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.

Theory:

Static Application Security Testing (SAST)

SAST is a testing methodology that analyzes source code to find security vulnerabilities, making applications less susceptible to attacks. It scans the application before the code is compiled, also known as white-box testing.

Problems SAST Solves:

- Early Detection: Identifies vulnerabilities early in the SDLC, allowing developers to resolve issues without breaking builds or passing vulnerabilities to the final release.
- Real-Time Feedback: Provides developers with immediate feedback as they code, helping them fix issues before moving to the next phase.
- Graphical Representations: Offers visual aids to navigate code, pinpointing exact locations of vulnerabilities and providing guidance on fixes.
- Regular Scanning: Should be run regularly, such as during daily/monthly builds, code check-ins, or code releases.

Importance of SAST

- Resource Efficiency: Developers outnumber security staff, making it challenging to perform manual code reviews. SAST tools can analyze 100% of the codebase quickly.
- Speed: Can scan millions of lines of code in minutes, identifying critical vulnerabilities like buffer overflows, SQL injection, and cross-site scripting with high confidence.

CI/CD Pipeline

A CI/CD pipeline refers to the Continuous Integration/Continuous Delivery pipeline, which is the backbone of the DevOps approach. It involves a series of tasks connected in sequence to facilitate quick software releases. The pipeline is responsible for building code, running tests, and deploying new software versions.

SonarQube

SonarQube is an open-source platform developed by SonarSource for continuous inspection of code quality. It performs static code analysis, providing detailed reports on bugs, code smells, vulnerabilities, and code duplications. It supports over 25 major programming languages and can be extended with various plugins.

Benefits of SonarQube:

- Sustainability: Reduces complexity, vulnerabilities, and code duplications, optimizing application lifespan.
- Increased Productivity: Lowers maintenance costs and risks, reducing the need for extensive code changes.
- Quality Code: Ensures code quality control is an integral part of software development.
- Error Detection: Automatically detects errors and alerts developers to fix them before output submission.
- Consistency: Identifies code criteria breaches, enhancing overall code quality.
- Business Scaling: Supports scaling without 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

