

## **ADVANCE DEVOPS EXPERIMENT 7**

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

**Theory:** 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.

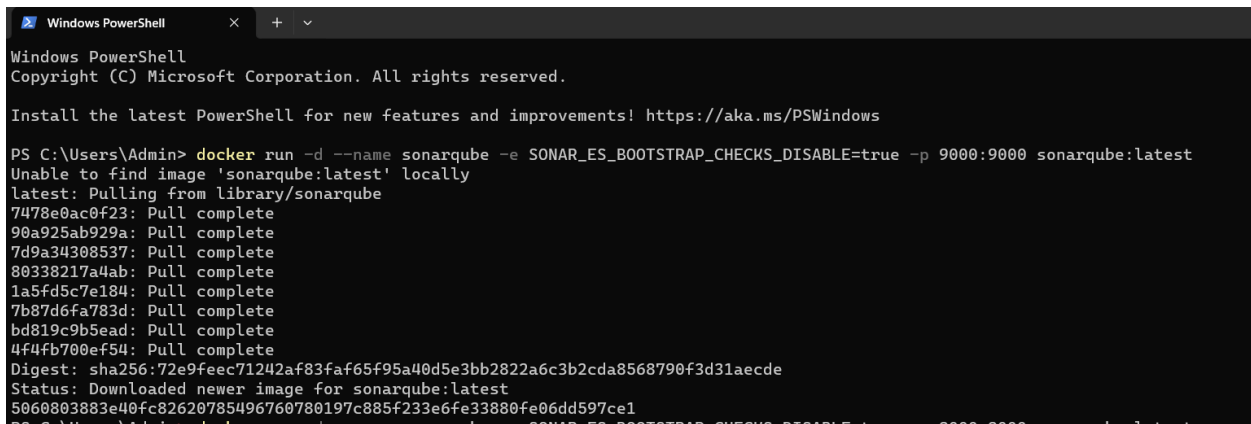
1. **Finalize the tool.** Select a static analysis tool that can perform code reviews of applications written in the programming languages you use. The tool should also be able to comprehend the underlying framework used by your software.
2. **Create the scanning infrastructure, and deploy the tool.** 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.

### Steps to integrate Jenkins with SonarQube

1. Open up Jenkins Dashboard on localhost, port 8080 .
2. Run SonarQube in a Docker container using this command -



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

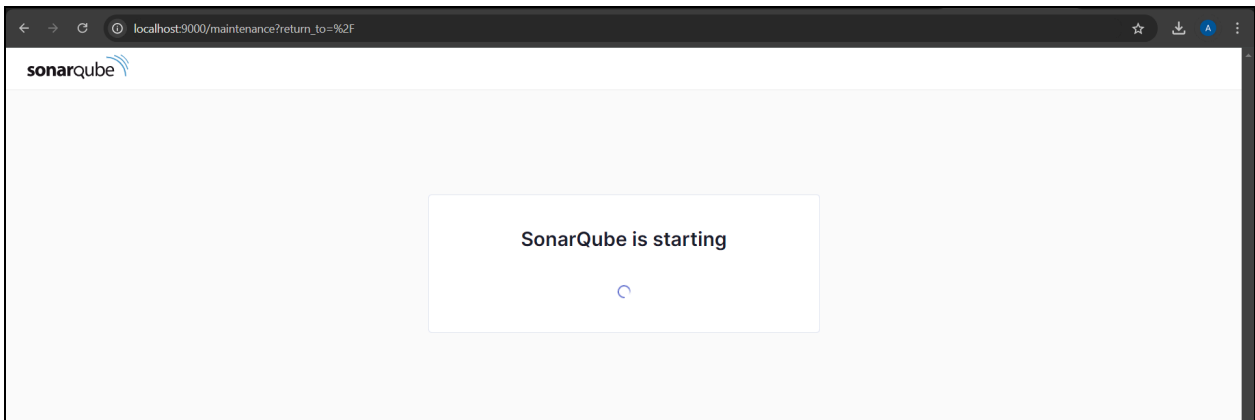
Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\Admin> 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: sha256:72e9feec71242af83faf65f95a40d5e3bb2822a6c3b2cda8568790f3d31aecd
Status: Downloaded newer image for sonarqube:latest
5060803883e40fc82620785496760780197c885f233e6fe33880fe06dd597ce1
```

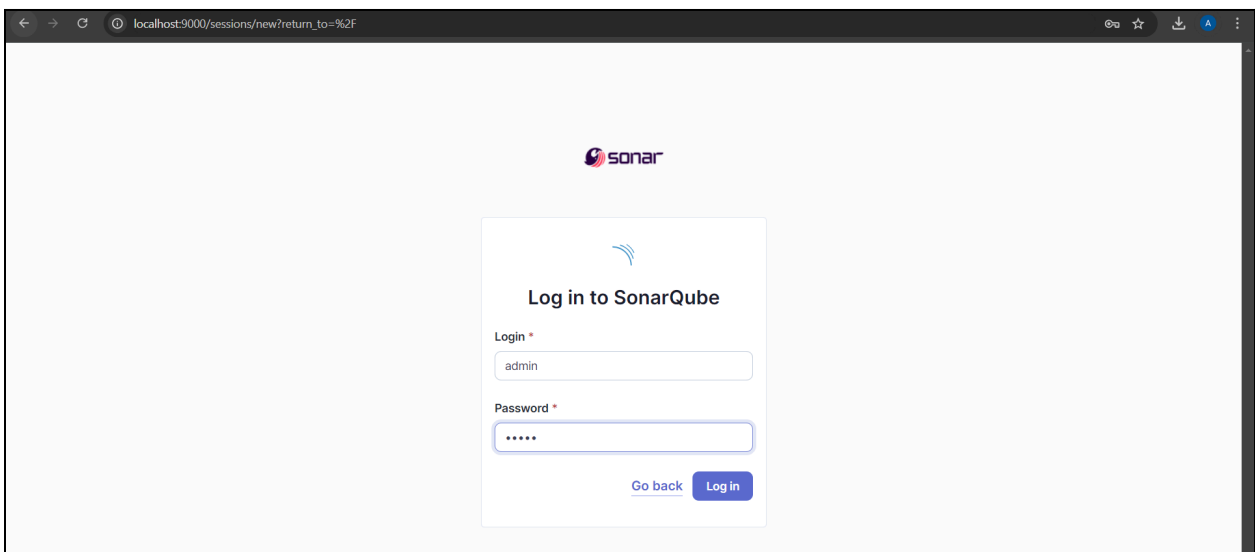
**Warning: run below command only once**

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

