

EXPERIMENT - ①

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 is also known as white box testing.
- SAST takes place very early in the software development life cycle as it does not require a working application.
- SAST tools give developers real-time feedback as they code, helping them fix issues before they pass the code to next phase of SDLC. This prevents security-related issues from being considered an after-thought. Some tools point out 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 code to fix them, without requiring deep security domain expertise.

- Developers dramatically outnumber security staff. They are much faster than manual source code reviewers. These tools can scan millions of lines of code. SAST tools automatically identify critical vulnerabilities. Thus integrating static analysis into SDLC can yield dramatic results in overall quality of code.

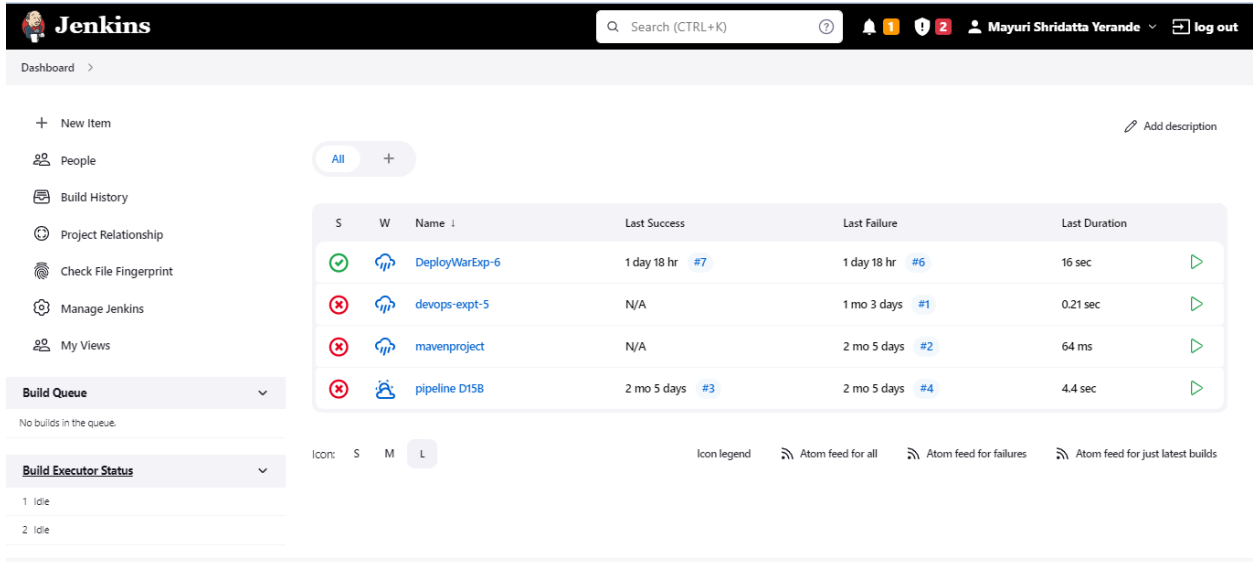
⇒ There are six simple steps needed to perform SAST efficiently in organizations that have a very large number of applications built with different languages, frameworks and platforms.

- ① Finalize the tool: The tool should be able to comprehend the underlying framework used by our software.
- ② Create the scanning infrastructure and deploy the tool: It involves handling licensing requirements, setting up access control and authorization and procuring the resources required to deploy the tool.
- ③ Customize the tool: Fine tune the tool to suit the needs of organization. For example, integrate tool into build environment, build custom reports.
- ④ Prioritize and onboard applications: Once the tool is ready, onboard your application. All your applications should be scanned regularly with application scans synced with release cycles, daily builds, or code check ins.

- ⑤ Analyse scan results:- It involves tagging the results of the scan to remove false positives.
- ⑥ Provide governance and learning:- It Ensures that your development teams are employing the scanning tools properly.

CONCLUSION: In this experiment, we understood importance of SAST and have successfully integrated jenkins with SonarQube for static analysis and code testing.

