

Name : Nidhi Rijhwani

Subject : Adv Devops Exp No 06

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

Theory:

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.

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.

- **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.
- **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.
- **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.
- **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.
- **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.
- **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:

Windows Installation

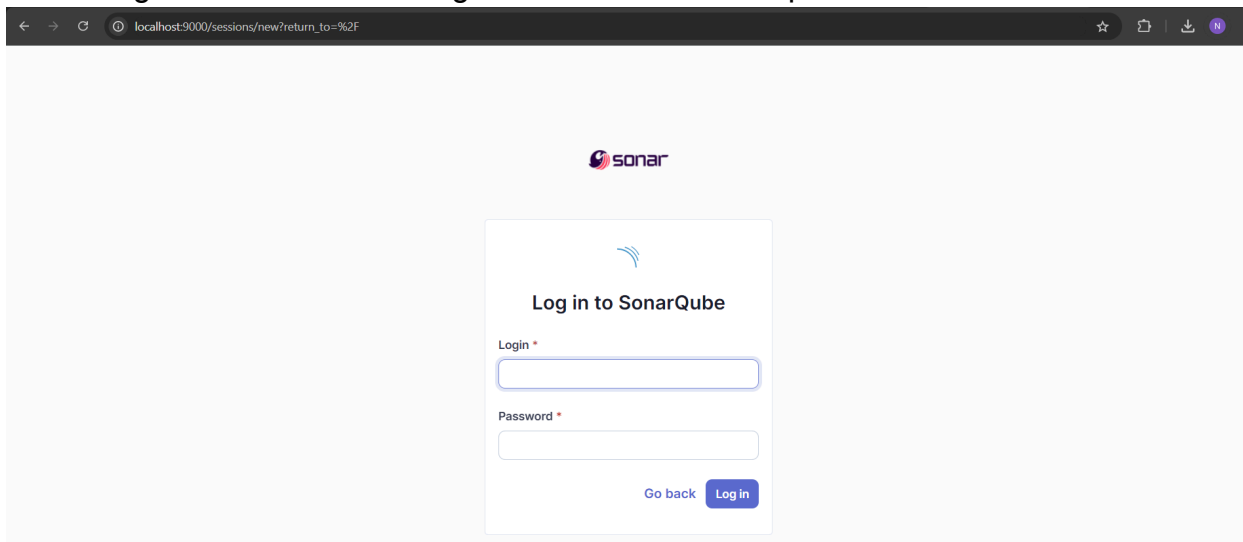
- **Step 1 : Install JDK 1.8**
- **Step 2 : Download and install jenkins**
 - <https://www.blazemeter.com/blog/how-to-install-jenkins-on-windows>
- **Requirements :**
 - [Jenkins installed](#)
 - [Docker Installed](#) (for SonarQube)
- **Ubuntu installation**
 - <https://www.digitalocean.com/community/tutorials/how-to-install-java-with-apt-on-ubuntu-20-04#installing-the-default-jre-jdk>
 - **Step 1 : Install JDK 1.8**
 - `sudo apt-get install openjdk-8-jre`
 - `sudo apt install default-jre`
- <https://www.digitalocean.com/community/tutorials/how-to-install-jenkins-on-ubuntu-20-04>
- [Open SSH](#)

