

**Advanced Devops
Experiment No:07**

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

installing-the-default-jre-jdk

Step 1 Install JDK 1.8

sudo apt-get install openjdk-8-jre

sudo apt install default-jre /

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

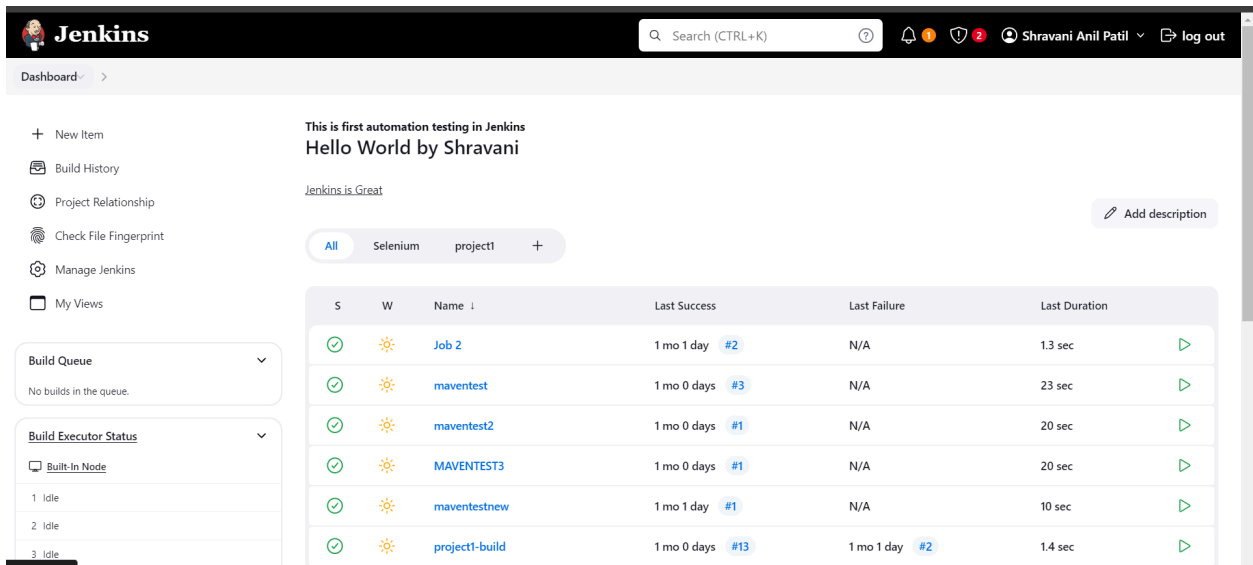
docker-ce-cli=5:20.10.15~3-0~ubuntu-jammy containerd.io

docker-compose-plugin)