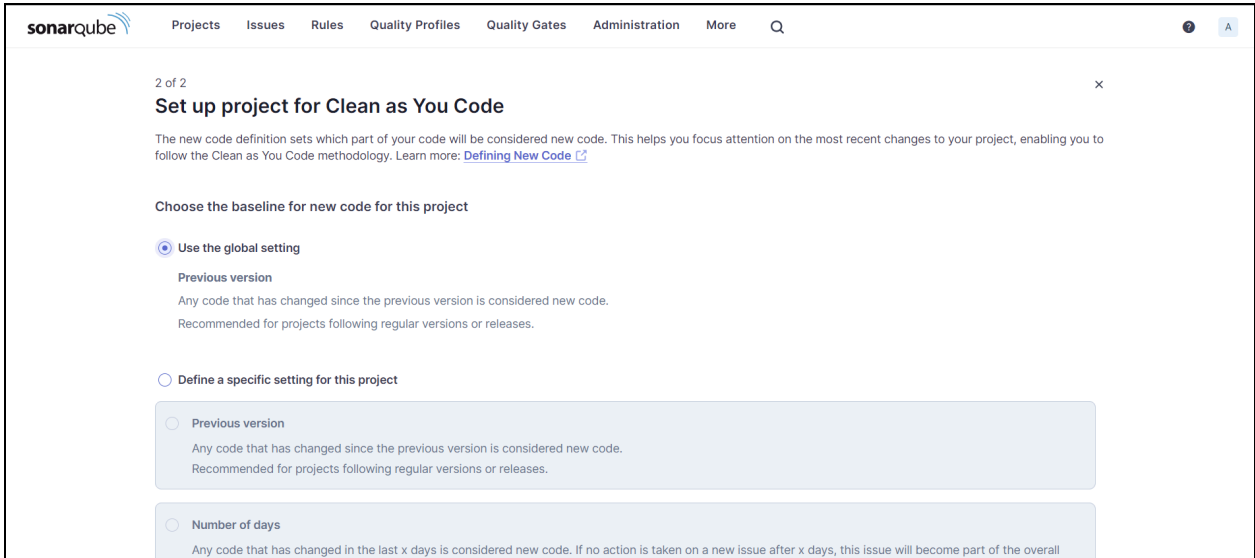
3. Once the container is up and running, you can check the status of SonarQube at localhost port 9000.



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