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



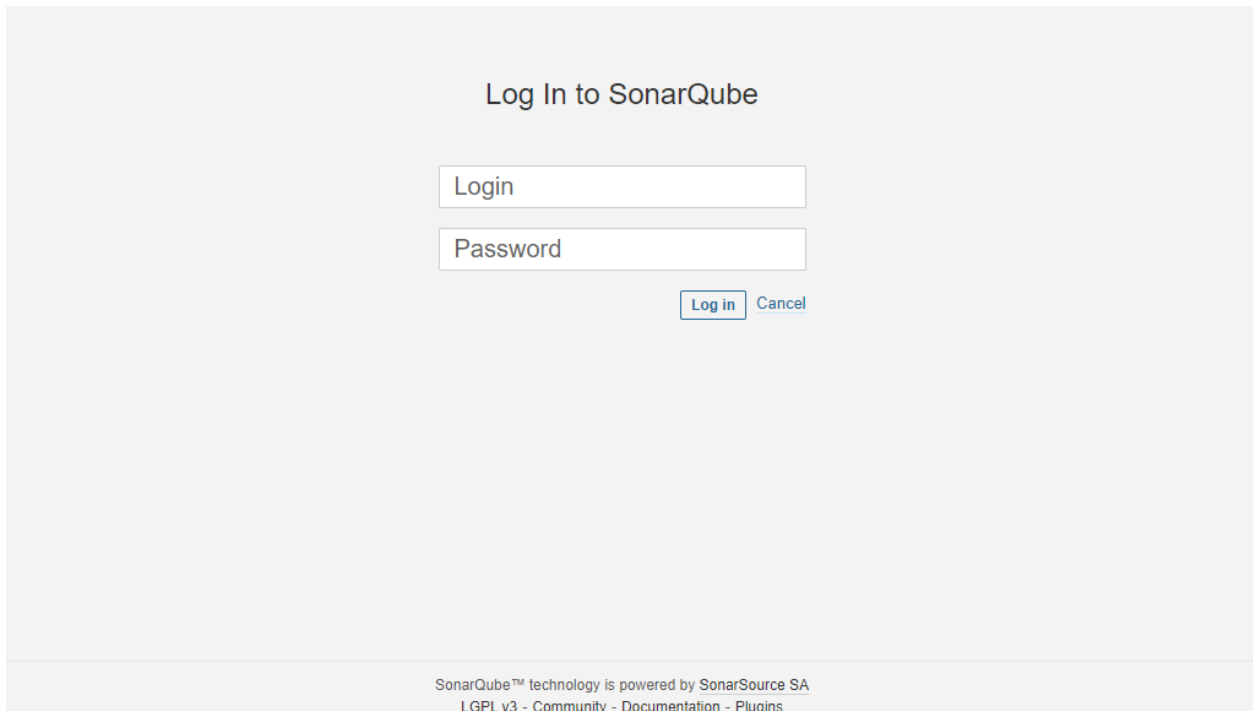
The screenshot shows the Jenkins Dashboard. The top navigation bar includes the Jenkins logo, a search bar, and user information (Mayuri Shridatta Yerande). The left sidebar contains links to 'New Item', 'People', 'Build History', 'Project Relationship', 'Check File Fingerprint', 'Manage Jenkins', and 'My Views'. The main content area displays a table of build jobs. The table has columns for status (S), icon (W), name, last success, last failure, and last duration. The jobs listed are 'DeployWarExp-6', 'devops-expt-5', 'mavenproject', and 'pipeline D15B'. The 'Build Queue' section shows 'No builds in the queue.' and the 'Build Executor Status' section shows two executors in an 'Idle' state.

