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
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:

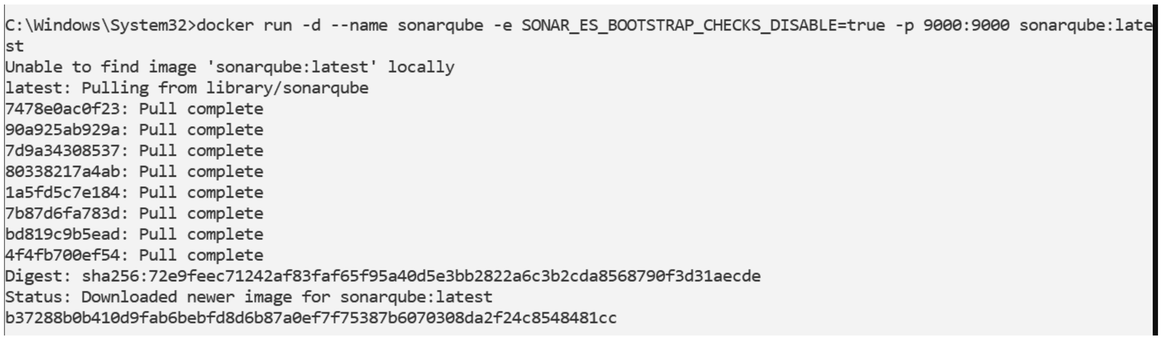
- Jenkins installed
- Docker Installed (for SonarQube)

(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 -

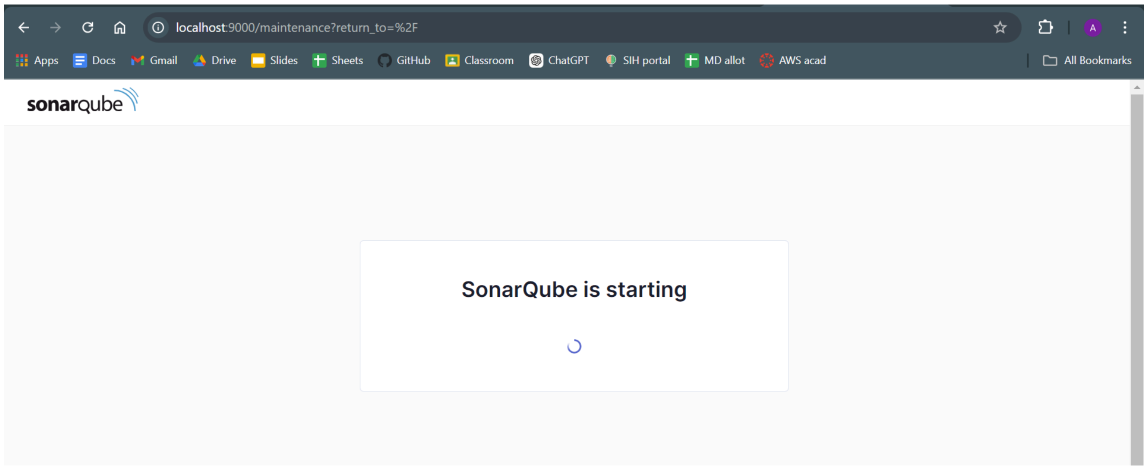


```
C:\Windows\System32>docker run -d --name sonarqube -e SONAR_ES_BOOTSTRAP_CHECKS_DISABLE=true -p 9000:9000 sonarqube:late
st
Unable to find image 'sonarqube:latest' locally
latest: Pulling from library/sonarqube
7478e0ac0f23: Pull complete
90a925ab929a: Pull complete
7d9a34308537: Pull complete
80338217a4ab: Pull complete
1a5fd5c7e184: Pull complete
7b87d6fa783d: Pull complete
bd819c9b5ead: Pull complete
4f4fb700ef54: Pull complete
Digest: sha256:72e9feec71242af83faf65f95a40d5e3bb2822a6c3b2cda8568790f3d31aecde
Status: Downloaded newer image for sonarqube:latest
b37288b0b410d9fab6bebfd8d6b87a0ef7f75387b6070308da2f24c8548481cc
```