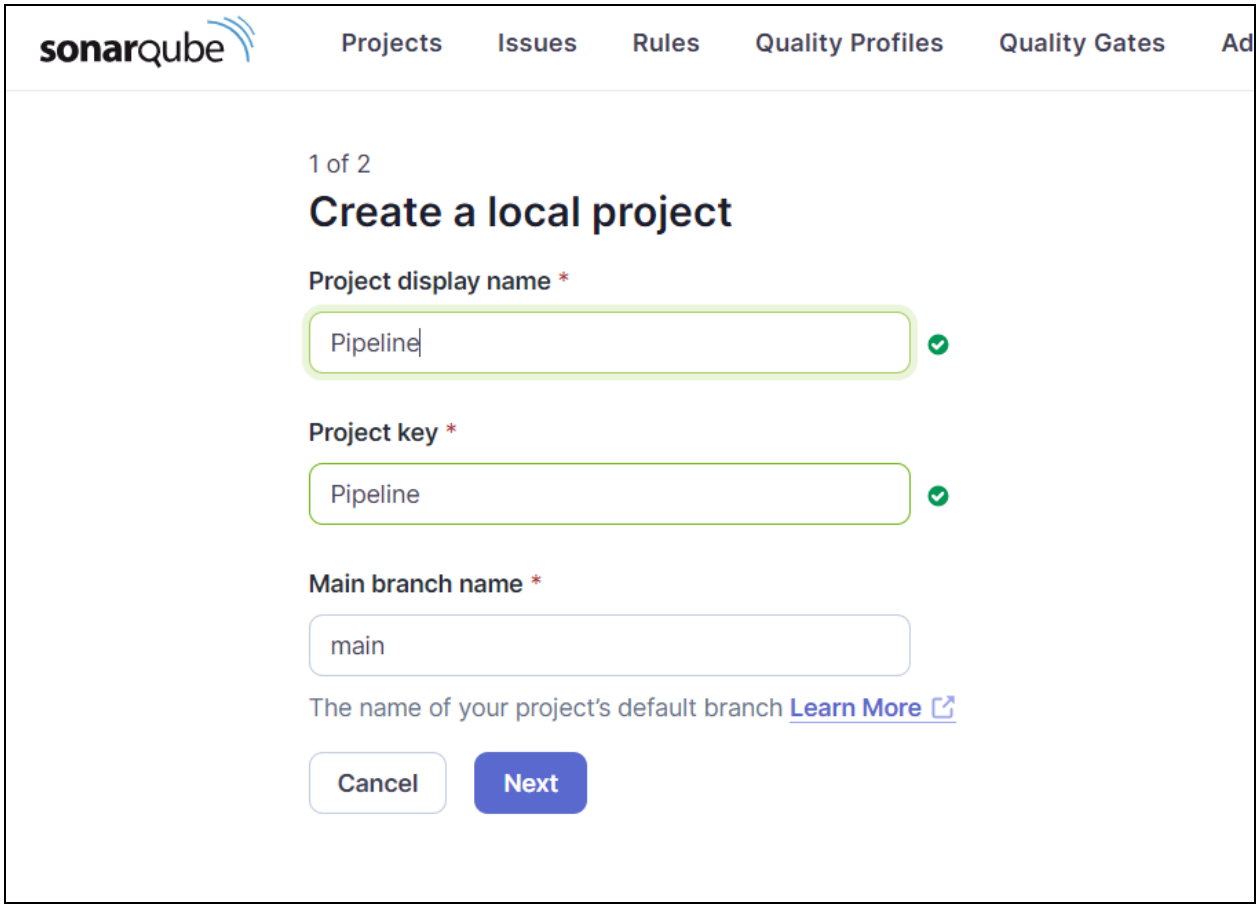
2. Run SonarQube in a Docker Container
    - Open a terminal and run the following command to start SonarQube in a Docker container

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

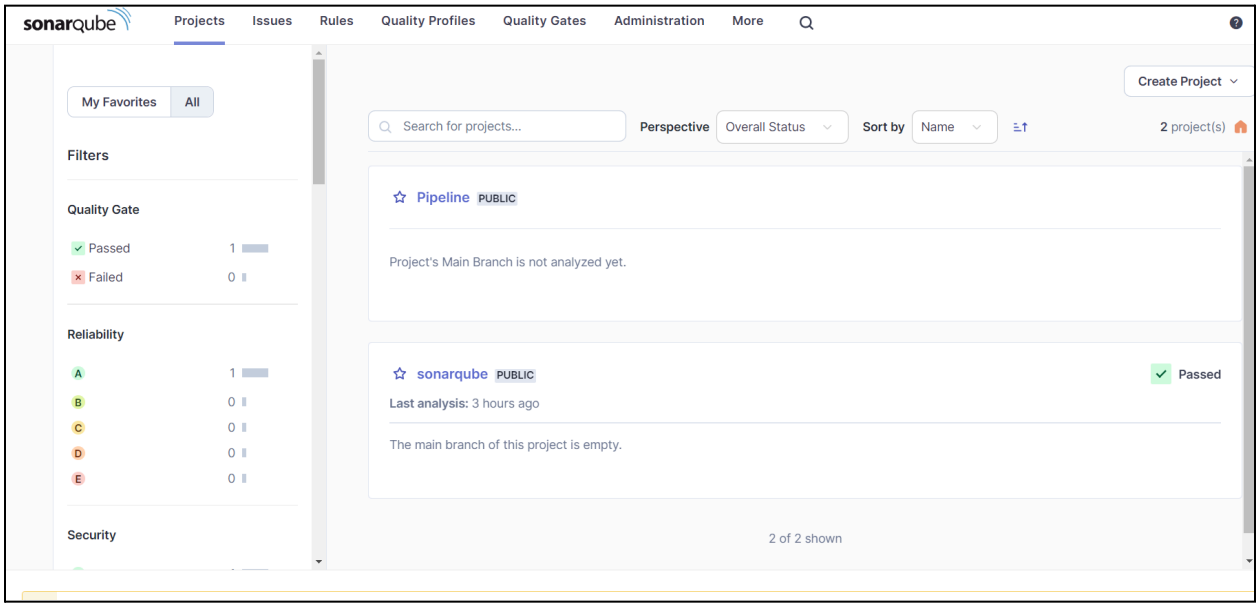
```
C:\Users\272241>docker run -d --name sonarqube -e SONAR_ES_BOOTSTRAP_CHECKS_
DISABLE=true -p 9000:9000 sonarqube:latest
Unable to find image 'sonarqube:latest' locally
latest: Pulling from library/sonarqube
7478e0ac0f23: Pull complete
90a925ab929a: Pull complete
7d9a34308537: Pull complete
80338217a4ab: Pull complete
1a5fd5c7e184: Pull complete
7b87d6fa783d: Pull complete
bd819c9b5ead: Pull complete
4f4fb700ef54: Pull complete
Digest: sha256:72e9feec71242af83faf65f95a40d5e3bb2822a6c3b2cda8568790f3d31ae
cde
Status: Downloaded newer image for sonarqube:latest
8b62aeca4d09887a0db32d349116529581a639b59222a8b20987b42d8cec6ef3
```