- SonarQube Docker Image

Steps to integrate Jenkins with SonarQube

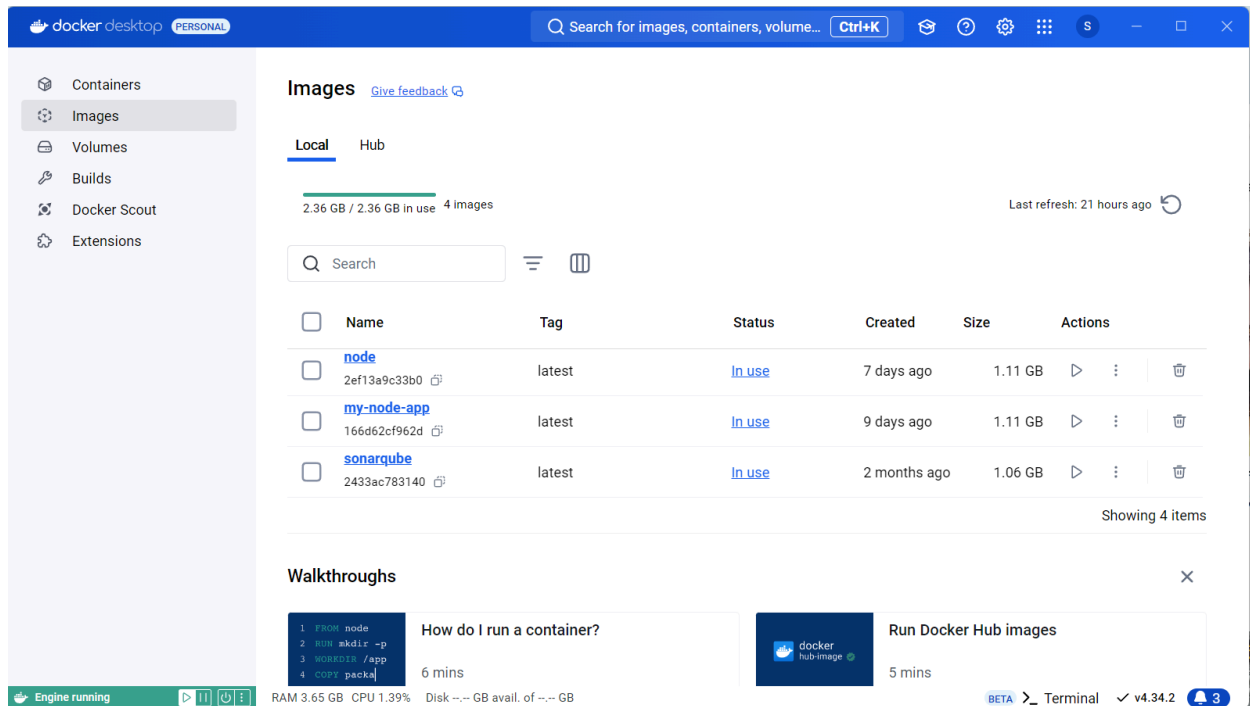
1. Open up Jenkins Dashboard on localhost, port 8080 or whichever port it is at for you.



The screenshot shows the Jenkins Dashboard interface. At the top, there's a header with the Jenkins logo, a search bar, and user information (Shravani Anil Patil). The main content area displays a message: "This is first automation testing in Jenkins Hello World by Shravani". Below this, there's a section titled "Jenkins is Great" with a filter bar showing "All", "Selenium", and "project1". A table lists several builds with columns for status (S), icon (W), name, last success, last failure, and last duration. The table shows builds for "Job 2", "maventest", "maventest2", "MAVENTEST3", "maventestnew", and "project1-build". On the left sidebar, there are links for "New Item", "Build History", "Project Relationship", "Check File Fingerprint", "Manage Jenkins", and "My Views". Below these, there are sections for "Build Queue" (showing no builds) and "Build Executor Status" (showing 3 idle executors).

S	W	Name	Last Success	Last Failure	Last Duration
✓	☀	Job 2	1 mo 1 day #2	N/A	1.3 sec
✓	☀	maventest	1 mo 0 days #3	N/A	23 sec
✓	☀	maventest2	1 mo 0 days #1	N/A	20 sec
✓	☀	MAVENTEST3	1 mo 0 days #1	N/A	20 sec
✓	☀	maventestnew	1 mo 1 day #1	N/A	10 sec
✓	☀	project1-build	1 mo 0 days #13	1 mo 1 day #2	1.4 sec