Warning: run below command only once

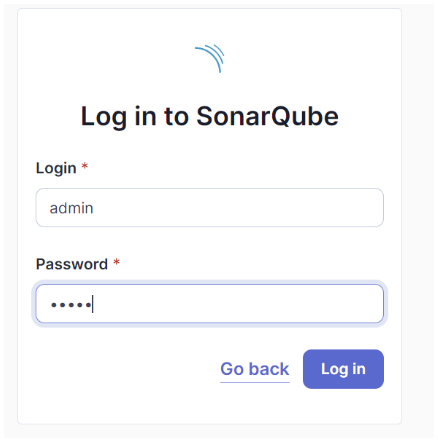
docker run -d --name sonarqube -e SONAR_ES_BOOTSTRAP_CHECKS_DISABLE=true -p 9000:9000 sonarqube:latest

3. Once the container is up and running, you can check the status of SonarQube at localhost port 9000.



SonarQube is starting

4. Login to SonarQube using username *admin* and password *admin*.



Log in to SonarQube

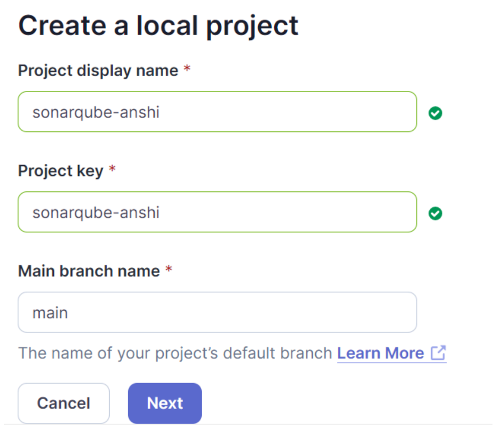
Login *

admin

Password *

•••••

5. Create a manual project in SonarQube with the name **sonarqube**



Create a local project

Project display name *

sonarqube-anshi

Project key *

sonarqube-anshi

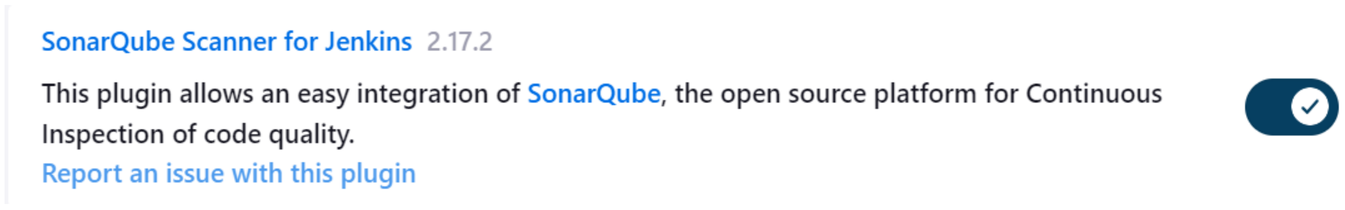
Main branch name *

main

The name of your project's default branch Learn More

Setup the project and come back to Jenkins Dashboard.

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



SonarQube Scanner for Jenkins 2.17.2

This plugin allows an easy integration of SonarQube, the open source platform for Continuous Inspection of code quality.

Report an issue with this plugin

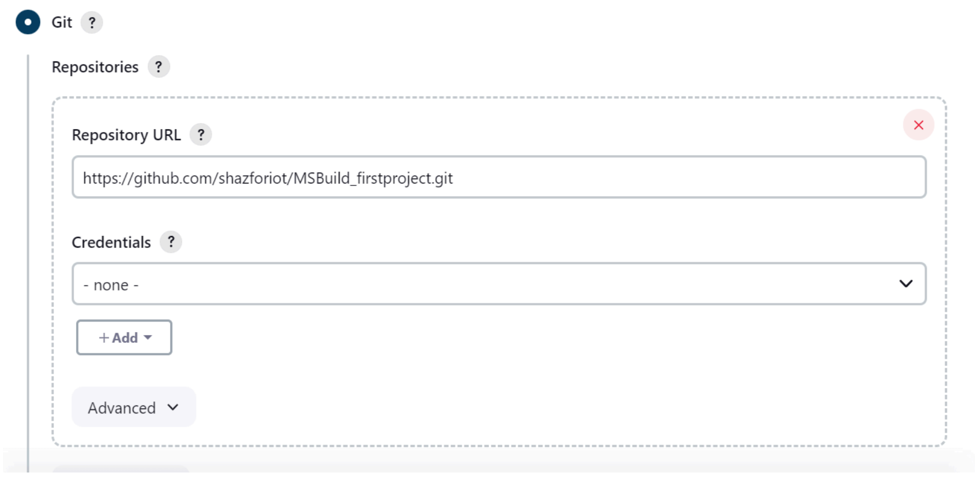
6. Under Jenkins 'Configure System', look for SonarQube Servers and enter the details.