3. Check SonarQube Status
    - Once the container is up and running, check the status of SonarQube by navigating to http://localhost:9000

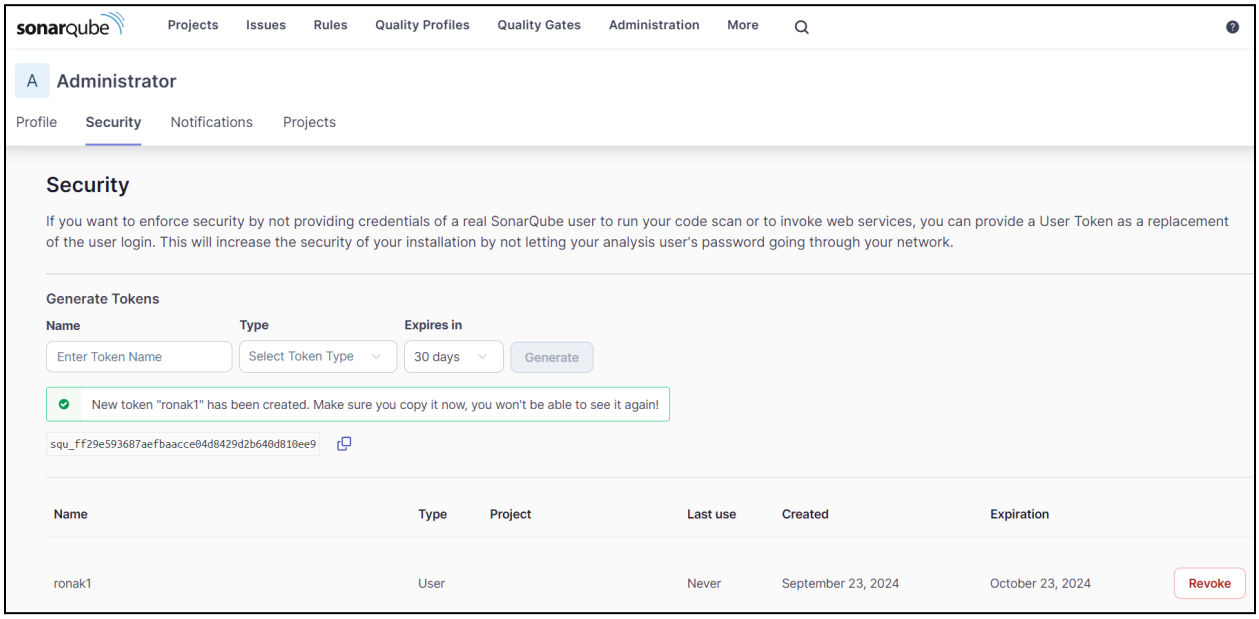
4. Login to SonarQube
    - Use the credentials to log in:



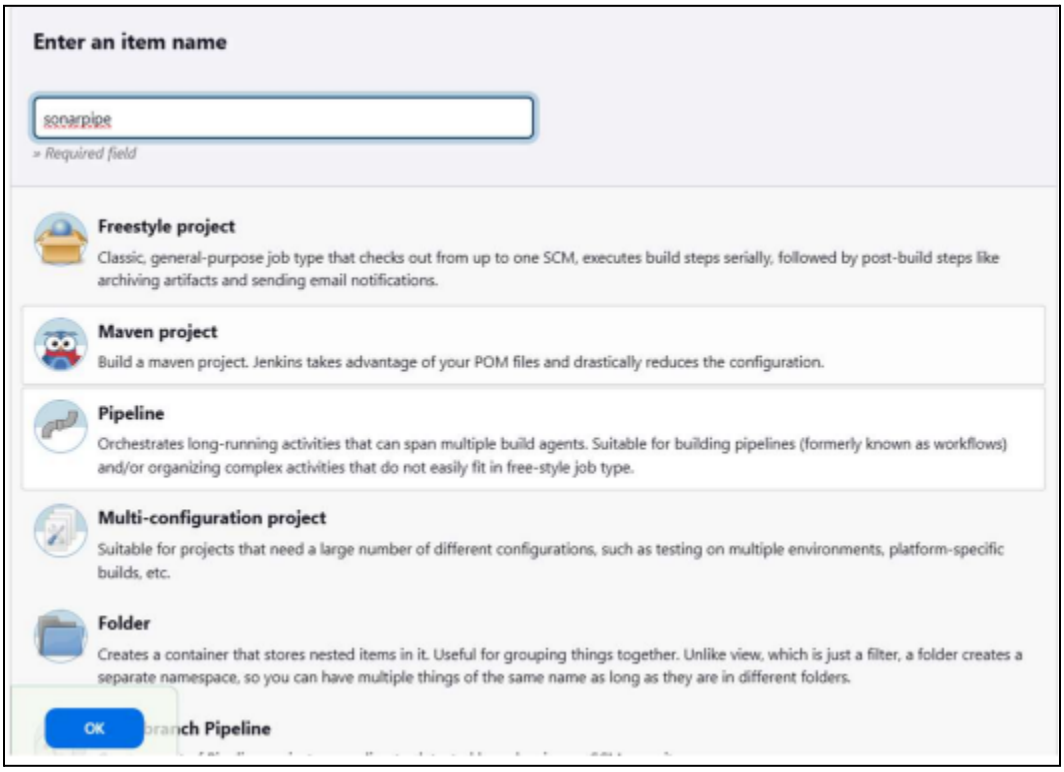
5. You can view the project:



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

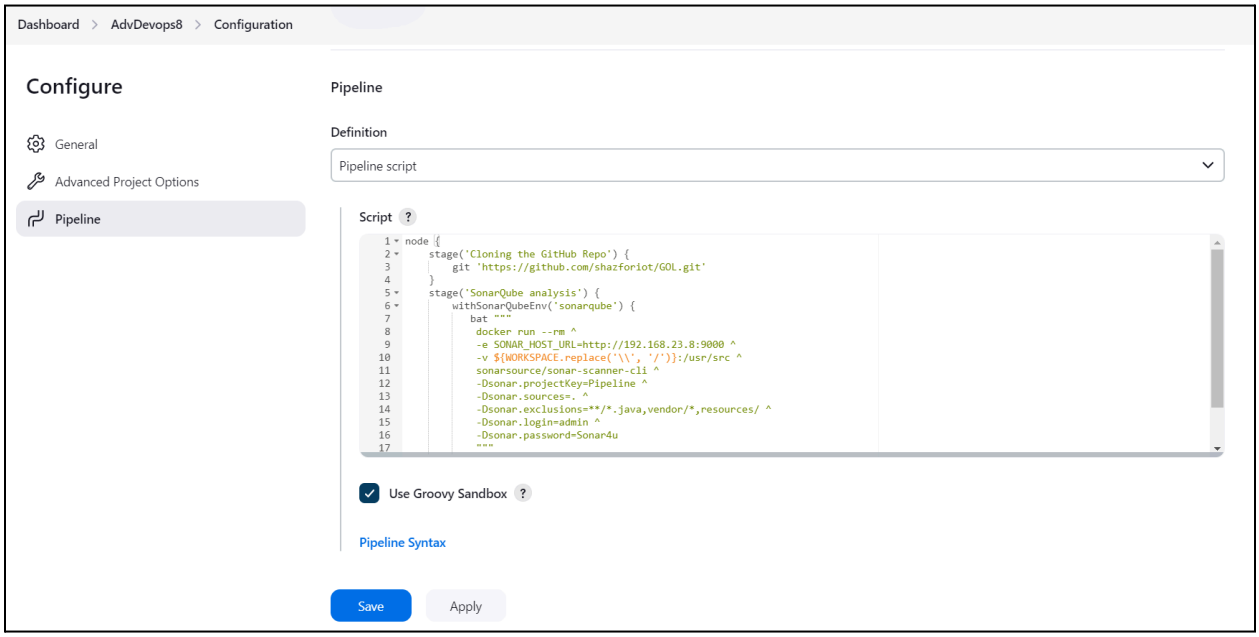


7. Create a Jenkins Pipeline
    - Go to Jenkins Dashboard, click New Item, and select Pipeline.



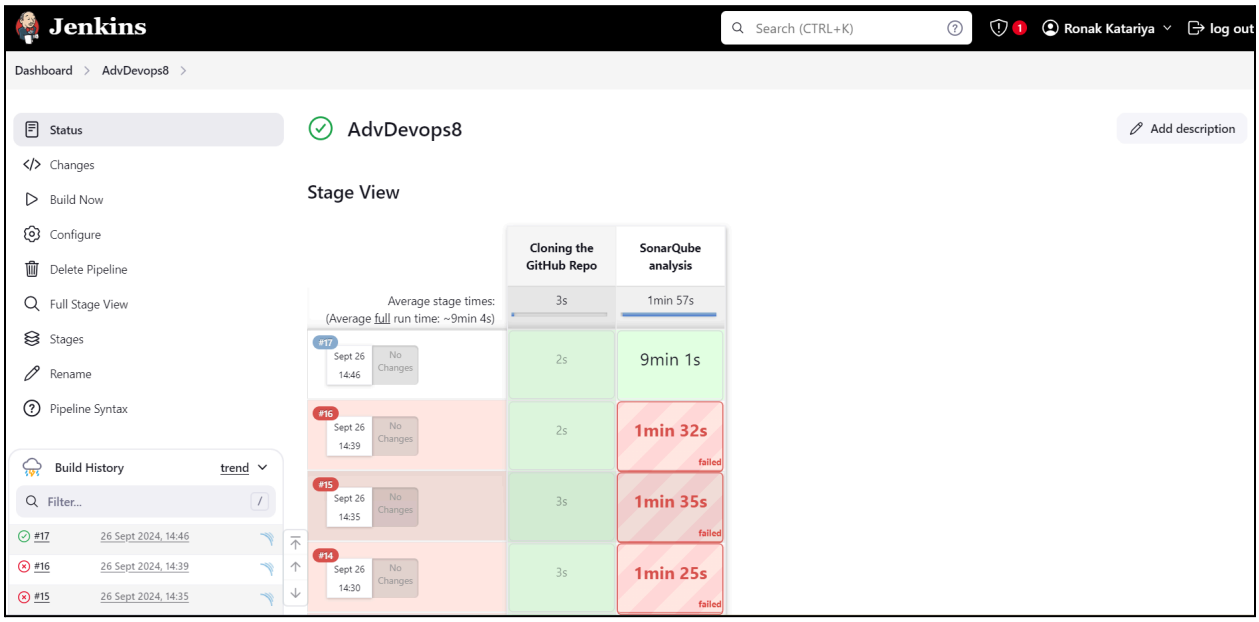
8. Under Pipeline Script, enter the following script:

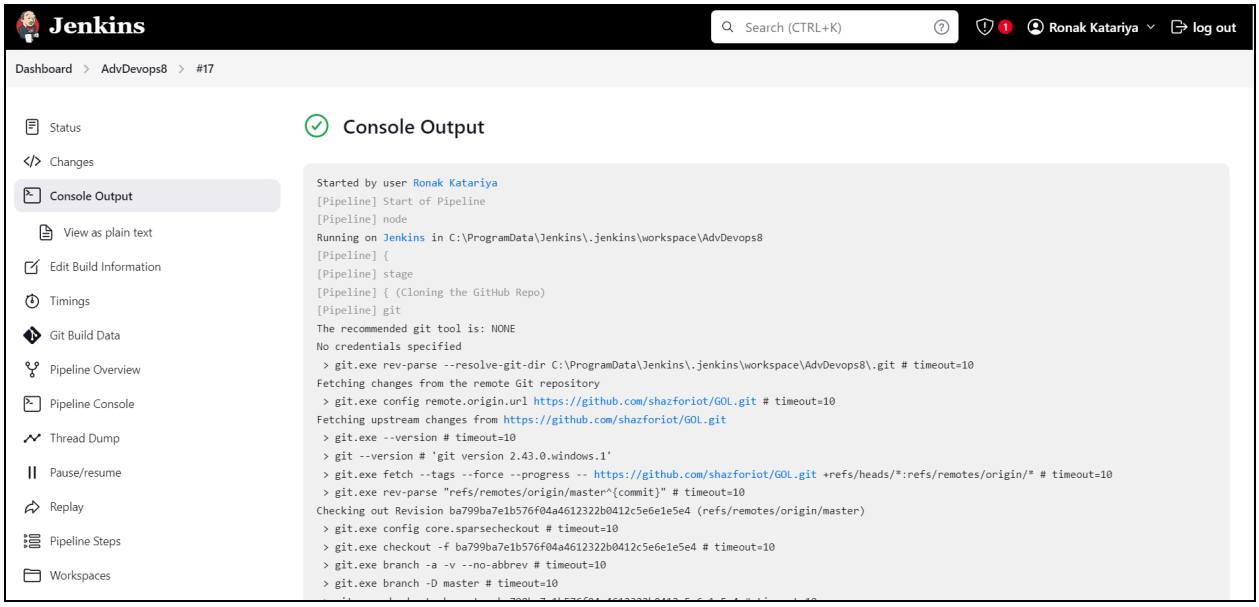
```
docker network create sonarnet
node {
 stage('Cloning the GitHub Repo') {
 git 'https://github.com/shazforiot/GOL.git'
 }
 stage('SonarQube analysis') {
 withSonarQubeEnv('sonarqube') {
 bat """
 docker run --rm --network host \
 -e SONAR_HOST_URL=http://<ip_address>:9000 \
 -e SONAR_LOGIN=admin \
 -e SONAR_PASSWORD=<Sonarqube_password> \
 -e SONAR_PROJECT_KEY=sonarqube-test \
 -v ${WORKSPACE.replace('\\', '/')}:/usr/src \
 sonarsource/sonar-scanner-cli \
 -Dsonar.projectKey=sonarqube-test \
 -Dsonar.exclusions=vendor/**,resources/**,**/*.java \
 -Dsonar.login=admin \
 -Dsonar.password=<Sonarqube_password>
 """
 }
 }
}
```