Steps To Integrate Jenkins with SonarQube

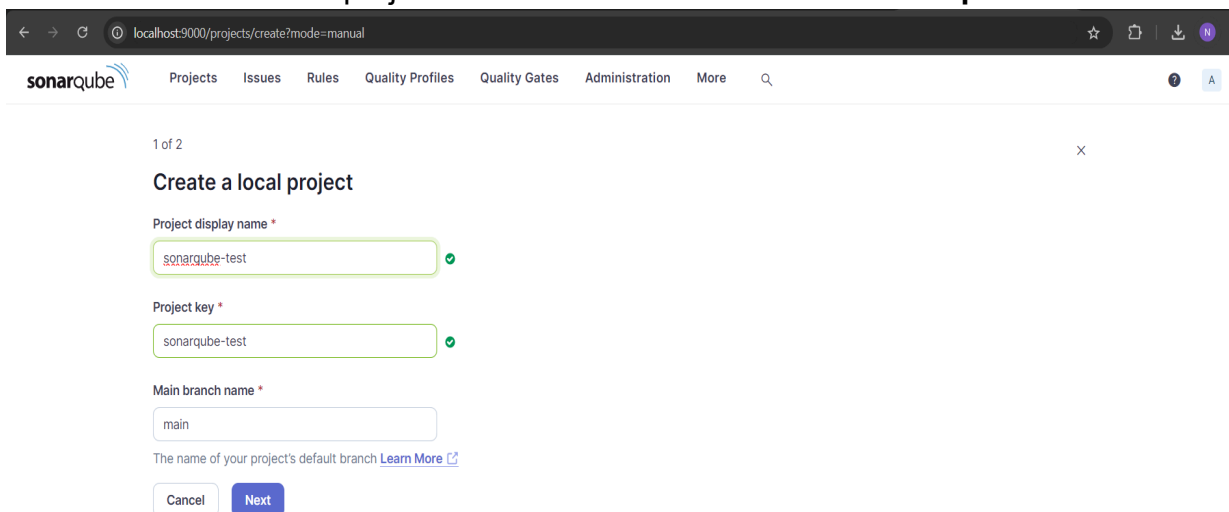
1. Open up Jenkins Dashboard on localhost, port 8080 or whichever it is for you.
2. Run SonarQube in a Docker container using this command.
 - **docker run -d --name sonarqube -e SONAR_ES_BOOTSTRAP_CHECKS_DISABLE=true -p 9000:9000 sonarqube:latest**

```
Windows PowerShell
PS C:\Users\rjhw> 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
90a925ab929a: Download complete
7478e0ac0f23: Download complete
```

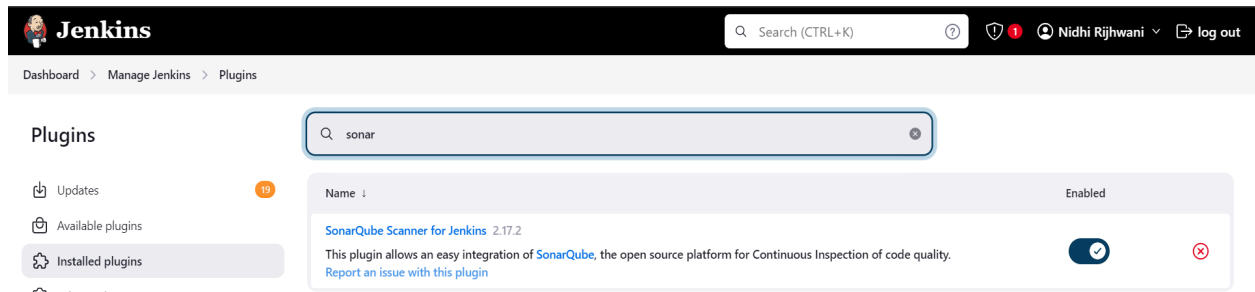
3. Once the container is up and running, you can check the status of SonarQube at localhost port 9000.
4. Login to SonarQube using username admin and password admin.



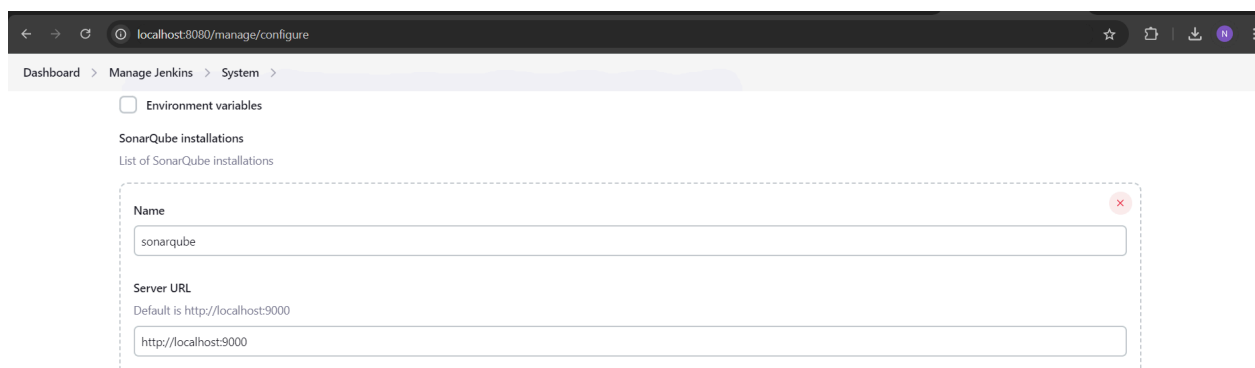
5. Create a manual project in SonarQube with the name **sonarqube**.