Enter the Server Authentication token if needed.

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.

9. 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

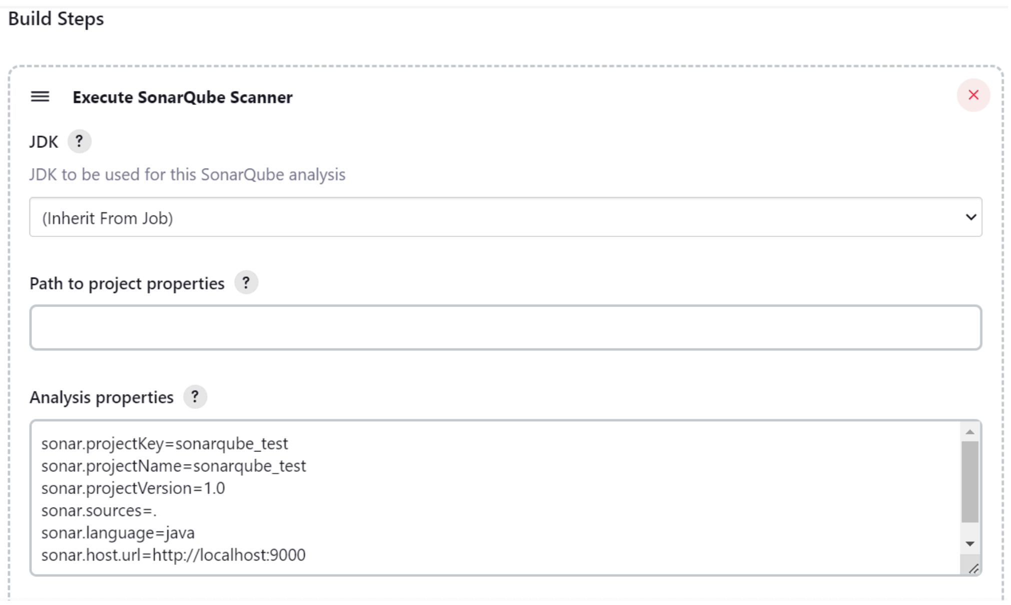


https://github.com/shazforiot/MSBuild_firstproject.git

- none -

the integration.

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



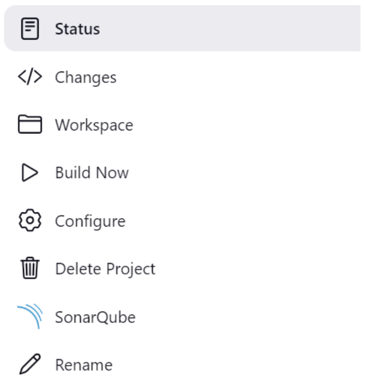
Build Steps
Execute SonarQube Scanner
JDK
JDK to be used for this SonarQube analysis

JDK: (Inherit From Job)

