

**Name: Abhishek Chaurasiya**  
**Class: D15B / 9**  
**ADVDEVOPS PRACTICAL 7**

Installation Links :Git : <https://git-scm.com/download/win>  
Jenkins : <https://www.jenkins.io/download/thank-you-downloading-windows-installer-stable>  
Docker : <https://www.docker.com/products/docker-desktop/>

**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

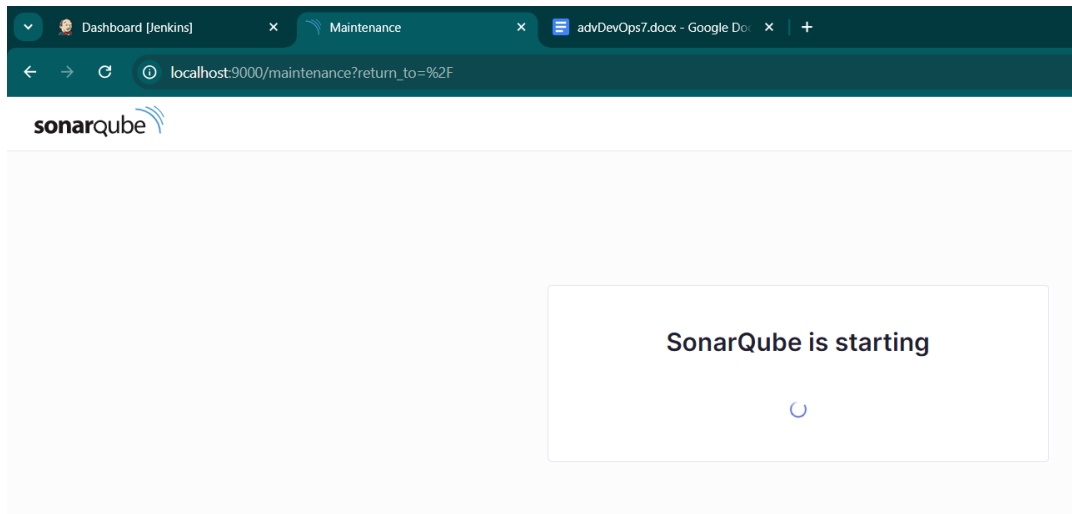
1. Open up Jenkins Dashboard on localhost, port 8080 or whichever port it is at for you.
2. Run SonarQube in a Docker container using this command -

```
Windows PowerShell
~ docker run -d --name sonarqube -e SONAR_ES_BOOTSTRAP_CHECKS_DISABLE=true -p 9000:9000 sonarqube:latest
```

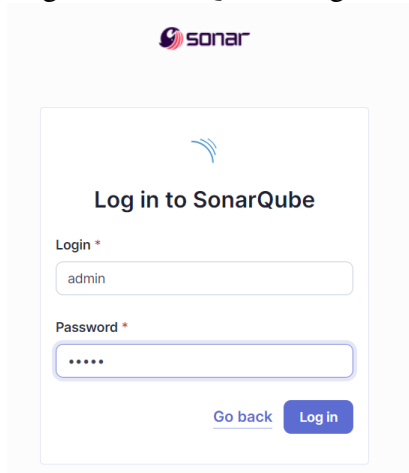
Warning: run below command only once

```
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**  
Setup the project and come back to Jenkins Dashboard.

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

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

Q sonar

Install Name ↓

☒ **SonarQube Scanner** 2.17.2

[External Site/Tool Integrations](#) [Build Reports](#)

This plugin allows an easy integration of [SonarQube](#), the open source platform for Continuous Inspection of code quality.

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

Dashboard > Manage Jenkins > System >

SonarQube installations

List of SonarQube installations

Name

sonarqube

Server URL

Default is http://localhost:9000

http://localhost:9000

Server authentication token

Enter the Server Authentication token if needed.

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

☰ SonarQube Scanner

Name

SonarQube Scanner

☒ Install automatically ?








☰ Install from Maven Central

Version

SonarQube Scanner 6.2.0.4584

Add Installer ▾

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

 **Jenkins**       Abhishek Chaurasiya ▾  log out


Dashboard > All > New Item

## New Item

Enter an item name

SonarCube

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.

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 integration.

Source Code Management

☐ None

☒ Git ?

Repositories ?

Repository URL ?

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

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

Dashboard > SonarQube > Configuration

---

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
sonar.login=admin
sonar.password=admin
sonar.sources=.
sonar.host.url=http://localhost:9000
```

