

AdvanceDevops 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.

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>[Open SSH](#)**Prerequisites:**

- [Jenkins installed](#)
- [Docker Installed](#) (for SonarQube)

(sudo apt-get install docker-ce=5:20.10.15~3-0~ubuntu-jammy
docker-ce-cli=5:20.10.15~3-0~ubuntu-jammy containerd.io docker-compose-plugin)

- SonarQube Docker Image

Steps to integrate Jenkins with SonarQube

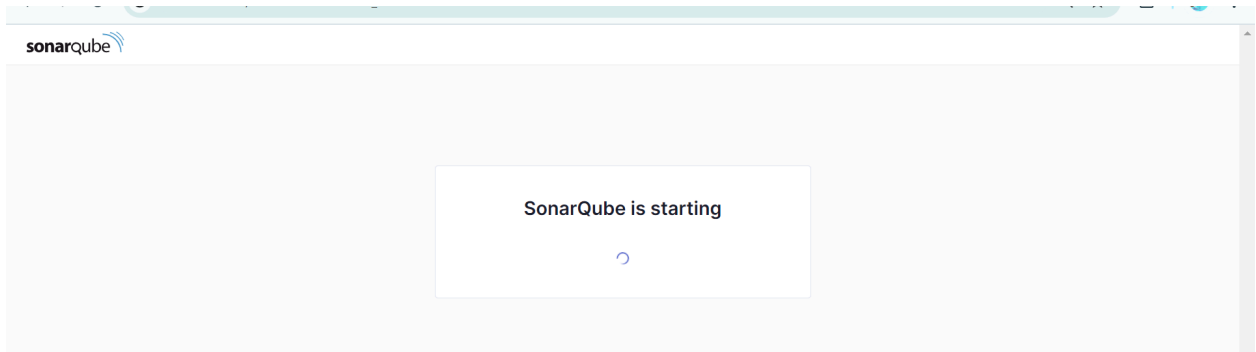
1. Open up Jenkins Dashboard on localhost, port 8080 or whichever port it is at for you.
2. Run SonarQube in a Docker container using this command -

```
PS C:\Users\91900> docker run -d --name sonarqube -e SONAR_ES_BOOTSTRAP_CHECKS_DISABLE=true -p 9000:9000 sonarqube:lates
t
8b2a833004ac39a9b009118bacac47e5808c9ec8df3f59f8657bd23fa23f48f2
```

