

## **Experiment 7**

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

### **Theory:**

#### **What is 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.

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:

### Prerequisites:

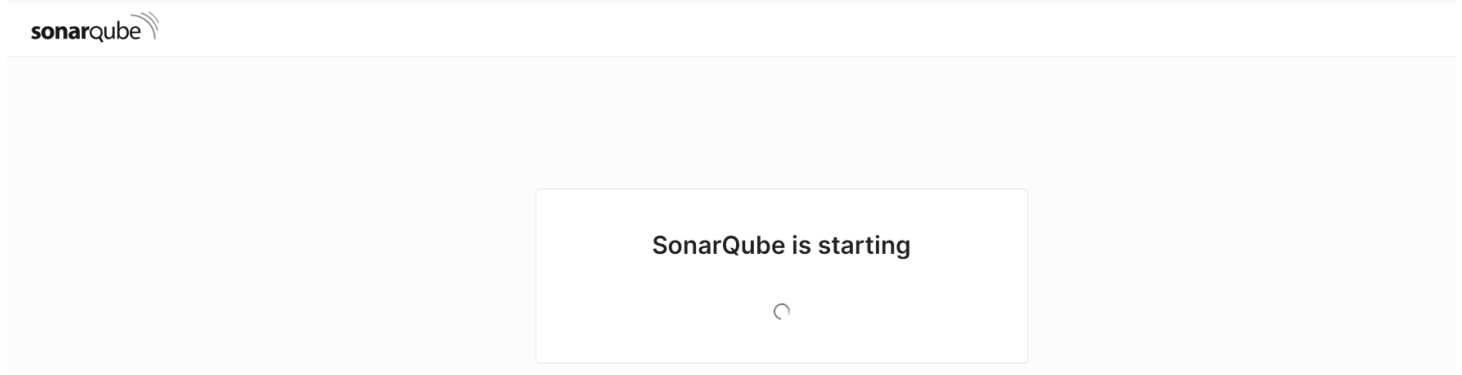
- Jenkins installed
- Docker Installed (for SonarQube)
- SonarQube Docker Image

## 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 -

```
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: sha256:72e9feec71242af83faf65f95a40d5e3bb2822a6c3b2cda8568790f3d31aecde
Status: Downloaded newer image for sonarqube:latest
de76efbeef2054aeb442b86ba54c2916039b8757b388482d9780ffc69f5d8bbe
```

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](#)

Cancel

Next

Setup the project and come back to Jenkins Dashboard.

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

A search bar at the top contains the text 'sonar'. To its right is an 'Install' button with a downward arrow. Below the search bar is a table with columns 'Install', 'Name', and 'Released'. The table contains one entry for 'SonarQube Scanner 2.17.2'. To the left of the entry is an unchecked checkbox. To the right of the entry is the text '7 mo 9 days ago'. Below the entry name are two tags: 'External Site/Tool Integrations' and 'Build Reports'. Below these tags is a description: 'This plugin allows an easy integration of SonarQube, the open source platform for Continuous Inspection of code quality.'

7. Under Jenkins ‘Configure System’, look for SonarQube Servers and enter the details.

Enter the Server Authentication token if needed.

The page has a breadcrumb trail: 'Dashboard > Manage Jenkins > System >'. The main heading is 'SonarQube installations' with a subtitle 'List of SonarQube installations'. Below this is a dashed box containing three sections: 'Name' with a text input field containing 'sonarqube'; 'Server URL' with a text input field containing 'http://localhost:9000' and a note 'Default is http://localhost:9000'; and 'Server authentication token' with a text input field containing '- none -' and a note 'SonarQube authentication token. Mandatory when anonymous access is disabled.'. Below the token field is a '+ Add' button. At the bottom of the dashed box is an 'Advanced' dropdown menu.

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

SonarQube Scanner installations

Add SonarQube Scanner

The page has a heading 'SonarQube Scanner' with a close button. Below it is a 'Name' text input field containing 'sonarqube'. Below that is a checked checkbox labeled 'Install automatically' with a help icon. Below this is a dashed box with a heading 'Install from Maven Central' and a close button. Inside this dashed box is a 'Version' text input field containing 'SonarQube Scanner 6.1.0.4477'. Below the dashed box is an 'Add Installer' dropdown menu.

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

## 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.

10. Choose this GitHub repository in Source Code Management.

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

Source Code Management

☐ None

☒ Git ?

Repositories ?

Repository URL ?

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

Credentials ?

- none -

+ Add ▾

Advanced ▾

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

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

Dashboard > SonarQube > Configuration

### Configure

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

#### Build Steps

##### Execute SonarQube Scanner

SonarQube Installation: ?  
sonarqube

JDK ?  
JDK to be used for this SonarQube analysis  
(Inherit From Job)

Path to project properties ?

