

Name:Komal Singh

Roll No:60

Div:D15B

Advance Devops-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:

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

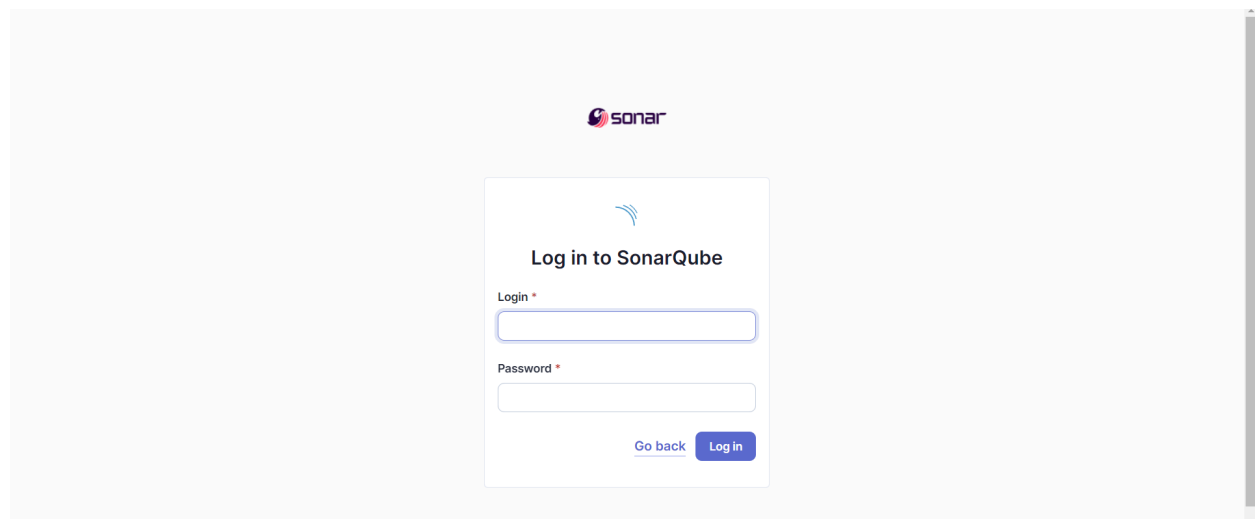
1. Open up Jenkins Dashboard on localhost, port 8080 or whichever port it is at for you.
2. Run SonarQube in a Docker container using this command -

```
PS C:\Users\ACER> 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
ee55c3fed1ebd978d24da39243f7be8f4e2b95a26c2b40b9bddb2c31a3e44ae2
PS C:\Users\ACER> _
```

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](#) 

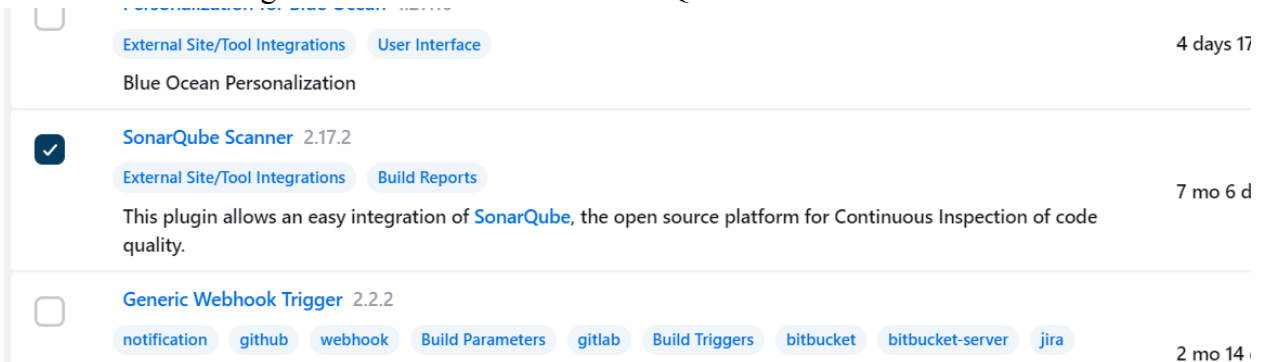
Cancel

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.



