Name: Shrusti Chintawar
Class: D15B / 10
ADVDEVOPS PRAC 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

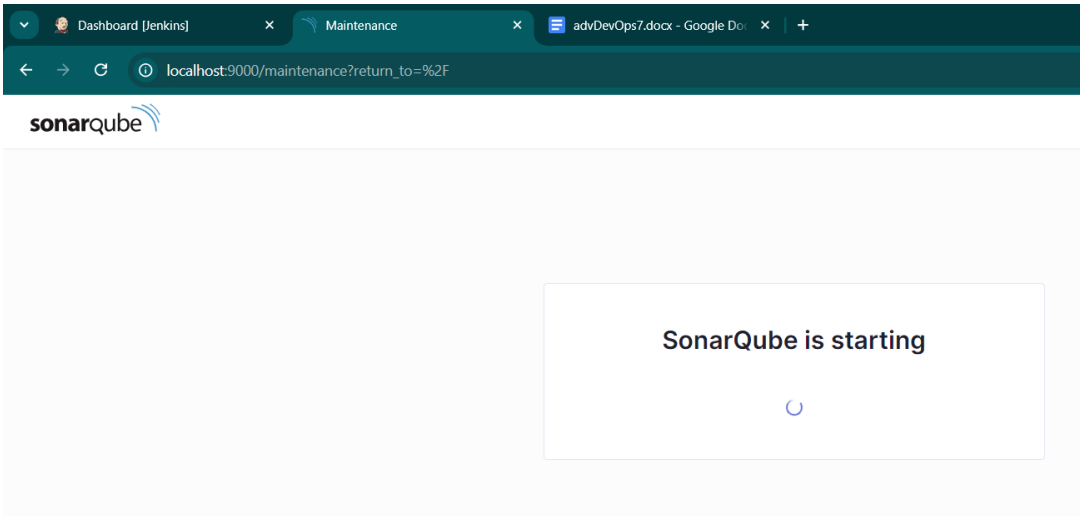
1. Open up Jenkins Dashboard on localhost, port 8080 or whichever port it is at for you.

2. Run SonarQube in a Docker container using this command -

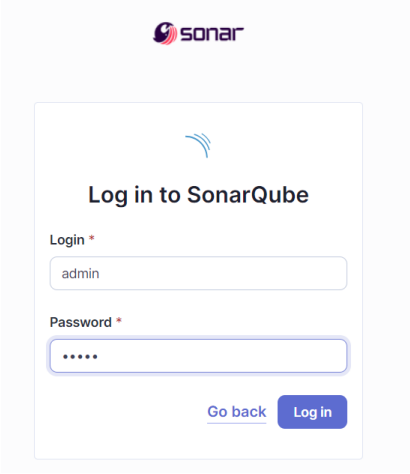
Warning: run below command only once

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

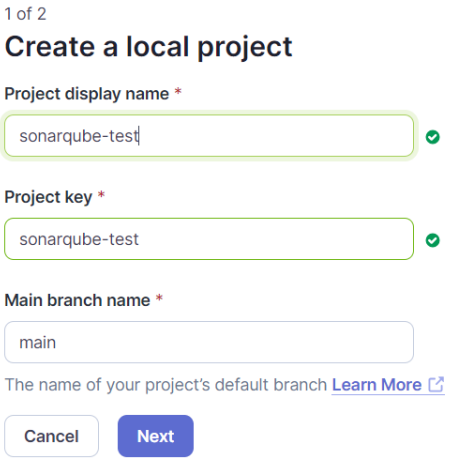
3. Once the container is up and running, you can check the status of SonarQube at localhost port 9000.



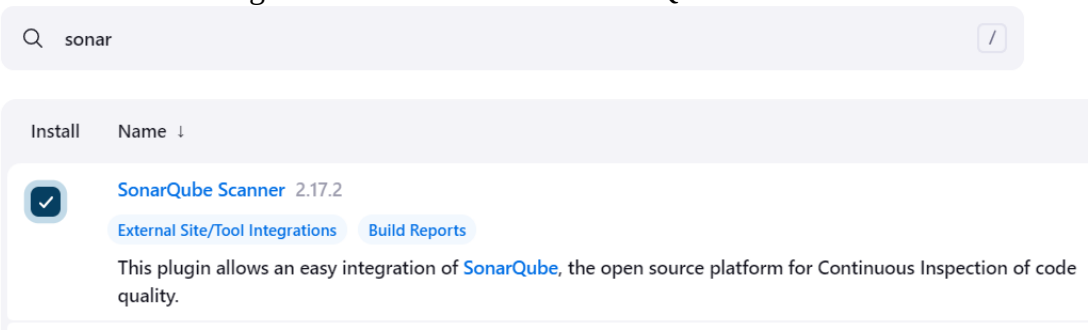
4. Login to SonarQube using username *admin* and password *admin*.



