

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
```

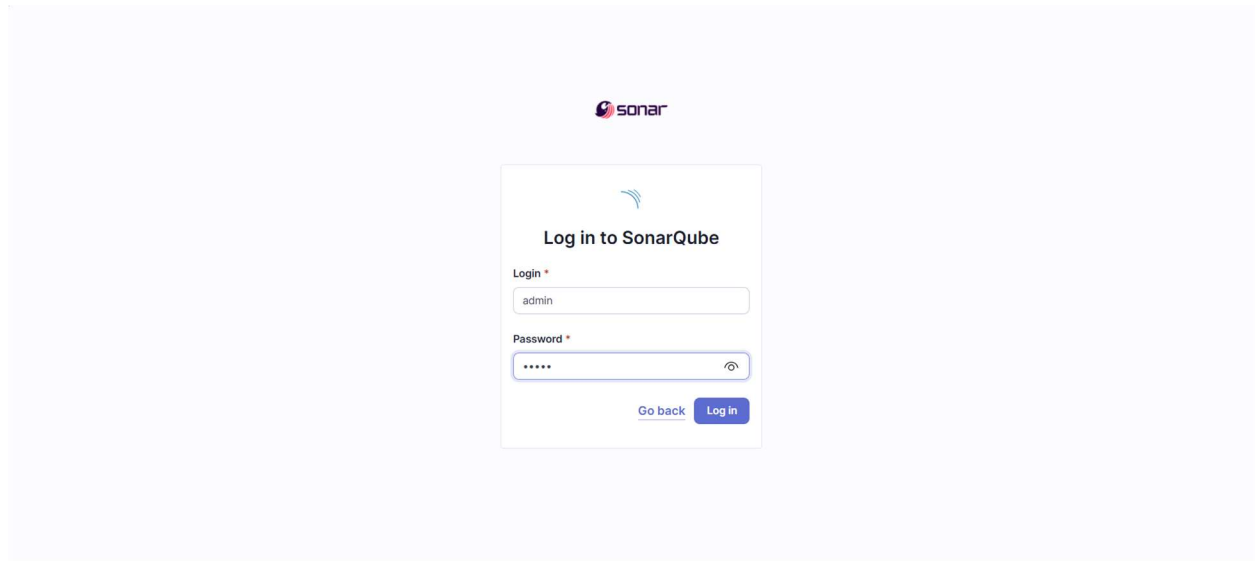
```
C:\Users\sbpol>docker run -d --name sonarqube -e SONAR_ES_BOOTSTRAP_CHECKS_DISABLE=true -p 9000:9000 sonarqube:latest
Unable to find image 'sonarqube:latest' locally
latest: Pulling from library/sonarqube
7478e0ac0f23: Pull complete
90a925ab929a: Pull complete
7d9a34308537: Pull complete
80338217a4ab: 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`.

