

Advanced DevOps Lab

Experiment 07

Roll No.	24
Name	Iyer Sreekesh Subramanian
Class	D15-A
Subject	Advanced DevOps Lab

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:

Prerequisites:

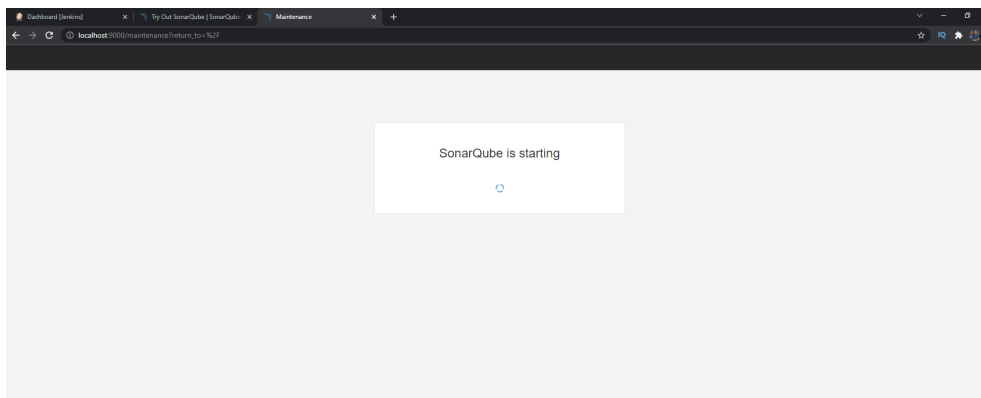
- Jenkins installed
- Docker Installed (for SonarQube)
- SonarQube Docker Image