Analysis properties ?  
sonar.projectKey=sonarqube  
sonar.login=admin  
sonar.password=admin123  
sonar.sources=C:\\ProgramData\\Jenkins\\jenkins\\workspace\\SonarQube  
sonar.host.url=http://127.0.0.1:9000

Additional arguments ?

JVM Options ?  
-Dsonar.ws.timeout=300

Save Apply

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

	Administer System ?	Administer ?	Execute Analysis ?	Create ?
A Administrator admin	<input 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

## Check the console output.

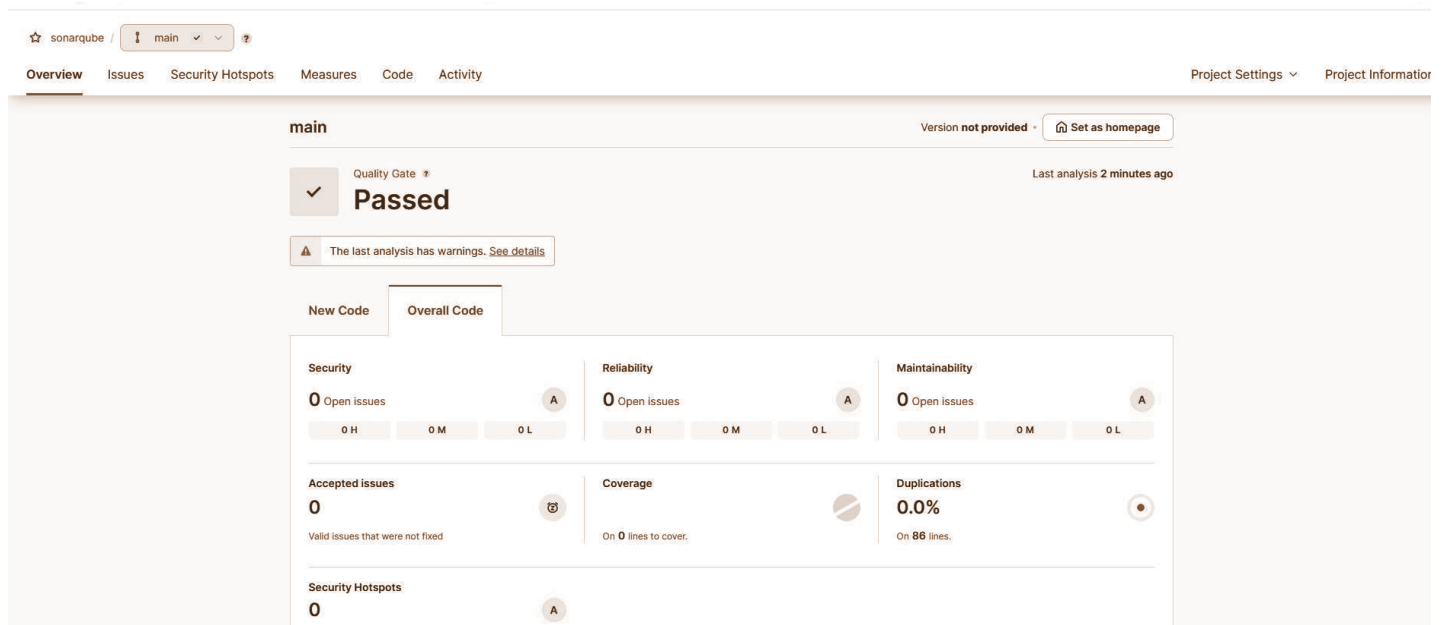
### ✓ Console Output

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

```
Started by user Shiven Bansal
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
> git.exe rev-parse --resolve-git-dir C:\ProgramData\Jenkins\jenkins\workspace\SonarQube\.git # timeout=10
Fetching changes from the remote Git repository
> 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
> git.exe --version # timeout=10
> git --version # 'git version 2.45.2.windows.1'
> git.exe fetch --tags --force --progress -- https://github.com/shazforiot/MSBuild_firstproject.git +refs/heads/*:refs/remotes/origin/* # timeout=10
> git.exe rev-parse 'refs/remotes/origin/master^{commit}' # timeout=10
Checking out Revision f2bc042c04c6e72427c380bcaee6d6fee7b49adf (refs/remotes/origin/master)
> git.exe config core.sparsecheckout # timeout=10
> git.exe checkout -f f2bc042c04c6e72427c380bcaee6d6fee7b49adf # timeout=10
Commit message: "updated"
> git.exe rev-list --no-walk f2bc042c04c6e72427c380bcaee6d6fee7b49adf # timeout=10
[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=admin -Dsonar.host.url=http://127.0.0.1:9000 -Dsonar.sources=C:\ProgramData\Jenkins\jenkins\workspace\SonarQube -Dsonar.password=admin123 -
Dsonar.projectBaseDir=C:\ProgramData\Jenkins\jenkins\workspace\SonarQube
16:16:39.198 WARN Property 'sonar.host.url' with value 'http://localhost:9000' is overridden with value 'http://127.0.0.1:9000'
16:16:39.206 INFO Scanner configuration file: C:\ProgramData\Jenkins\jenkins\tools\udson.plugins.sonar.SonarRunnerInstallation\sonarqube\bin\..\conf\sonar-scanner.properties
16:16:39.206 INFO Project root configuration file: NONE
16:16:39.230 INFO SonarScanner CLI 6.1.0.4477
16:16:39.230 INFO Java 21.0.4 Eclipse Adoptium (64-bit)
16:16:39.230 INFO Windows 11 10.0 amd64
16:16:39.230 INFO SONAR_SCANNER_OPTS=-Dsonar.ws.timeout=300
16:16:39.254 INFO User cache: C:\Windows\system32\config\systemprofile\.sonar\cache

16:16:58.734 INFO Using git CLI to retrieve untracked files
16:16:58.791 INFO Analyzing language associated files and files included via "sonar.text.inclusions" that are tracked by git
16:16:58.856 INFO 14 source files to be analyzed
16:16:59.154 INFO 14/14 source files have been analyzed
16:16:59.154 INFO Sensor TextAndSecretsSensor [text] (done) | time=1306ms
