EXPERIMENT NO. 7

NAME : PRANAV POL CLASS : D15A ROLL NO. : 42

Aim: To understand Static Analysis SAST process and learn to integrate Jenkins SAST to SonarQube/GitLab.

Theory:

Static Application Security Testing (SAST)

SAST is a method of security testing that analyzes source code to identify vulnerabilities without executing the program. It is also known as white-box testing. Here's a breakdown of the SAST process:

1. **Code Parsing**: The source code is parsed to create an abstract syntax tree (AST), which represents the code structure.
2. **Pattern Matching**: The AST is analyzed using predefined rules to detect patterns that may indicate security vulnerabilities.
3. **Data Flow Analysis**: This step examines how data moves through the code to identify potential security issues like SQL injection or cross-site scripting (XSS).
4. **Control Flow Analysis**: This involves analyzing the paths that the code execution might take to find logical errors or vulnerabilities.
5. **Reporting**: The tool generates a report highlighting the vulnerabilities found, their severity, and recommendations for fixing them.

Benefits of SAST

- **Early Detection**: Identifies vulnerabilities early in the development lifecycle, reducing the cost and effort required to fix them.
- **Comprehensive Coverage**: Can analyze 100% of the codebase, including all possible execution paths.
- **Automated and Scalable**: Suitable for large codebases and can be integrated into CI/CD pipelines for continuous monitoring.

SonarQube and SAST

SonarQube is a popular tool that provides static code analysis to detect bugs, code smells, and security vulnerabilities. Here's how SonarQube fits into the SAST process:

1. **Integration**: SonarQube can be integrated into your CI/CD pipeline to automatically analyze code every time it is committed.
2. **Rule Sets**: It uses a comprehensive set of rules to detect security vulnerabilities, coding standards violations, and code quality issues.

3. **Dashboards and Reports**: SonarQube provides detailed dashboards and reports that help developers understand and fix issues.
4. **Continuous Improvement**: By continuously analyzing code, SonarQube helps maintain high code quality and security standards over time.

Implementation:

1. Open Jenkins Dashboard

- Access your Jenkins Dashboard by navigating to `http://localhost:8080` (or the port you have configured Jenkins to run on).

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

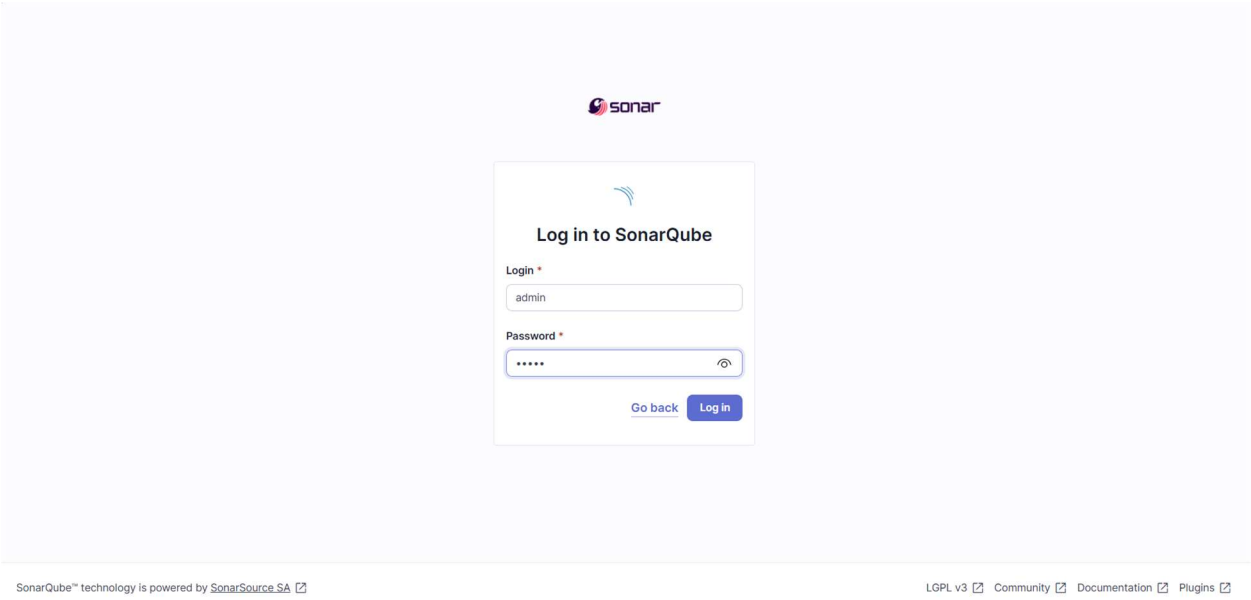
```
C:\Users\sbpol>docker run -d --name sonarqube -e SONAR_ES_BOOTSTRAP_CHECKS_DISABLE=true -p 9000:9000 sonarqube:latest
Unable to find image 'sonarqube:latest' locally
latest: Pulling from library/sonarqube
7478e0ac0f23: Pull complete
90a925ab929a: Pull complete
7d9a34308537: Pull complete
80338217a4ab: Pull complete
1a5fd5c7e184: Pull complete
7b87d6fa783d: Pull complete
bd819c9b5ead: Pull complete
4f4fb700ef54: Pull complete
Digest: sha256:72e9feec71242af83faf65f95a40d5e3bb2822a6c3b2cda8568790f3d31aecde
Status: Downloaded newer image for sonarqube:latest
2f213117ce50f08304d681a60dc0e2a4dd6c3c8e46f5725be7fb40fd0d48bb5d
```