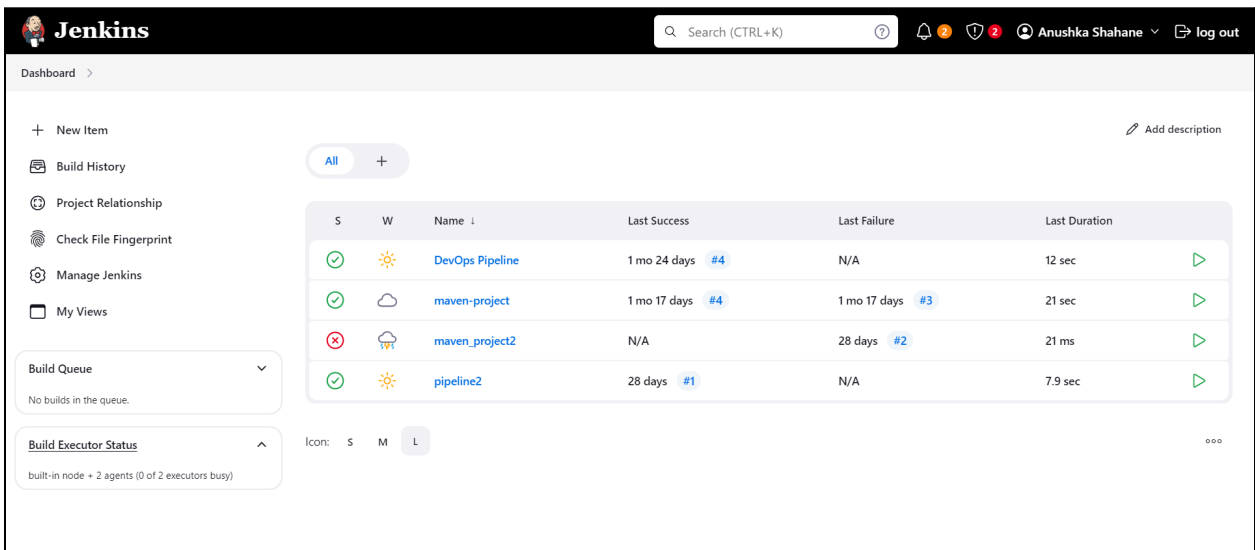


5. Create a manual project in SonarQube with the name sonarqube

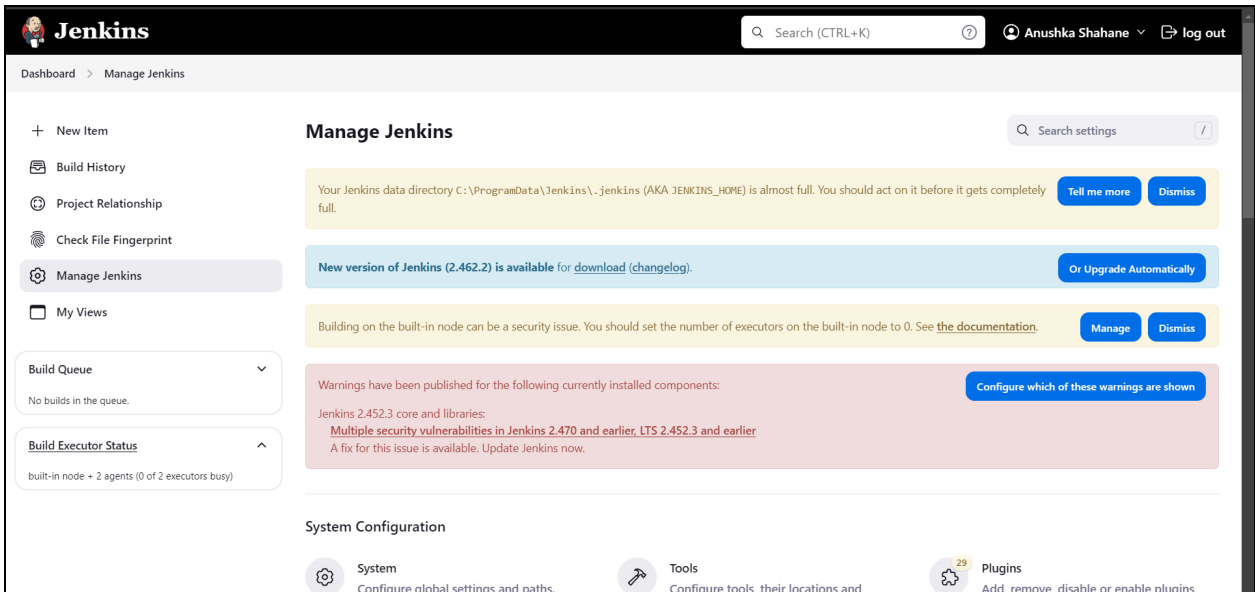


Setup the project and come back to Jenkins Dashboard.

