

Installation Links :Git : <https://git-scm.com/download/win>

Jenkins : <https://www.jenkins.io/download/thank-you-downloading-windows-installer-stable>

Docker : <https://www.docker.com/products/docker-desktop/>

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.

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>

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](https://www.digitalocean.com/community/tutorials/how-to-install-jenkins-on-ubuntu-20-04)

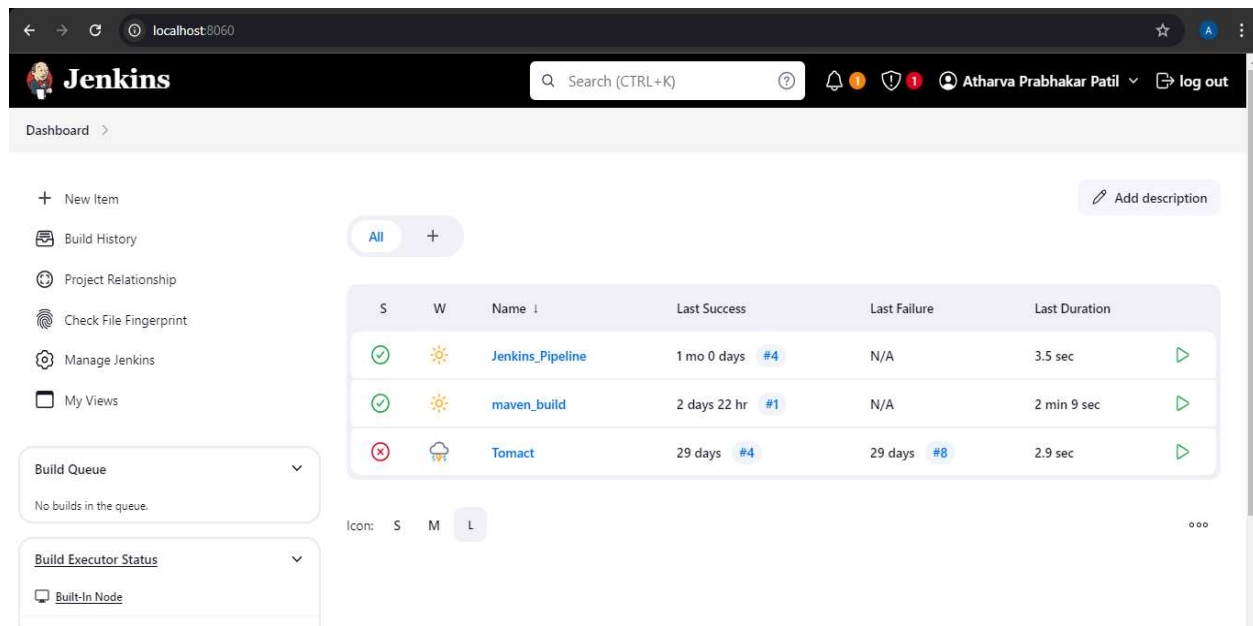
[Open SSH](#)

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 -
docker -v

docker pull sonarqube

try the new cross-platform powershell <https://aka.ms/powershell>

```
PS C:\Users\sushmita> docker -v
Docker version 27.1.1, build 6312585
PS C:\Users\sushmita> docker pull sonarqube
Using default tag: latest
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
docker.io/library/sonarqube:latest
```

Warning: run below command only once

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

```
PS C:\Users\sushmita> docker run -d --name sonarqube -e SONAR_ES_BOOTSTRAP_CHECKS_DISABLE=true -p 9000:9000 sonarqube:latest
ee4dad9f96364a57945728ce2499989ee0787873adacfc3d36d9aaa39258c93f3
PS C:\Users\sushmita>
```