3. Check SonarQube Status

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

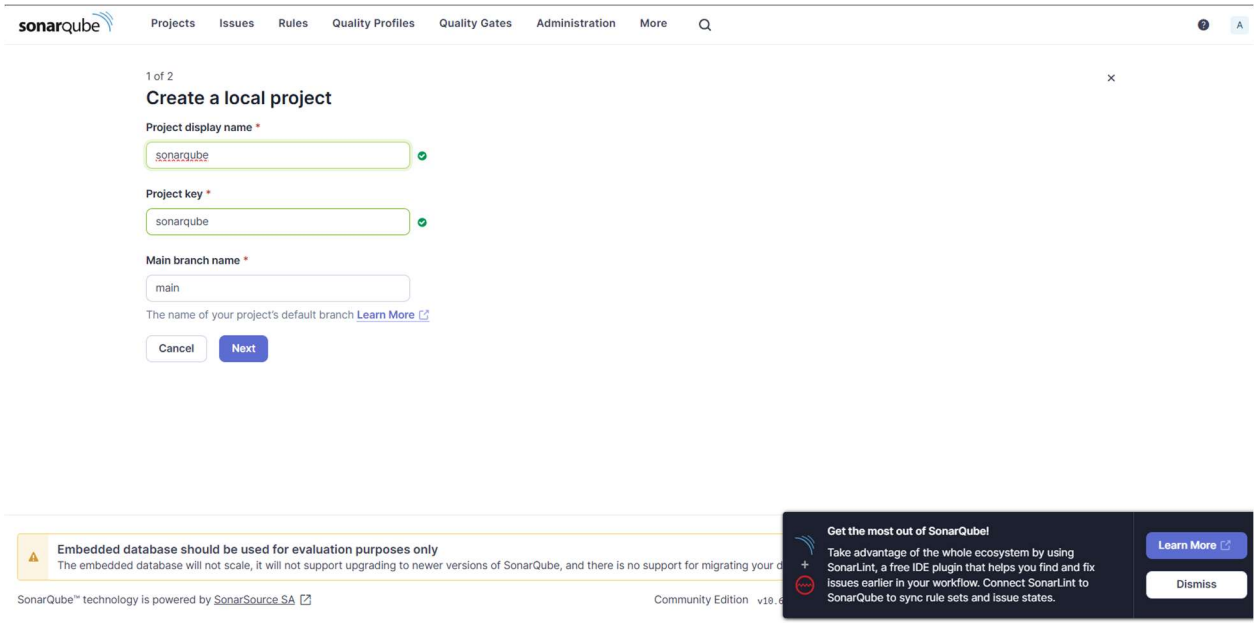
4. Login to SonarQube

- Use the default credentials to log in:
  - **Username:** `admin`
  - **Password:** `admin`