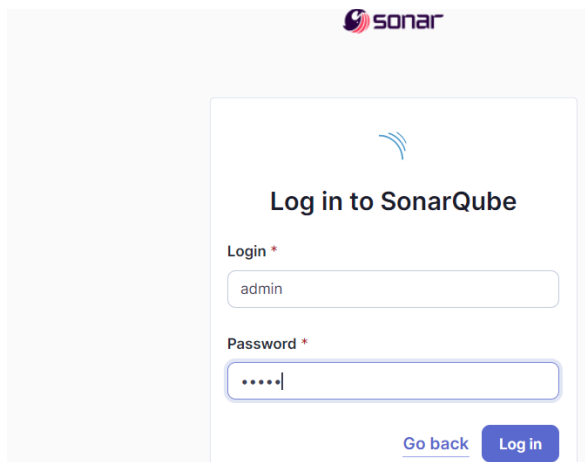
Warning: run below command only once

```
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**

1 of 2

Create a local project

Project display name *

 ✓

Project key *

 ✓

Main branch name *

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

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 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.

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.

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

Server URL

Default is `http://localhost:9000`

Server authentication token

SonarQube authentication token. Mandatory when anonymous access is disabled.

- none -

+ Add

Advanced

Save Apply

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

Add SonarScanner for MSBuild

SonarQube Scanner installations

Add SonarQube Scanner

SonarQube Scanner

Name

☒ Install automatically ?

Install from Maven Central

Version

SonarQube Scanner 6.2.0.4584

Add Installer

Add SonarQube Scanner


Save Apply


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


New Item


Enter an item name


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.

**Folder**
Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a

OK

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

Configure

- General
- Source Code Management**
- Build Triggers
- Build Environment
- Build Steps
- Post-build Actions

None

Git ?

Repositories ?

Repository URL ?

Please enter Git repository.

Credentials ?

- none -

+ Add

Advanced

Add Repository

Branches to build ?

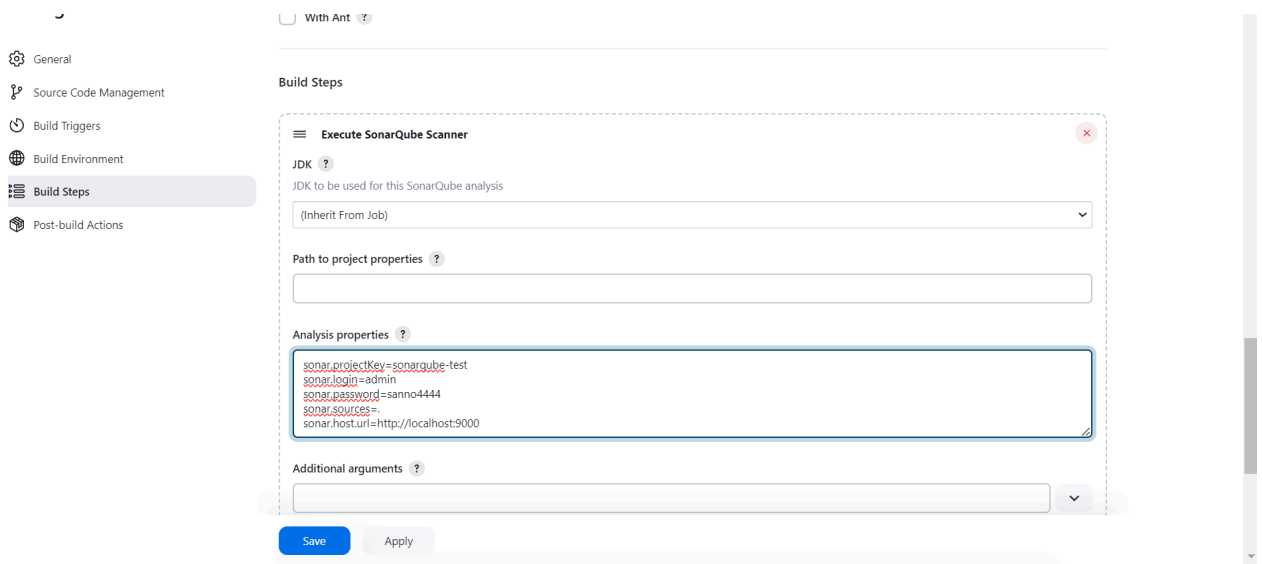
Branch Specifier (blank for 'any') ?

Save

Apply

the integration.

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



With Ant ?

Build Steps

Execute SonarQube Scanner

JDK ?

JDK to be used for this SonarQube analysis

(Inherit From Job)

Path to project properties ?

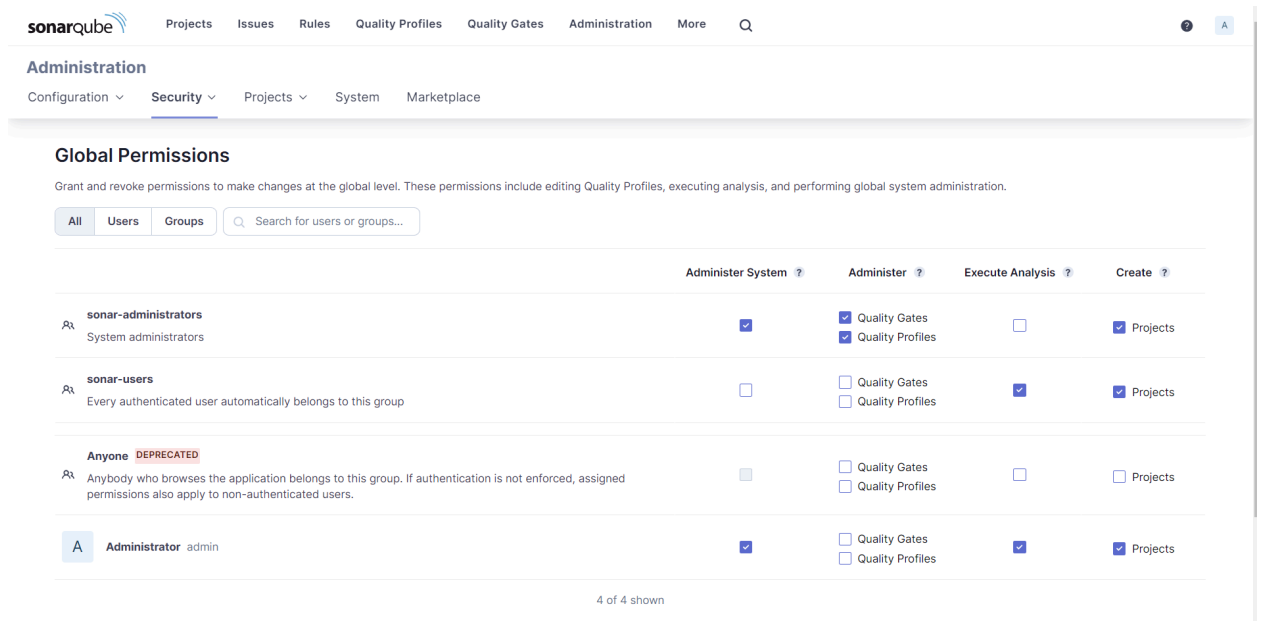
Analysis properties ?

