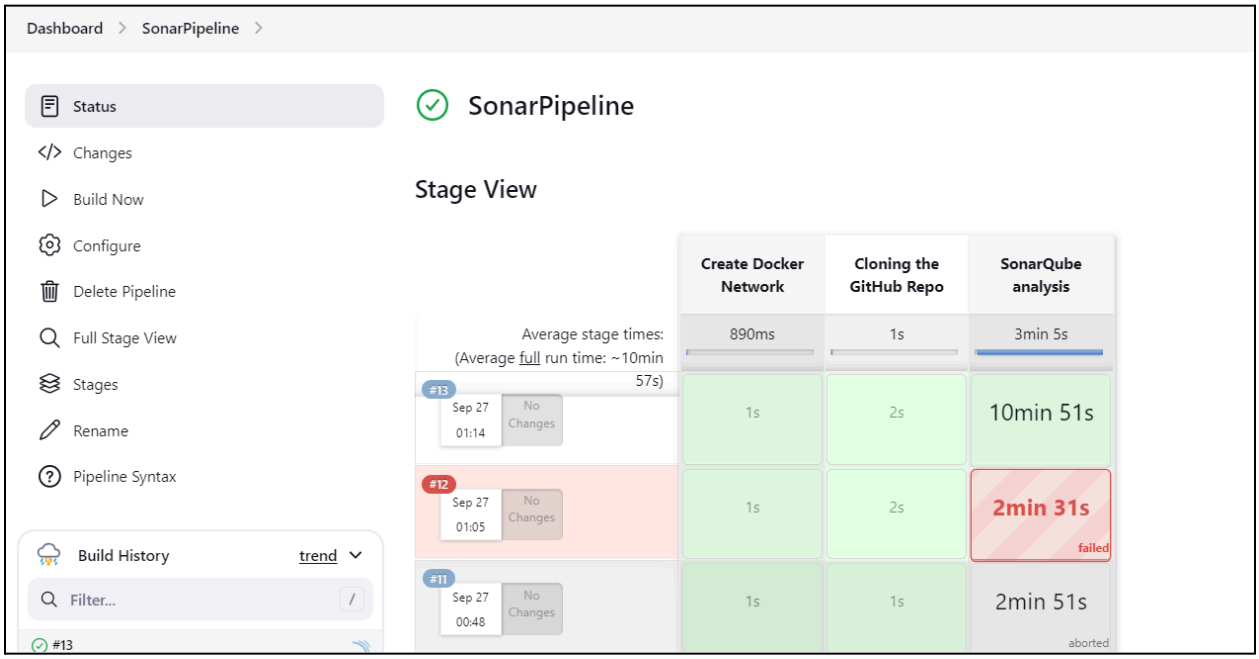
```
pipeline {
    agent any

    stages {
        stage('Create Docker Network') {
            steps {
                script {
                    bat 'docker network rm sonarnet || echo "Network not found, creating a new one."'
                    bat 'docker network create sonarnet'
                }
            }
        }

        stage('Cloning the GitHub Repo') {
            steps {
                git 'https://github.com/shazforiot/GOL.git'
```

```
        }
    }

    stage('SonarQube analysis') {
        steps {
            withSonarQubeEnv('sonarqube') {
                bat """
                docker run --rm --network sonarnet ^
                    -e SONAR_HOST_URL=http://192.168.133.16:9000 ^
                    -e SONAR_LOGIN=admin ^
                    -e SONAR_PASSWORD=Shravani@0212 ^
                    -e SONAR_PROJECT_KEY=sonarqube-test ^
                    -v ${WORKSPACE}:/usr/src ^
                    sonarsource/sonar-scanner-cli ^
                    -Dsonar.projectKey=sonarqube-test ^
                    -Dsonar.exclusions=vendor/**,resources/**,**/*.java ^
                    -Dsonar.login=admin ^
                    -Dsonar.password=Shravani@0212
                """
            }
        }
    }
  }
}
```