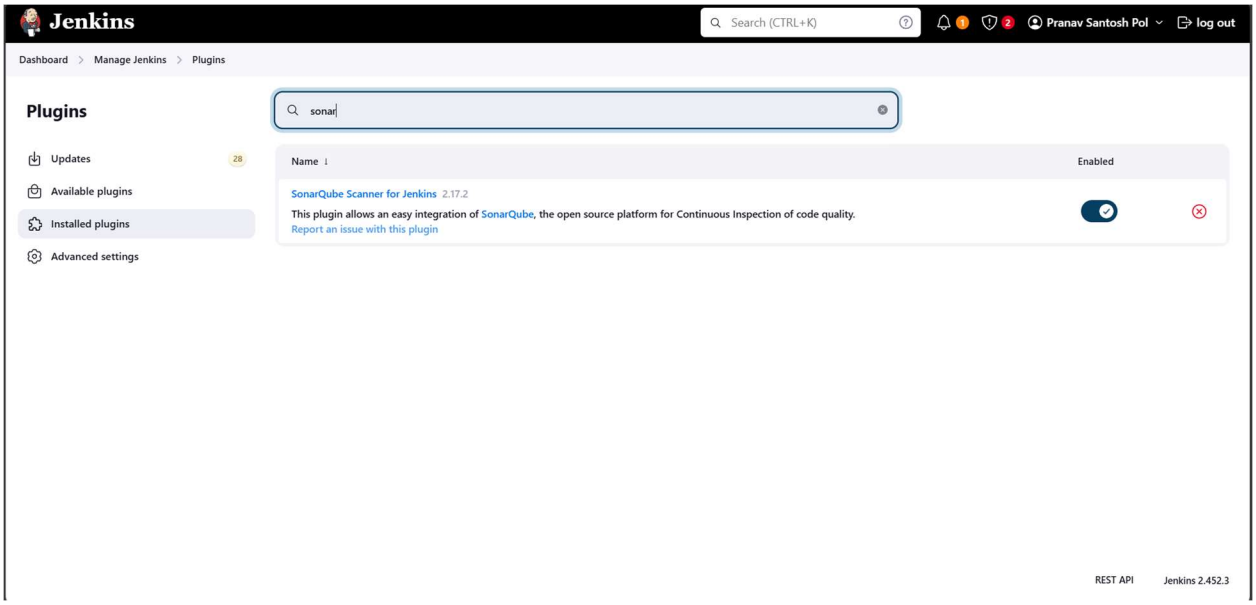
5. Create a Project in SonarQube

- Create a new project manually in SonarQube and name it `sonarqube`.



6. Install SonarQube Scanner for Jenkins

- Go back to the Jenkins Dashboard.
- Navigate to `Manage Jenkins` > `Manage Plugins`.
- Search for `SonarQube Scanner for Jenkins` and install it.

