Experiment No 7

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 -

Warning: run below command only once
docker run -d --name sonarqube -e SONAR_ES_BOOTSTRAP_CHECKS_DISABLE=true -p
9000:9000 sonarqube:latest

```
PS C:\Users\akank> docker run -d --name sonarqube -e SONAR_ES_BOOTSTRAP_CHECKS_DISABLE=true -p 9000:9000 sonarqube:lates
t
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
781f61a017c237dcaf6a61148bd90ff1e62e3704f1f80d9ed77c290ac3b2bd32
PS C:\Users\akank>
```

3. Once the container is up and running, you can check the status of SonarQube at localhost port 9000.
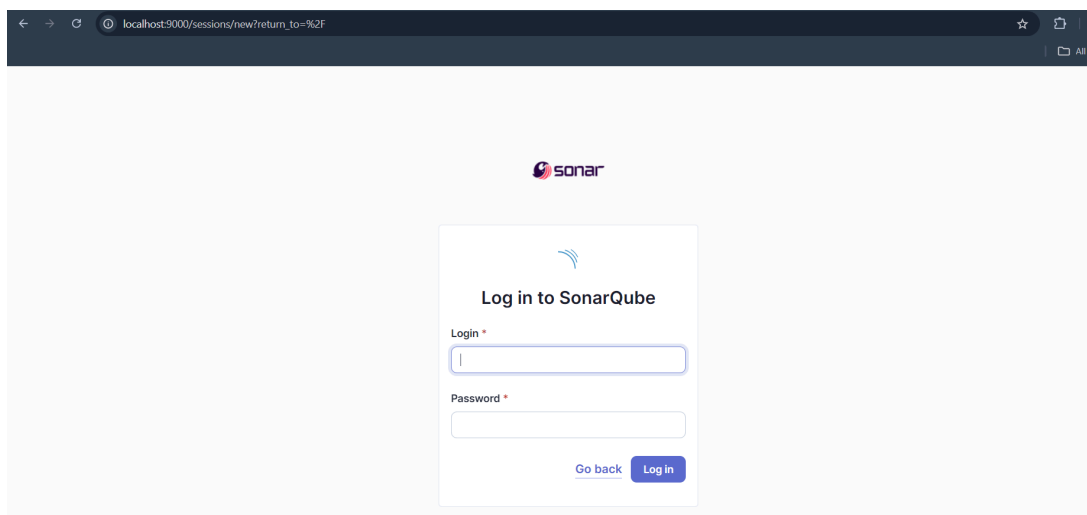
4. Login to SonarQube using username *admin* and password *admin*.
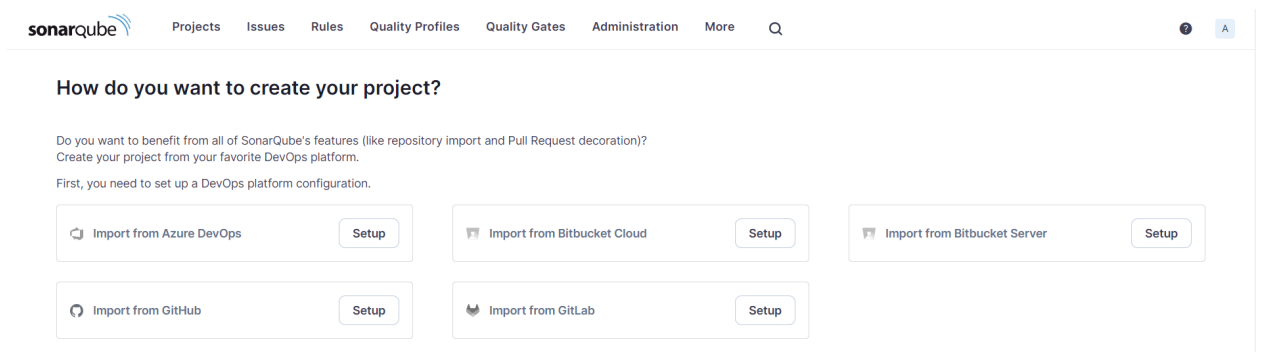
5. Click on create local project which will be somewhere in the left of the screen. Create a manual project in SonarQube with the name **sonarqube**

1 of 2

Create a local project

Project display name *

sonarqube-akanksha ✓

Project key *

sonarqube-akanksha ✓

Main branch name *

main

The name of your project's default branch **Learn More** ⬈

Cancel   **Next**

After clicking on next select use the global setting and click create project.