```
sonar.projectKey=sonarqube-test
sonar.login=admin
sonar.password=sanno4444
sonar.sources=
sonar.host.url=http://localhost:9000
```

Additional arguments ?

Save Apply

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



sonarqube Projects Issues Rules Quality Profiles Quality Gates Administration More

Administration

Configuration Security Projects System Marketplace

Global Permissions

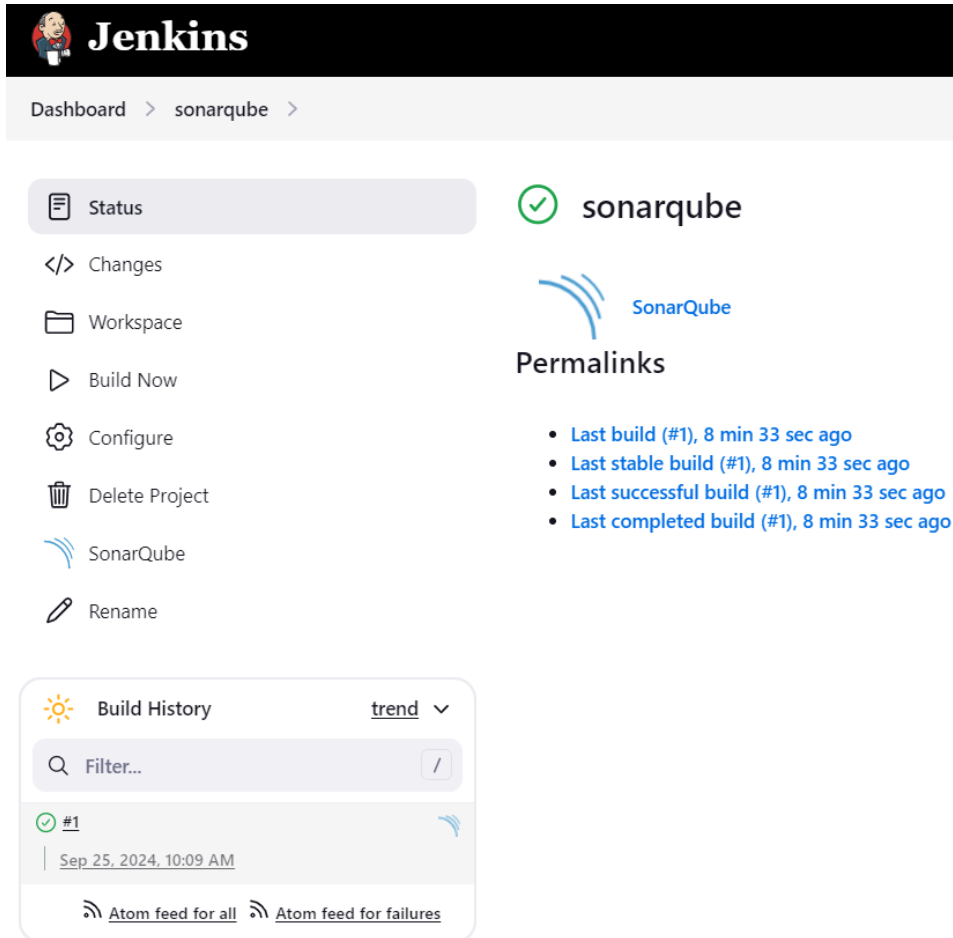
Grant and revoke permissions to make changes at the global level. These permissions include editing Quality Profiles, executing analysis, and performing global system administration.

All Users Groups Search for users or groups...

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

4 of 4 shown

12. Run The Build.



The Jenkins dashboard for the 'sonarqube' project shows a 'Status' section with various actions: Changes, Workspace, Build Now, Configure, Delete Project, SonarQube, and Rename. A 'sonarqube' status indicator with a green checkmark is present. Below this, a 'Permalinks' section lists four build links, all indicating a duration of 8 min 33 sec ago. A 'Build History' section shows a single build (#1) from Sep 25, 2024, 10:09 AM, with links for Atom feeds for all and failures.

Status

- Changes
- Workspace
- Build Now
- Configure
- Delete Project
- SonarQube
- Rename

sonarqube

Permalinks

- Last build (#1), 8 min 33 sec ago
- Last stable build (#1), 8 min 33 sec ago
- Last successful build (#1), 8 min 33 sec ago