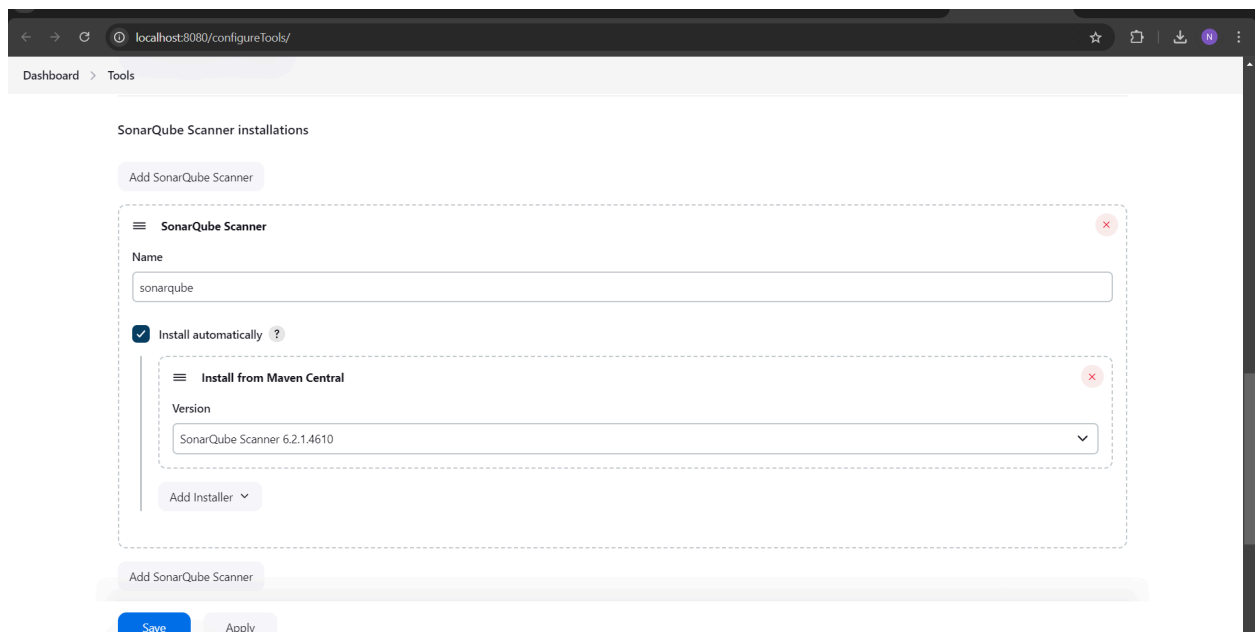
SetUp the project and come back to Jenkins Dashboard.
Go to Manage Jenkins and search for SonarQube Scanner for Jenkins and install it.