S	W	Name	Last Success	Last Failure	Last Duration
✓	🔗	DeployWarExp-6	1 day 18 hr #7	1 day 18 hr #6	16 sec
✗	🔗	devops-expt-5	N/A	1 mo 3 days #1	0.21 sec
✗	🔗	mavenproject	N/A	2 mo 5 days #2	64 ms
✗	🔗	pipeline D15B	2 mo 5 days #3	2 mo 5 days #4	4.4 sec

Step 2 : Start sonarQube in cmd using command “StartSonar” in the path “C:\sonarqube-9.6.1.59531\bin\windows-x86-64”

Step 3 : Open sonarQube on the localhost port 9000

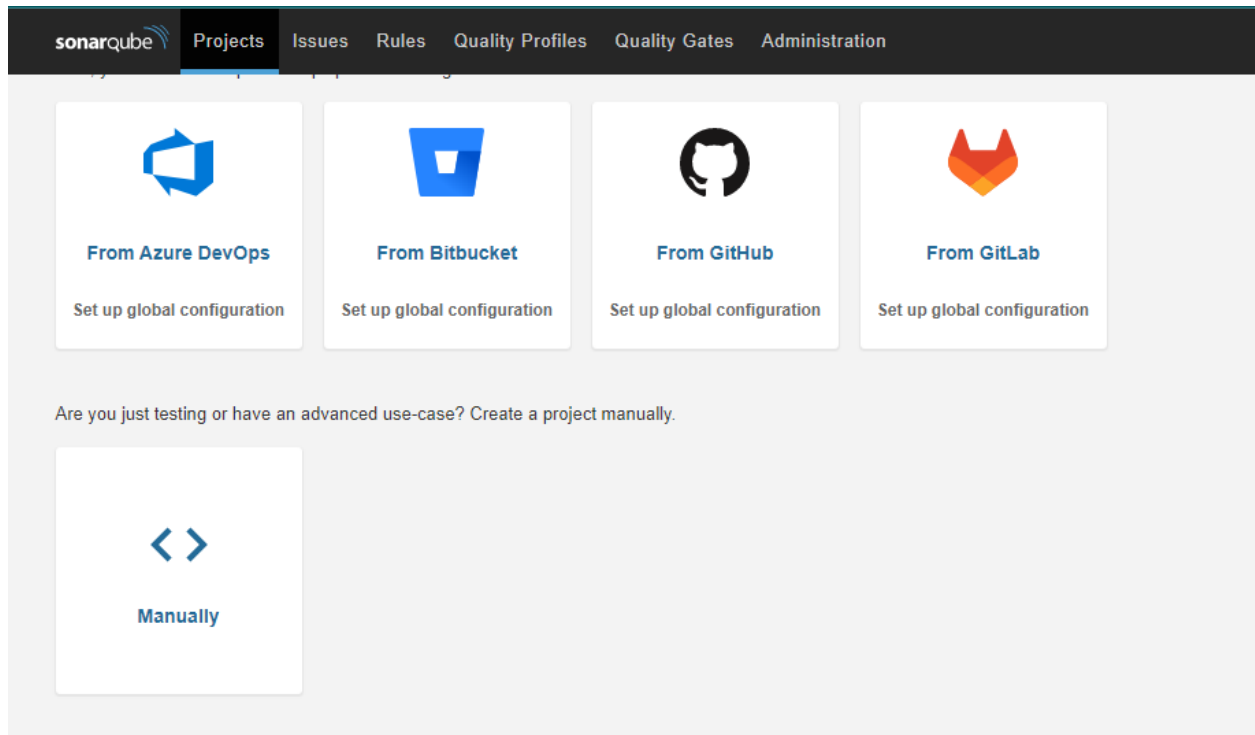
Step 4 : Login to SonarQube using username admin and password admin.



The screenshot shows the SonarQube login interface. It features a title 'Log In to SonarQube' and two input fields labeled 'Login' and 'Password'. Below the fields are 'Log in' and 'Cancel' buttons. The footer contains the text: 'SonarQube™ technology is powered by SonarSource SA', 'LGPL v3 - Community - Documentation - Plugins'.

Step 5: Create a manual project in SonarQube with the name sonarqube

Step 6: Setup the project and come back to Jenkins Dashboard.



