Name: **Dev Gaonkar** Div/Roll no.: **D15C/12**

Experiment 7

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

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. Thus, integrating static analysis into the SDLC can yield dramatic results in the overall quality of the code developed.

What are the key steps to run SAST effectively?

There are six simple steps needed to perform SAST efficiently in organizations that have a very large number of applications built with different languages, frameworks, and platforms.

1. **Finalize the tool.** Select a static analysis tool that can perform code reviews of applications written in the programming languages you use. The tool should also be able to comprehend the underlying framework used by your software.
2. **Create the scanning infrastructure, and deploy the tool.** This step involves handling the licensing requirements, setting up access control and authorization, and procuring the resources required (e.g., servers and databases) to deploy the tool.
3. **Customize the tool.** Fine-tune the tool to suit the needs of the organization. For example, you might configure it to reduce false positives or find additional security vulnerabilities by writing new rules or updating existing ones. Integrate the tool into the build environment, create dashboards for tracking scan results, and build custom reports.
4. **Prioritize and onboard applications.** Once the tool is ready, onboard your applications. If you have a large number of applications, prioritize the high-risk applications to scan first. Eventually, all your applications should be onboarded and scanned regularly, with application scans synced with release cycles, daily or monthly builds, or code check-ins.
5. **Analyze scan results.** This step involves triaging the results of the scan to remove false positives. Once the set of issues is finalized, they should be tracked and provided to the deployment teams for proper and timely remediation.
6. **Provide governance and training.** Proper governance ensures that your development teams are employing the scanning tools properly. The software security touchpoints should be present within the SDLC. SAST should be incorporated as part of your application development and deployment process.

Integrating Jenkins with SonarQube:

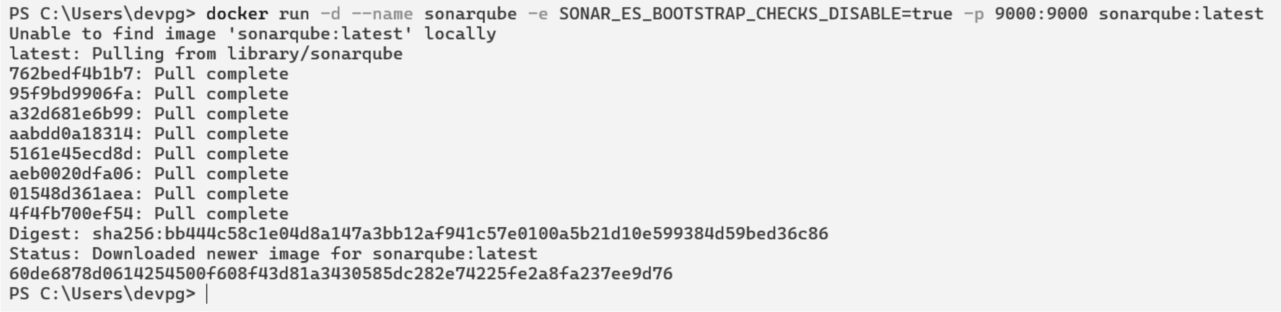
Prerequisites:

- Jenkins installed
- Docker Installed (for SonarQube)
- SonarQube Docker Image

Steps to integrate Jenkins with SonarQube

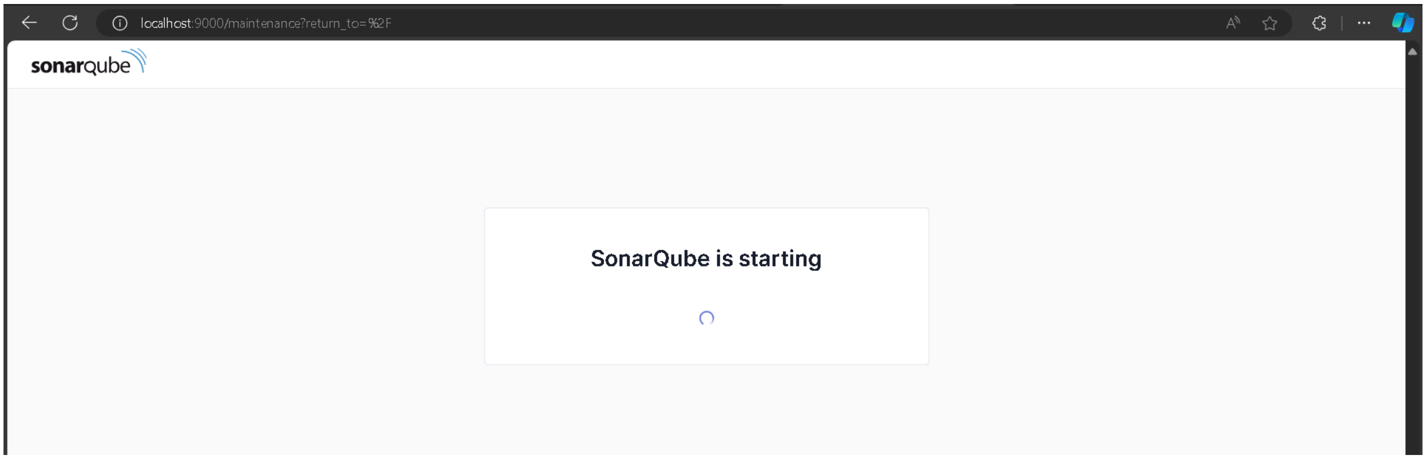
1. Open up Jenkins Dashboard on localhost, port 8080 or whichever port it is at for you.