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
```

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**

All fields marked with * are required

Project display name *

✓

Up to 255 characters. Some scanners might override the value you provide.

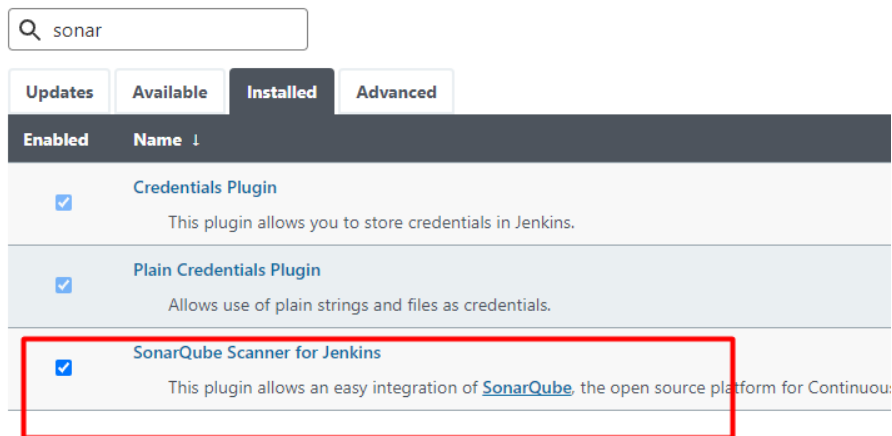
Project key *

✓

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.

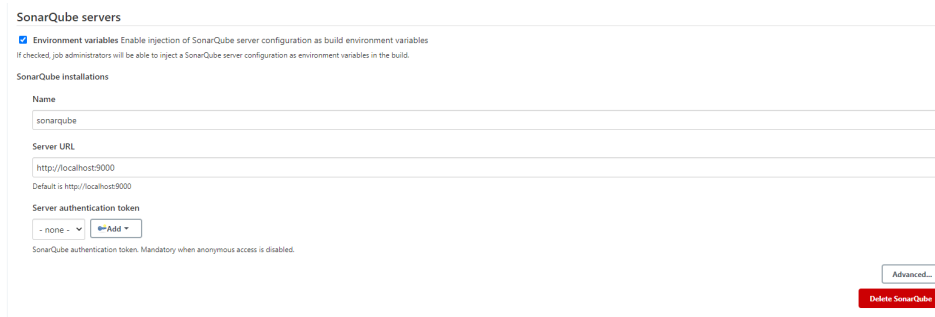
Setup the project and come back to Jenkins Dashboard.

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

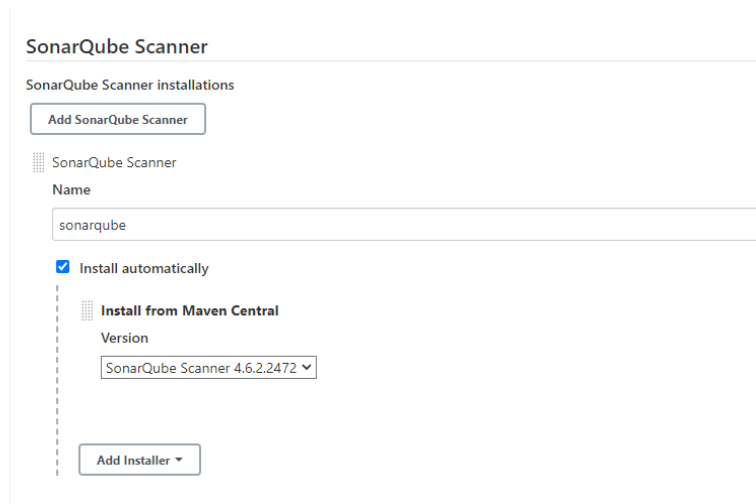


7. Under Jenkins 'Configure System', look for SonarQube Servers and enter the details.

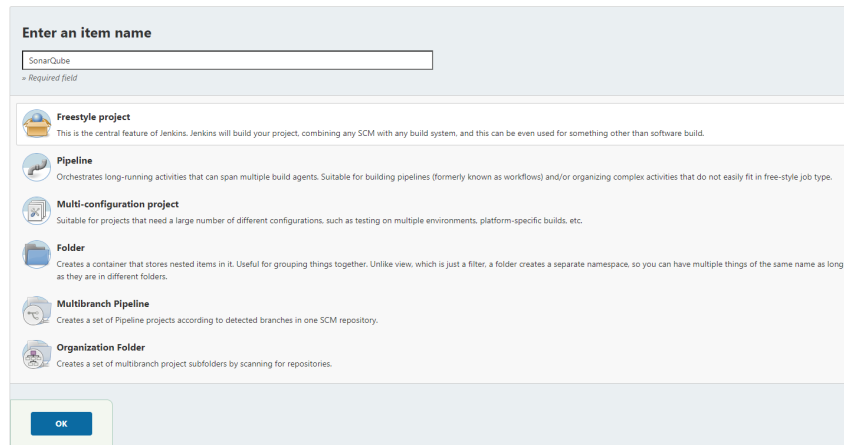
Enter the Server Authentication token if needed.



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



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



Enter an item name

SonarQube

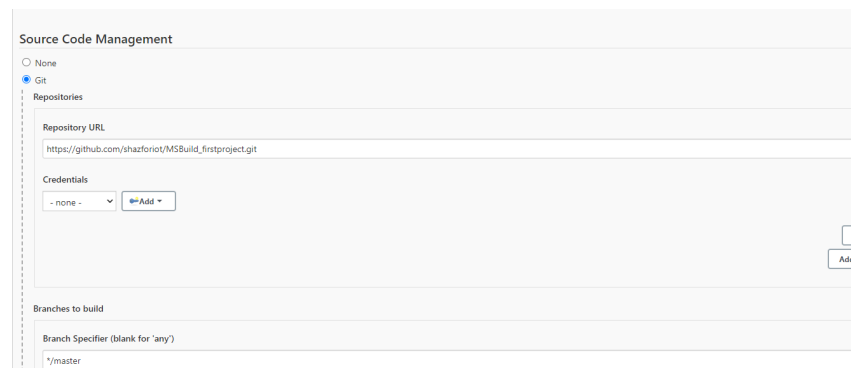
= Required field

- Freestyle project**
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.
- Pipeline**
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.
- Multi-configuration project**
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.
- Folder**
Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.
- Multibranch Pipeline**
Creates a set of Pipeline projects according to detected branches in one SCM repository.
- Organization Folder**
Creates a set of multibranch project subfolders by scanning for repositories.

OK

10. Choose this GitHub repository in Source Code Management.

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



Source Code Management

☐ None

☒ Git

Repositories

Repository URL

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

Credentials

Add

Branches to build

Branch Specifier (blank for 'any')

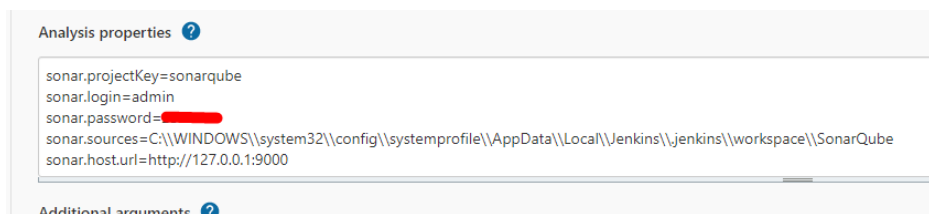
`*/master`

Add

Add Re

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

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



Analysis properties ?

sonar.projectKey=sonarqube

sonar.login=admin

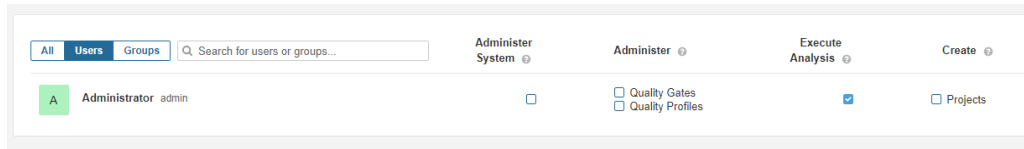
sonar.password=[redacted]

sonar.sources=C:\\WINDOWS\\system32\\config\\systemprofile\\AppData\\Local\\Jenkins\\jenkins\\workspace\\SonarQube

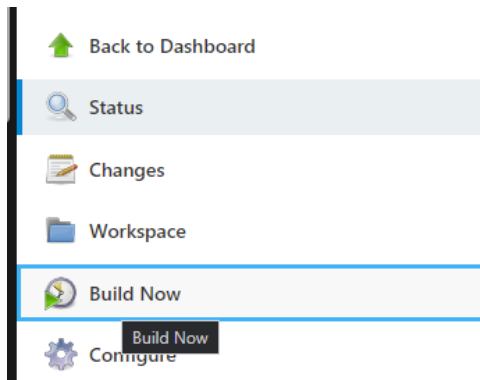
sonar.host.url=http://127.0.0.1:9000

Additional arguments ?

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



13. Run The Build.



Check the console output.