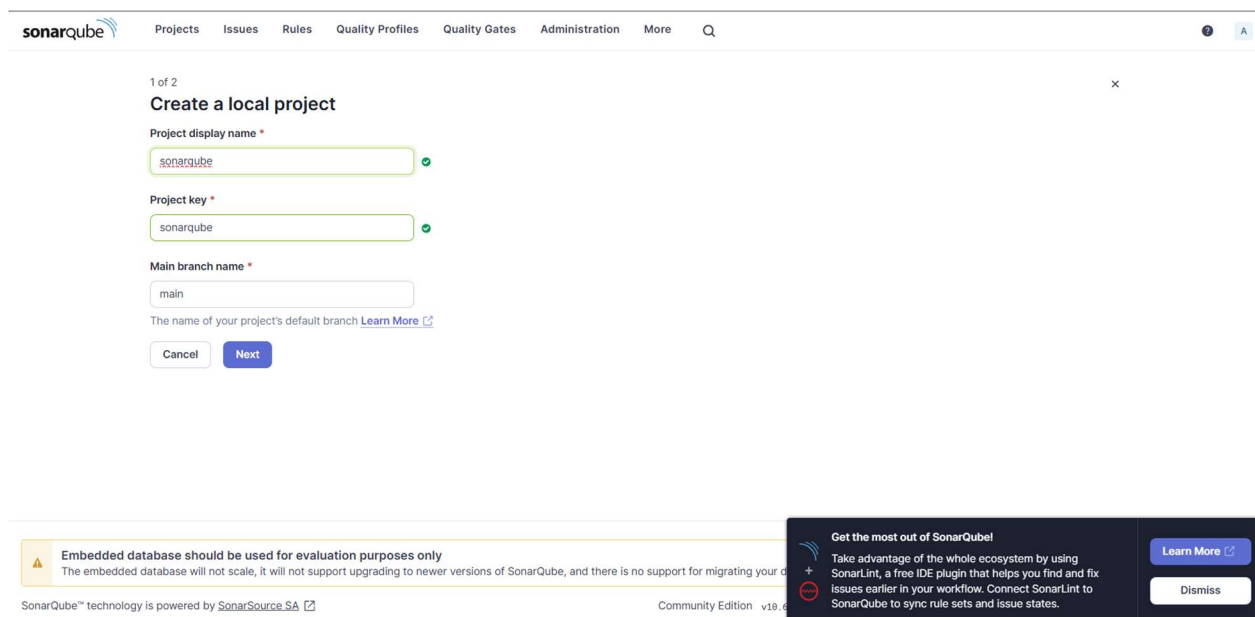
4. Login to SonarQube

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



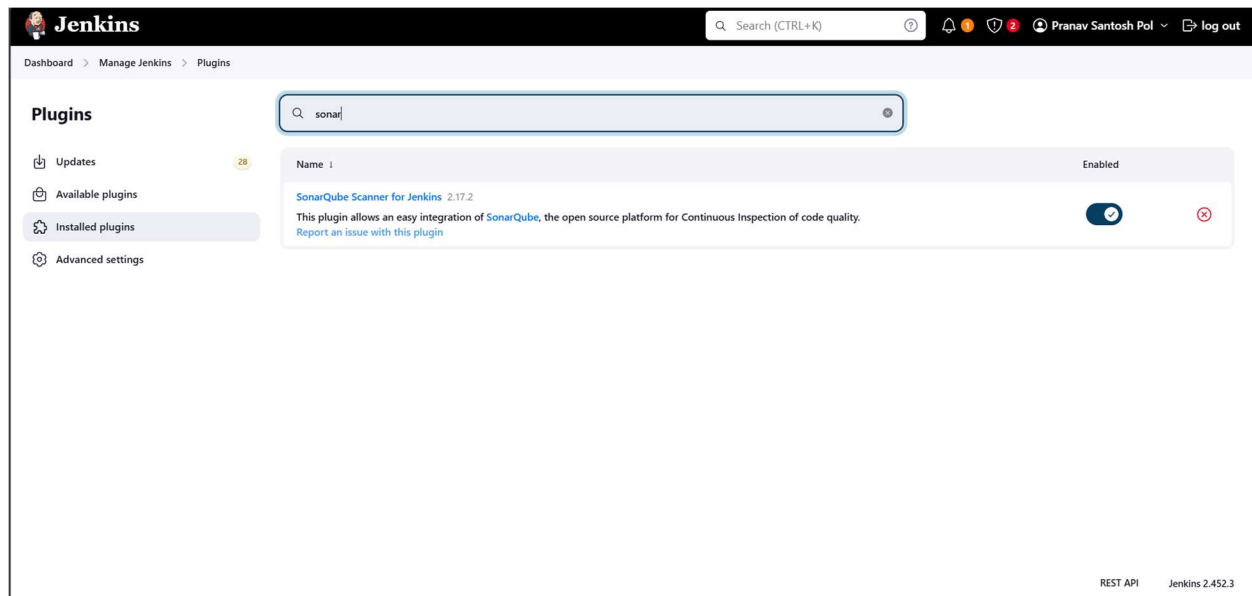
5. Create a Project in SonarQube

- 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)

SonarQube servers

If checked, job administrators will be able to inject a SonarQube server configuration as environment variables in the build.

☐ Environment variables

SonarQube installations

List of SonarQube installations

Name

Server URL

Default is http://localhost:9000

Server authentication token

SonarQube authentication token. Mandatory when anonymous access is disabled.