Choose the baseline for new code for this project

● **Use the global setting**

Previous version

Any code that has changed since the previous version is considered new code.

Recommended for projects following regular versions or releases.

○ **Define a specific setting for this project**

○ **Previous version**

Any code that has changed since the previous version is considered new code.

Recommended for projects following regular versions or releases.

○ **Number of days**

Any code that has changed in the last x days is considered new code. If no action is taken on a new issu

Recommended for projects following continuous delivery.

○ **Reference branch**

Choose a branch as the baseline for the new code.

Recommended for projects using feature branches.

Back   **Create project**

Setup the project and come back to Jenkins Dashboard.

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

it.

6. Go to manage jenkins and click on 'System', look for SonarQube Servers and enter details.

   Enter the Server Authentication token if needed. (not necessary I skipped it).

   Do the following task and click save.

7. Go to manage jenkins and click tools then search for SonarQube Scanner under Global Tool Configuration. Choose the latest configuration and choose Install automatically and click save.

SonarScanner for MSBuild installations

Add SonarScanner for MSBuild

☰   **SonarScanner for MSBuild**

Name

sonarqubeScannerExp7

☑ Install automatically   ?

☰   **Install from GitHub**

Version
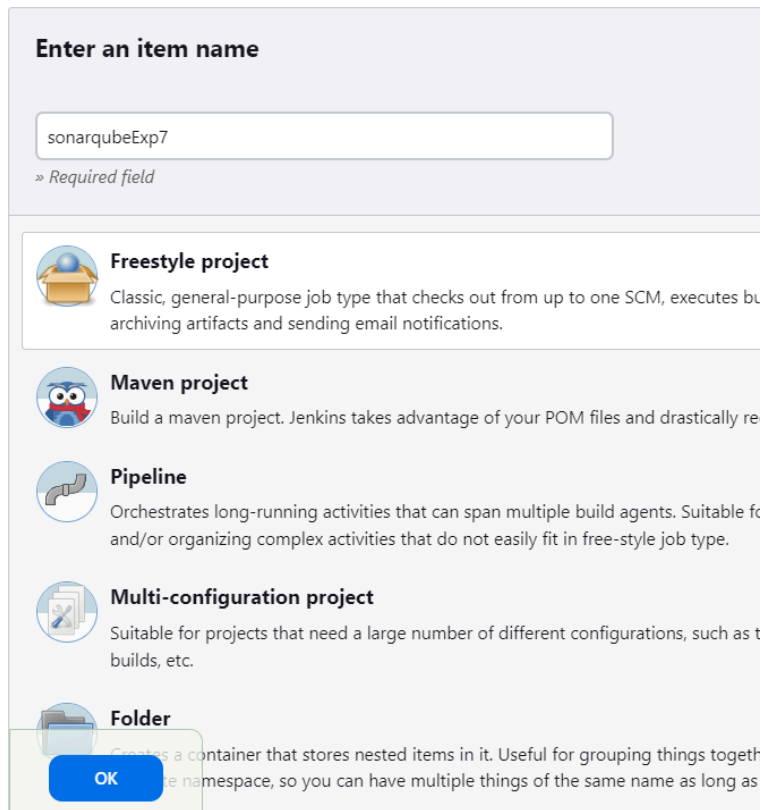
SonarScanner for .NET Framework 8.0.3.99785