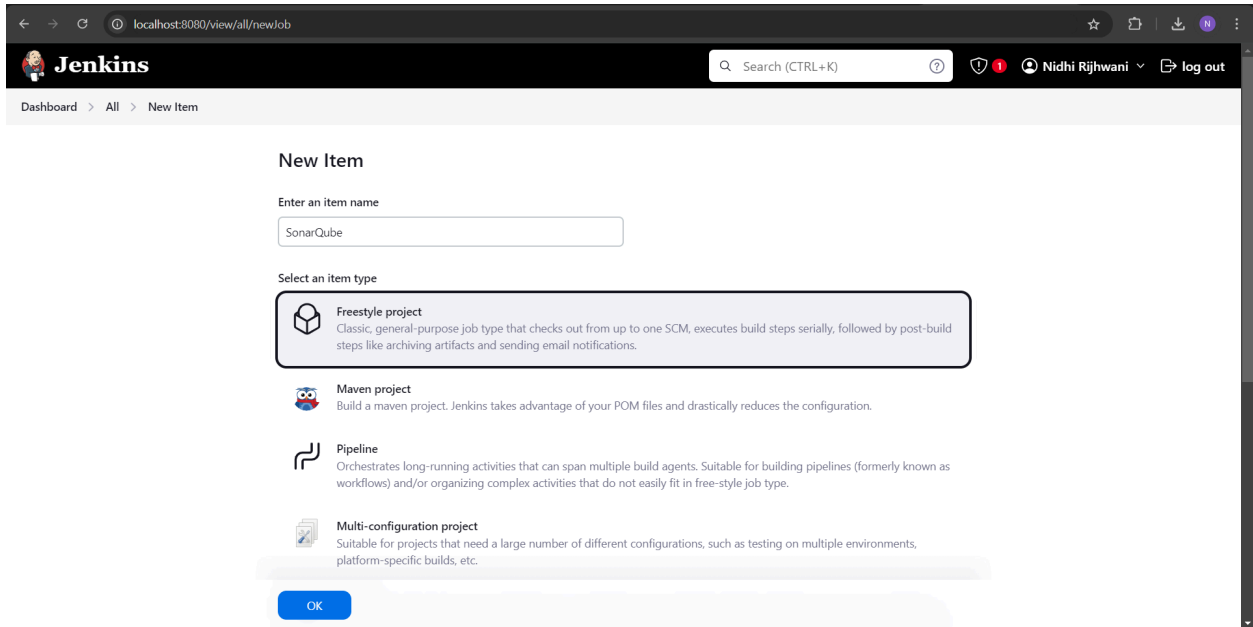
6. Under Jenkins 'Configure System', look for SonarQube Servers and enter the details.



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



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



The screenshot shows the Jenkins 'New Item' page. The browser address bar is 'localhost:8080/view/all/newJob'. The Jenkins logo and name are at the top left. A search bar with 'Search (CTRL+K)' is at the top right. Below the header, the breadcrumb is 'Dashboard > All > New Item'. The main section is titled 'New Item'. It has a text input field 'Enter an item name' with 'SonarQube' entered. Below it is a section 'Select an item type' with four options: 'Freestyle project' (selected), 'Maven project', 'Pipeline', and 'Multi-configuration project'. Each option has a brief description. At the bottom is an 'OK' button.

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.

Maven project
Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.

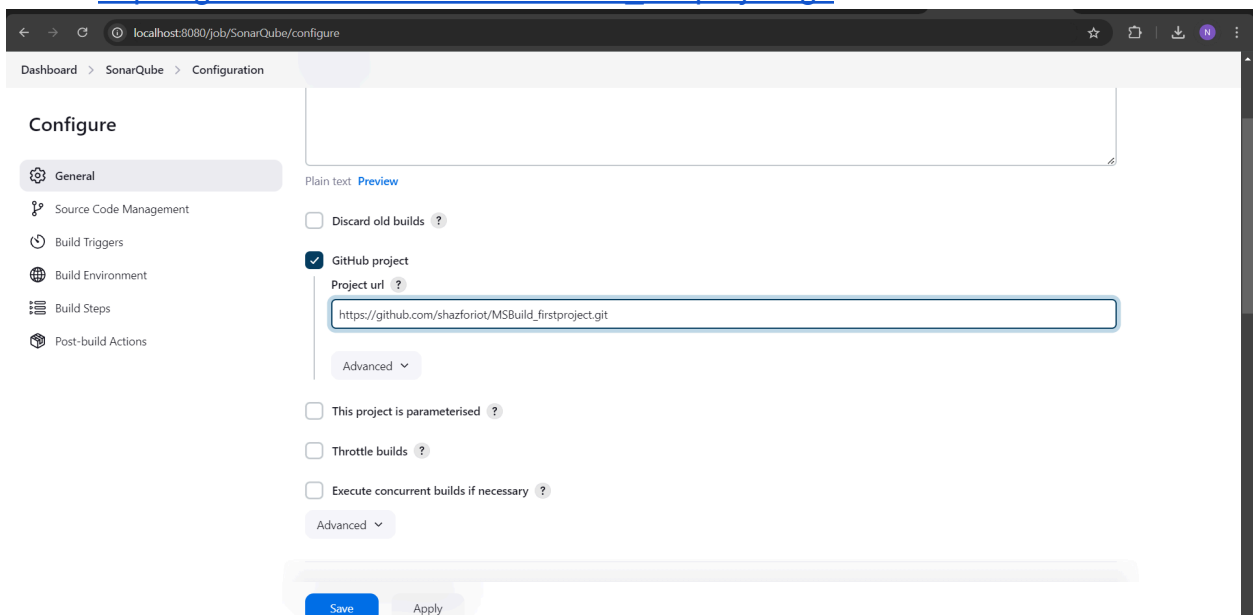
Pipeline
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