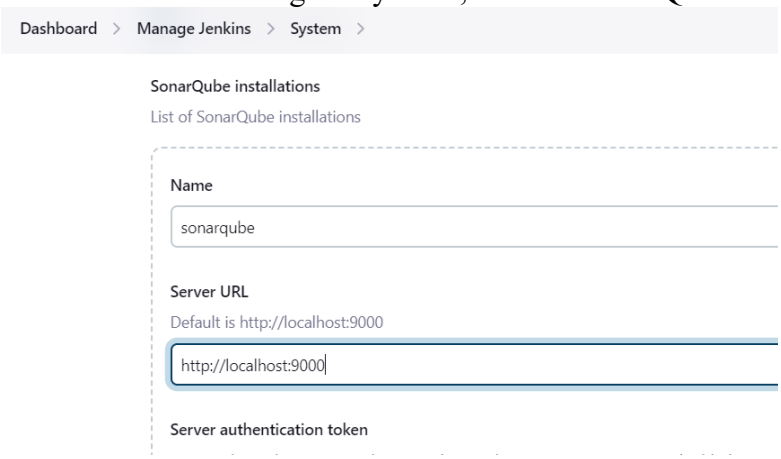
5. Create a manual project in SonarQube with the name **sonarqube**
   Setup the project and come back to Jenkins Dashboard.



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

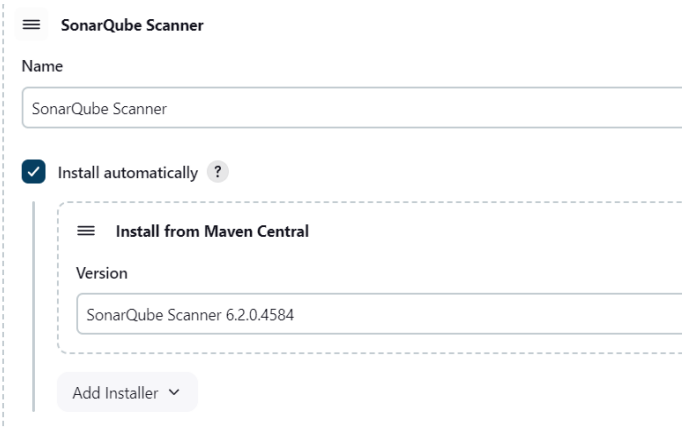


6. Under Jenkins 'Configure System', look for SonarQube Servers and enter the details.

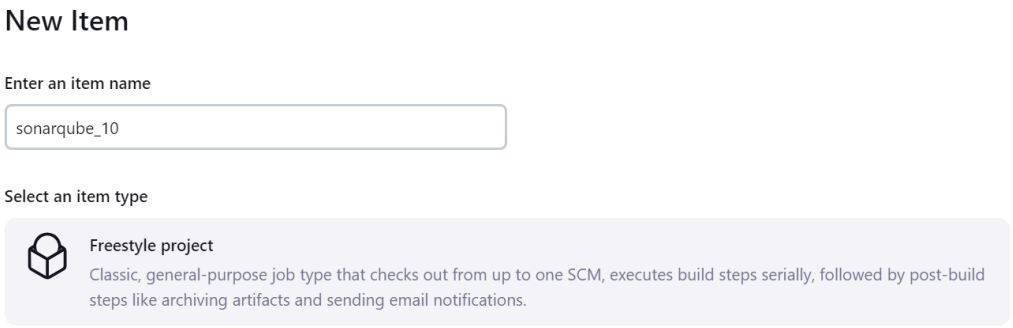


Enter the Server Authentication token if needed.