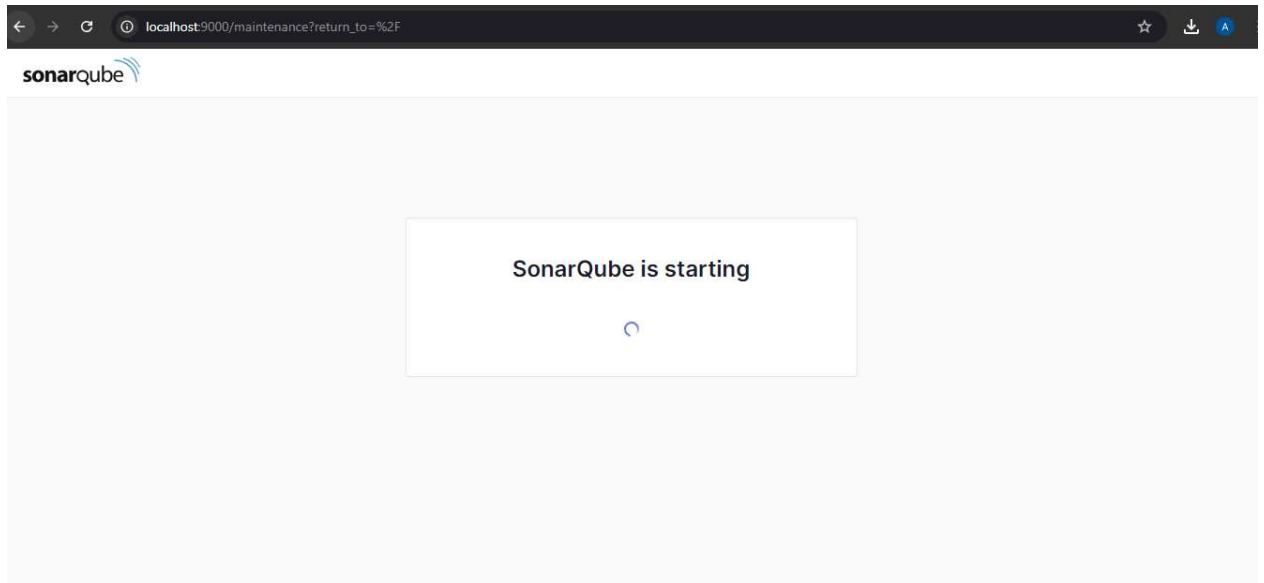
RAM 0.96 GB CPU 11.11% Disk -- GB avail. of -- GB

BETA

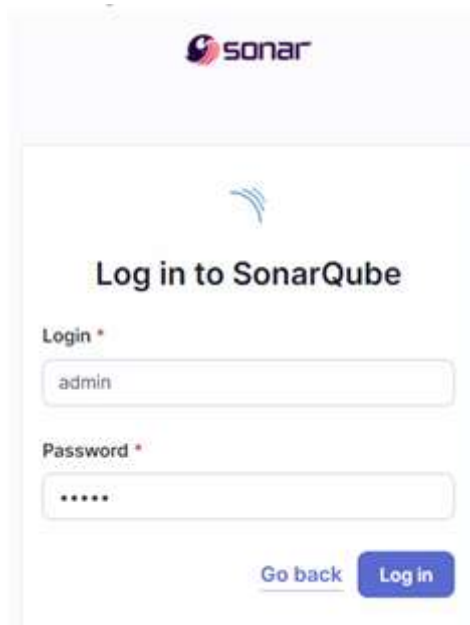
Terminal

New version

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**

1 of 2

Create a local project

Project display name *

sonarqube

Project key *

sonarqube

Main branch name *

main

The name of your project's default branch [Learn More](#)

CancelNext

2 of 2

Set up project for Clean as You Code

The new code definition sets which part of your code will be considered new code. This helps you focus attention on the most recent changes to your project, enabling you to follow the Clean as You Code methodology. Learn more: [Defining New Code](#)

Choose the baseline for new code for this project

☒ Use the global setting

Previous version

Any code that has changed since the previous version is considered new code.
Recommended for projects following regular versions or releases.

Define a specific setting for this project

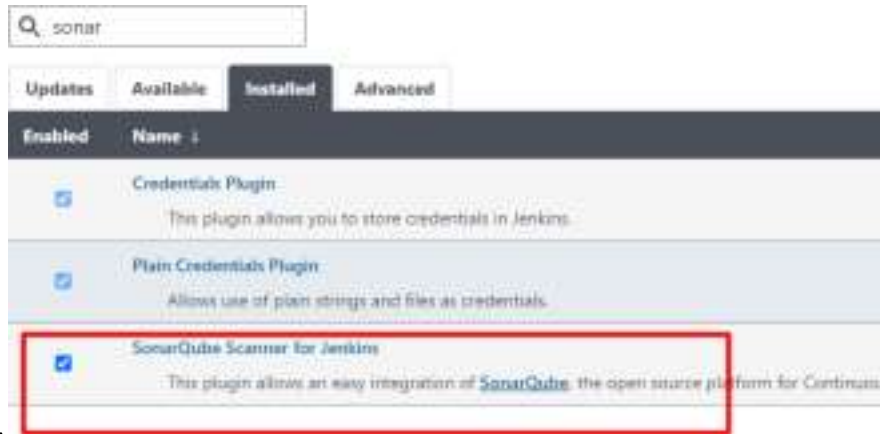
Previous version

Any code that has changed since the previous version is considered new code.
Recommended for projects following regular versions or releases.