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

Projects Issues Rules Quality Profiles Quality Gates Administration More

ion

Security Projects System Marketplace

### Permissions

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

Users Groups

	Administer System ?	Administer ?	Execute Analysis ?	Create ?
administrators	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/> Quality Gates <input checked="" type="checkbox"/> Quality Profiles	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/> Projects
administrators				

12. Run The Build.

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

Check the console output.

Abhishek Chaurasiya
log out

Dashboard > SonarCube > #1

Status

**Console Output**

Download

Copy

View as plain text

```

Started by user Abhishek Chaurasiya
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 2.43.0.windows.1'
> git.exe fetch --tags --force --progress -- https://github.com/shazforiot/MSBuild_firstproject.git +refs/heads/*:re
> git.exe rev-parse "refs/remotes/origin/master^{commit}" # timeout=10
Checking out Revision f2bc042c04c6e72427c380bcaee6d6fee7b49adf (refs/remotes/origin/master)

16:21:33.894 INFO   SCM Publisher 4/4 source files have been analyzed (done) | time=803ms
16:21:33.898 INFO   CPD Executor Calculating CPD for 0 files
16:21:33.898 INFO   CPD Executor CPD calculation finished (done) | time=0ms
16:21:33.914 INFO   SCM revision ID 'f2bc042c04c6e72427c380bcaee6d6fee7b49adf'
16:21:34.450 INFO   Analysis report generated in 218ms, dir size=199.9 kB
16:21:34.569 INFO   Analysis report compressed in 99ms, zip size=22.3 kB
16:21:37.534 INFO   Analysis report uploaded in 2963ms
16:21:37.537 INFO   ANALYSIS SUCCESSFUL, you can find the results at: http://localhost:9000/dashboard?id=sonarqube
16:21:37.538 INFO   Note that you will be able to access the updated dashboard once the server has processed the submitted analysis report
16:21:37.539 INFO   More about the report processing at http://localhost:9000/api/ce/task?id=56bd7dcb-696a-421b-b9f8-52995e991328
16:21:37.577 INFO   Analysis total time: 46.557 s
16:21:37.581 INFO   SonarScanner Engine completed successfully
16:21:37.675 INFO   EXECUTION SUCCESS
16:21:37.678 INFO   Total time: 1:14.844s
Finished: SUCCESS

```



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

The screenshot displays the SonarQube project page for a project named 'sonarqube' (PUBLIC). The status is 'Passed' with a green checkmark. The last analysis was 3 minutes ago. A message states: 'The main branch of this project is empty.'

Below this, the 'main' branch is selected, and the version is 'not provided'. A button 'Set as homepage' is visible. The 'Quality Gate' is 'Passed' with a green checkmark, and the last analysis was 5 minutes ago. A warning message states: 'The last analysis has warnings. [See details](#)'.

The 'Overall Code' tab is selected, showing the following metrics:

Metric	Value	Grade
Security	0 Open issues	A
Reliability	0 Open issues	A
Maintainability	0 Open issues	A
Accepted issues	0	
Coverage	On 0 lines to cover.	
Duplications	0.0% On 86 lines.	
Security Hotspots	0	A

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.

## Additional resources

Sonarqube installation on aws Ubuntu

<https://www.coachdevops.com/2020/04/install-sonarqube-on-ubuntu-how-to.html>

<https://awstip.com/installing-sonarqube-on-aws-ec2-instance-and-integrating-it-with-aws-codepipeline-abec99416ba4>

`docker-compose up -d && docker ps`