Add Installer ⌄

Save   Apply

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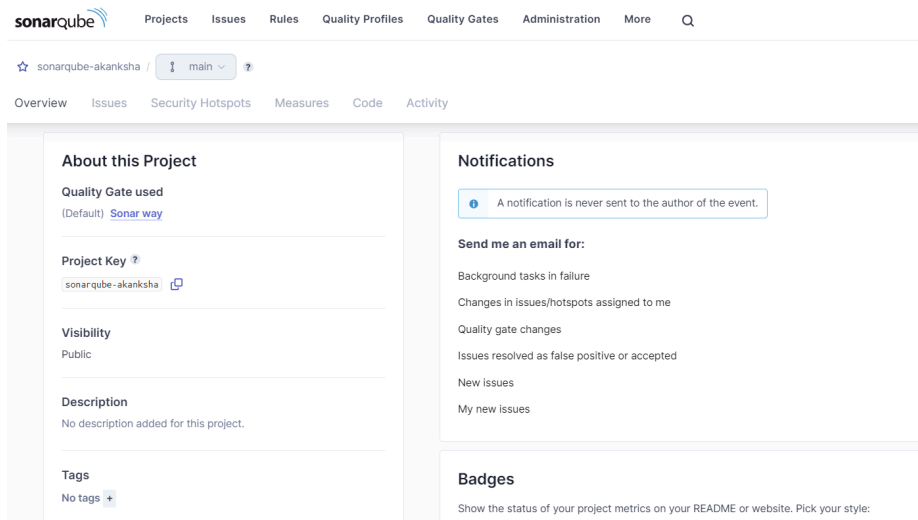It is a sample hello-world project with no vulnerabilities and issues just to test.
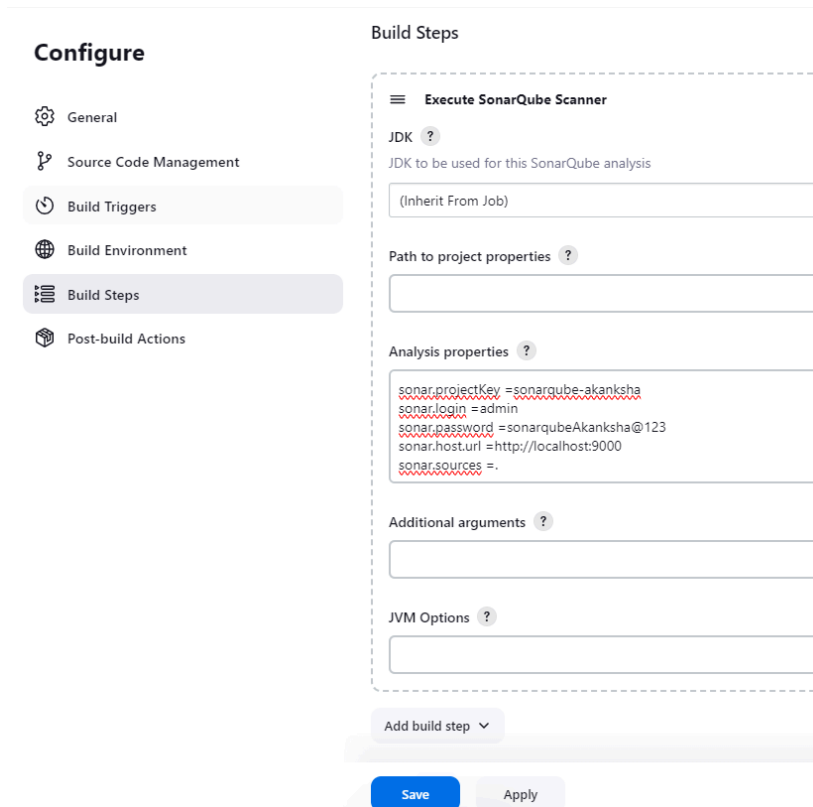
the integration.

10. Under myProject -> configuration -> Build-> Execute SonarQube Scanner -> enter these Analysis properties(to get this go back to sonarqube and click on your project name after that click project information at the right of the screen).

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.