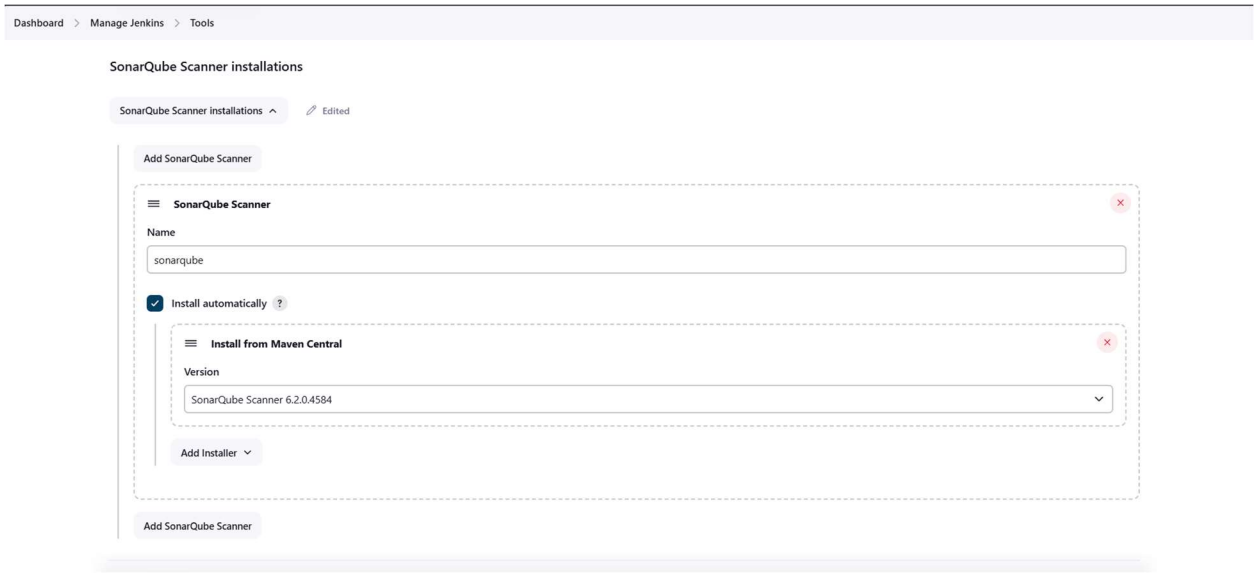


7. Configure SonarQube in Jenkins

- Go to `Manage Jenkins` > `Configure System`.
- Scroll down to the `SonarQube Servers` section and enter the required details:
  - **Name:** Any name you prefer.
  - **Server URL:** `http://localhost:9000`
  - **Server Authentication Token:** (Generate this token in SonarQube under `My Account` > `Security` > `Generate Tokens`).
  - **Add Jenkins**: Select Kind - Secret Text > Secret (Paste Generated Token)

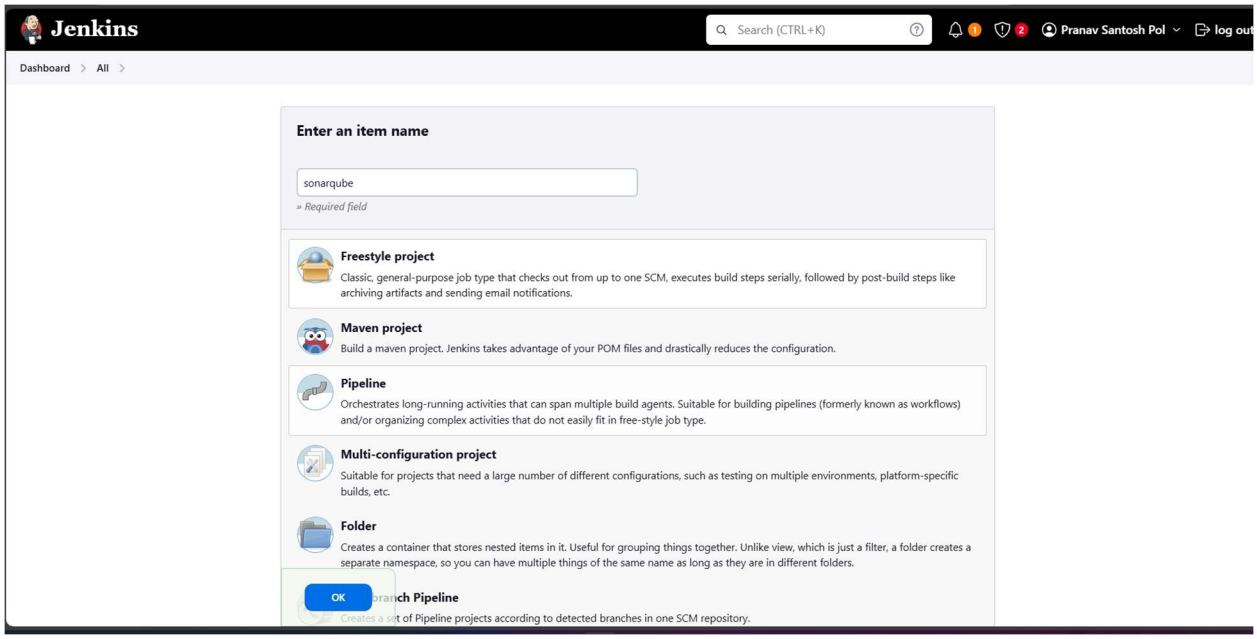
8. Configure SonarQube Scanner in Jenkins