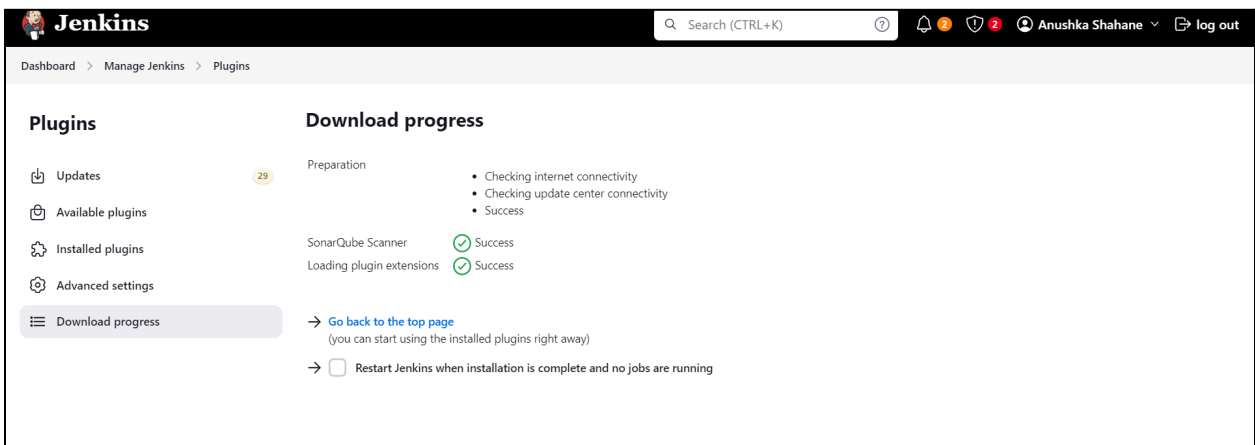
Go to Manage Jenkins and search for SonarQube Scanner for Jenkins and install it



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

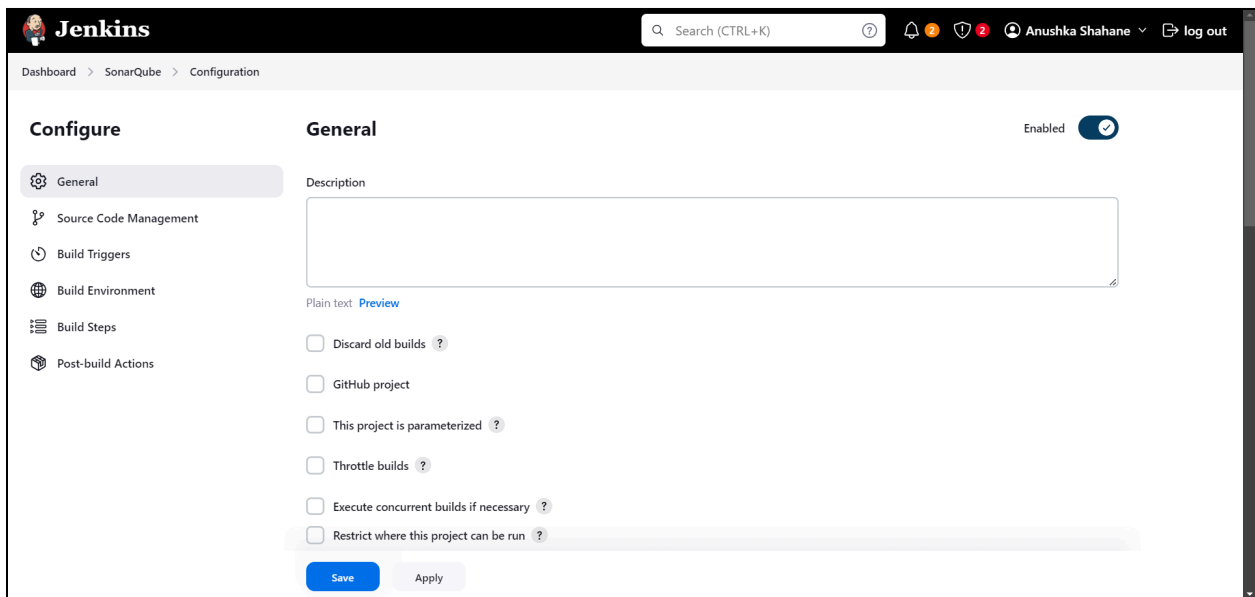
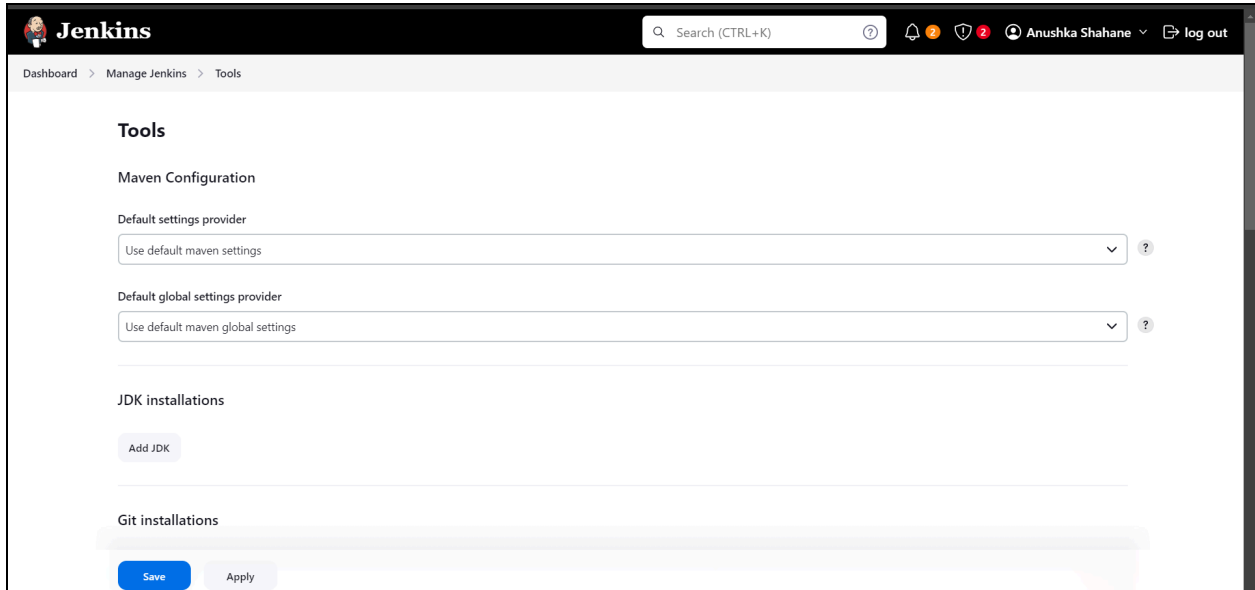


The screenshot shows the Jenkins 'Manage Jenkins' dashboard. The top navigation bar includes the Jenkins logo, a search bar, and the user 'Anushka Shahane' with a 'log out' button. The left sidebar contains links for 'New Item', 'Build History', 'Project Relationship', 'Check File Fingerprint', 'Manage Jenkins' (selected), and 'My Views'. The main content area is titled 'Manage Jenkins' and features several informational banners: a yellow warning about the Jenkins data directory being full, a blue banner for a new version (2.462.2) available for download, and a yellow banner about the built-in node security issue. Below these is a red warning banner about security vulnerabilities in Jenkins 2.470 and earlier. At the bottom, there is a 'System Configuration' section with three tabs: 'System', 'Tools', and 'Plugins' (selected).



The screenshot shows the 'Plugins' section of the Jenkins dashboard. The top navigation bar is the same as the previous screenshot. The left sidebar has 'Download progress' selected. The main content area is titled 'Download progress' and shows the status of plugin installation. It includes a 'Preparation' section with a list of tasks: 'Checking internet connectivity', 'Checking update center connectivity', and 'Success'. Below this, 'SonarQube Scanner' and 'Loading plugin extensions' are both marked as 'Success'. At the bottom, there are two links: 'Go back to the top page' and 'Restart Jenkins when installation is complete and no jobs are running'.

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



9. Choose this GitHub repository in Source Code Management.

[https://github.com/shazforiot/MSBuild\\_firstproject.git](https://github.com/shazforiot/MSBuild_firstproject.git)

It is a sample hello-world project with no vulnerabilities and issues,

The screenshot shows the SonarQube Configuration page for the 'Execute SonarQube Scanner' build step. The left sidebar contains a 'Configure' section with a list of options: General, Source Code Management, Build Triggers, Build Environment, Build Steps (highlighted), and Post-build Actions. The main content area is titled 'Execute SonarQube Scanner' and contains several configuration fields:

- JDK**: A dropdown menu with the value '(Inherit From Job)'.
- Path to project properties**: A text input field.