Number of days

Setup the project and come back to Jenkins Dashboard.

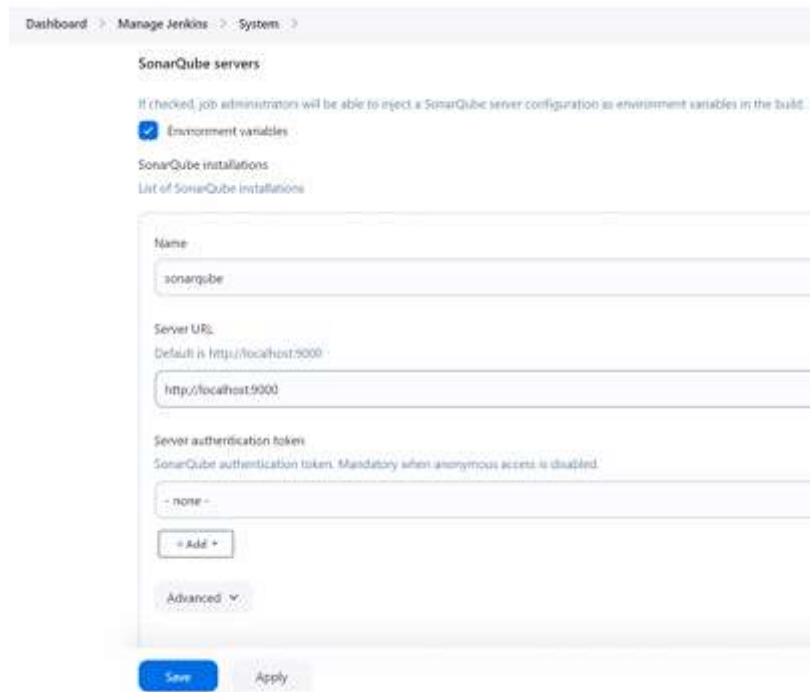
Go to Manage Jenkins and search for SonarQube Scanner for Jenkins and install



it.

6. Under Manage Jenkins, Systems look for SonarQube Servers and enter the details.

Enter the Server Authentication token if needed.

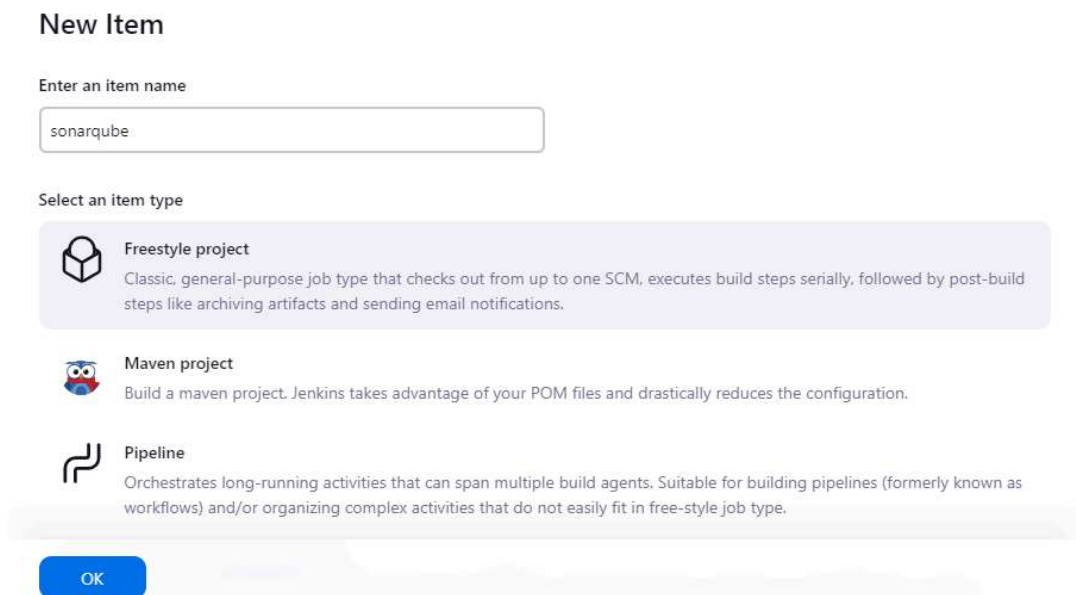


7. Search for SonarQube Scanner under Global Tool Configuration. Choose the latest configuration and choose Install automatically.