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

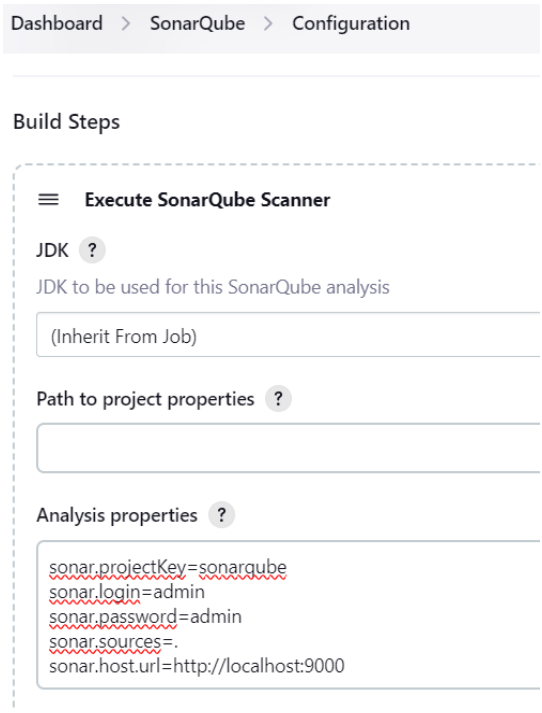


8. After the configuration, create a New Item in Jenkins, choose a freestyle project.

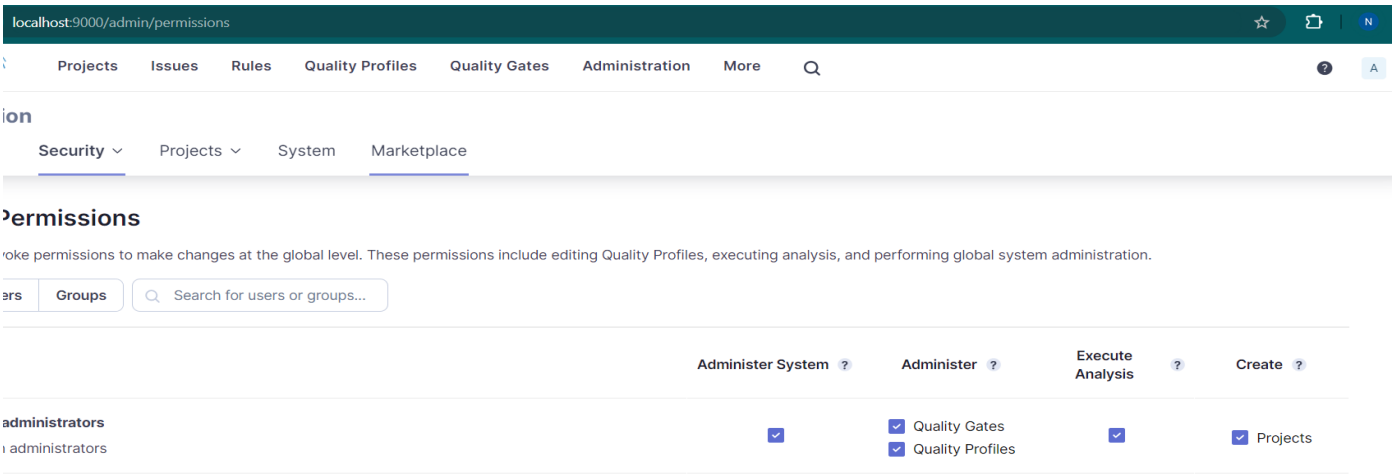


9. Choose this GitHub repository in Source Code Management.
   https://github.com/shazforiot/MSBuild_firstproject.git
   It is a sample hello-world project with no vulnerabilities and issues, just to test the integration.

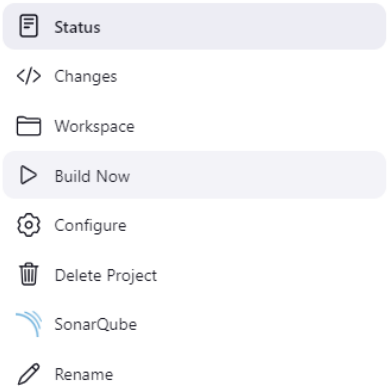
10. Under Build-> Execute SonarQube Scanner, enter these Analysis properties. Mention the SonarQube Project Key, Login, Password, Source path and Host URL.