9. Run the Pipeline

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



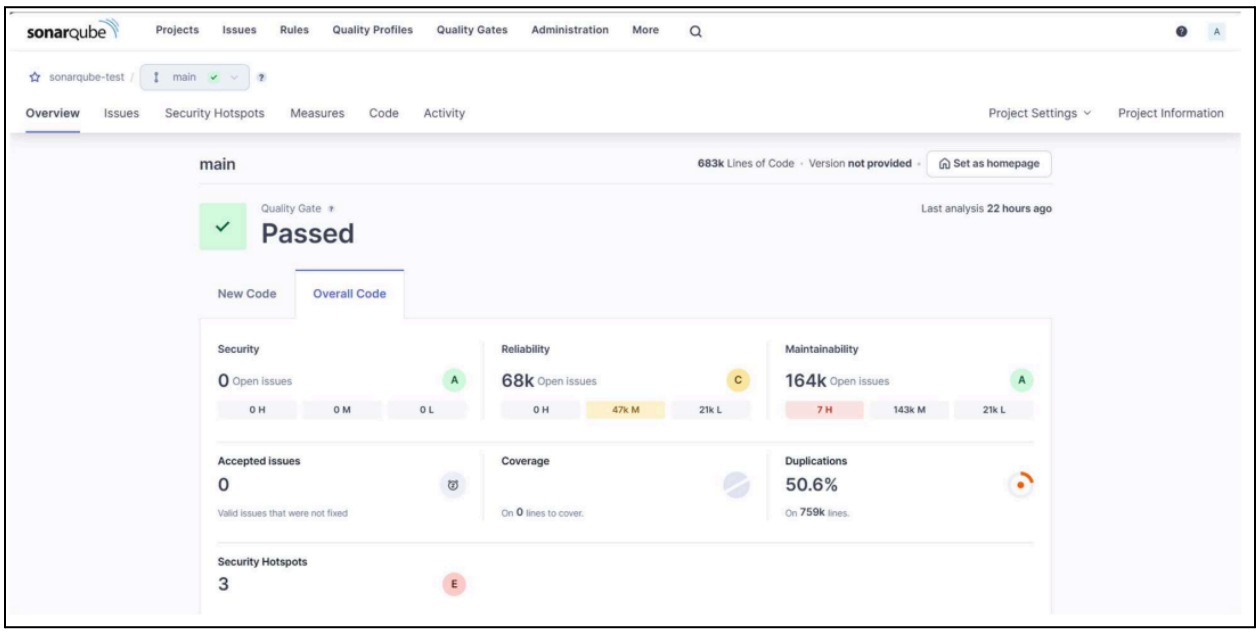


Dashboard > AdvDevops8 > #17

```
09:24:35.453 WARN  Too many duplication references on file gameoflife-web/tools/jmeter/docs/api/org/apache/jmeter/assertions/BeanShellAssertion.html
for block at line 41. Keep only the first 100 references.
09:24:35.453 WARN  Too many duplication references on file gameoflife-web/tools/jmeter/docs/api/org/apache/jmeter/assertions/BeanShellAssertion.html
for block at line 17. Keep only the first 100 references.
09:24:35.453 WARN  Too many duplication references on file gameoflife-web/tools/jmeter/docs/api/org/apache/jmeter/assertions/BeanShellAssertion.html
for block at line 698. Keep only the first 100 references.
09:24:35.453 WARN  Too many duplication references on file gameoflife-web/tools/jmeter/docs/api/org/apache/jmeter/assertions/BeanShellAssertion.html
for block at line 75. Keep only the first 100 references.
09:24:35.454 INFO  CPD Executor CPD calculation finished (done) | time=119165ms
09:24:35.488 INFO  SCM revision ID 'ba799ba7e1b576f04a4612322b0412c5e6e1e5e4'
09:24:50.535 INFO  Analysis report generated in 14994ms, dir size=127.2 MB