Put the following code in analysis properties and put your credentials in it and then apply and save.
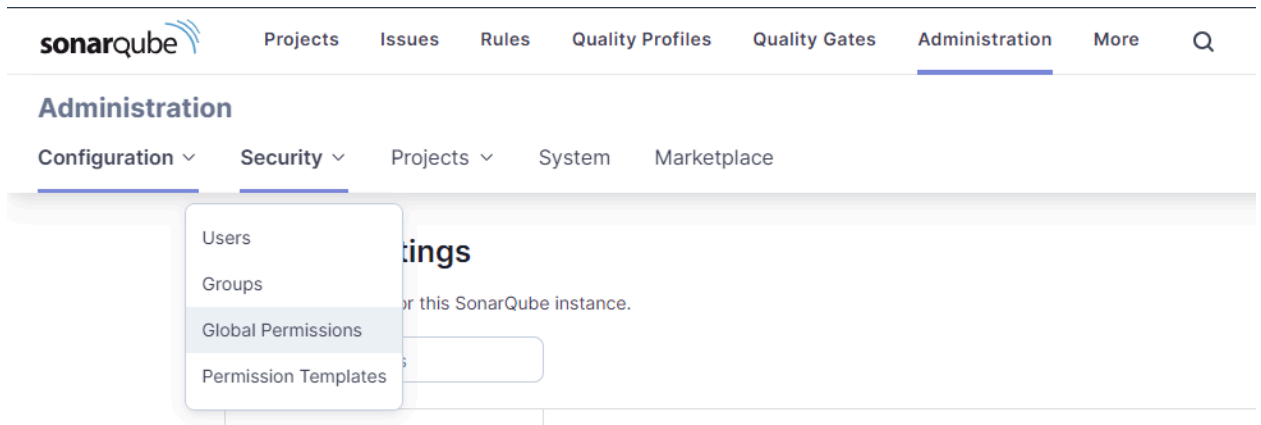
Go to sonarqube and click on administration and click on security and select global permissions from drop down menu

Make sure to check all the checkboxes that I have selected.
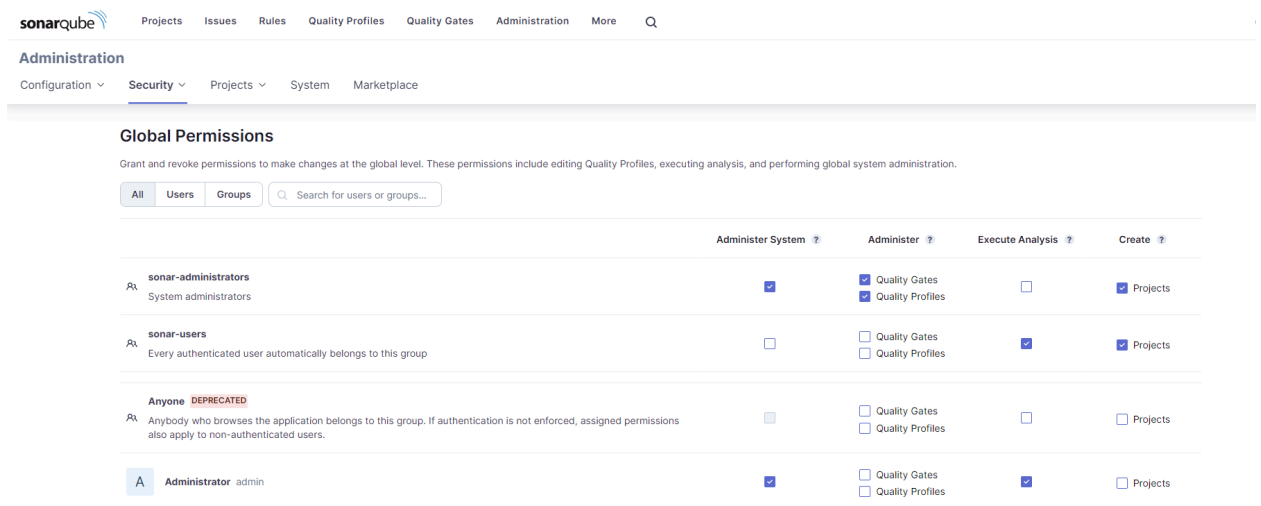
12. Go to jenkins and click Build Now.