11. Go to http://localhost:9000/<user_name>/permissions and allow Execute Permissions to the Admin user.



12. Run The Build.



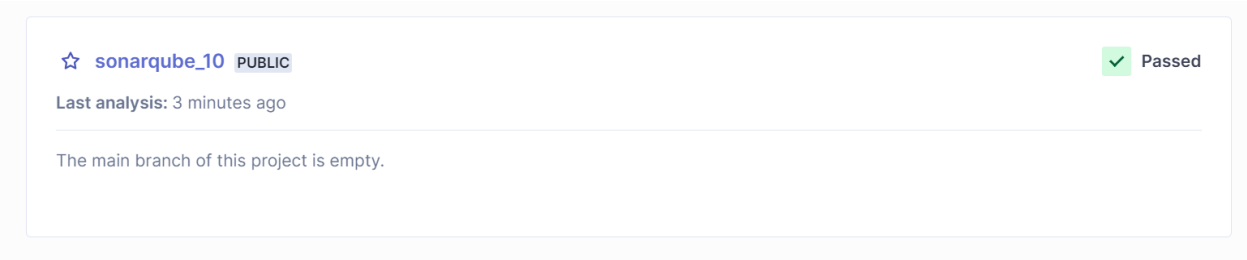
Check the console output.

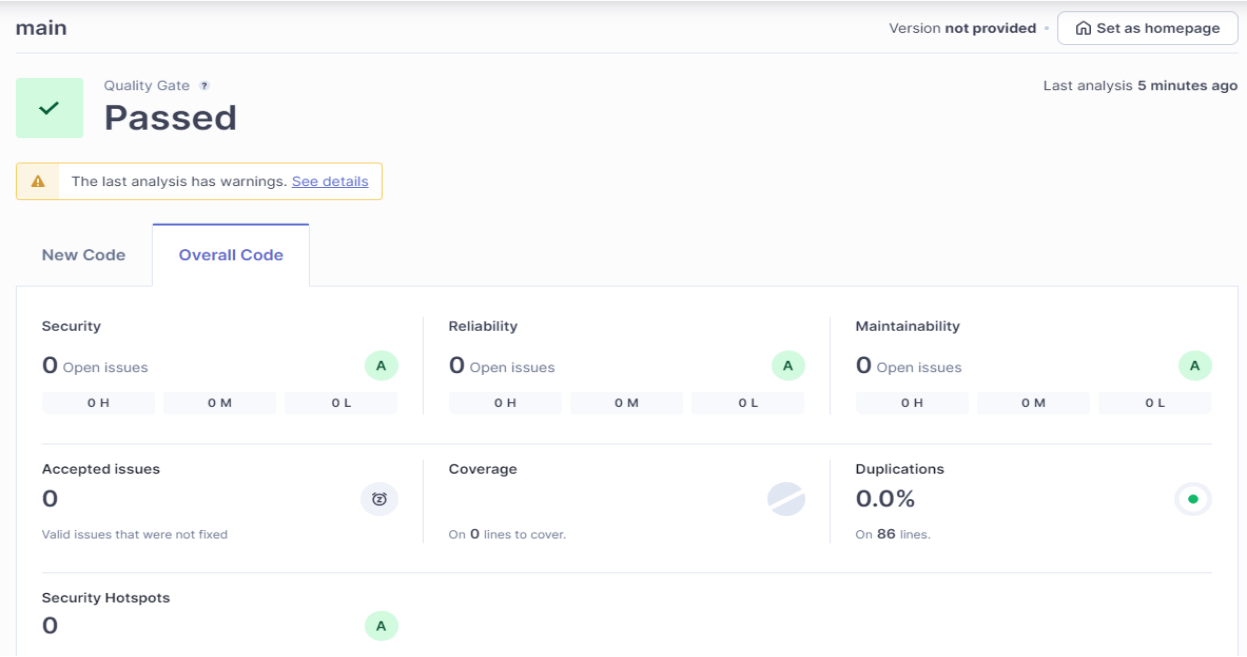
```
Started by user Shrusti Chintawar
Running as SYSTEM
Building remotely on devopnode in workspace c:\Jenkins\workspace\sonarqube_10
Unpacking https://repo1.maven.org/maven2/org/sonarsource/scanner/cli/sonar-scanner-cli/6.2.1.4610/sonar-scanner-cli-6.2.1.4610.zip to
c:\Jenkins\tools\hudson.plugins.sonar.SonarRunnerInstallation\sonarqube on devopnode
[sonarqube_10] $ c:\Jenkins\tools\hudson.plugins.sonar.SonarRunnerInstallation\sonarqube\bin\sonar-scanner.bat -
Dsonar.host.url=http://localhost:9000 -Dsonar.projectKey=sonarqube_10 -Dsonar.login=admin -Dsonar.host.url=http://localhost:9000 -
Dsonar.password=shrustibest -Dsonar.projectBaseDir=c:\Jenkins\workspace\sonarqube_10
22:29:55.528 WARN  Property 'sonar.host.url' with value 'http://localhost:9000' is overridden with value 'http://localhost:9000'
22:29:55.541 INFO  Scanner configuration file: c:\jenkins\tools\hudson.plugins.sonar.SonarRunnerInstallation\sonarqube\bin\..\conf\sonar-
scanner.properties
22:29:55.555 INFO  Project root configuration file: NONE
22:29:55.577 INFO  SonarScanner CLI 6.2.1.4610
22:29:55.580 INFO  Java 17.0.12 Oracle Corporation (64-bit)
22:29:55.582 INFO  Windows 11 10.0 amd64
22:29:55.605 INFO  User cache: C:\Users\Shhrustiiii\.sonar\cache
22:29:56.611 INFO  JRE provisioning: os[windows], arch[amd64]
22:30:00.907 INFO  Communicating with SonarQube Server 10.6.0.92116
22:30:01.622 INFO  Starting SonarScanner Engine...
22:30:01.623 INFO  Java 17.0.11 Eclipse Adoptium (64-bit)
22:30:02.824 INFO  Load global settings
22:30:03.120 INFO  Load global settings (done) | time=295ms
22:30:03.125 INFO  Server id: 147B411E-AZJn5ik7pWlJzYmlAUSX
22:30:03.138 INFO  Loading required plugins
22:30:03.139 INFO  Load plugins index
22:30:03.316 INFO  Load plugins index (done) | time=175ms
22:30:03.320 INFO  Load/download plugins
22:30:05.931 INFO  Load/download plugins (done) | time=2617ms