The screenshot shows the 'Create a project' form in the SonarQube web interface. The form has two main input fields: 'Project display name' and 'Project key'. Both fields contain the text 'sonarqube' and have a green checkmark icon to their right, indicating they are valid. Below the 'Project display name' field, there is a note: 'Up to 255 characters. Some scanners might override the value you provide.' Below the 'Project key' field, there is a note: 'The project key is a unique identifier for your project. It may contain up to 400 characters. Allowed characters are alphanumeric, '-' (dash), '_' (underscore), '.' (period) and ':' (colon), with at least one non-digit.' At the bottom of the form, there is a 'Set Up' button. The top navigation bar includes the SonarQube logo and links to 'Projects', 'Issues', 'Rules', 'Quality Profiles', 'Quality Gates', and 'Administration'. A search bar is visible on the right side of the navigation bar.

The screenshot shows a notification for the 'SonarQube Scanner for Jenkins' plugin. The notification includes the plugin name and version '2.14'. Below this, it states: 'This plugin allows an easy integration of SonarQube, the open source platform for Continuous Inspection of code quality.' There is a link to 'Report an issue with this plugin'. On the right side of the notification, there is a toggle switch that is currently turned on, indicated by a blue circle with a white checkmark. To the right of the toggle switch is a red circle with a white 'X' icon.

Step 7: Go to Manage Jenkins and search for SonarQube Scanner for Jenkins and install it.

Step 8: Under Jenkins ‘Configure System’, look for SonarQube Servers and enter the details.

Step 9: Enter the Server Authentication token if needed.

Step 10: Search for SonarQube Scanner under Global Tool Configuration. Choose the latest configuration and choose Install automatically.

List of SonarQube Scanner installations on this system

Add SonarQube Scanner

☰ SonarQube Scanner

Name

SonarQube Scanner

☒ Install automatically ?

☰ Install from Maven Central

Version

SonarQube Scanner 4.7.0.2747 ▼

Add Installer ▼

Add SonarQube Scanner

Save

Apply

SonarQube installations

List of SonarQube installations

Name

SonarQube

Server URL

Default is http://localhost:9000

http://localhost:9000

Server authentication token

SonarQube authentication token. Mandatory when anonymous access is disabled.

- none - ▼

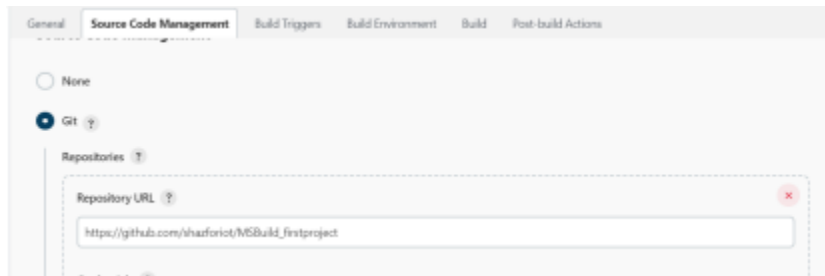
+ Add

Advanced...