- Go to `Manage Jenkins` > `Global Tool Configuration`.
- Scroll down to `SonarQube Scanner`.
- Choose the latest version and select `Install automatically`



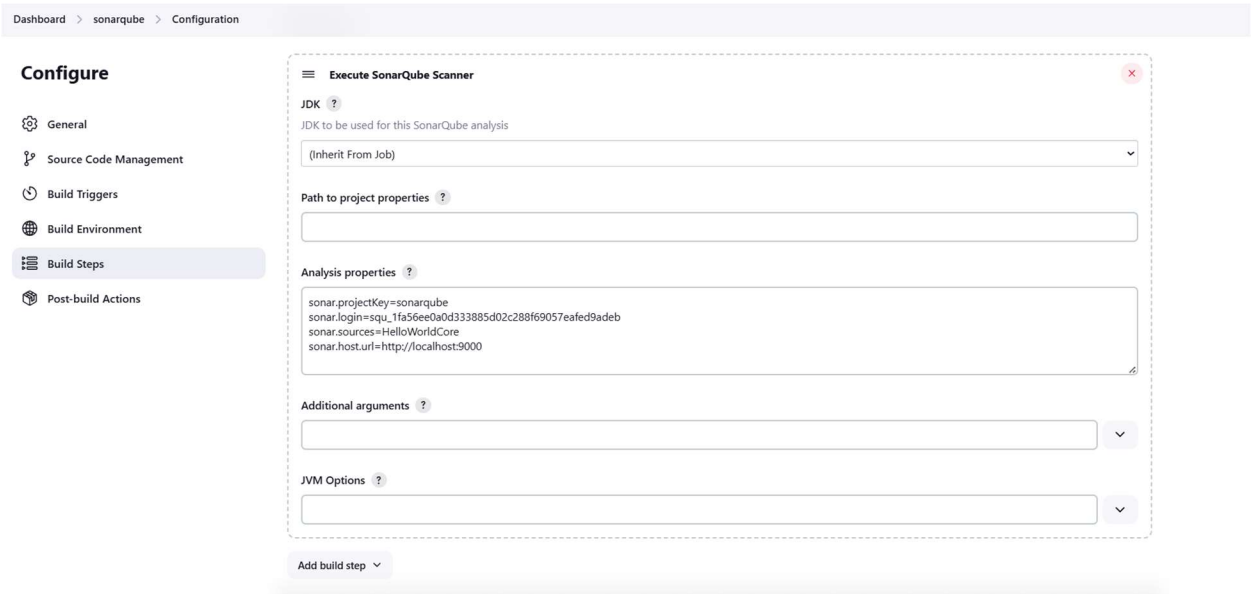
9. Create a New Jenkins Job

- In Jenkins, create a new item and select `Freestyle project`.
- Under `Source Code Management`, choose `Git` and enter the repository URL:
- https://github.com/shazforiot/MSBuild_firstproject.git