Multi-configuration project
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

OK

9. Choose this GitHub repository in Source Code Management.

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



The screenshot shows the Jenkins 'Configuration' page for the 'SonarQube' job. The browser address bar is 'localhost:8080/job/SonarQube/configure'. The breadcrumb is 'Dashboard > SonarQube > Configuration'. The left sidebar has a 'Configure' section with a list of tabs: 'General' (selected), 'Source Code Management', 'Build Triggers', 'Build Environment', 'Build Steps', and 'Post-build Actions'. The main area is titled 'Configure' and has a 'Plain text' preview. It contains several checkboxes: 'Discard old builds' (unchecked), 'GitHub project' (checked), 'This project is parameterised' (unchecked), 'Throttle builds' (unchecked), and 'Execute concurrent builds if necessary' (unchecked). The 'GitHub project' section has a 'Project url' field with the value 'https://github.com/shazforiot/MSBuild_firstproject.git'. At the bottom are 'Save' and 'Apply' buttons.

Configure

General

Source Code Management

Build Triggers

Build Environment

Build Steps

Post-build Actions

Plain text [Preview](#)

☐ Discard old builds ?

☒ **GitHub project** ?

Project url ?

[https://github.com/shazforiot/MSBuild_firstproject.git](#)

Advanced ▾

☐ This project is parameterised ?

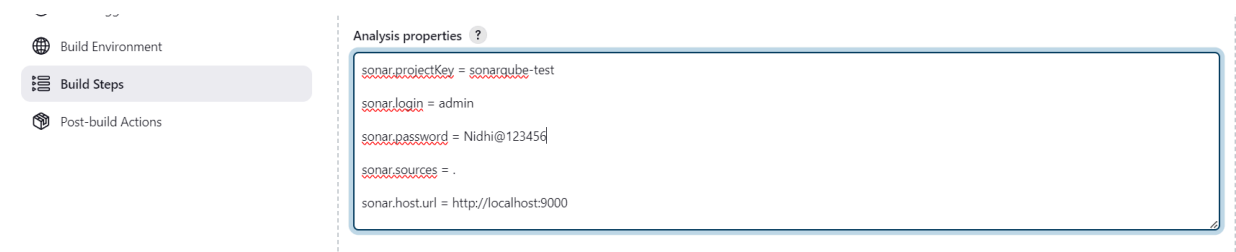
☐ Throttle builds ?

☐ Execute concurrent builds if necessary ?

Advanced ▾

Save Apply

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



The screenshot shows the Jenkins 'Build Steps' page for the 'SonarQube' job. The browser address bar is 'localhost:8080/job/SonarQube/configure'. The breadcrumb is 'Dashboard > SonarQube > Configuration'. The left sidebar has a 'Build Steps' section with a list of tabs: 'Build Environment' (selected), 'Build Steps' (selected), and 'Post-build Actions'. The main area is titled 'Build Steps' and has a 'Plain text' preview. It contains a text area with the following analysis properties: 'sonar.projectKey = sonarqube-test', 'sonar.login = admin', 'sonar.password = Nidhi@123456', 'sonar.sources = .', and 'sonar.host.url = http://localhost:9000'. At the bottom are 'Save' and 'Apply' buttons.