16:16:59.163 INFO ----- Run sensors on project
16:16:59.373 INFO Sensor C# [csharp]
16:16:59.373 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
16:16:59.373 INFO Sensor C# [csharp] (done) | time=0ms
16:16:59.373 INFO Sensor Analysis Warnings import [csharp]
16:16:59.379 INFO Sensor Analysis Warnings import [csharp] (done) | time=0ms
16:16:59.379 INFO Sensor C# File Caching Sensor [csharp]
16:16:59.379 WARN Incremental PR analysis: Could not determine common base path, cache will not be computed. Consider setting 'sonar.projectBaseDir' property.
16:16:59.379 INFO Sensor C# File Caching Sensor [csharp] (done) | time=6ms
16:16:59.379 INFO Sensor Zero Coverage Sensor
16:16:59.389 INFO Sensor Zero Coverage Sensor (done) | time=10ms
16:16:59.389 INFO SCM Publisher SCM provider for this project is: git
16:16:59.389 INFO SCM Publisher 4 source files to be analyzed
16:16:59.838 INFO SCM Publisher 4/4 source files have been analyzed (done) | time=449ms
16:16:59.846 INFO CPD Executor Calculating CPD for 0 files
16:16:59.846 INFO CPD Executor CPD calculation finished (done) | time=0ms
16:16:59.854 INFO SCM revision ID 'f2bc042c04c6e72427c380bcaee6d6fee7b49adf'
16:17:00.121 INFO Analysis report generated in 120ms, dir size=201.1 kB
16:17:00.195 INFO Analysis report compressed in 57ms, zip size=22.4 kB
16:17:00.393 INFO Analysis report uploaded in 195ms
16:17:00.394 INFO ANALYSIS SUCCESSFUL, you can find the results at: http://127.0.0.1:9000/dashboard?id=sonarqube
16:17:00.395 INFO Note that you will be able to access the updated dashboard once the server has processed the submitted analysis report
16:17:00.395 INFO More about the report processing at http://127.0.0.1:9000/api/ce/task?id=acd819f5-9e70-42ab-bff7-3cc893e2cae4
16:17:00.405 INFO Analysis total time: 18.743 s
16:17:00.408 INFO SonarScanner Engine completed successfully
16:17:00.494 INFO EXECUTION SUCCESS
16:17:00.494 INFO Total time: 21.288s
Finished: SUCCESS
```

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



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

### **Conclusion:**

In this experiment, I learned how to integrate Jenkins with SonarQube for performing Static Application Security Testing (SAST). I set up SonarQube in a Docker container and configured Jenkins to use the SonarQube scanner. By creating a manual project in SonarQube and configuring the necessary authentication and tools in Jenkins, I established a seamless connection between Jenkins and SonarQube for static code analysis.

I tested the integration with a sample GitHub repository, successfully running a build and analyzing the project's code quality through SonarQube. This hands-on experience enhanced my understanding of the SAST process, Jenkins automation, and SonarQube's capabilities for identifying potential code vulnerabilities.