The screenshot shows the Jenkins 'Manage Jenkins' interface. At the top, there are tabs for 'External Site/Tool Integrations' and 'User Interface'. Below these, the 'Blue Ocean Personalization' section is visible. The main list of plugins shows 'SonarQube Scanner 2.17.2' with a checked checkbox, indicating it is installed. Below the plugin name are tabs for 'External Site/Tool Integrations' and 'Build Reports'. A description states: 'This plugin allows an easy integration of SonarQube, the open source platform for Continuous Inspection of code quality.' The date '7 mo 6 d' is shown to the right. Below this, the 'Generic Webhook Trigger 2.2.2' plugin is listed with an unchecked checkbox and various integration tabs like 'notification', 'github', 'webhook', etc. The date '2 mo 14 d' is shown to the right.

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

Enter the Server Authentication token if needed.

☒ Environment variables

SonarQube installations

List of SonarQube installations

Name

sonarqube

Server URL

Default is http://localhost:9000

http://localhost:9000

Server authentication token

8. Search for SonarQube Scanner under Global Tool Configuration. Choose the latest configuration and choose Install automatically.
9. After the configuration, create a New Item in Jenkins, choose a freestyle project.

New Item

Enter an item name

SonarKomal

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.

10. 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 ? ✕

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

Please enter Git repository.

Credentials ?

- none - ▼

+ Add ▼

Advanced ▼

11. Under Build-> Execute SonarQube Scanner, enter these Analysis properties. Mention the SonarQube Project Key, Login, Password, Source path and Host URL.
12. Go to http://localhost:9000/<user_name>/permissions and allow Execute Permissions to the Admin user.

Users Create User

Create and administer individual users.

Filter by All users ?

Name	SCM Accounts	Last connection	Last SonarLint connection ?	Groups	Tokens	Actions
A Administrator admin		< 1 hour ago	Never	2 ⋮	1 ⋮	⋮

1 of 1 shown

13.

14. Run The Build.



Status



Changes



Workspace



Build Now



Configure



Delete Project



SonarQube



Rename

15.

Check the console output.



Console Output

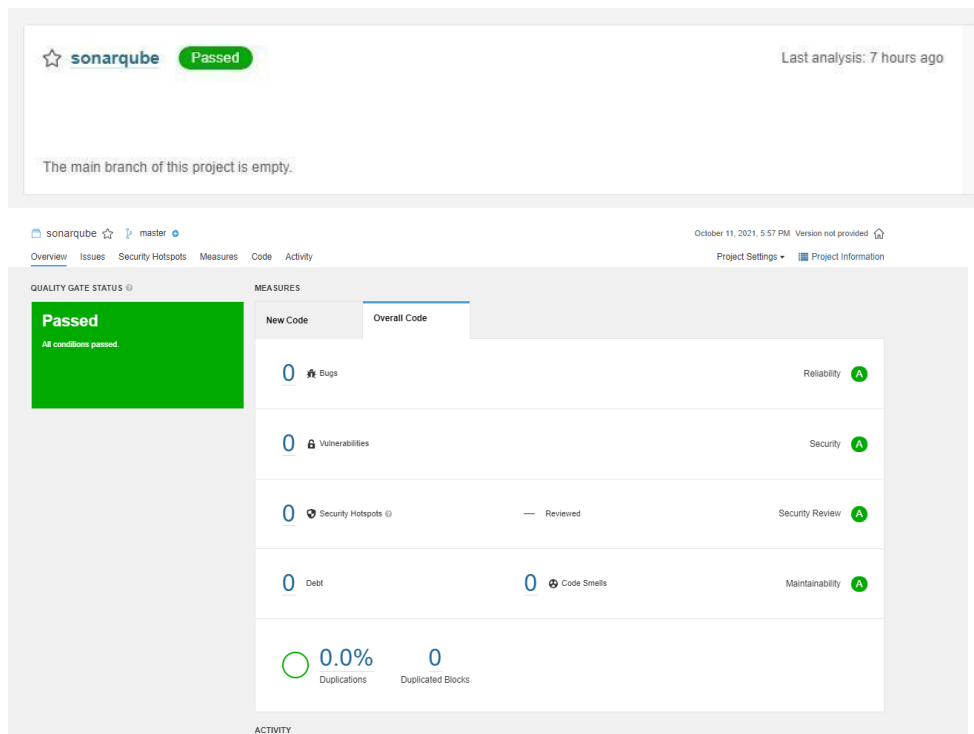
Download

Copy

View as

```
Started by user admin
Running as SYSTEM
Building on the built-in node in workspace C:\ProgramData\Jenkins\.jenkins\workspace\SonarKomal
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\SonarKomal # timeout=10
Fetching upstream changes from https://github.com/shazforiot/MSBuild_firstproject.git
> git.exe --version # timeout=10
> git --version # 'git version 2.43.0.windows.1'
> 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.
Finished: SUCCESS
```


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