2. Run SonarQube in a Docker container using this command
Ensure Docker engine is running

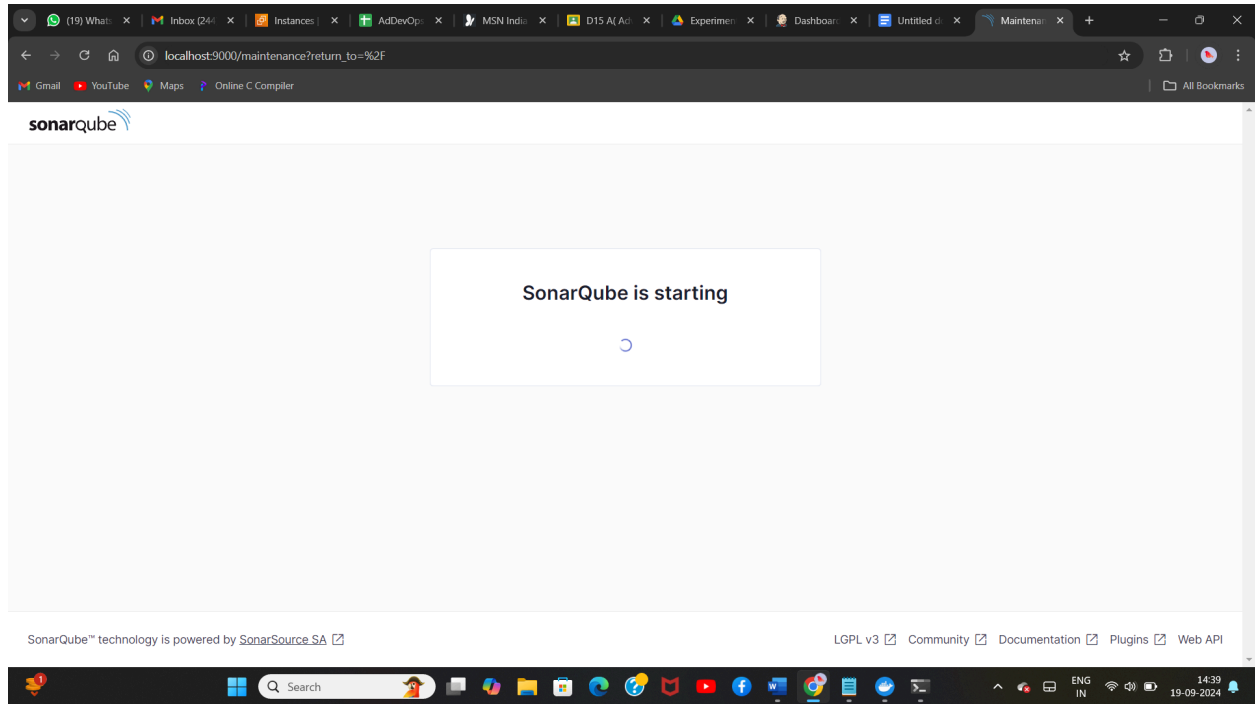


```
Microsoft Windows [Version 10.0.22631.4169]
(c) Microsoft Corporation. All rights reserved.

C:\Users\shrav>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: sha256:72e9feec71242af83faf65f95a40d5e3bb2822a6c3b2cda8568790f3d31aecde
Status: Downloaded newer image for sonarqube:latest
e4d231de58da6871a773bbd5b77ac510bb10d0ff003c24d82542d1f5cbce4ea1

C:\Users\shrav>
```

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 SonarQube1 with the name sonarqube1

1 of 2

Create a local project

Project display name *



Project key *



Main branch name *

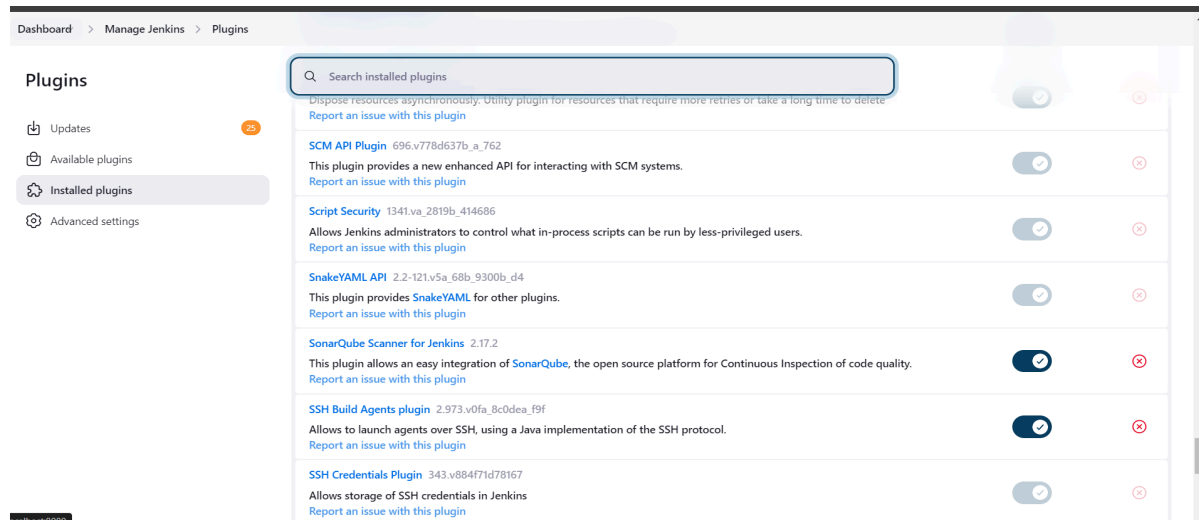
The name of your project's default branch [Learn More](#) 