```

Console Output

Started by user Sreekesh Iyer
Running as SYSTEM
Building in workspace C:\WINDOWS\system32\config\systemprofile\AppData\Local\Jenkins\.jenkins\work
The recommended git tool is: NONE
No credentials specified
> git.exe rev-parse --resolve-git-dir C:\WINDOWS\system32\config\systemprofile\AppData\Local\Jen
Fetching changes from the remote Git repository
> git.exe config remote.origin.url https://github.com/shazforiot/MSBuild_firstproject.git # timeo
Fetching upstream changes from https://github.com/shazforiot/MSBuild_firstproject.git
> git.exe --version # timeout=10
> git --version # 'git version 2.29.2.windows.2'
> git.exe fetch --tags --force --progress -- https://github.com/shazforiot/MSBuild_firstproject.g
> 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"
First time build. Skipping changelog.
[SonarQube] $ C:\WINDOWS\system32\config\systemprofile\AppData\Local\Jenkins\.jenkins\tools\hudson

```

```

INFO: Load project repositories (done) | time=204ms
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=303ms
INFO: CPD Executor Calculating CPD for 0 files
INFO: CPD Executor CPD calculation finished (done) | time=0ms
INFO: Analysis report generated in 86ms, dir size=105.2 kB
INFO: Analysis report compressed in 173ms, zip size=15.5 kB
INFO: Analysis report uploaded in 248ms
INFO: ANALYSIS SUCCESSFUL, you can browse http://127.0.0.1:9000/dashboard?id=sonarqube
INFO: Note that you will be able to access the updated dashboard once the server has processed the s
INFO: More about the report processing at http://127.0.0.1:9000/api/ce/task?id=AXxvUtkueVymuXkA1Ro
INFO: Analysis total time: 11.138 s
INFO: -----
INFO: EXECUTION SUCCESS
INFO: -----
INFO: Total time: 13.011s
INFO: Final Memory: 7M/30M
INFO: -----
Finished: SUCCESS

```

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

The screenshot displays the SonarQube web interface. At the top, there's a navigation bar with the SonarQube logo and a green 'Passed' badge. To the right, it says 'Last analysis: 7 hours ago'. Below the navigation bar, a message states 'The main branch of this project is empty.' The main content area is divided into two sections: 'QUALITY GATE STATUS' and 'MEASURES'. The 'QUALITY GATE STATUS' section shows a green 'Passed' indicator with the text 'All conditions passed.' The 'MEASURES' section is a table with two columns: 'New Code' and 'Overall Code'. It lists various metrics: Bugs (0), Vulnerabilities (0), Security Hotspots (0), Debt (0), Code Smells (0), Reliability (A), Security (A), Security Review (A), and Maintainability (A). At the bottom, there's a section for 'ACTIVITY' showing '0.0%' for Duplications and '0' for Duplicated Blocks.

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.