- Analysis properties**: A text area containing the following properties:

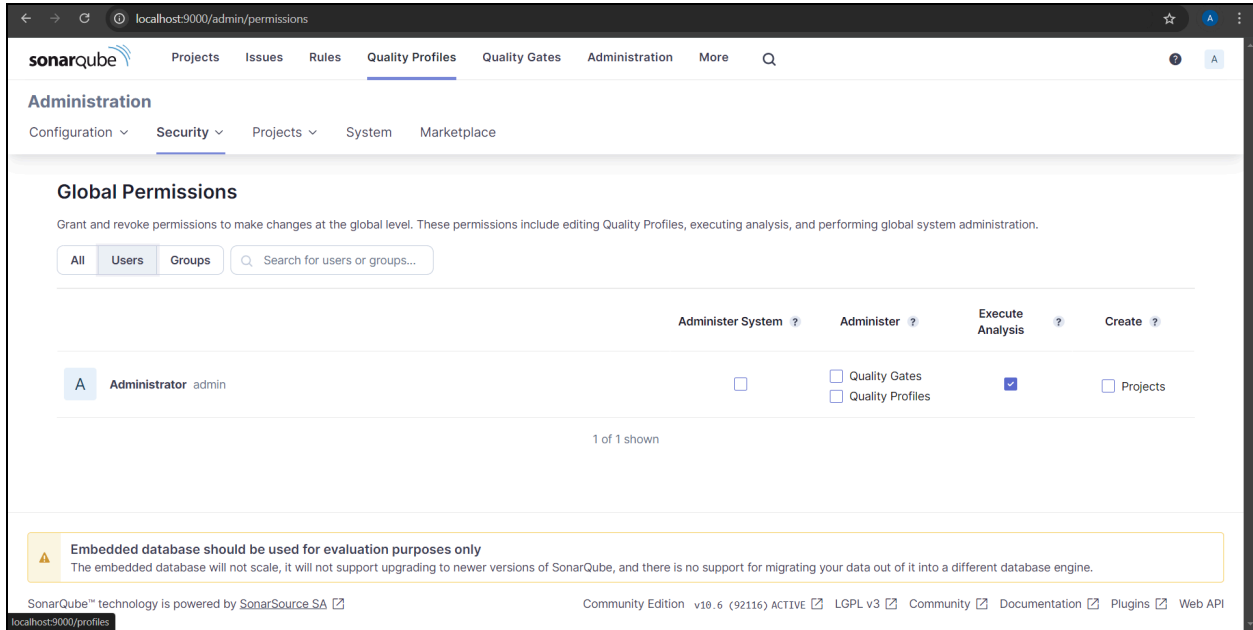
```
sonar.projectKey=sonarqube-test
sonar.login=admin
sonar.password=Anushka32
sonar.hosturl=http://sonarqube:9000
```
- Additional arguments**: A text input field.
- JVM Options**: A text input field.

At the bottom of the configuration area, there are two buttons: 'Save' and 'Apply'.

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

11. Go to [http://localhost:9000/<user\\_name>/permissions](http://localhost:9000/<user_name>/permissions) and allow Execute Permissions to the Admin user.





The screenshot shows the SonarQube Administration interface, specifically the 'Global Permissions' page. The page is titled 'Administration' and has a sub-header 'Global Permissions'. Below the header, there is a search bar and a table of permissions. The table has columns for 'All', 'Users', 'Groups', and a search bar. The 'Users' tab is selected, and the table shows one user, 'Administrator', with permissions for 'Administer System', 'Administer', 'Execute Analysis', and 'Create'. The 'Execute Analysis' permission is checked. Below the table, there is a warning message: 'Embedded database should be used for evaluation purposes only'. The footer of the page contains the SonarQube logo, version information, and links to documentation and plugins.

**Global Permissions**

Grant and revoke permissions to make changes at the global level. These permissions include editing Quality Profiles, executing analysis, and performing global system administration.