📄 Status	**Exp7**
</> Changes	
📁 Workspace	
▷ Build Now	SonarQ̲
⚙ Configure	**Permalinks**
🗑 Delete Project	
〜 SonarQube	
✏ Rename	

☀ Build History   trend ⌄

🔍 Filter...   /

✓ #1   〜
  Sep 26, 2024, 4:58 PM

📶 Atom feed for all   📶 Atom feed for failures

Check the console output.

✓ **Console Output** 🗗

```
Started by user akanksha
Running as SYSTEM
Building on the built-in node in workspace C:\ProgramData\Jenkins\.jenkins\workspace\Exp7
The recommended git tool is: NONE
No credentials specified
 > C:\Program Files\Git\bin\git.exe rev-parse --resolve-git-dir C:\ProgramData\Jenkins\.jenkins\workspace\Exp7\.git # timeout=10
Fetching changes from the remote Git repository
 > C:\Program Files\Git\bin\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\bin\git.exe --version # timeout=10
 > git --version # 'git version 2.43.0.windows.1'
 > C:\Program Files\Git\bin\git.exe fetch --tags --force --progress -- https://github.com/shazforiot/MSBuild_firstproject.git +refs/heads/*:refs/remotes/origin/* #
timeout=10
 > C:\Program Files\Git\bin\git.exe rev-parse "refs/remotes/origin/master^{commit}" # timeout=10
Checking out Revision f2bc042c04c6e72427c380bcaee6d6fee7b49adf (refs/remotes/origin/master)
 > C:\Program Files\Git\bin\git.exe config core.sparsecheckout # timeout=10
 > C:\Program Files\Git\bin\git.exe checkout -f f2bc042c04c6e72427c380bcaee6d6fee7b49adf # timeout=10
Commit message: "updated"
First time build. Skipping changelog.
[Exp7] $ C:\ProgramData\Jenkins\.jenkins\tools\hudson.plugins.sonar.SonarRunnerInstallation\sonarqube-scannerExp7\bin\sonar-scanner.bat -
Dsonar.host.url=http://localhost:9000 -Dsonar.projectKey=sonarqube-akanksha -Dsonar.login=admin -Dsonar.host.url=http://localhost:9000 -Dsonar.sources=. -
Dsonar.password=admin123 -Dsonar.projectBaseDir=C:\ProgramData\Jenkins\.jenkins\workspace\Exp7
16:59:00.866 WARN  Property 'sonar.host.url' with value 'http://localhost:9000' is overridden with value 'http://localhost:9000'
16:59:00.874 INFO  Scanner configuration file: C:\ProgramData\Jenkins\.jenkins\tools\hudson.plugins.sonar.SonarRunnerInstallation\sonarqube-scannerExp7\bin\..\conf\son
```

```
5.x or higher, see https://redirect.sonarsource.com/doc/install-configure-scanner-msbuild.html
16:59:30.166 INFO  Sensor C# [csharp] (done) | time=1ms
16:59:30.166 INFO  Sensor Analysis Warnings import [csharp]
16:59:30.168 INFO  Sensor Analysis Warnings import [csharp] (done) | time=1ms
16:59:30.168 INFO  Sensor C# File Caching Sensor [csharp]
16:59:30.169 WARN  Incremental PR analysis: Could not determine common base path, cache will not be computed. Consider setting 'sonar.projectBaseDir' property.
16:59:30.170 INFO  Sensor C# File Caching Sensor [csharp] (done) | time=1ms
16:59:30.170 INFO  Sensor Zero Coverage Sensor
16:59:30.183 INFO  Sensor Zero Coverage Sensor (done) | time=15ms
16:59:30.188 INFO  SCM Publisher SCM provider for this project is: git
16:59:30.191 INFO  SCM Publisher 4 source files to be analyzed
16:59:30.693 INFO  SCM Publisher 4/4 source files have been analyzed (done) | time=502ms
16:59:30.697 INFO  CPD Executor Calculating CPD for 0 files
16:59:30.697 INFO  CPD Executor CPD calculation finished (done) | time=0ms
16:59:30.705 INFO  SCM revision ID 'f2bc042c04c6e72427c380bcaee6d6fee7b49adf'
16:59:31.003 INFO  Analysis report generated in 118ms, dir size=201.0 kB
16:59:31.071 INFO  Analysis report compressed in 53ms, zip size=22.4 kB
16:59:31.272 INFO  Analysis report uploaded in 197ms
16:59:31.274 INFO  ANALYSIS SUCCESSFUL, you can find the results at: http://localhost:9000/dashboard?id=sonarqube-akanksha
16:59:31.275 INFO  Note that you will be able to access the updated dashboard once the server has processed the submitted analysis report
16:59:31.275 INFO  More about the report processing at http://localhost:9000/api/ce/task?id=903576df-83d0-4126-b8a2-6abcd1f0b97e
16:59:31.291 INFO  Analysis total time: 20.478 s
16:59:31.292 INFO  SonarScanner Engine completed successfully
16:59:31.330 INFO  EXECUTION SUCCESS
16:59:31.332 INFO  Total time: 30.459s
Finished: SUCCESS
```