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



```
PS C:\Users\devpg> 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
762bedf4b1b7: Pull complete
95f9bd9906fa: Pull complete
a32d681e6b99: Pull complete
aabdd0a18314: Pull complete
5161e45ecd8d: Pull complete
aeb0020dfa06: Pull complete
01548d361aea: Pull complete
4f4fb700ef54: Pull complete
Digest: sha256:bb444c58c1e04d8a147a3bb12af941c57e0100a5b21d10e599384d59bed36c86
Status: Downloaded newer image for sonarqube:latest
60de6878d0614254500f608f43d81a3430585dc282e74225fe2a8fa237ee9d76
PS C:\Users\devpg> |
```

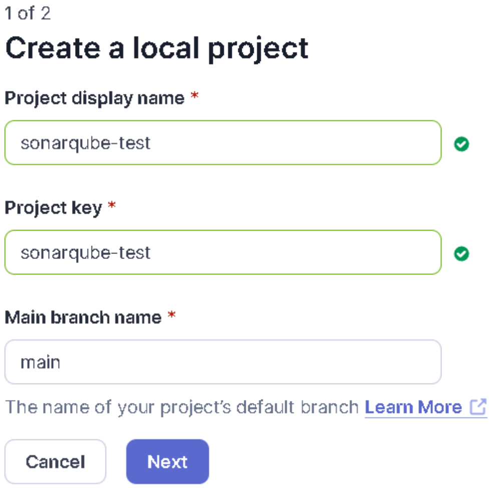
3. Once the container is up and running, you can check the status of SonarQube at localhost port 9000.



SonarQube is starting

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

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



1 of 2

Create a local project

Project display name *: sonarqube-test

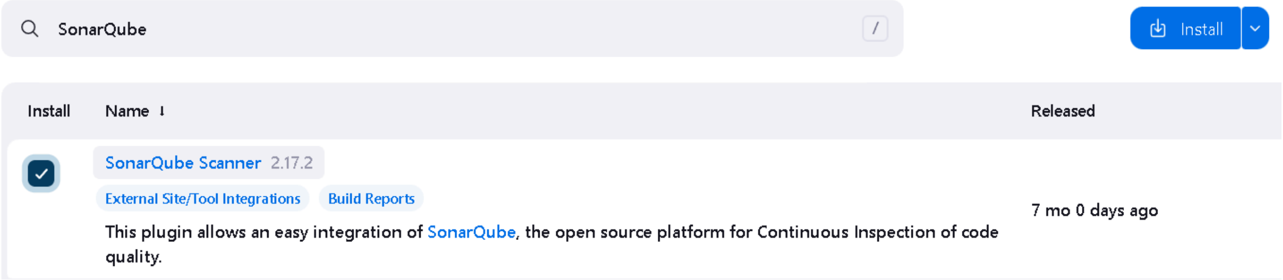
Project key *: sonarqube-test

Main branch name *: main

The name of your project's default branch Learn More

Setup the project and come back to Jenkins Dashboard.