- Last completed build (#1), 8 min 33 sec ago

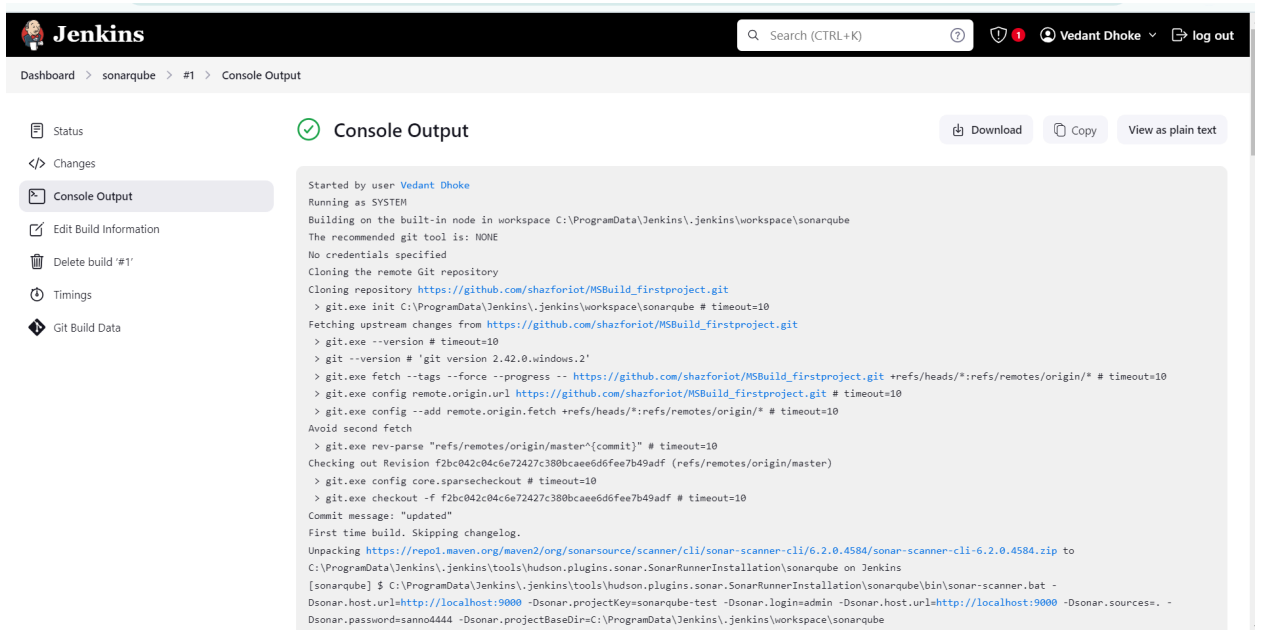
Build History trend

Filter...

#1
Sep 25, 2024, 10:09 AM

Atom feed for all Atom feed for failures

Check the console output.



The Jenkins console output for build #1 shows the execution of a SonarScanner CLI command. The output includes the repository URL, git commands, and the SonarScanner command. The build is successful, and the console output is displayed in a scrollable area.

Console Output

Started by user Vedant Dhoke
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.exe init C:\ProgramData\Jenkins\jenkins\workspace\sonarqube # 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.2'
> git.exe fetch --tags --force --progress -- https://github.com/shazforiot/MSBuild_firstproject.git +refs/heads/*:refs/remotes/origin/* # timeout=10
> git.exe config remote.origin.url https://github.com/shazforiot/MSBuild_firstproject.git # timeout=10
> git.exe config --add remote.origin.fetch +refs/heads/*:refs/remotes/origin/* # timeout=10
Avoid second fetch
> 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"
First time build. Skipping changelog.
Unpacking https://repo1.maven.org/maven2/org/sonarsource/scanner/cli/sonar-scanner-cli-6.2.0.4584/sonar-scanner-cli-6.2.0.4584.zip to C:\ProgramData\Jenkins\jenkins\tools\hudson.plugins.sonar.SonarRunnerInstallation\sonarqube on Jenkins
[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=sanno4444 -Dsonar.projectBaseDir=C:\ProgramData\Jenkins\jenkins\workspace\sonarqube

