

## **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:

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

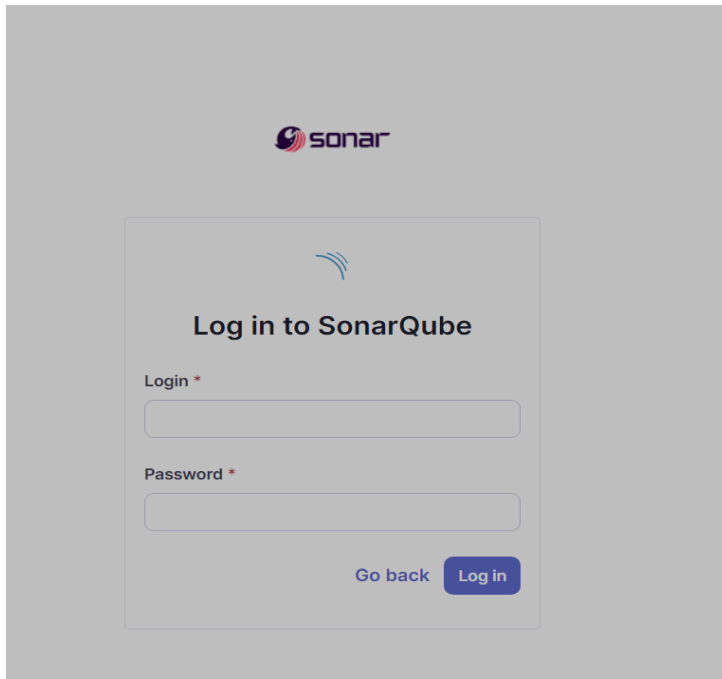
1. Open up Jenkins Dashboard on localhost, port 8080 or whichever port it is at for you.
2. Run SonarQube in a Docker container using this command -

**docker run -d --name sonarqube -e  
SONAR\_ES\_BOOTSTRAP\_CHECKS\_DISABLE=true -p 9000:9000**

## sonarqube:latest

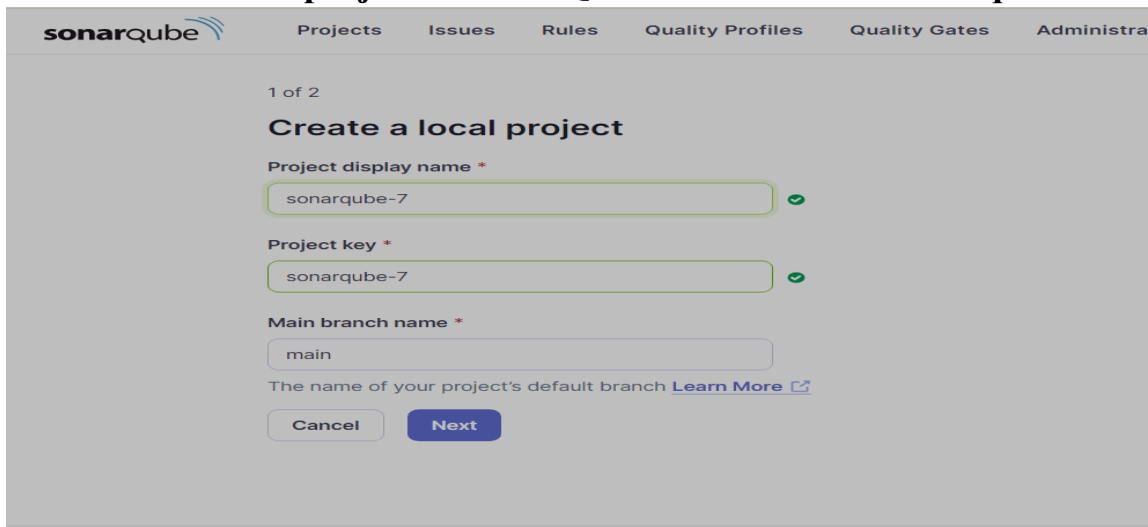
```
C:\Users\tarun>docker run -d --name sonarqube -e SONAR_ES_BOOTSTRAP_CHECKS_DISABLE=true -p 9000:9000 sonarqube:latest 53d4af904582e15d29003562ae68704002c13d93fb8c2256b6d198f5a0724743  
C:\Users\tarun>
```