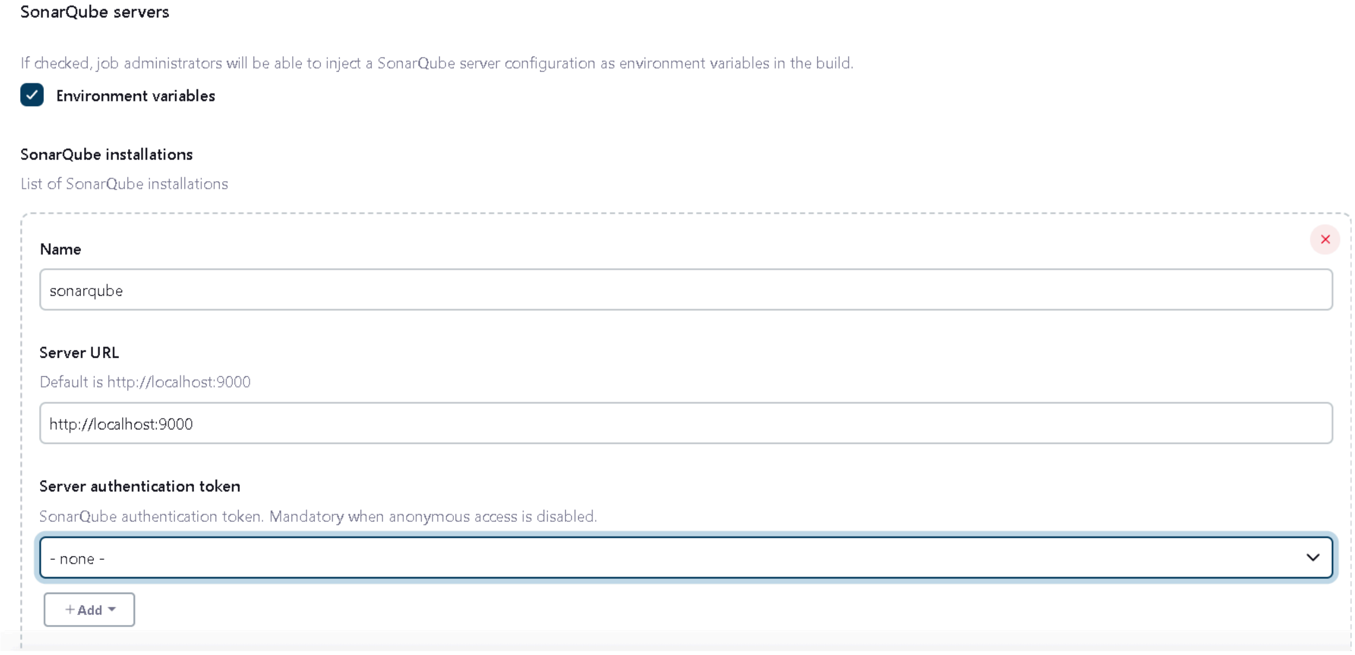
6. Go to Manage Jenkins and search for SonarQube Scanner for Jenkins and install it.



Install	Name ↓	Released
☑	SonarQube Scanner 2.17.2 External Site/Tool Integrations Build Reports This plugin allows an easy integration of SonarQube, the open source platform for Continuous Inspection of code quality.	7 mo 0 days ago

7. Under Jenkins 'Configure System', look for SonarQube Servers and enter the details.

Enter the Server Authentication token if needed.


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

sonarqube

Server URL

Default is http://localhost:9000

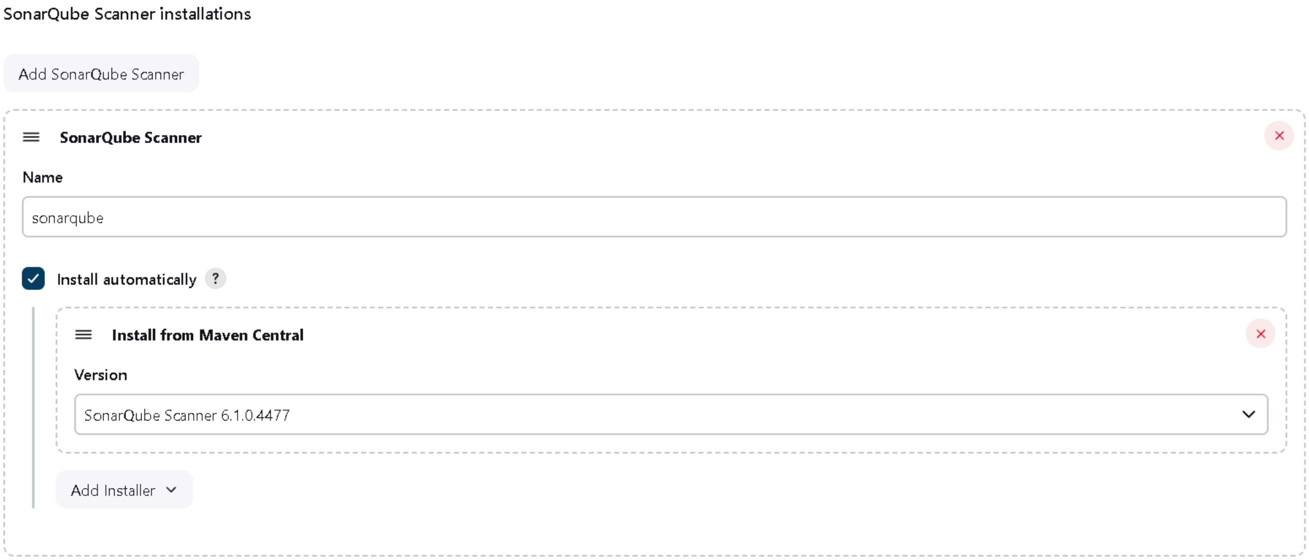
http://localhost:9000

Server authentication token

SonarQube authentication token. Mandatory when anonymous access is disabled.