Cancel

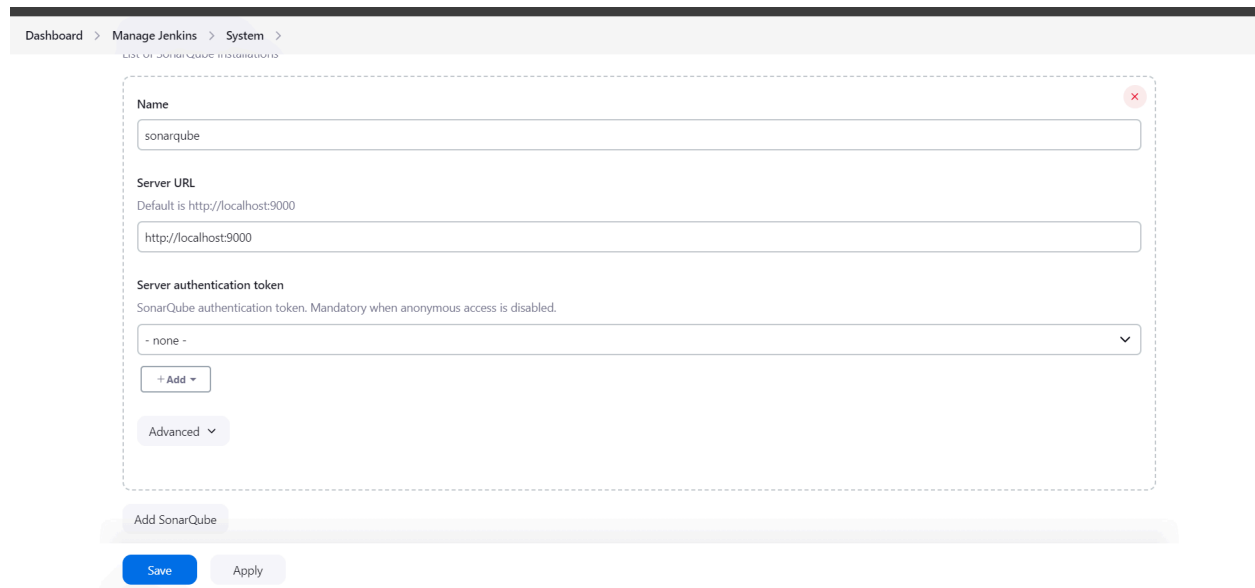
Next

6.Setup the project and come back to Jenkins Dashboard.

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



**7. Under Jenkins ‘Configure System’, look for SonarQube Servers and enter the details.
Enter the Server Authentication token if needed.**



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

Add SonarQube Scanner

SonarQube Scanner

Name

SonarQube

☒ Install automatically ?

Install from Maven Central

Version

SonarQube Scanner 6.2.0.4584

Add Installer

Add SonarQube Scanner

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

Dashboard > All > 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.

Folder
Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different

OK

10. Choose this GitHub repository in Source Code Management.

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

Git ?

Repositories ?

Repository URL ?

Credentials ?

Branches to build ?

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




Path to project properties ?

Analysis properties ?

Additional arguments ?

JVM Options ?

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

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

13. Run The Build.

 Status


 Changes

 Workspace

 Build Now


 Configure

 Delete Project






 SonarQube

 Rename


14. Console Output


**Jenkins**


Search (CTRL+K)


 Shravani Anil Patil  log out


Dashboard > SonarQube > #1 > Console Output


 Status


 Changes


 Console Output


 Edit Build Information


 Delete build '#1'


 Timings

 Git Build Data

 Console Output

 Download

 Copy

 View as plain text

```
Started by user Shravani Anil 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
> 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 2.46.0.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"
First time build. Skipping changelog.
[SonarQube] $ C:\ProgramData\Jenkins\jenkins\tools\udson.plugins.sonar.SonarRunnerInstallation\SonarQube\bin\sonar-scanner.bat -
Dsonar.host.url=http://localhost:9000 -Dsonar.projectKey=sonarqube -Dsonar.Login=squ_32cbb4ccc16482530991b7a25a9c5e5b980193fa -
Dsonar.host.url=http://localhost:9000 -Dsonar.sources=HelloWorldCore -Dsonar.projectBaseDir=C:\ProgramData\Jenkins\jenkins\workspace\SonarQube
```