The screenshot shows the 'SonarQube Scanner' configuration window. It has a title bar with a menu icon and a close button. The 'Name' field contains 'sonarqube'. Below it, a red error message 'Required' is displayed. The 'Install automatically' checkbox is checked. A sub-section titled 'Install from Maven Central' contains a 'Version' dropdown menu with 'SonarQube Scanner 6.2.0.4584' selected. At the bottom, there is an 'Add Installer' button with a dropdown arrow.

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



The screenshot shows the 'New Item' page in Jenkins. The title is 'New Item'. Below it, the 'Enter an item name' field contains 'sonarqube'. The 'Select an item type' section shows three options: 'Freestyle project' (selected), 'Maven project', and 'Pipeline'. The 'Freestyle project' option is highlighted with a description: '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.' The 'Maven project' option has a description: 'Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.' The 'Pipeline' option has a description: '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.' At the bottom, there is an 'OK' button.

9. Choose this GitHub repository in Source Code Management.

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

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



Git ?

Repositories ?

Repository URL ?

Credentials ?

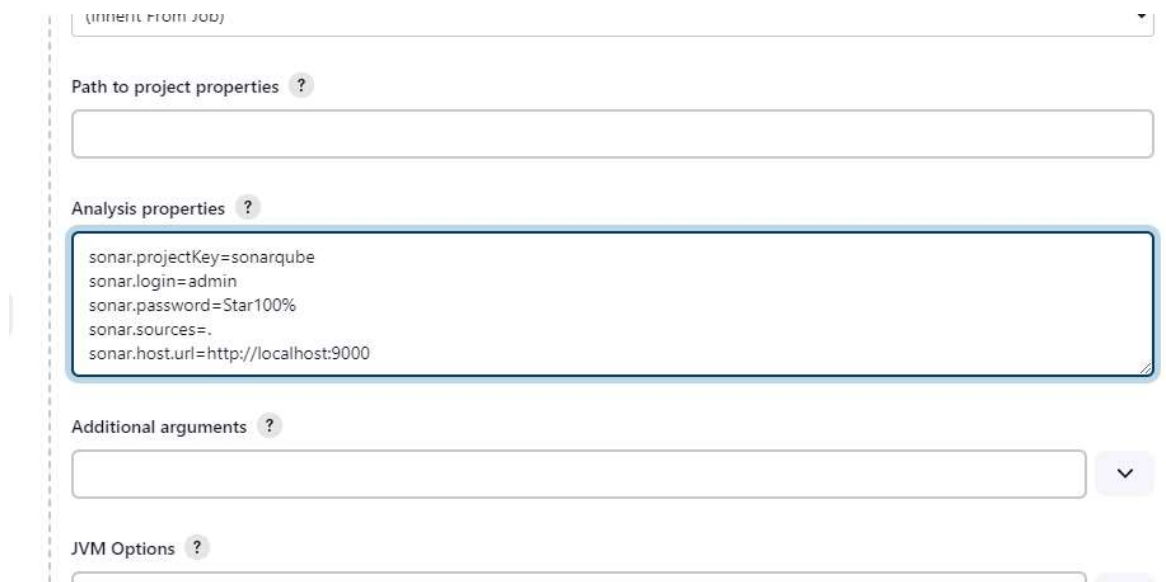
+ Add

Advanced

Add Repository

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 and allow Execute Permissions to the Admin user.

sonar.projectKey=sonarqube
sonar.login=admin
sonar.password= Star100%
sonar.sources=.
sonar.host.url=http://localhost:9000



(inherited from job)

Path to project properties ?

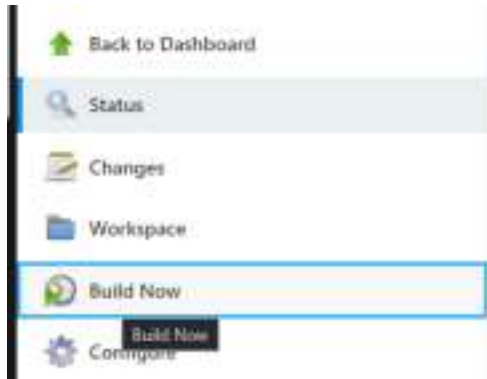
Analysis properties ?

sonar.projectKey=sonarqube
sonar.login=admin
sonar.password=Star100%
sonar.sources=.
sonar.host.url=http://localhost:9000

Additional arguments ?