- none -

8. Search for SonarQube Scanner under Global Tool Configuration. Choose the latest configuration and choose Install automatically.


SonarQube Scanner installations

SonarQube Scanner

Name

sonarqube

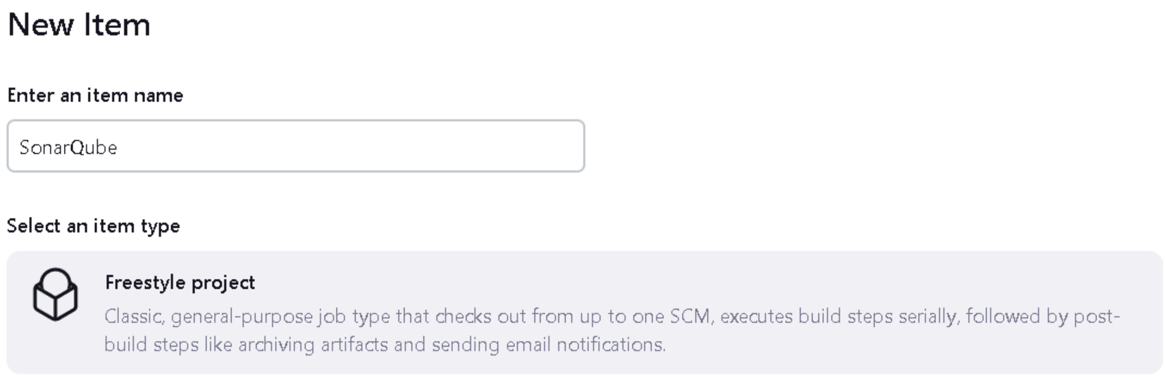
☑ Install automatically ?

Install from Maven Central

Version

SonarQube Scanner 6.1.0.4477

9. After the configuration, create a New Item in Jenkins, choose a freestyle project.



New Item

Enter an item name

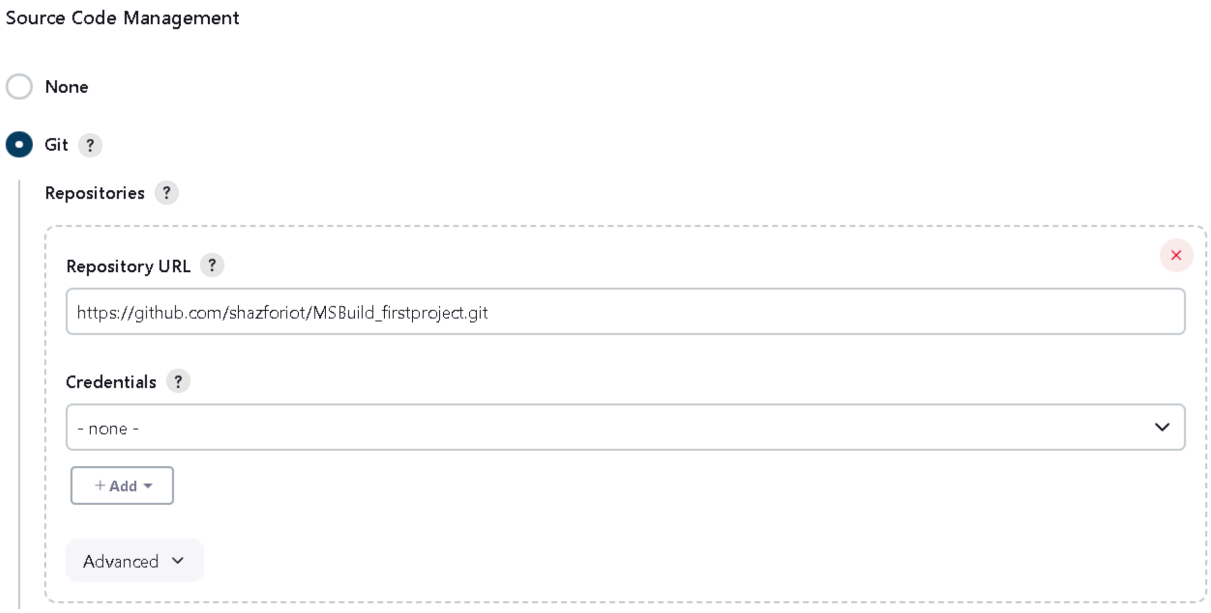
SonarQube

Select an item type

Freestyle project

Classic, general-purpose job type that checks out from up to one SCM, executes build steps serially, followed by post-build steps like archiving artifacts and sending email notifications.