Dashboard > SonarQube > #1 > Console Output

```
Dsonar.host.url=http://localhost:9000 -Dsonar.projectKey=sonarqube -Dsonar.Login=squ_32cbb4ccc16482530991b7a25a9c5e5b980193fa -
Dsonar.host.url=http://localhost:9000 -Dsonar.sources=HelloWorldCore -Dsonar.projectBaseDir=C:\ProgramData\Jenkins\jenkins\workspace\SonarQube
16:22:47.189 WARN Property 'sonar.host.url' with value 'http://localhost:9000' is overridden with value 'http://localhost:9000'
16:22:47.224 INFO Scanner configuration file:
C:\ProgramData\Jenkins\jenkins\tools\udson.plugins.sonar.SonarRunnerInstallation\SonarQube\bin\..\conf\sonar-scanner.properties
16:22:47.230 INFO Project root configuration file: NONE
16:22:47.294 INFO SonarScanner CLI 6.2.0.4584
16:22:47.299 INFO Java 21 Oracle Corporation (64-bit)
16:22:47.311 INFO Windows 11 10.0 amd64
16:22:47.392 INFO User cache: C:\windows\system32\config\systemprofile\.sonar\cache
16:22:49.719 INFO JRE provisioning: os[windows], arch[amd64]
16:22:51.182 INFO Communicating with SonarQube Server 10.6.0.92116
16:22:52.539 INFO Starting SonarScanner Engine...
16:22:52.541 INFO Java 17.0.11 Eclipse Adoptium (64-bit)
16:22:55.497 INFO Load global settings
16:22:56.059 INFO Load global settings (done) | time=560ms
16:22:56.072 INFO Server id: 147B411E-AZIOS_TCFdjvGUUATc6Z
16:22:56.105 INFO Loading required plugins
16:22:56.107 INFO Load plugins index
16:22:56.344 INFO Load plugins index (done) | time=237ms
16:22:56.345 INFO Load/download plugins
16:22:56.509 INFO Load/download plugins (done) | time=165ms
16:22:57.306 INFO Process project properties
16:22:57.326 INFO Process project properties (done) | time=20ms
16:22:57.360 INFO Project key: sonarqube
16:22:57.362 INFO Base dir: C:\ProgramData\Jenkins\jenkins\workspace\SonarQube
16:22:57.363 INFO Working dir: C:\ProgramData\Jenkins\jenkins\workspace\SonarQube\scannerwork
16:22:57.384 INFO Load project settings for component key: 'sonarqube'
16:22:57.532 INFO Load quality profiles
```

```
16:23:40.924 INFO Load analysis cache (404) | time=21ms
16:23:41.084 WARN The property 'sonar.login' is deprecated and will be removed in the future. Please use the 'sonar.token' property instead when
passing a token.
16:23:41.154 INFO Preprocessing files...
16:23:42.333 INFO 2 languages detected in 23 preprocessed files
16:23:42.334 INFO 0 files ignored because of scm ignore settings
16:23:42.339 INFO Loading plugins for detected languages
16:23:42.341 INFO Load/download plugins
16:23:42.242 INFO Load/download plugins (done) | time=902ms
16:23:43.487 INFO Executing phase 2 project builders
16:23:43.490 INFO Executing phase 2 project builders (done) | time=2ms
16:23:43.514 INFO Load project repositories
16:23:43.546 INFO Load project repositories (done) | time=29ms
16:23:43.611 INFO Indexing files...
16:23:43.613 INFO Project configuration:
16:23:43.700 INFO 23 files indexed
16:23:43.704 INFO Quality profile for cs: Sonar way
16:23:43.705 INFO Quality profile for json: Sonar way
16:23:43.707 INFO ----- Run sensors on module sonarqube
16:23:43.874 INFO Load metrics repository
16:23:43.937 INFO Load metrics repository (done) | time=63ms
16:23:46.020 INFO Sensor C# Project Type Information [csharp]
16:23:46.037 INFO Sensor C# Project Type Information [csharp] (done) | time=17ms
16:23:46.042 INFO Sensor C# Analysis Log [csharp]
16:23:46.111 INFO Sensor C# Analysis Log [csharp] (done) | time=76ms
16:23:46.112 INFO Sensor C# Properties [csharp]
16:23:46.113 INFO Sensor C# Properties [csharp] (done) | time=0ms
16:23:46.115 INFO Sensor JaCoCo XML Report Importer [jacoco]
16:23:46.118 INFO 'sonar.coverage.jacoco.xmlReportPaths' is not defined. Using default locations: target/site/jacoco/jacoco.xml,target/site/jacoco-
st/jacoco.xml,build/reports/jacoco/task/jacocoTaskReport.xml
```

```
16:23:49.539 INFO Sensor C# File Caching Sensor [csharp]
16:23:49.540 WARN Incremental PR analysis: Could not determine common base path, cache will not be computed. Consider setting 'sonar.projectBaseDir'
property.
16:23:49.541 INFO Sensor C# File Caching Sensor [csharp] (done) | time=2ms
16:23:49.542 INFO Sensor Zero Coverage Sensor
16:23:49.554 INFO Sensor Zero Coverage Sensor (done) | time=15ms
16:23:49.570 INFO SCM Publisher SCM provider for this project is: git
16:23:49.572 INFO SCM Publisher 2 source files to be analyzed
16:23:51.571 INFO SCM Publisher 2/2 source files have been analyzed (done) | time=1997ms
16:23:51.583 INFO CPD Executor Calculating CPD for 0 files
16:23:51.589 INFO CPD Executor CPD calculation finished (done) | time=0ms
16:23:51.614 INFO SCM revision ID 'f2bc042c04c6e72427c380bcaee6d6fee7b49adf'
16:23:51.889 INFO Analysis report generated in 265ms, dir size=197.9 kB
16:23:52.000 INFO Analysis report compressed in 108ms, zip size=20.5 kB
16:23:55.249 INFO Analysis report uploaded in 3245ms
16:23:55.259 INFO ANALYSIS SUCCESSFUL, you can find the results at: http://localhost:9000/dashboard?id=sonarqube
16:23:55.260 INFO Note that you will be able to access the updated dashboard once the server has processed the submitted analysis report
16:23:55.262 INFO More about the report processing at http://localhost:9000/api/ce/task?id=3bd7337b-a267-400d-9686-a4a28c011b88
16:23:55.352 INFO Analysis total time: 58.695 s
16:23:55.356 INFO SonarScanner Engine completed successfully
16:23:55.469 INFO EXECUTION SUCCESS
16:23:55.471 INFO Total time: 1:08.257s
Finished: SUCCESS
```