**Users** | Search for users or groups...

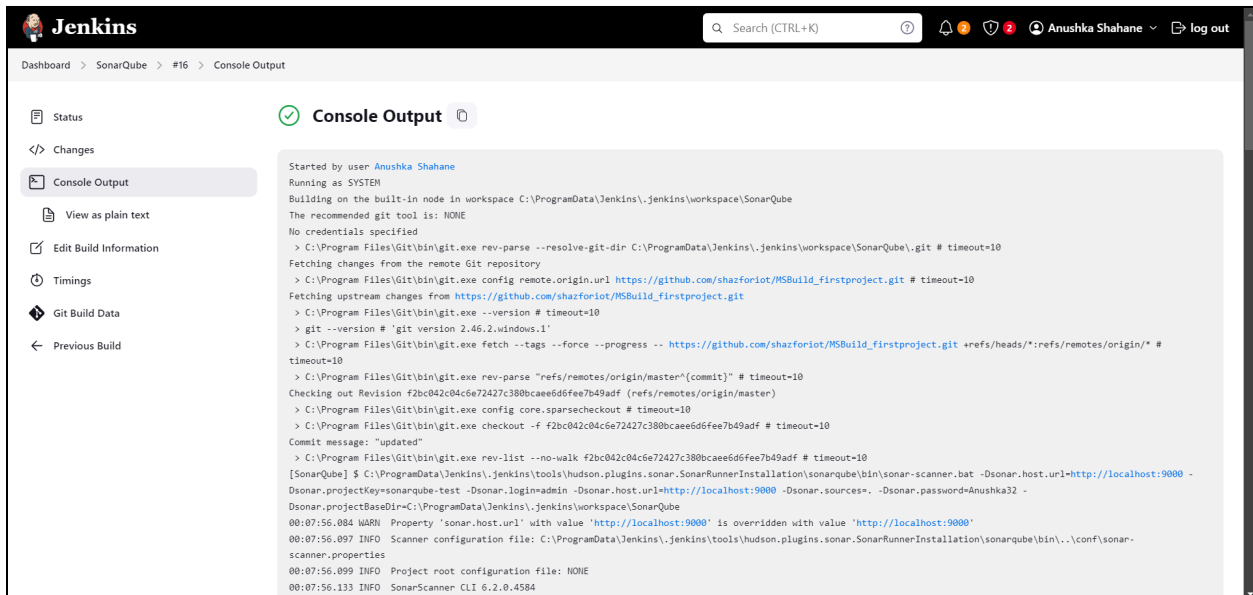
	Administer System ?	Administer ?	Execute Analysis ?	Create ?
<b>A</b> Administrator admin	<input type="checkbox"/>	<input type="checkbox"/> Quality Gates <input type="checkbox"/> Quality Profiles	<input checked="" type="checkbox"/>	<input type="checkbox"/> Projects

1 of 1 shown

**Warning:** Embedded database should be used for evaluation purposes only  
The embedded database will not scale, it will not support upgrading to newer versions of SonarQube, and there is no support for migrating your data out of it into a different database engine.

SonarQube™ technology is powered by SonarSource SA | Community Edition v10.6 (92116) ACTIVE | LGPL v3 | Community | Documentation | Plugins | Web API

Check the console output.



The screenshot shows the Jenkins console output for a build. The build is titled 'Console Output' and is in a 'Completed' state. The output shows the build process, including the checkout of the repository, the execution of the SonarScanner CLI, and the generation of the SonarQube report. The output is displayed in a monospace font, with line numbers on the left. The build was started by user 'Anushka Shahane' and is running as 'SYSTEM'.

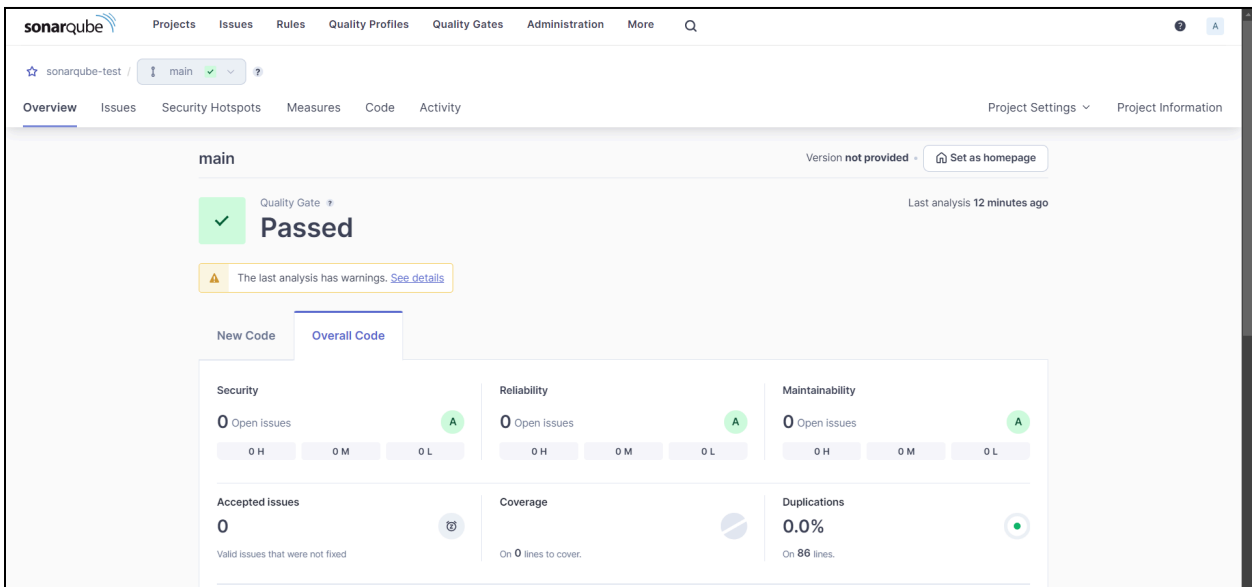
**Console Output**

```
Started by user Anushka Shahane
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 2.46.2.windows.1'
> 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 f2bc042c04c6e72427c380bcae6d6fee7b49adf (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 f2bc042c04c6e72427c380bcae6d6fee7b49adf # timeout=10
Commit message: "updated"
> C:\Program Files\Git\bin\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-test -Dsonar.login=admin -Dsonar.host.url=http://localhost:9000 -Dsonar.sources=. -Dsonar.password=Anushka32 -Dsonar.projectBaseDir=C:\ProgramData\Jenkins\jenkins\workspace\SonarQube
00:07:56.084 WARN Property 'sonar.host.url' with value 'http://localhost:9000' is overridden with value 'http://localhost:9000'
00:07:56.097 INFO Scanner configuration file: C:\ProgramData\Jenkins\jenkins\tools\hudson.plugins.sonar.SonarRunnerInstallation\sonarqube\bin\..\conf\sonar-scanner.properties
00:07:56.099 INFO Project root configuration file: NONE
00:07:56.133 INFO SonarScanner CLI 6.2.0.4584
```

```
Dashboard > SonarQube > #16 > Console Output

00:09:43.074 INFO 14 source files to be analyzed