10. Choose this GitHub repository in Source Code Management. https://github.com/shazforiot/MSBuild_firstproject.git



Source Code Management

○ None

● Git ?

Repositories ?

Repository URL ?

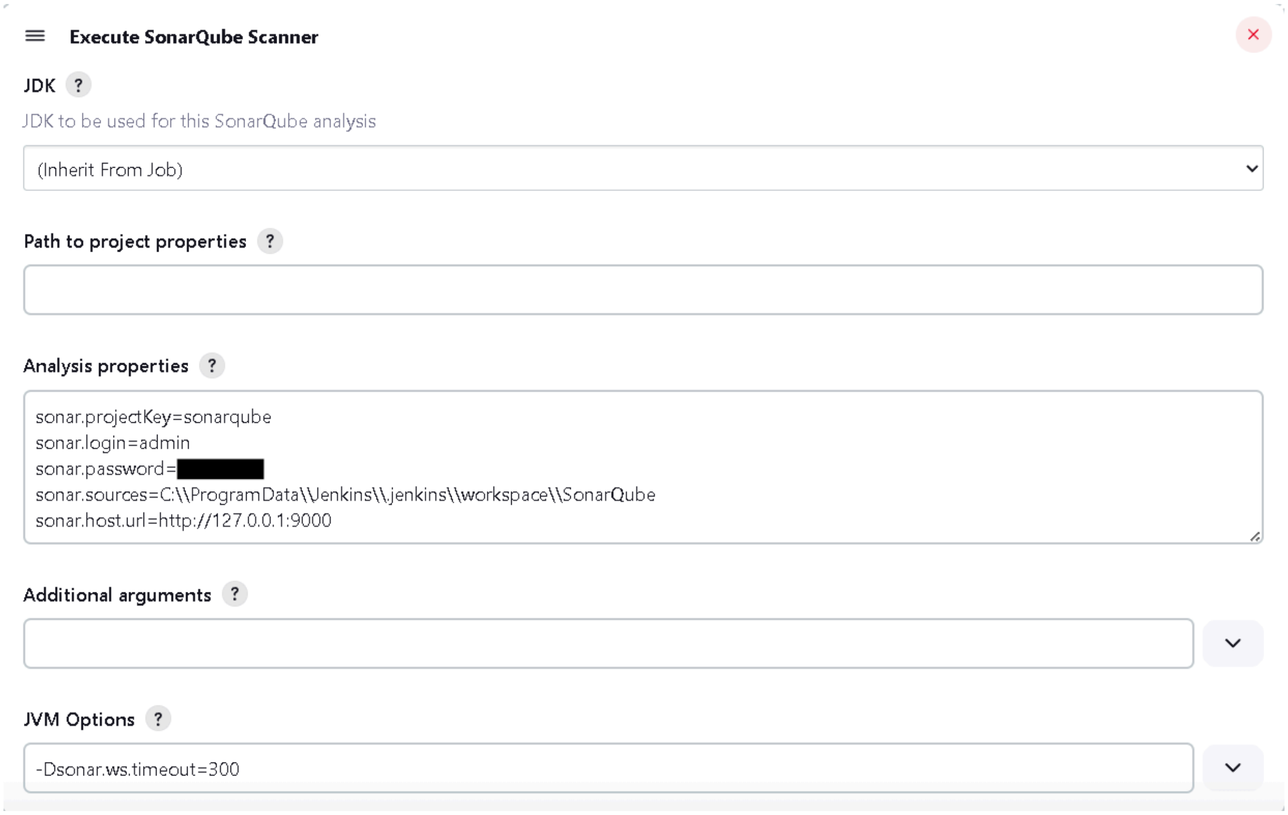
https://github.com/shazforiot/MSBuild_firstproject.git

Credentials ?

- none -

It is a sample hello-world project with no vulnerabilities and issues, just to test the integration.

11. Under Build-> Execute SonarQube Scanner, enter these Analysis properties. Mention the SonarQube Project Key, Login, Password, Source path and Host URL.



Execute SonarQube Scanner

JDK ?

JDK to be used for this SonarQube analysis

(Inherit From Job)

Path to project properties ?

Analysis properties ?