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



**5. 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.**

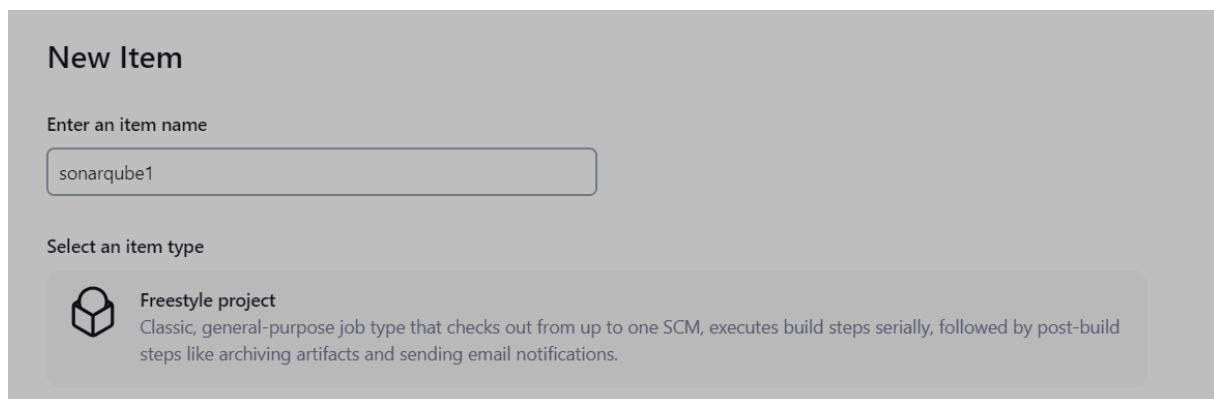


The screenshot shows the 'SonarQube installations' configuration page in Jenkins. It features a list of installations, with one entry visible: 'sonarqube'. The configuration fields for this entry are: 'Name' (sonarqube), 'Server URL' (http://localhost:9000), and 'Server authentication token' (token). There is an '+ Add' button and an 'Advanced' dropdown menu.

**7. Search for SonarQube Scanner under Global Tool**

**Configuration. Choose the latest configuration and choose Install automatically.**

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



The screenshot shows the 'New Item' page in Jenkins. It has a title 'New Item' and a subtitle 'Enter an item name'. The input field contains 'sonarqube1'. Below this, it says 'Select an item type'. There is a list of item types, with 'Freestyle project' selected. The description for 'Freestyle project' is: '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.'

## 9. Choose this GitHub repository in Source Code Management.


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

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



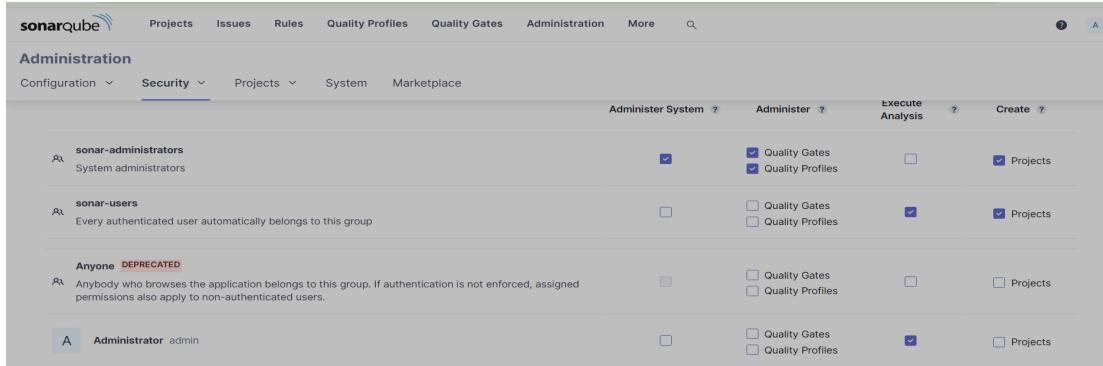
The screenshot shows a 'Git' configuration window with a 'Repositories' tab. A dashed box highlights the 'Repository URL' field, which contains the URL 'https://github.com/shazforiot/MSBuild\_firstproject.git'. Below it, the 'Credentials' dropdown is set to '- none -'. There is an '+ Add' button and an 'Advanced' dropdown. At the bottom of the dashed box is an 'Add Repository' button. Below the dashed box is a 'Branches to build' field.