## 7. Run the Pipeline

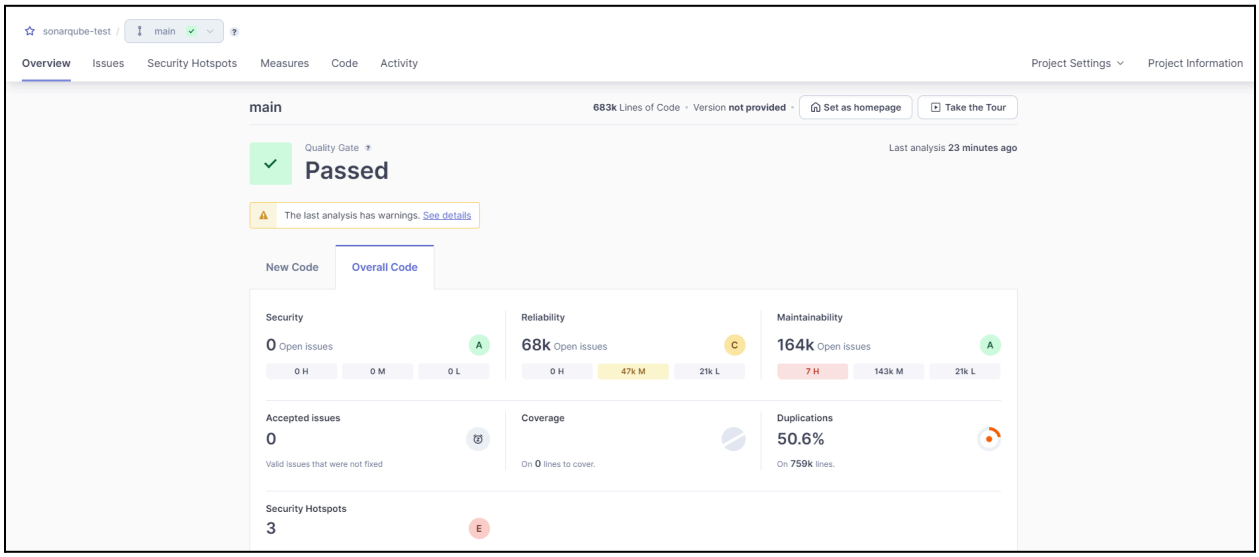
- Save the pipeline and click Build Now
- Monitor the console output for any errors