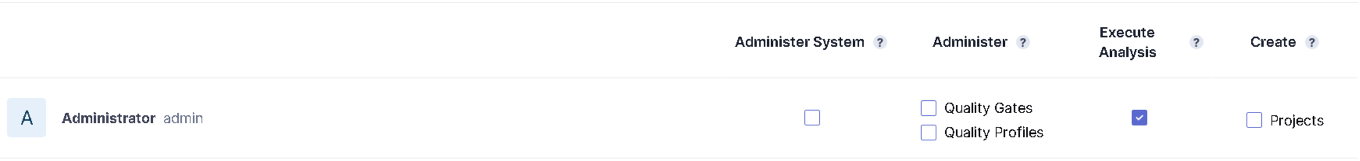
```
sonar.projectKey=sonarqube
sonar.login=admin
sonar.password=[redacted]
sonar.sources=C:\\ProgramData\\Jenkins\\.jenkins\\workspace\\SonarQube
sonar.host.url=http://127.0.0.1:9000
```

Additional arguments ?

JVM Options ?

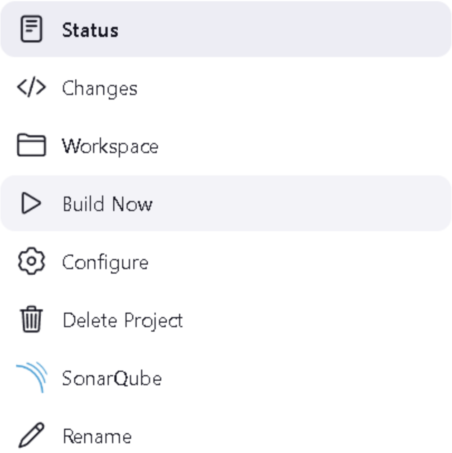
-Dsonar.ws.timeout=300

12. Go to http://localhost:9000/<user_name>/permissions and allow Execute Permissions to the Admin user.

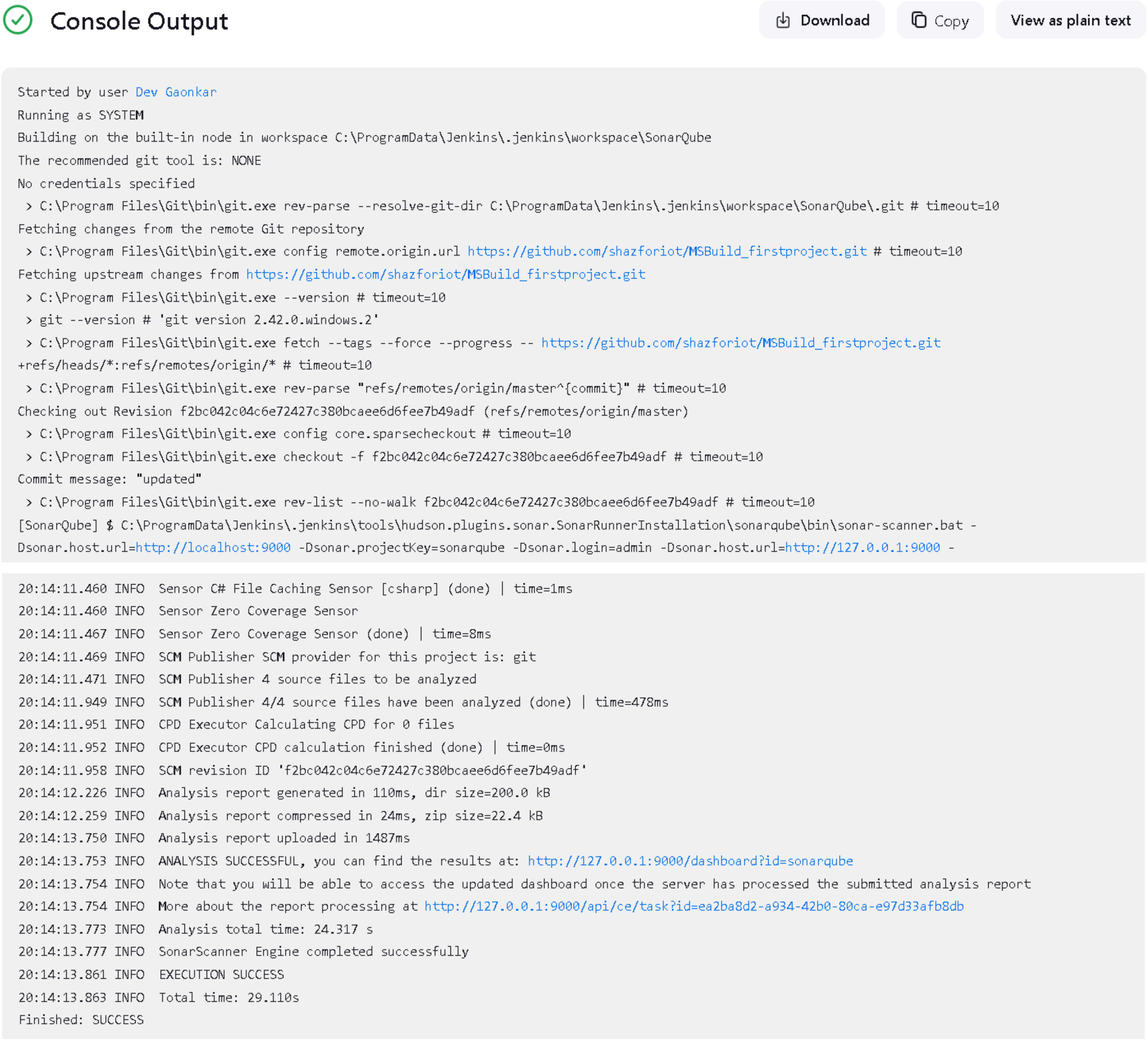


	Administer System ?	Administer ?	Execute Analysis ?	Create ?
A Administrator admin	□	□ Quality Gates □ Quality Profiles	☑	□ Projects