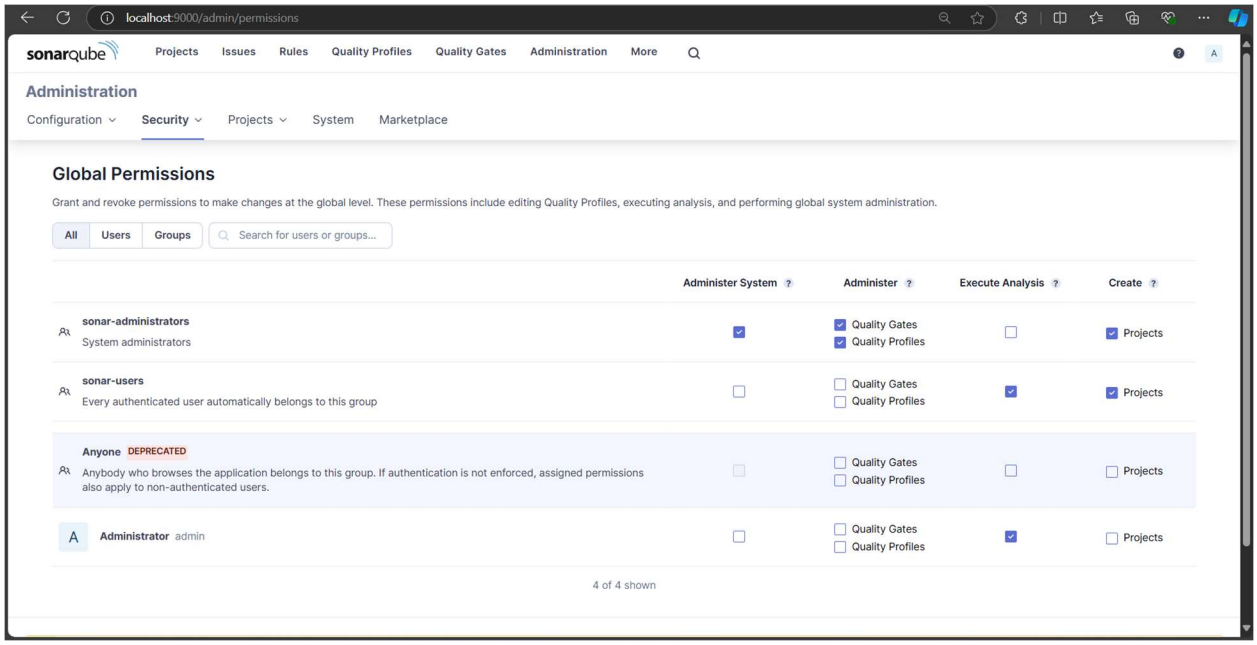
10. Configure Build Steps

- Under the `Build` section, add a build step to `Execute SonarQube Scanner`.
- Enter the following analysis properties:
  - sonar.projectKey=my_project_name
  - sonar.login=your_generated_token
  - sonar.sources=HelloWorldCore
  - sonar.host.url=http://localhost:9000