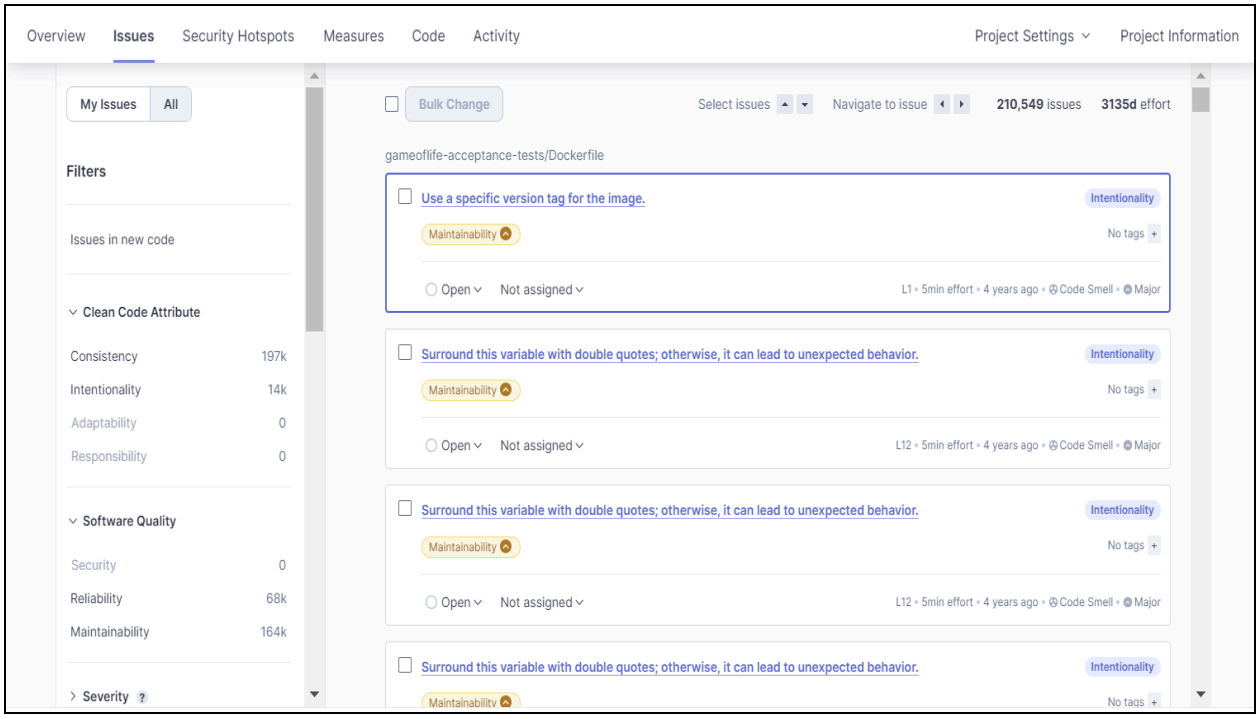
9. Check SonarQube for Analysis Results

● Go to your SonarQube dashboard and check the project for issues such as bugs, code smells, and security vulnerabilities.