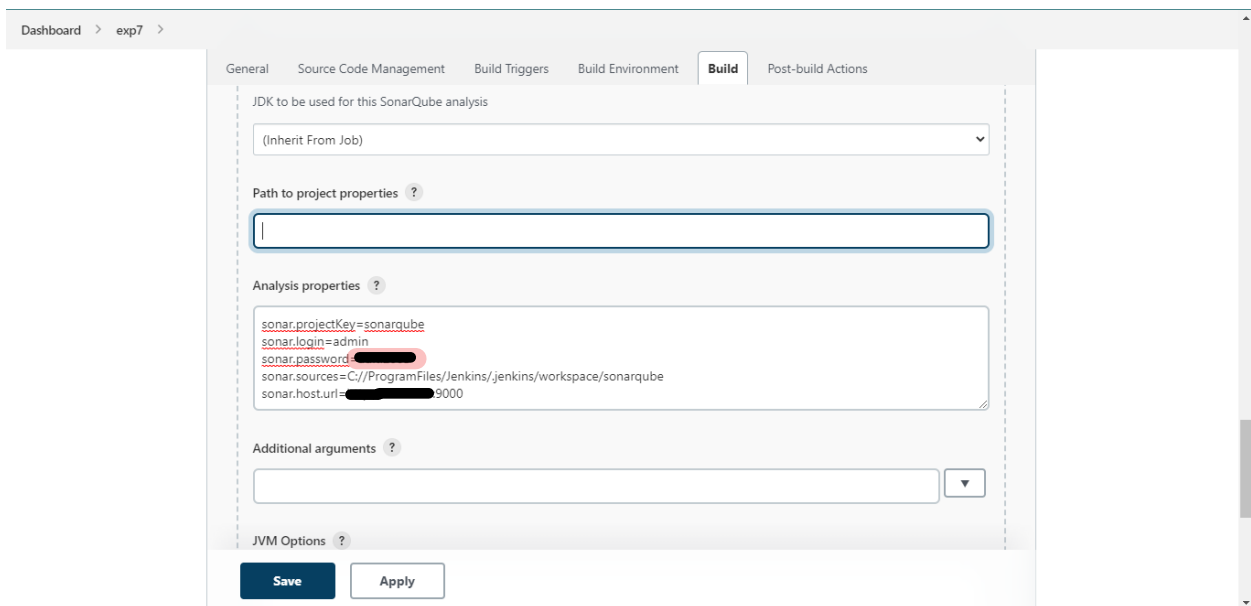
Save

Apply

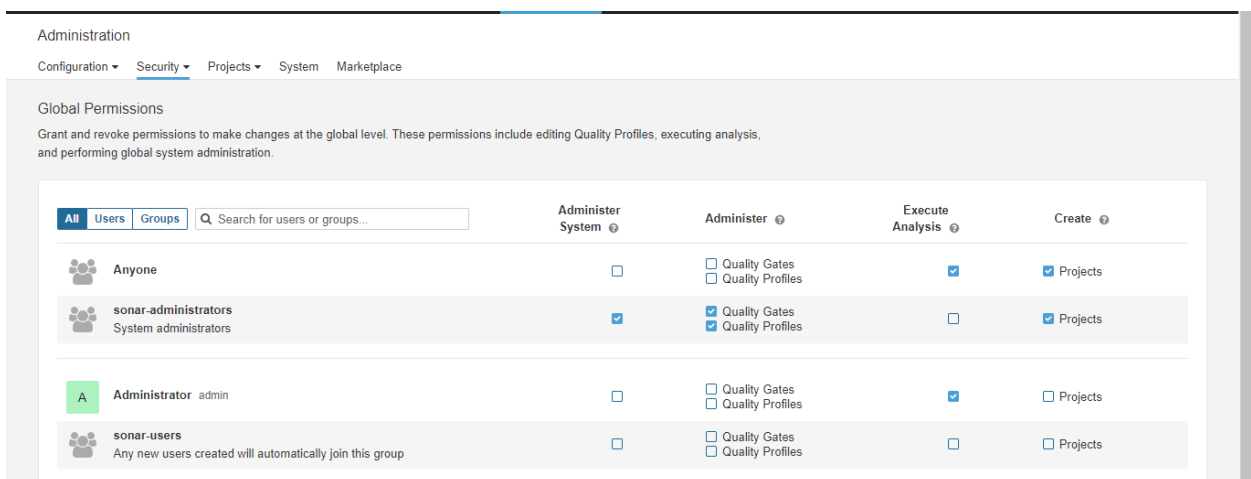
Step 11: Choose this GitHub repository in Source Code Management.
https://github.com/shazforiot/MSBuild_firstproject.git



Step 12: Under Build-> Execute SonarQube Scanner, enter these Analysis properties. Mention the SonarQube Project Key, Login, Password, Source path and Host URL.



Step 13: Go to <http://localhost:9000/permissions> and allow Execute Permissions to the Admin user.



Step 14: Run The Build.

Check the console output.