+ Add +

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

Dashboard > Manage Jenkins > Tools

SonarQube Scanner installations

SonarQube Scanner installations ^ Edited

Add SonarQube Scanner

SonarQube Scanner

Name

☒ Install automatically ?

Install from Maven Central

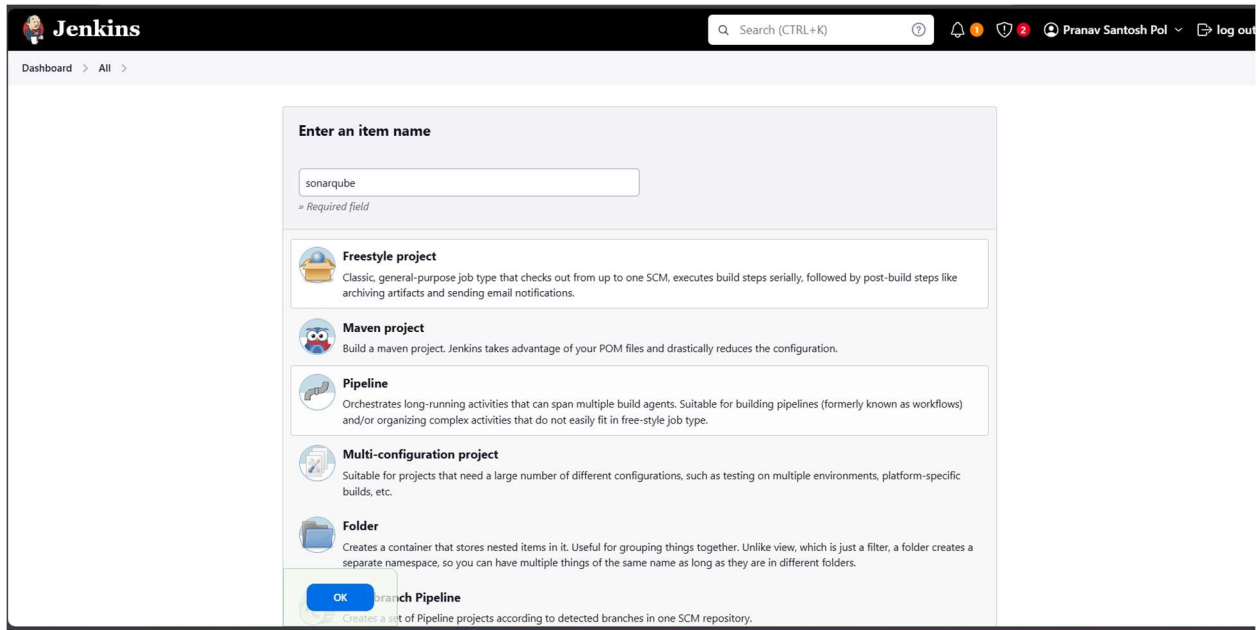