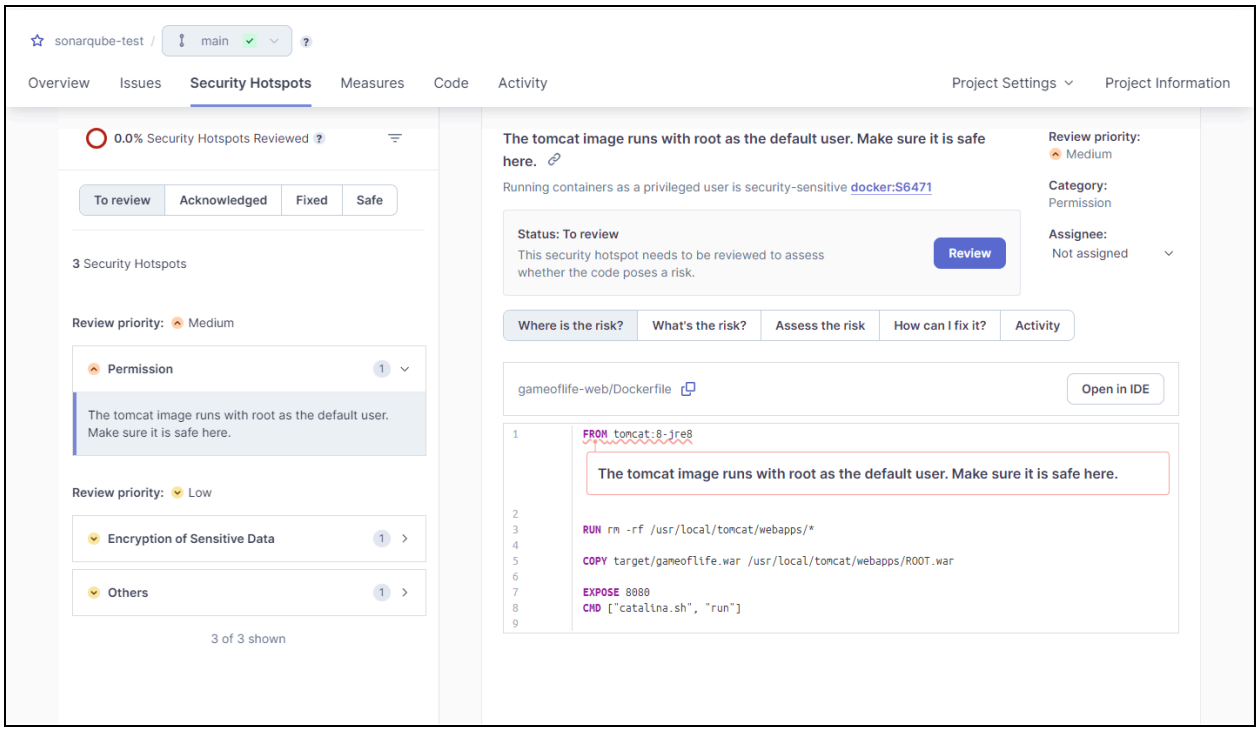


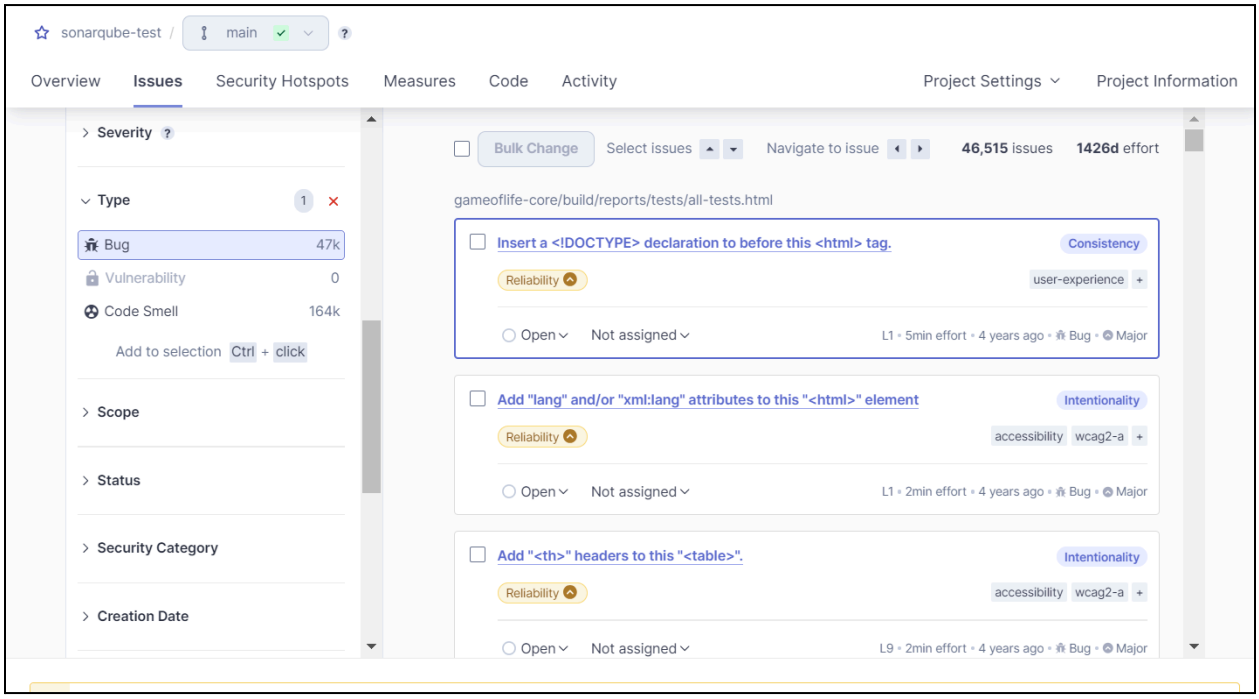

**10. Checking SonarQube for Analysis Results of a Code File with Bugs , Code Smells, Security Vulnerabilities, Cyclomatic Complexities and Duplicates .**