Dashboard > sonarqube > #6

↑ Back to Project

📄 Status

</> Changes

🔍 Console Output

📄 View as plain text

⚙️ Edit Build Information

🗑️ Delete build '#6'

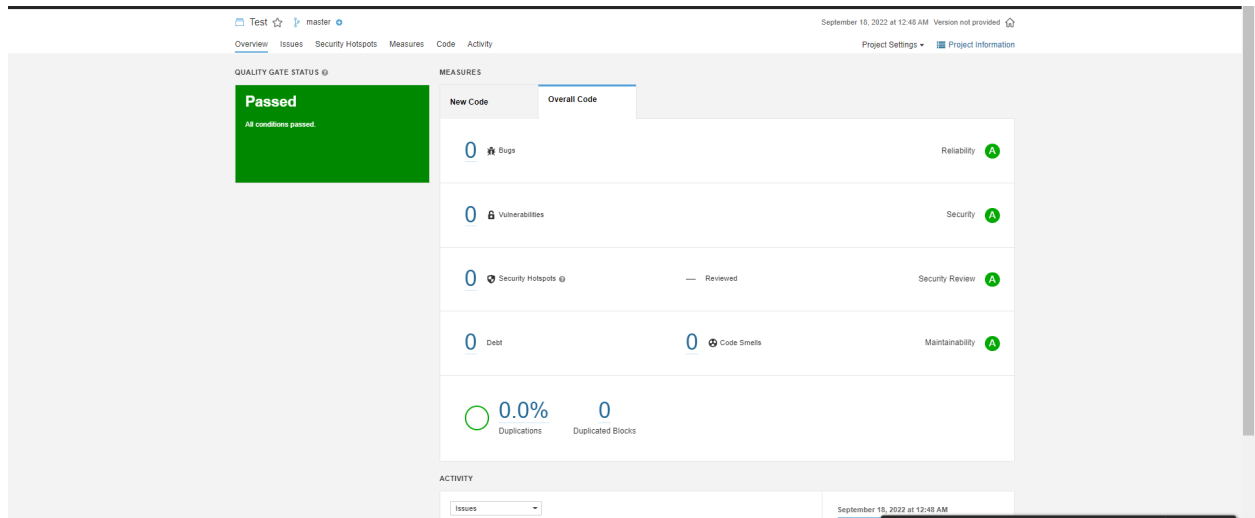
🔴 Git Build Data

← Previous Build

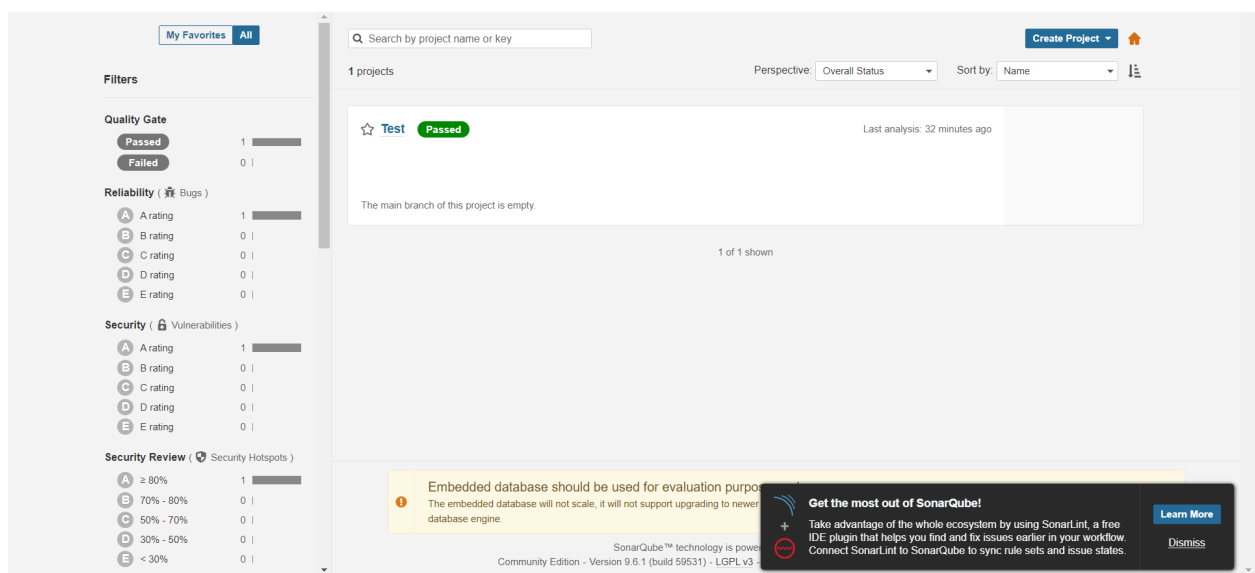
🟢 Console Output

Started by user **user**
Running as SYSTEM
Building 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 # timeout=10
Fetching upstream changes from https://github.com/shazforiot/MSBuild_firstproject
> git.exe --version # timeout=10
> git --version # 'git version 2.37.1.windows.1'
> git.exe fetch --tags --force --progress -- https://github.com/shazforiot/MSBuild_firstproject +refs/heads/*:refs/remotes/origin/* # timeout=10
> git.exe rev-parse "refs/remotes/origin/master:{commit}" # timeout=10
Checking out Revision f2bc042c04c6e72427c380bcae6d6fee7b49adf (refs/remotes/origin/master)
> git.exe config core.sparsecheckout # timeout=10
> git.exe checkout -f f2bc042c04c6e72427c380bcae6d6fee7b49adf # timeout=10
Commit message: "updated"
> git.exe rev-list --no-walk f2bc042c04c6e72427c380bcae6d6fee7b49adf # timeout=10
[sonarqube] \$ C:\ProgramData\Jenkins\jenkins\tools\udson.plugins.sonar.SonarRunnerInstallation\SonarQube_Scanner\bin\sonar-scanner.bat -
Dsonar.host.url=http://localhost:9000 -Dsonar.projectKey=Test -Dsonar.login=admin -Dsonar.host.url=http://127.0.0.1:9000 -
Dsonar.sources=C:\ProgramData\Jenkins\jenkins\workspace\sonarqube -Dsonar.password=Sam -
Dsonar.projectBaseDir=C:\ProgramData\Jenkins\jenkins\workspace\sonarqube
WARN: Property 'sonar.host.url' with value 'http://localhost:9000' is overridden with value 'http://127.0.0.1:9000'