Version

+ Add Installer +

Add SonarQube Scanner

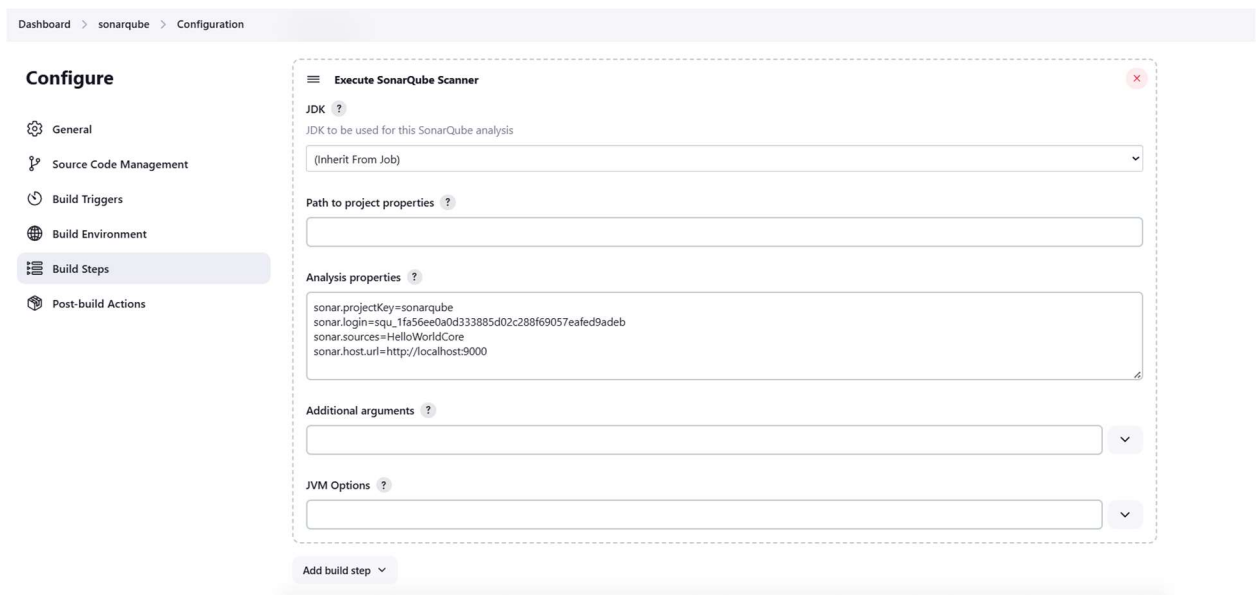
9. Create a New Jenkins Job

- 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



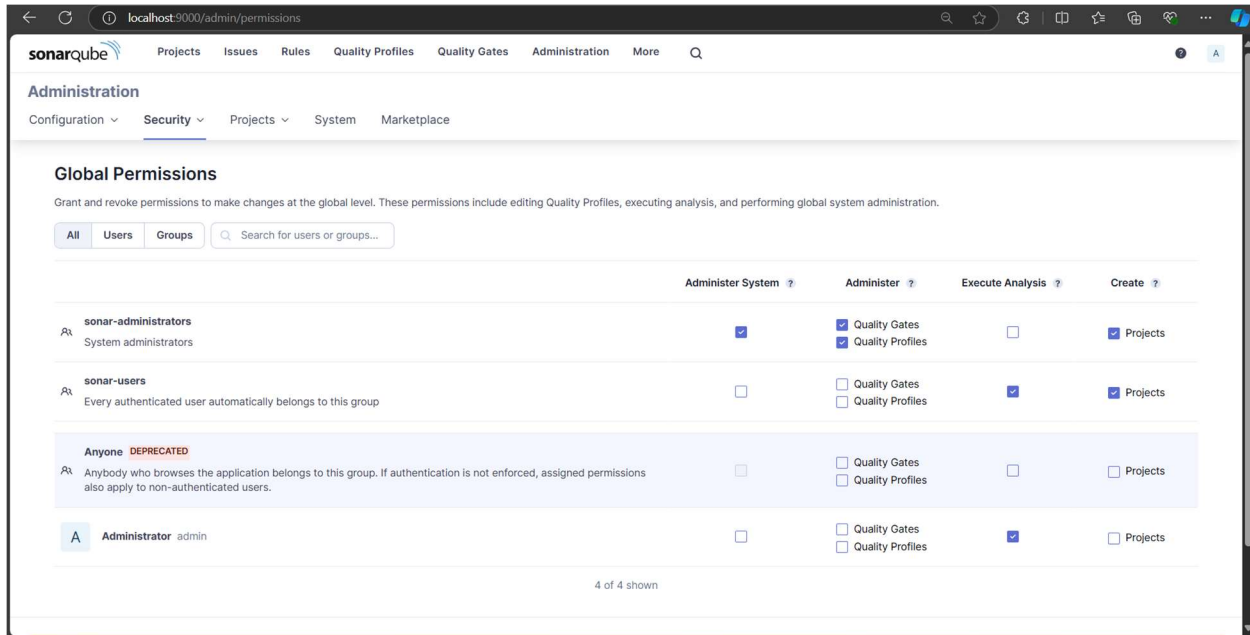
10. Configure Build Steps

- 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.