13. Run The Build.



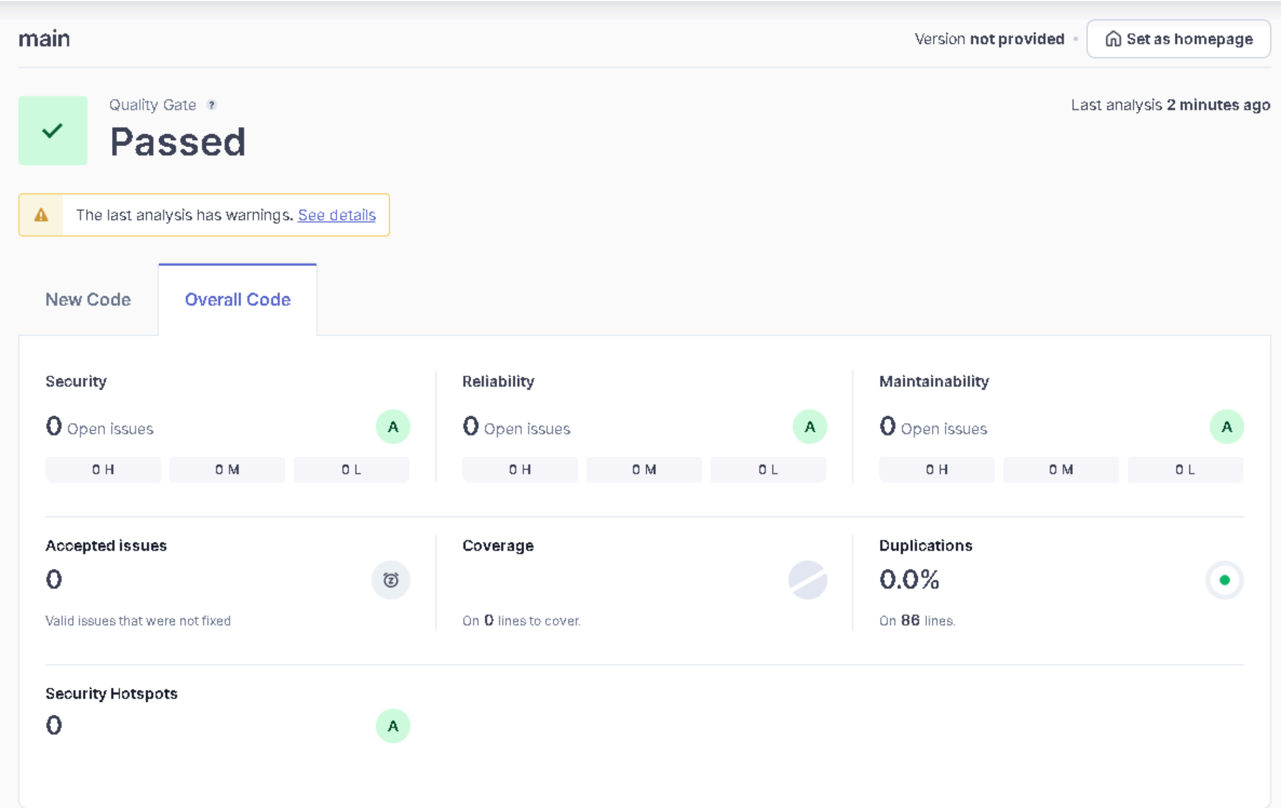
Check the console output.



```
Started by user Dev Gaonkar
Running as SYSTEM
Building on the built-in node in workspace C:\ProgramData\Jenkins\.jenkins\workspace\SonarQube
The recommended git tool is: NONE
No credentials specified
 > C:\Program Files\Git\bin\git.exe rev-parse --resolve-git-dir C:\ProgramData\Jenkins\.jenkins\workspace\SonarQube\.git # timeout=10
Fetching changes from the remote Git repository
 > C:\Program Files\Git\bin\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
 > C:\Program Files\Git\bin\git.exe --version # timeout=10
 > git --version # 'git version 2.42.0.windows.2'
 > C:\Program Files\Git\bin\git.exe fetch --tags --force --progress -- https://github.com/shazforiot/MSBuild_firstproject.git
+refs/heads/*:refs/remotes/origin/* # timeout=10
 > C:\Program Files\Git\bin\git.exe rev-parse "refs/remotes/origin/master^{commit}" # timeout=10
Checking out Revision f2bc042c04c6e72427c380bcaee6d6fee7b49adf (refs/remotes/origin/master)
 > C:\Program Files\Git\bin\git.exe config core.sparsecheckout # timeout=10
 > C:\Program Files\Git\bin\git.exe checkout -f f2bc042c04c6e72427c380bcaee6d6fee7b49adf # timeout=10
Commit message: "updated"
 > C:\Program Files\Git\bin\git.exe rev-list --no-walk f2bc042c04c6e72427c380bcaee6d6fee7b49adf # 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=admin -Dsonar.host.url=http://127.0.0.1:9000 -
```

```
20:14:11.460 INFO  Sensor C# File Caching Sensor [csharp] (done) | time=1ms
20:14:11.460 INFO  Sensor Zero Coverage Sensor
20:14:11.467 INFO  Sensor Zero Coverage Sensor (done) | time=8ms
20:14:11.469 INFO  SCM Publisher SCM provider for this project is: git
20:14:11.471 INFO  SCM Publisher 4 source files to be analyzed
20:14:11.949 INFO  SCM Publisher 4/4 source files have been analyzed (done) | time=478ms
20:14:11.951 INFO  CPD Executor Calculating CPD for 0 files
20:14:11.952 INFO  CPD Executor CPD calculation finished (done) | time=0ms
20:14:11.958 INFO  SCM revision ID 'f2bc042c04c6e72427c380bcaee6d6fee7b49adf'