Build Steps

Build Environment

Build Steps

Post-build Actions

Plain text [Preview](#)

Analysis properties ?

`sonar.projectKey = sonarqube-test`

`sonar.login = admin`

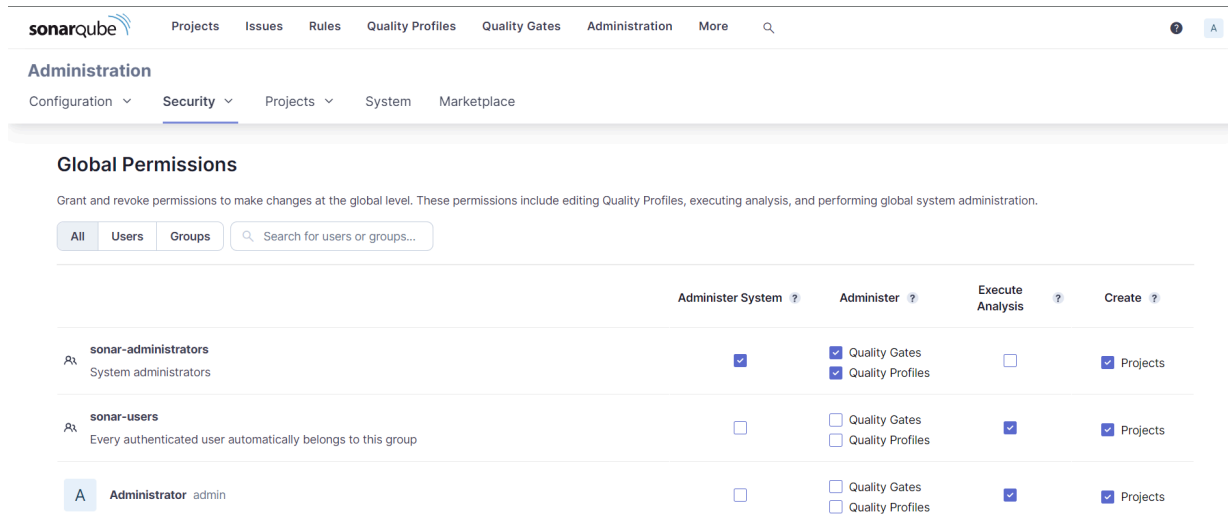
`sonar.password = Nidhi@123456`

`sonar.sources = .`

`sonar.host.url = http://localhost:9000`

Save Apply

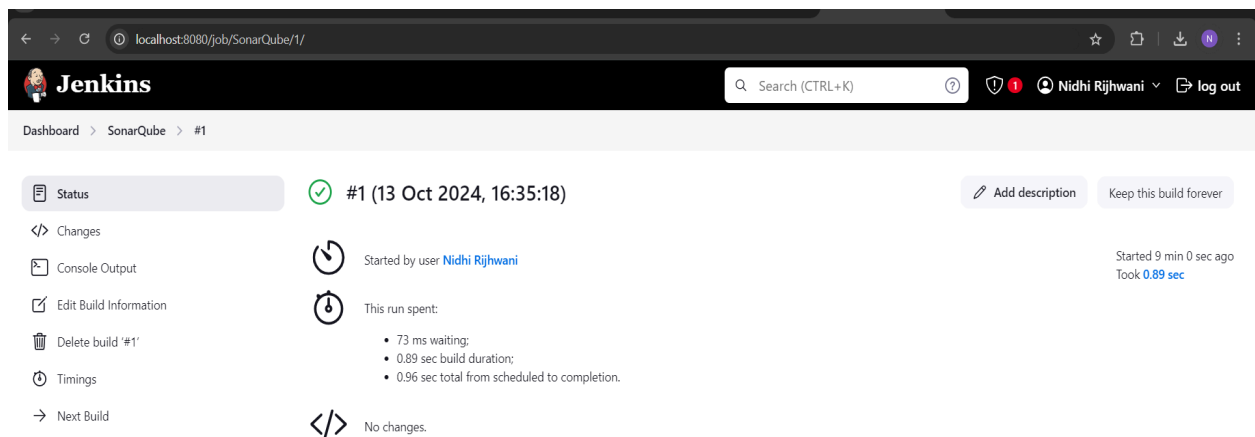
11. Go to http://localhost:9000/<user_name>/permissions and allow Execute Permissions to the Admin user.