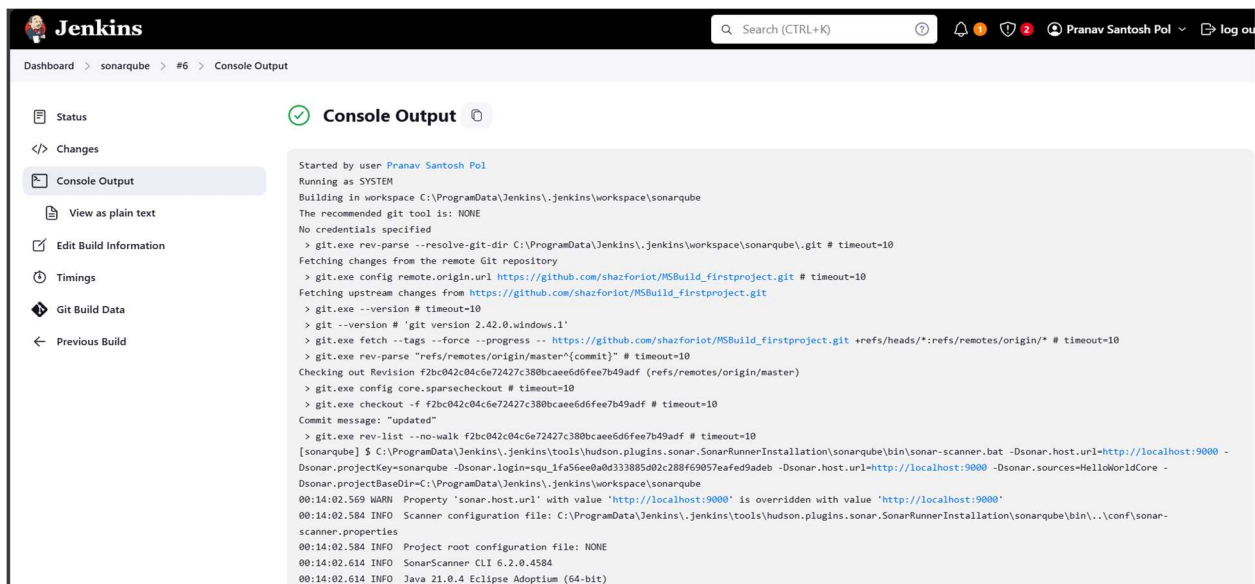
The screenshot shows the SonarQube Administration interface. The 'Security' tab is selected under 'Administration'. The 'Global Permissions' section is active, showing a table of permissions for different user groups. The 'Administrator' user is highlighted, and the 'Execute Analysis' permission is checked for them.

	Administer System ?	Administer ?	Execute Analysis ?	Create ?
sonar-administrators System administrators	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/> Quality Gates <input checked="" type="checkbox"/> Quality Profiles	<input type="checkbox"/>	<input checked="" type="checkbox"/> Projects
sonar-users Every authenticated user automatically belongs to this group	<input type="checkbox"/>	<input type="checkbox"/> Quality Gates <input type="checkbox"/> Quality Profiles	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/> Projects
Anyone <small>DEPRECATED</small> Anybody who browses the application belongs to this group. If authentication is not enforced, assigned permissions also apply to non-authenticated users.	<input type="checkbox"/>	<input type="checkbox"/> Quality Gates <input type="checkbox"/> Quality Profiles	<input type="checkbox"/>	<input type="checkbox"/> Projects
Administrator admin	<input type="checkbox"/>	<input type="checkbox"/> Quality Gates <input type="checkbox"/> Quality Profiles	<input checked="" type="checkbox"/>	<input type="checkbox"/> Projects