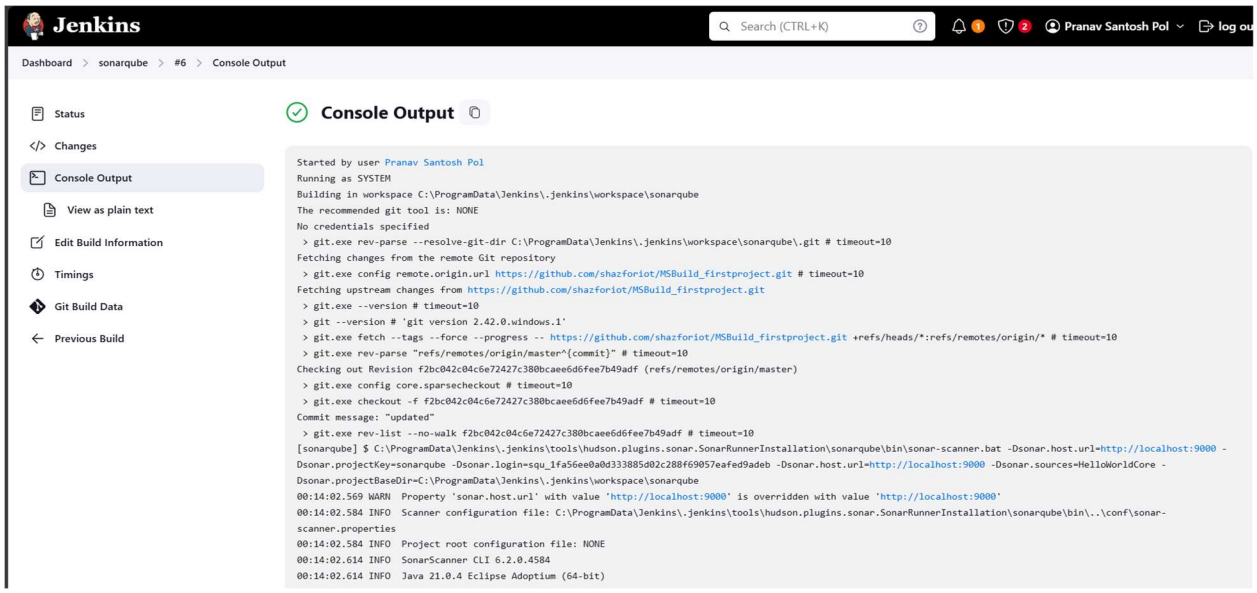
11. Set Permissions in SonarQube

- Navigate to `http://localhost:9000/<user_name>/permissions`.
- Allow `Execute Permissions` to the `Admin` user.



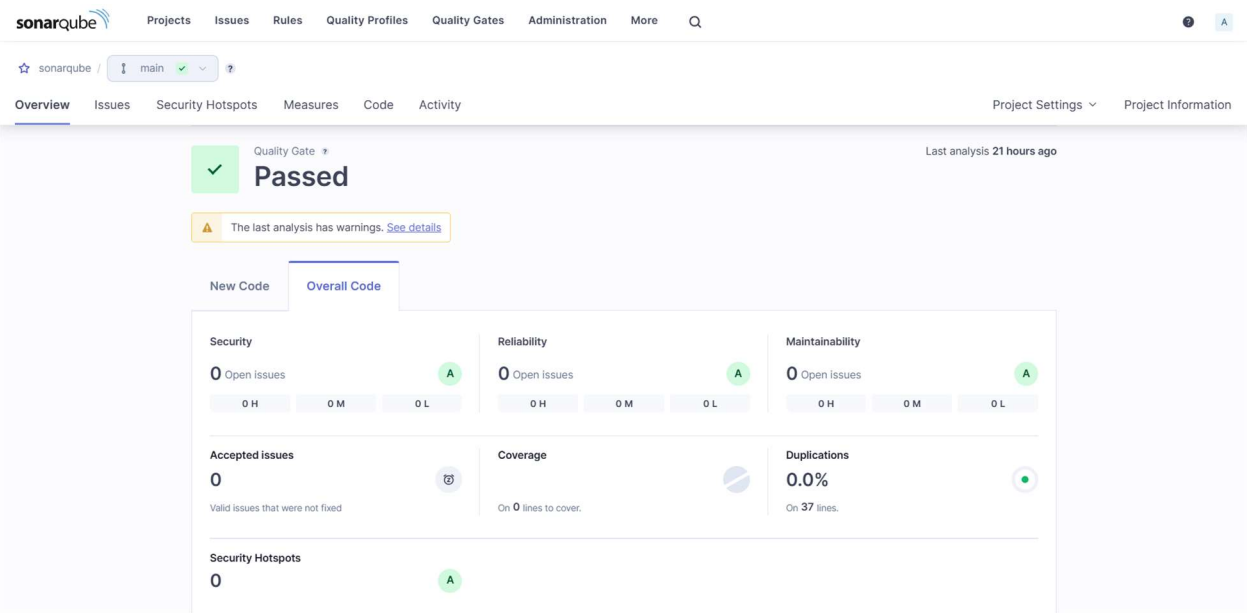
12. Run the Build

- Go back to Jenkins and run the build.
- Check the console output for any errors or issues.



13. Verify in SonarQube

- Once the build is complete, check the project in SonarQube to see the analysis results.



Conclusion: In this experiment, we have understood the importance of SAST and have successfully integrated Jenkins with SonarQube for Static Analysis and Code Testing.