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



The screenshot shows the 'Execute SonarQube Scanner' configuration window. It has several sections: 'JDK' with a dropdown set to '(Inherit From Job)'; 'Path to project properties' with an empty text field; 'Analysis properties' with a text area containing the following properties:  
sonar.projectKey=sonarqube-7  
sonar.login=admin  
sonar.password=T@runs207000  
sonar.sources=.  
sonar.host.url=http://localhost:9000  
'Additional arguments' with an empty text field; and 'JVM Options' with an empty text field.

## 11. 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 ?
<b>sonar-administrators</b> 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
<b>sonar-users</b> 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
<b>Anyone</b> <small>DEPRECATED</small> 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
<b>Administrator</b> admin	<input type="checkbox"/>	<input type="checkbox"/> Quality Gates <input type="checkbox"/> Quality Profiles	<input checked="" type="checkbox"/>	<input type="checkbox"/> Projects

## 12. Run The Build.

```
Started by user Tarunkumar sharma
Running as SYSTEM
Building on the built-in node in workspace C:\ProgramData\Jenkins\.jenkins\workspace\sonarqube1
The recommended git tool is: NONE
No credentials specified
> C:\Program Files\Git\cmd\git.exe rev-parse --resolve-git-dir C:\ProgramData\Jenkins\.jenkins\workspace\sonarqube1\.git # timeout=10
Fetching changes from the remote Git repository
> C:\Program Files\Git\cmd\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\cmd\git.exe --version # timeout=10
> git --version # 'git version 2.46.0.windows.1'
> C:\Program Files\Git\cmd\git.exe fetch --tags --force --progress -- https://github.com/shazforiot/MSBuild_firstproject.git
+refs/heads/*:refs/remotes/origin/* # timeout=10
```

```
20:48:00.668 INFO Sensor C# File Caching Sensor [csharp] (done) | time=1ms
20:48:00.672 INFO Sensor Zero Coverage Sensor
20:48:00.748 INFO Sensor Zero Coverage Sensor (done) | time=82ms
20:48:00.753 INFO SCM Publisher SCM provider for this project is: git
20:48:00.756 INFO SCM Publisher 4 source files to be analyzed
20:48:02.732 INFO SCM Publisher 4/4 source files have been analyzed (done) | time=1973ms
20:48:02.736 INFO CPD Executor Calculating CPD for 0 files
20:48:02.739 INFO CPD Executor CPD calculation finished (done) | time=0ms
20:48:02.754 INFO SCM revision ID 'f2bc042c04c6e72427c380bcae6d6fee7b49adf'
20:48:04.160 INFO Analysis report generated in 314ms, dir size=201.0 kB
20:48:04.285 INFO Analysis report compressed in 86ms, zip size=22.2 kB
20:48:06.673 INFO Analysis report uploaded in 2374ms
20:48:06.688 INFO ANALYSIS SUCCESSFUL, you can find the results at: http://localhost:9000/dashboard?id=swayam-test
20:48:06.689 INFO Note that you will be able to access the updated dashboard once the server has processed the submitted analysis report
20:48:06.690 INFO More about the report processing at http://localhost:9000/api/ce/task?id=941ac1cc-19ac-45a3-a656-887de2d6805f
20:48:06.744 INFO Analysis total time: 2:22.044 s
20:48:06.766 INFO SonarScanner Engine completed successfully
20:48:06.951 INFO EXECUTION SUCCESS
20:48:07.092 INFO Total time: 3:31.032s
Finished: SUCCESS
```

**Conclusion:**

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