22:30:06.327 INFO  Process project properties
22:30:06.328 INFO  Process project properties (done) | time=1ms
```

```
22:30:26.933 INFO  The property 'sonar.tests' is not set. To improve the analysis accuracy, we categorize a file as a test file if any of the
following is true:
  * The filename starts with "test"
  * The filename contains "test." or "tests."
  * Any directory in the file path is named: "doc", "docs", "test" or "tests"
  * Any directory in the file path has a name ending in "test" or "tests"

22:30:27.124 INFO  Using git CLI to retrieve untracked files
22:30:27.271 WARN  Analyzing only language associated files, make sure to run the analysis inside a git repository to make use of inclusions
specified via "sonar.text.inclusions"
22:30:27.277 INFO  Sensor TextAndSecretsSensor [text] (done) | time=2430ms
22:30:27.297 INFO  ------------ Run sensors on project
22:30:27.627 INFO  Sensor Zero Coverage Sensor
22:30:27.628 INFO  Sensor Zero Coverage Sensor (done) | time=1ms
22:30:27.636 INFO  SCM Publisher No SCM system was detected. You can use the 'sonar.scm.provider' property to explicitly specify it.
22:30:27.746 INFO  CPD Executor Calculating CPD for 0 files
22:30:27.766 INFO  CPD Executor CPD calculation finished (done) | time=0ms
22:30:28.301 INFO  Analysis report generated in 431ms, dir size=196.4 kB
22:30:28.353 INFO  Analysis report compressed in 33ms, zip size=19.2 kB
22:30:28.942 INFO  Analysis report uploaded in 564ms
22:30:28.943 INFO  ANALYSIS SUCCESSFUL, you can find the results at: http://localhost:9000/dashboard?id=sonarqube_10
22:30:28.943 INFO  Note that you will be able to access the updated dashboard once the server has processed the submitted analysis report
22:30:28.943 INFO  More about the report processing at http://localhost:9000/api/ce/task?id=9b35db7c-9c45-4e79-8ecb-3557a1964fa0
22:30:28.965 INFO  Analysis total time: 22.933 s
22:30:28.968 INFO  SonarScanner Engine completed successfully
22:30:29.025 INFO  EXECUTION SUCCESS
22:30:29.026 INFO  Total time: 33.488s
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

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
```