09:24:59.094 INFO  Analysis report compressed in 8556ms, zip size=29.6 MB
09:25:01.759 INFO  Analysis report uploaded in 2664ms
09:25:01.760 INFO  ANALYSIS SUCCESSFUL, you can find the results at: http://192.168.23.8:9000/dashboard?id=Pipeline
09:25:01.760 INFO  Note that you will be able to access the updated dashboard once the server has processed the submitted analysis report
09:25:01.761 INFO  More about the report processing at http://192.168.23.8:9000/api/ce/task?id=f8f82a37-2582-451e-8839-ef19560deada
09:25:03.386 INFO  Analysis total time: 8:41.698 s
09:25:03.389 INFO  SonarScanner Engine completed successfully
09:25:03.783 INFO  EXECUTION SUCCESS
09:25:03.785 INFO  Total time: 8:57.529s
[Pipeline] }
WARN: Unable to locate 'report-task.txt' in the workspace. Did the SonarScanner succeed?
[Pipeline] // withSonarQubeEnv
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

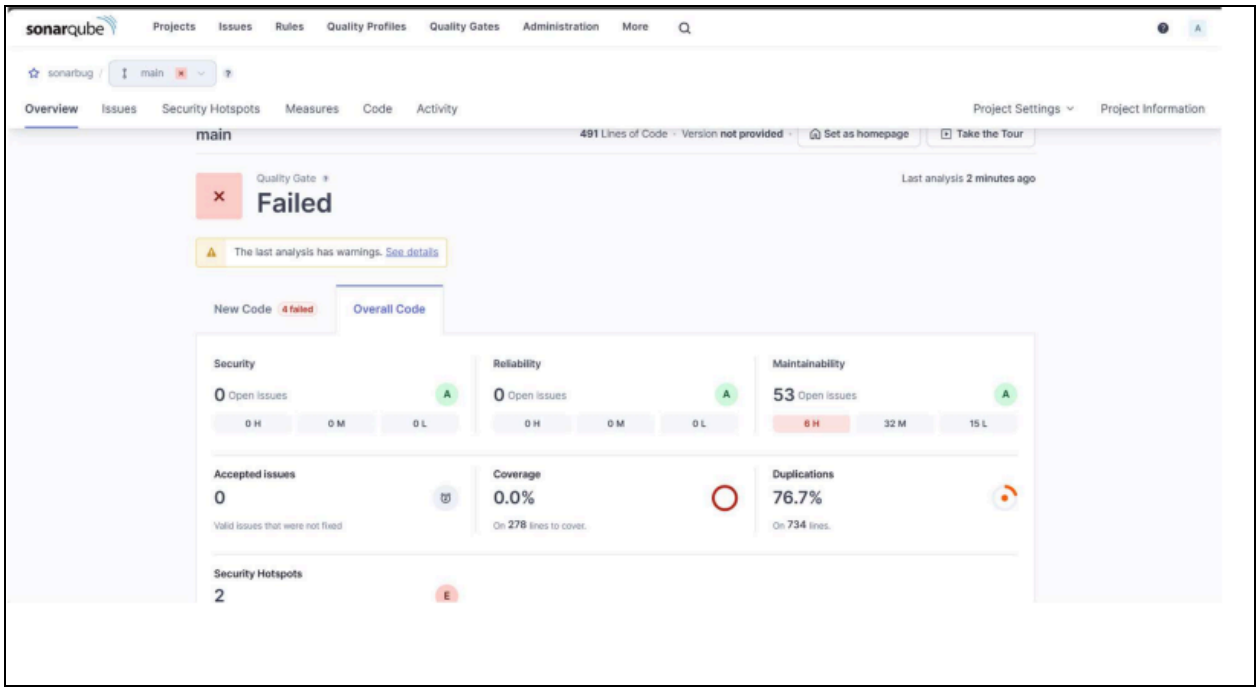
10. Check SonarQube for Analysis Results

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



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

- Overview -



Conclusion:

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