```

SonarScanner for .NET 5.x or higher, see https://redirect.sonarsource.com/doc/install-configure-scanner-msbuild.html
10:10:57.397 INFO Sensor C# [csharp] (done) | time=2ms
10:10:57.397 INFO Sensor Analysis Warnings import [csharp]
10:10:57.399 INFO Sensor Analysis Warnings import [csharp] (done) | time=4ms
10:10:57.401 INFO Sensor C# File Caching Sensor [csharp]
10:10:57.405 WARN Incremental PR analysis: Could not determine common base path, cache will not be computed. Consider setting 'sonar.projectBaseDir'
property.
10:10:57.405 INFO Sensor C# File Caching Sensor [csharp] (done) | time=5ms
10:10:57.405 INFO Sensor Zero Coverage Sensor
10:10:57.424 INFO Sensor Zero Coverage Sensor (done) | time=19ms
10:10:57.428 INFO SCM Publisher SCM provider for this project is: git
10:10:57.430 INFO SCM Publisher 4 source files to be analyzed
10:10:58.315 INFO SCM Publisher 4/4 source files have been analyzed (done) | time=883ms
10:10:58.324 INFO CPD Executor Calculating CPD for 0 files
10:10:58.363 INFO CPD Executor CPD calculation finished (done) | time=0ms
10:10:58.372 INFO SCM revision ID 'f2bc042c04c6e72427c380bcaee6d6fee7b49adf'
10:10:58.843 INFO Analysis report generated in 226ms, dir size=201.0 kB
10:10:58.903 INFO Analysis report compressed in 45ms, zip size=22.2 kB
10:10:59.397 INFO Analysis report uploaded in 491ms
10:10:59.401 INFO ANALYSIS SUCCESSFUL, you can find the results at: http://localhost:9000/dashboard?id=sonarqube-test
10:10:59.402 INFO Note that you will be able to access the updated dashboard once the server has processed the submitted analysis report
10:10:59.403 INFO More about the report processing at http://localhost:9000/api/ce/task?id=2620d8fe-827f-4a3c-999f-1ed8a7b15249
10:10:59.429 INFO Analysis total time: 30.223 s
10:10:59.431 INFO SonarScanner Engine completed successfully
10:10:59.519 INFO EXECUTION SUCCESS
10:10:59.521 INFO Total time: 47.815s
Finished: SUCCESS

```

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

The screenshot shows the SonarQube web interface for a project named 'main'. The top navigation bar includes links for Projects, Issues, Rules, Quality Profiles, Quality Gates, Administration, and More. The main content area shows the project overview with a 'Passed' status and a green checkmark. Below this, there are tabs for 'New Code' and 'Overall Code'. The 'Overall Code' tab is active, displaying various metrics: Security (0 Open issues, A grade), Reliability (0 Open issues, A grade), Maintainability (0 Open issues, A grade), Accepted issues (0), Coverage (0.0%), and Duplications (0.0%). The interface also shows a warning that the last analysis has warnings and a link to 'See details'.

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

Conclusion

1. Docker Container Issues: The SonarQube container might not start because your system doesn't have enough memory or processing power. SonarQube needs around 2GB of RAM to work properly, so if your system is low on resources, the container won't run.

2. Login Problems in SonarQube: You might have trouble logging in with the default username (admin) and password (admin). This could happen if there was a configuration issue with SonarQube or if the default password was changed during previous setups.

3. Jenkins Plugin Installation Errors: While installing the SonarQube Scanner plugin in Jenkins, you might encounter failures due to network issues or proxy settings, preventing the plugin from downloading correctly.

4. Incorrect SonarQube Configuration in Jenkins: While configuring SonarQube in Jenkins, entering the wrong project key, username, or password can cause the scan to fail. Ensuring accurate information is critical for a successful scan.