JVM Options ?

12. Run The Build.



Check the console output.



Console Output

[Download](#)
[Copy](#)
[View as plain text](#)

```
Started by user Atharva Prabhakar Patil
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
Cloning the remote Git repository
Cloning repository https://github.com/shazforiot/MSBuild_firstproject.git
> git init C:\ProgramData\Jenkins\.jenkins\workspace\sonarqube # timeout=10
Fetching upstream changes from https://github.com/shazforiot/MSBuild_firstproject.git
> git --version # timeout=10
> git --version # 'git version 2.43.0.windows.1'
> git fetch --tags --force --progress -- https://github.com/shazforiot/MSBuild_firstproject.git
+refs/heads/*:refs/remotes/origin/* # timeout=10
> git config remote.origin.url https://github.com/shazforiot/MSBuild_firstproject.git # timeout=10
> git config --add remote.origin.fetch +refs/heads/*:refs/remotes/origin/* # timeout=10
Avoid second fetch
> git rev-parse "refs/remotes/origin/master^{commit}" # timeout=10
Checking out Revision f2bc042c04c6e72427c380bcaee6d6fee7b49adf (refs/remotes/origin/master)
```

```
01:48:52.594 WARN Incremental PR analysis: Could not determine common base path, cache will not be computed. Consider
setting 'sonar.projectBaseDir' property.
01:48:52.596 INFO Sensor C# File Caching Sensor [csharp] (done) | time=9ms
01:48:52.597 INFO Sensor Zero Coverage Sensor
01:48:52.641 INFO Sensor Zero Coverage Sensor (done) | time=44ms
01:48:52.653 INFO SCM Publisher SCM provider for this project is: git
01:48:52.656 INFO SCM Publisher 4 source files to be analyzed
01:48:55.556 INFO SCM Publisher 4/4 source files have been analyzed (done) | time=2899ms
01:48:55.560 INFO CPD Executor Calculating CPD for 0 files
01:48:55.565 INFO CPD Executor CPD calculation finished (done) | time=0ms
01:48:55.607 INFO SCM revision ID 'f2bc042c04c6e72427c380bcaee6d6fee7b49adf'
01:48:56.427 INFO Analysis report generated in 606ms, dir size=201.0 kB
01:48:56.710 INFO Analysis report compressed in 281ms, zip size=22.2 kB
01:48:57.922 INFO Analysis report uploaded in 1208ms
01:48:57.925 INFO ANALYSIS SUCCESSFUL, you can find the results at: http://localhost:9000/dashboard?id=sonarqube
01:48:57.927 INFO Note that you will be able to access the updated dashboard once the server has processed the submitted
analysis report
01:48:57.929 INFO More about the report processing at http://localhost:9000/api/ce/task?id=8b35a5b7-a50e-424b-a77c-
6aa39d925923
01:48:57.966 INFO Analysis total time: 1:09.987 s
01:48:57.995 INFO SonarScanner Engine completed successfully
01:48:58.203 INFO EXECUTION SUCCESS
01:48:58.205 INFO Total time: 3:30.482s
Finished: SUCCESS
```

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

The screenshot displays the SonarQube web interface. The top navigation bar includes links for Projects, Issues, Rules, Quality Profiles, Quality Gates, Administration, and More. The main content area shows a list of projects, with 'sonarqube' selected. The project status is 'Passed' with a green checkmark. The last analysis was 8 minutes ago. The main branch is empty. The left sidebar shows filters for Quality Gate (Passed: 1, Failed: 0) and Reliability (A: 1, B: 0, C: 0, D: 0, E: 0). Below the project list, the 'Overview' tab is active, showing a 'Passed' status with a green checkmark. A warning message states: 'The last analysis has warnings. See details'. The 'Overall Code' tab is selected, showing three metrics: Security (0 Open issues, A), Reliability (0 Open issues, A), and Maintainability (0 Open issues, A). Each metric has a bar chart with 'H', 'M', and 'L' categories.

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

Conclusion

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

Additional resources

Sonarqube installation on aws Ubuntu

<https://www.coachdevops.com/2020/04/install-sonarqube-on-ubuntu-how-to.html>

<https://awstip.com/installing-sonarqube-on-aws-ec2-instance-and-integrating-it-with-aws-code-pipeline-abec99416ba4>

`docker-compose up -d && docker ps`