00:09:43.536 INFO 14/14 source files have been analyzed
00:09:43.538 INFO Sensor TextAndSecretsSensor [text] (done) | time=681ms
00:09:43.557 INFO ----- Run sensors on project
00:09:43.934 INFO Sensor C# [csharp]
00:09:43.936 WARN Your project contains C# files which cannot be analyzed with the scanner you are using. To analyze C# or VB.NET, you must use the SonarScanner for .NET
5.x or higher, see https://redirect.sonarsource.com/doc/install-configure-scanner-msbuild.html
00:09:43.938 INFO Sensor C# [csharp] (done) | time=7ms
00:09:43.939 INFO Sensor Analysis Warnings import [csharp]
00:09:43.941 INFO Sensor Analysis Warnings import [csharp] (done) | time=2ms
00:09:43.942 INFO Sensor C# File Caching Sensor [csharp]
00:09:43.951 WARN Incremental PR analysis: Could not determine common base path, cache will not be computed. Consider setting 'sonar-projectBaseDir' property.
00:09:43.952 INFO Sensor C# File Caching Sensor [csharp] (done) | time=9ms
00:09:43.953 INFO Sensor Zero Coverage Sensor
00:09:44.036 INFO Sensor Zero Coverage Sensor (done) | time=84ms
00:09:44.042 INFO SCM Publisher SCM provider for this project is: git
00:09:44.043 INFO SCM Publisher 4 source files to be analyzed
00:09:45.201 INFO SCM Publisher 4/4 source files have been analyzed (done) | time=1146ms
00:09:45.206 INFO CPD Executor Calculating CPD for 0 files
00:09:45.236 INFO CPD Executor CPD calculation finished (done) | time=1ms
00:09:45.252 INFO SCM revision ID 'f2bc042c04c5e72427c380bcae6d6fee7b49adf'
00:09:46.446 INFO Analysis report generated in 845ms, dir size=201.0 kB
00:09:46.554 INFO Analysis report compressed in 77ms, zip size=22.1 kB
00:09:48.616 INFO Analysis report uploaded in 2053ms
00:09:48.633 INFO ANALYSIS SUCCESSFUL, you can find the results at: http://localhost:9000/dashboard?id=sonarqube-test
00:09:48.634 INFO Note that you will be able to access the updated dashboard once the server has processed the submitted analysis report
00:09:48.634 INFO More about the report processing at http://localhost:9000/api/ce/task?id=56715393-877a-4755-994a-bace678af666
00:09:48.685 INFO Analysis total time: 1:25.553 s
00:09:48.717 INFO SonarScanner Engine completed successfully
00:09:48.900 INFO EXECUTION SUCCESS
00:09:49.053 INFO Total time: 1:52.816s
Finished: SUCCESS
```

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



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