● Issues -

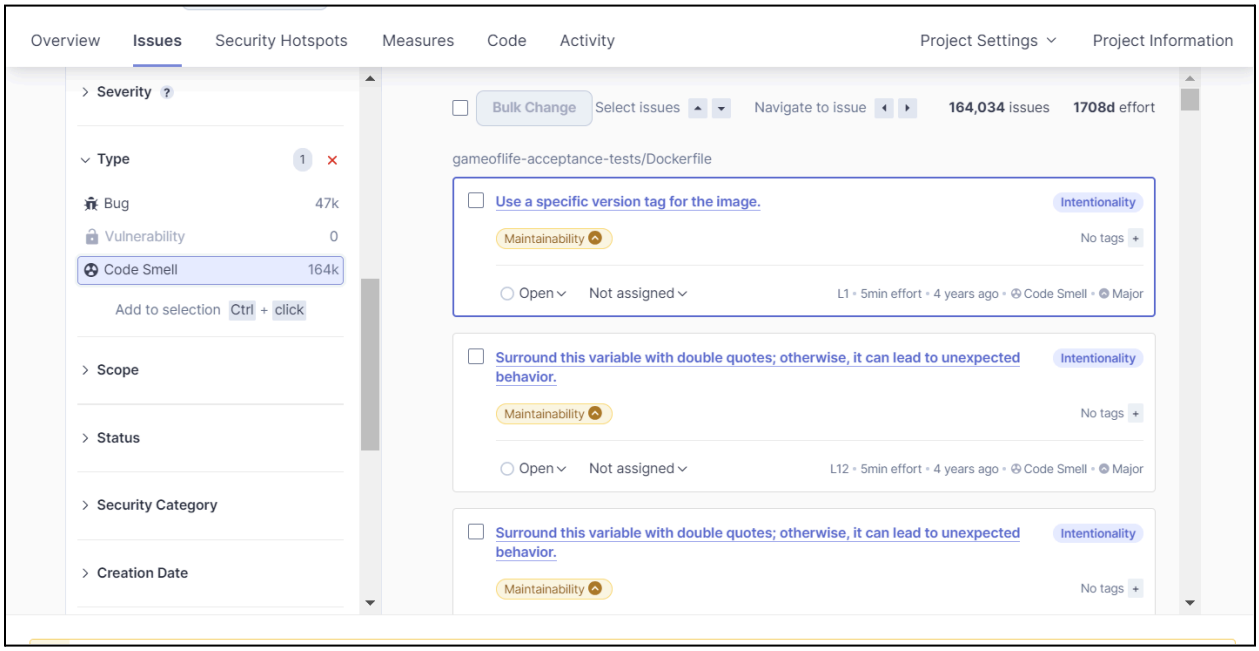


SHRAVANI RASAM D15A 46

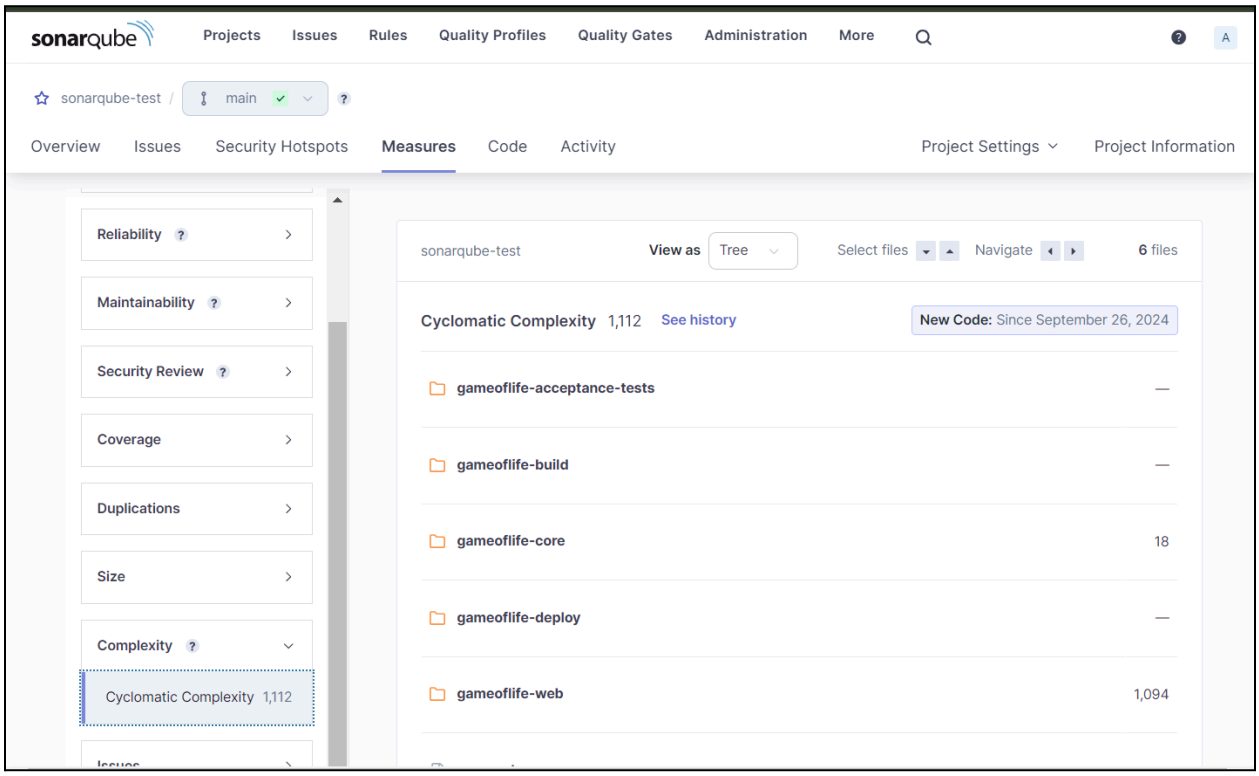


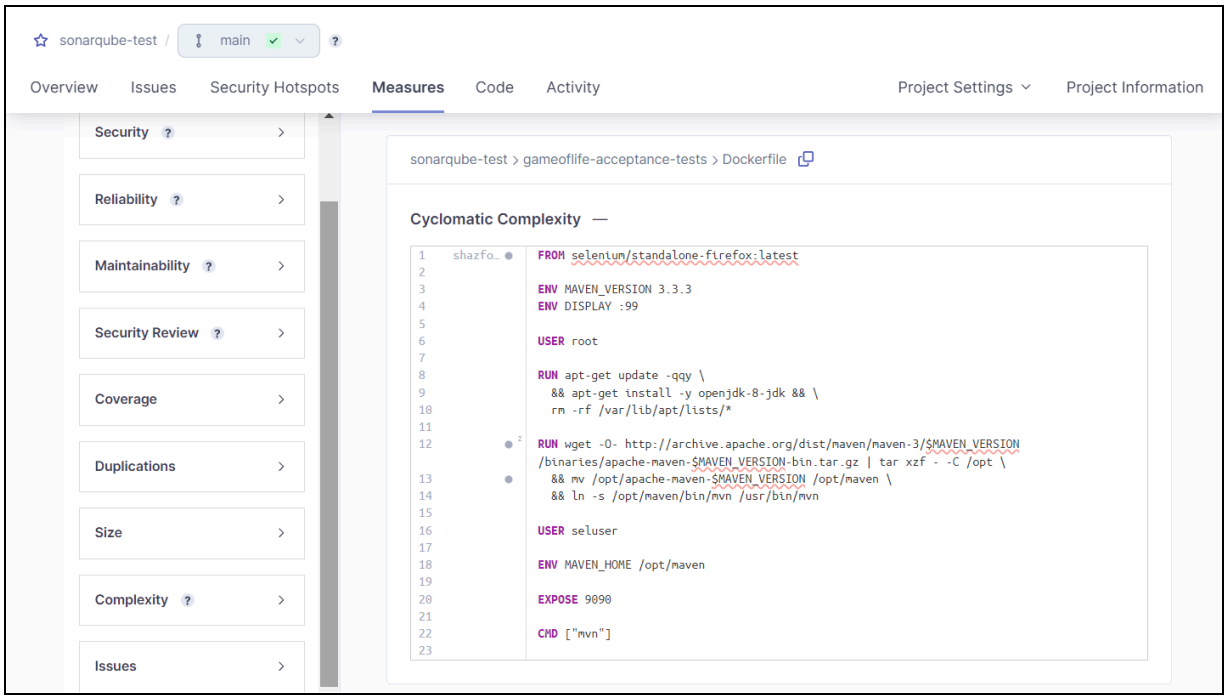