INFO: Scanner configuration file:

WARN: Your project contains C# files which cannot be analyzed with the scanner you are using. To analyze C# or VB.NET, you must use the SonarScanner for .NET 5.x or higher, see <https://redirect.sonarsource.com/doc/install-configure-scanner-msbuild.html>
INFO: Sensor C# [csharp] (done) | time=2ms
INFO: Sensor Analysis Warnings import [csharp]
INFO: Sensor Analysis Warnings import [csharp] (done) | time=1ms
INFO: Sensor Zero Coverage Sensor
INFO: Sensor Zero Coverage Sensor (done) | time=11ms
INFO: SCM Publisher SCM provider for this project is: git
INFO: SCM Publisher 4 source files to be analyzed
INFO: SCM Publisher 4/4 source files have been analyzed (done) | time=553ms
INFO: CPD Executor Calculating CPD for 0 files
INFO: CPD Executor CPD calculation finished (done) | time=0ms
INFO: Analysis report generated in 152ms, dir size=119.3 kB
INFO: Analysis report compressed in 44ms, zip size=16.9 kB
INFO: Analysis report uploaded in 375ms
INFO: ANALYSIS SUCCESSFUL, you can find the results at: <http://127.0.0.1:9000/dashboard?id=Test>
INFO: Note that you will be able to access the updated dashboard once the server has processed the submitted analysis report
INFO: More about the report processing at http://127.0.0.1:9000/api/ce/task?id=AYWMywOQSikg_mrkWq_h
INFO: Analysis total time: 13.775 s
INFO: -----
INFO: EXECUTION SUCCESS
INFO: -----
INFO: Total time: 19.525s
INFO: Final Memory: 16M/57M
INFO: -----
Finished: SUCCESS



Step 15: Once the build is complete, check the project in SonarQube.



Conclusion: We have understood Static Analysis SAST process and learnt to integrate Jenkins SAST to SonarQube/GitLab.