13. Once the build is complete, check the project in SonarQube.
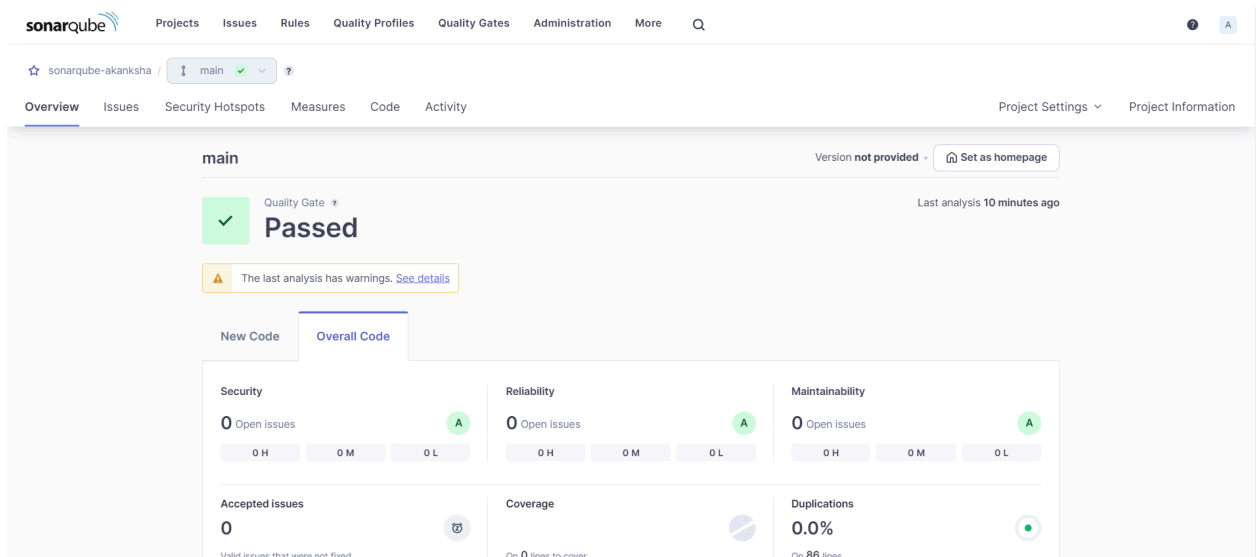
In this way, we have integrated Jenkins with SonarQube for SAST.

Conclusion

To conclude, I have understood the importance of SAST and have successfully integrated Jenkins with SonarQube for Static Analysis and Code Testing. In the first testing didn't go right there was the error of JAVA_HOME exists but not being able to point it. So I solved it by setting the path of java bin in manage jenkins and then global settings again I created a freestyle and project and was successfully able to perform the experiment.

Name: Akanksha Shinde   Class: D15C   Roll No: 53