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

The screenshot shows the SonarQube web interface. At the top, there is a navigation bar with links for Projects, Issues, Rules, Quality Profiles, Quality Gates, Administration, and More. Below this, the breadcrumb trail shows 'sonarqube / main'. The main header area includes 'Overview', 'Issues', 'Security Hotspots', 'Measures', 'Code', and 'Activity' tabs, along with 'Project Settings' and 'Project Information' links. The main content area is titled 'main' and shows a 'Quality Gate' status of 'Passed' with a green checkmark. A warning message states 'The last analysis has warnings. See details'. Below this, there are tabs for 'New Code' and 'Overall Code'. The 'Overall Code' tab is active, displaying three metrics: Security, Reliability, and Maintainability, each with a grade of 'A' and '0 Open issues'. At the bottom, there are sections for 'Accepted issues', 'Coverage', and 'Duplications'.

sonarqube

Projects Issues Rules Quality Profiles Quality Gates Administration More

sonarqube / main

Overview Issues Security Hotspots Measures Code Activity

Project Settings Project Information

main

Version not provided Set as homepage

Quality Gate

Passed

Last analysis 3 minutes ago

The last analysis has warnings. See details

New Code Overall Code

Security

0 Open issues

0 H 0 M 0 L A

Reliability

0 Open issues

0 H 0 M 0 L A

Maintainability

0 Open issues

0 H 0 M 0 L A

Accepted issues Coverage Duplications