20:14:12.226 INFO  Analysis report generated in 110ms, dir size=200.0 kB
20:14:12.259 INFO  Analysis report compressed in 24ms, zip size=22.4 kB
20:14:13.750 INFO  Analysis report uploaded in 1487ms
20:14:13.753 INFO  ANALYSIS SUCCESSFUL, you can find the results at: http://127.0.0.1:9000/dashboard?id=sonarqube
20:14:13.754 INFO  Note that you will be able to access the updated dashboard once the server has processed the submitted analysis report
20:14:13.754 INFO  More about the report processing at http://127.0.0.1:9000/api/ce/task?id=ea2ba8d2-a934-42b0-80ca-e97d33afb8db
20:14:13.773 INFO  Analysis total time: 24.317 s
20:14:13.777 INFO  SonarScanner Engine completed successfully
20:14:13.861 INFO  EXECUTION SUCCESS
20:14:13.863 INFO  Total time: 29.110s
Finished: SUCCESS
```

14. Once the build is complete, check the project in SonarQube.



Quality Gate Passed

Last analysis 2 minutes ago

The last analysis has warnings. See details

Security: 0 Open issues, A (0 H, 0 M, 0 L)

Reliability: 0 Open issues, A (0 H, 0 M, 0 L)

Maintainability: 0 Open issues, A (0 H, 0 M, 0 L)

Accepted issues: 0. Valid issues that were not fixed

Coverage: On 0 lines to cover.

Duplications: 0.0%. On 86 lines.

Security Hotspots: 0, A

In this way, we have integrated Jenkins with SonarQube for SAST.

Conclusion:

In this experiment, we explored the importance of Static Application Security Testing (SAST) and its role in identifying security vulnerabilities early in the software development lifecycle. By integrating Jenkins with SonarQube, we demonstrated an automated process for static code analysis. Using SonarQube’s capabilities, we scanned a sample project for vulnerabilities, ensuring that code is secure and free of potential threats. This integration allows continuous code analysis during development, making it easier to address security issues before deployment. Overall, the experiment highlighted the efficiency and necessity of incorporating SASTtools into modern DevOps workflows for secure development.