4 of 4 shown

12. Run the Build

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

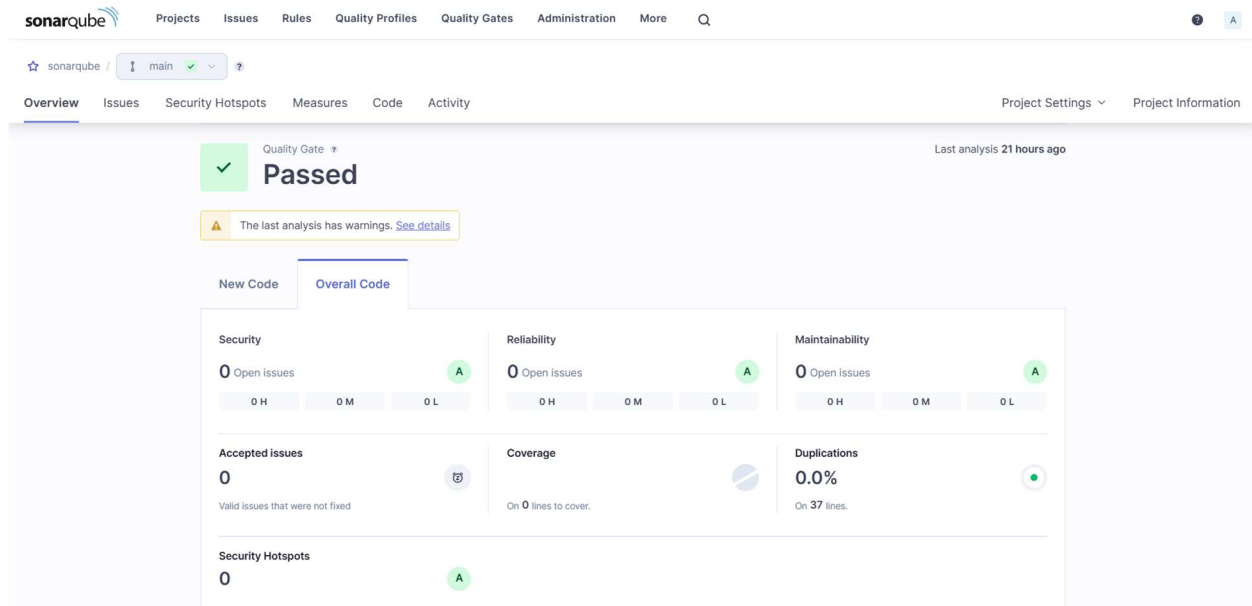


The screenshot shows the Jenkins console output for a build. The build is successful, and the output shows the following steps:

```
Started by user Pranav Santosh Pol
Running as SYSTEM
Building in workspace C:\ProgramData\Jenkins\jenkins\workspace\sonarqube
The recommended git tool is: NONE
No credentials specified
> git.exe rev-parse --resolve-git-dir C:\ProgramData\Jenkins\jenkins\workspace\sonarqube\git # timeout=10
Fetching changes from the remote Git repository
> git.exe config remote.origin.url https://github.com/shazforiot/MSBuild_firstproject.git # timeout=10
Fetching upstream changes from https://github.com/shazforiot/MSBuild_firstproject.git
> git.exe --version # timeout=10
> git --version # 'git version 2.42.0.windows.1'
> git.exe fetch --tags --force --progress -- https://github.com/shazforiot/MSBuild_firstproject.git +refs/heads/*:refs/remotes/origin/* # timeout=10
> git.exe rev-parse "refs/remotes/origin/master^{commit}" # timeout=10
Checking out Revision f2bc042c04c6e72427c380bcae6d6fee7b49adf (refs/remotes/origin/master)
> git.exe config core.sparsecheckout # timeout=10
> git.exe checkout -f f2bc042c04c6e72427c380bcae6d6fee7b49adf # timeout=10
Commit message: "updated"
> git.exe rev-list --no-walk f2bc042c04c6e72427c380bcae6d6fee7b49adf # timeout=10
[sonarqube] $ C:\ProgramData\Jenkins\jenkins\tools\hudson.plugins.sonar.SonarRunnerInstallation\sonarqube\bin\sonar-scanner.bat -Dsonar.host.url=http://localhost:9000 -Dsonar.projectKey=sonarqube -Dsonar.login=squ_1fa56ee0a0d333885d02c288f69057eafed9adeb -Dsonar.host.url=http://localhost:9000 -Dsonar.sources=HelloWorldCore -Dsonar.projectBaseDir=C:\ProgramData\Jenkins\jenkins\workspace\sonarqube
00:14:02.569 WARN Property 'sonar.host.url' with value 'http://localhost:9000' is overridden with value 'http://localhost:9000'
00:14:02.584 INFO Scanner configuration file: C:\ProgramData\Jenkins\jenkins\tools\hudson.plugins.sonar.SonarRunnerInstallation\sonarqube\bin\...\conf\sonar-scanner.properties
00:14:02.584 INFO Project root configuration file: NONE
00:14:02.614 INFO SonarScanner CLI 6.2.0.4584
00:14:02.614 INFO Java 21.0.4 Eclipse Adoptium (64-bit)
```

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.