The screenshot shows the SonarQube Administration interface, specifically the 'Global Permissions' section under 'Security'. It lists permissions for three groups: 'sonar-administrators', 'sonar-users', and 'Administrator admin'. The 'Administrator admin' group has 'Execute Analysis' and 'Create' permissions checked.

	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
A Administrator admin	<input type="checkbox"/>	<input type="checkbox"/> Quality Gates <input type="checkbox"/> Quality Profiles	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/> Projects

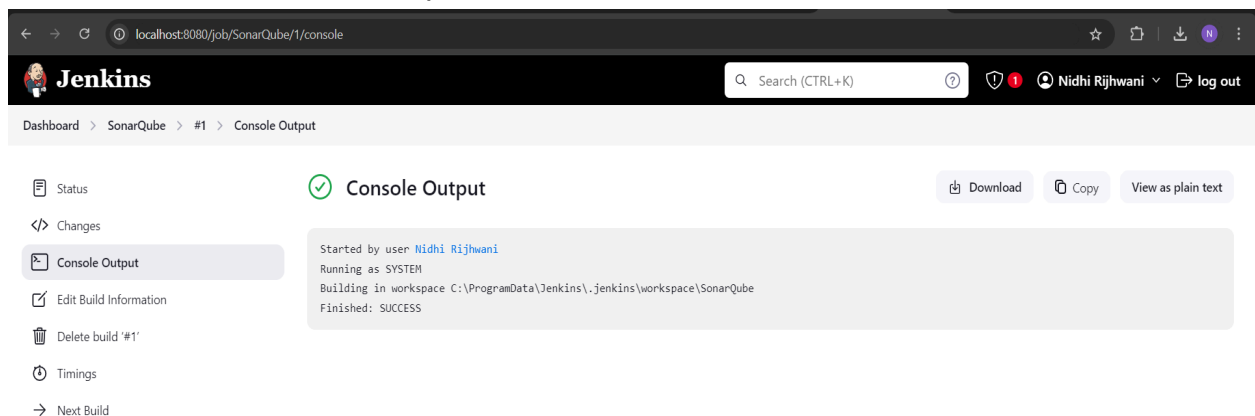
12. Run the build.



The screenshot shows the Jenkins job #1 console output. The build was started by user 'Nidhi Rijhwani' and completed successfully. The console output shows the build process and the final status 'SUCCESS'.

```
Started by user Nidhi Rijhwani
This run spent:
  • 73 ms waiting;
  • 0.89 sec build duration;
  • 0.96 sec total from scheduled to completion.
No changes.
```

13. Check the console output.



The screenshot shows the Jenkins job #1 console output. The build was started by user 'Nidhi Rijhwani' and completed successfully. The console output shows the build process and the final status 'SUCCESS'.

```
Started by user Nidhi Rijhwani
Running as SYSTEM
Building in workspace C:\ProgramData\Jenkins\jenkins\workspace\SonarQube
Finished: SUCCESS
```

Jenkins

Search (CTRL+K)

?

1

Nidhi Rijhwani

▼

log out

Dashboard >

+ New Item

Build History

Manage Jenkins

My Views

Build Queue

No builds in the queue.

Build Executor Status

1 Idle

2 Idle

All

+

S	W	Name ↓	Last Success	Last Failure	Last Duration	
		Multiply-two-numbers-test	10 days #2	10 days #1	20 sec	
		SonarQube	14 min #1	N/A	0.89 sec	

Icon: S M L

...

Conclusion :

Integrating Static Application Security Testing (SAST) with Jenkins and tools like SonarQube or GitLab is essential for enhancing software security. By automating code scans during the build process, teams can identify and address vulnerabilities early, improving overall code quality. This integration promotes a security-focused culture within DevOps, enabling more secure and resilient applications while fostering collaboration among development, security, and operations teams.