- Security Hotspot (Security Vulnerabilities) -

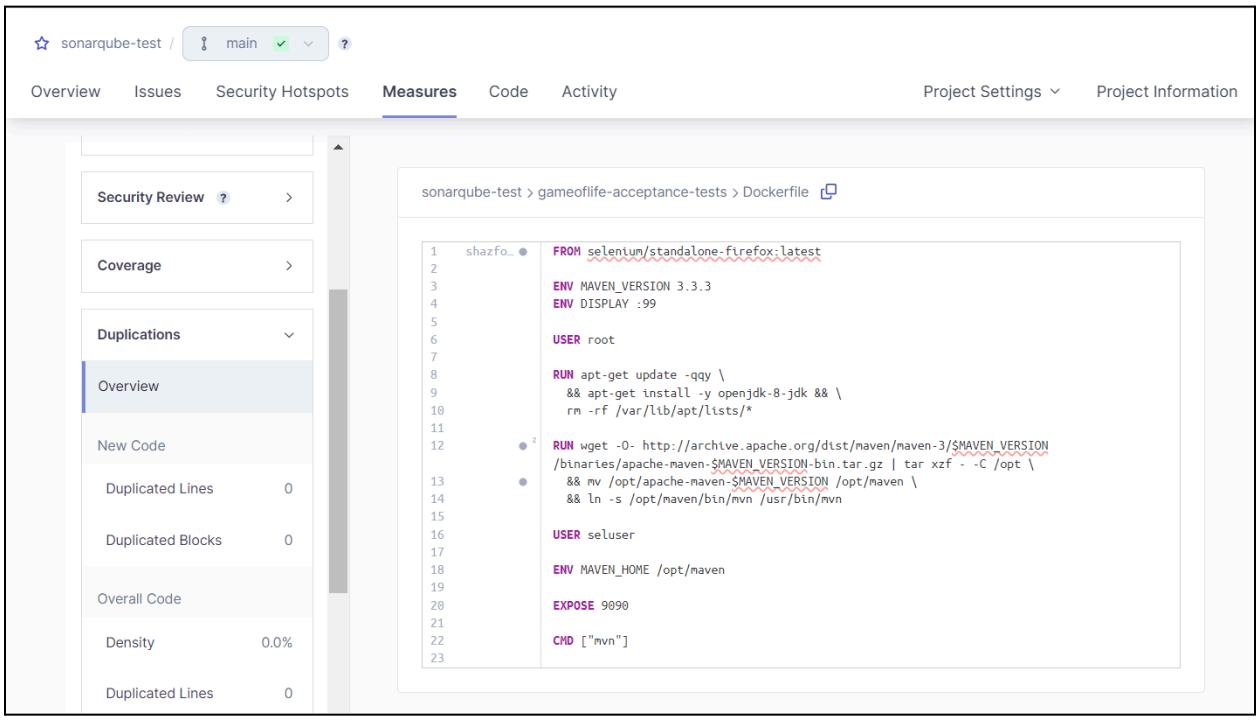



Cyclomatic Complexity -






● Duplications -




**Conclusion:**

In this experiment, we performed a static analysis of the code to detect bugs, code smells, and security vulnerabilities on our sample codes.