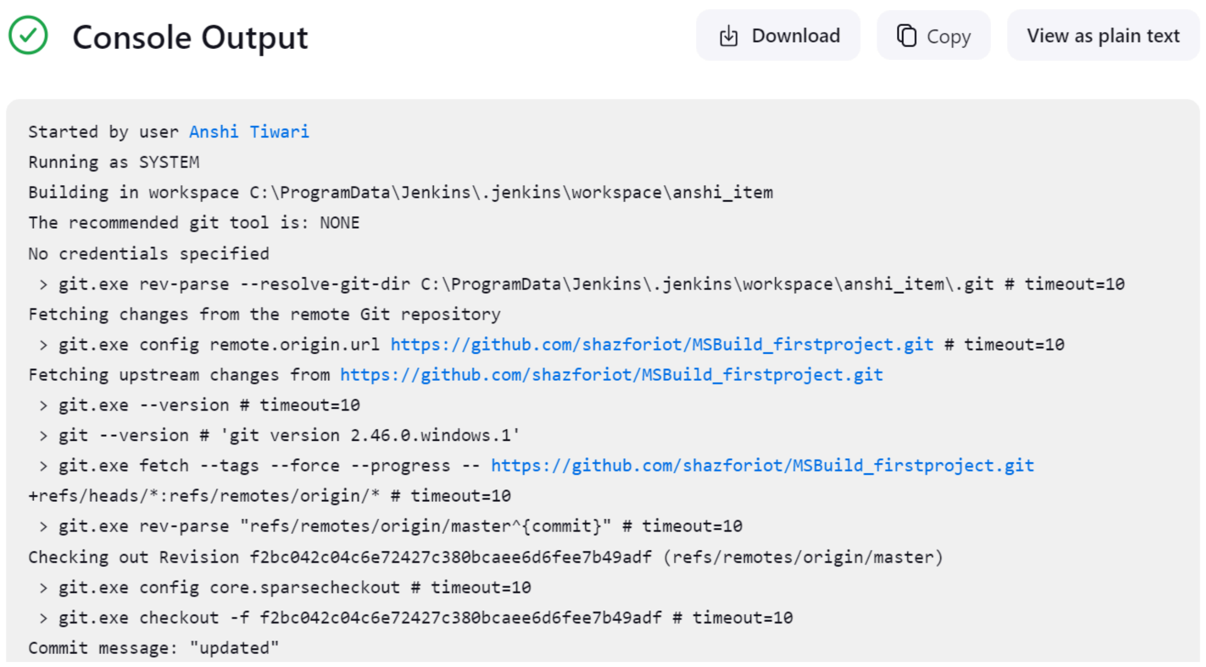
Path to project properties
Analysis properties

```
sonar.projectKey=sonarqube_test
sonar.projectName=sonarqube_test
sonar.projectVersion=1.0
sonar.sources=.
sonar.language=java
sonar.host.url=http://localhost:9000
```

12. Run The Build.

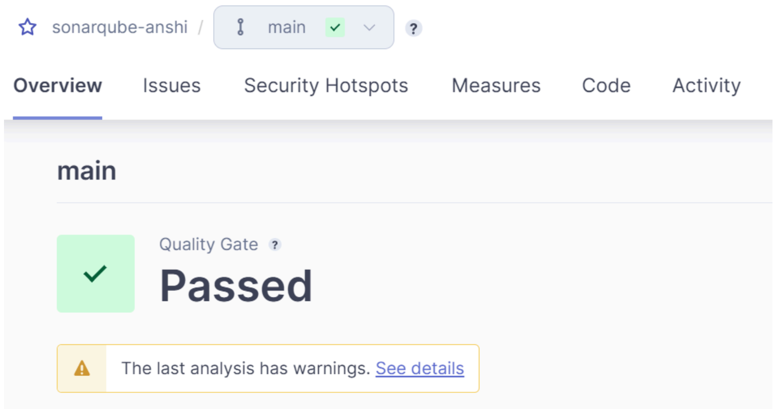


Check the console output.



```
Started by user Anshi Tiwari
Running as SYSTEM
Building in workspace C:\ProgramData\Jenkins\.jenkins\workspace\anshi_item
The recommended git tool is: NONE
No credentials specified
 > git.exe rev-parse --resolve-git-dir C:\ProgramData\Jenkins\.jenkins\workspace\anshi_item\.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 2.46.0.windows.1'
 > git.exe fetch --tags --force --progress -- https://github.com/shazforiot/MSBuild_firstproject.git
+refs/heads/*:refs/remotes/origin/* # timeout=10
 > git.exe 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"
```

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



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

Conclusion of the Experiment

This experiment focused on integrating Jenkins with SonarQube for Static Application Security Testing (SAST). Throughout the process, several challenges emerged:

1. **Token Authentication Issues**: Encountering a 401 Unauthorized error emphasized the importance of correctly managing authentication tokens and user permissions within SonarQube.

2. **Project Configuration**: Initial confusion regarding the configuration parameters in Jenkins highlighted the need for attention to detail. Ensuring the sonar.login property was set correctly was crucial for successful integration.
3. **Maven Misunderstanding**: Misunderstanding the role of Maven led to unnecessary command-line attempts. Recognizing that Jenkins handles the build process streamlined the workflow.
4. **Error Navigation**: Reading console logs proved vital in troubleshooting and identifying issues, enhancing my problem-solving skills.