

# Advanced DevOps Lab

## Experiment 7

**Name: Yash Rahate**

**Class: D15B**

**Roll no.: 48**

### **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.

### Steps to integrate Jenkins with SonarQube:

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

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

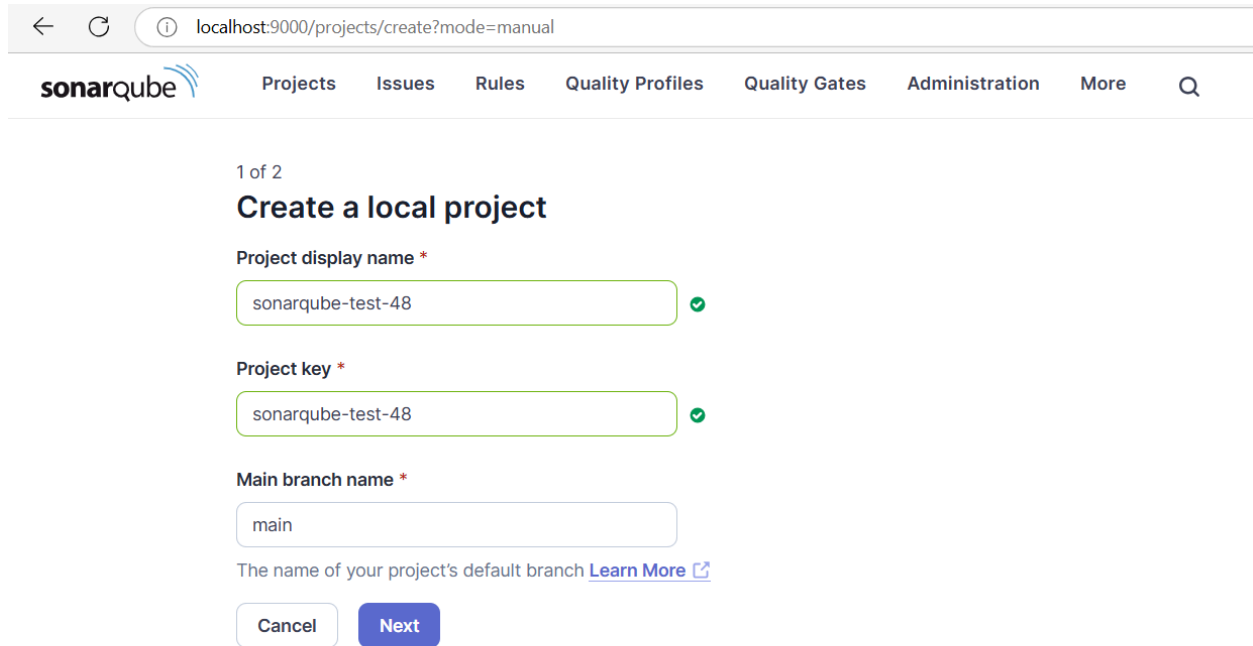


```
C:\Users\acer>docker run -d --name sonarqube -e SONAR_ES_BOOTSTRAP_CHECKS_DISABLE=true -p 9000:9000 sonarqube:latest
Unable to find image 'sonarqube:latest' locally
latest: Pulling from library/sonarqube
7478e0ac0f23: Pull complete
90a925ab929a: Pull complete
7d9a34308537: Pull complete
80338217a4ab: Pull complete
1a5fd5c7e184: Pull complete
7b87d6fa783d: Pull complete
bd819c9b5ead: Pull complete
4f4fb700ef54: Pull complete
Digest: sha256:72e9feec71242af83faf65f95a40d5e3bb2822a6c3b2cda8568790f3d31aecde
Status: Downloaded newer image for sonarqube:latest
e40af92b2267b1f9010bce98466607f969cec21a99ba588aa8d2e0f8038127d2

C:\Users\acer>docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS                    NAMES
e40af92b2267   sonarqube:latest  "/opt/sonarqube/dock..."  2 hours ago   Up 2 hours   0.0.0.0:9000->9000/tcp    sonarqube

C:\Users\acer>
```

2. Once the container is up and running, you can check the status of SonarQube at localhost port 9000.
3. Login to SonarQube using username *admin* and password *admin*.
4. Create a manual project in SonarQube with the name **sonarqube-test-48**



The screenshot shows the SonarQube web interface at the URL `localhost:9000/projects/create?mode=manual`. The page is titled "1 of 2 Create a local project". It contains three input fields, each with a green checkmark indicating successful validation:

- Project display name \***: `sonarqube-test-48`
- Project key \***: `sonarqube-test-48`
- Main branch name \***: `main`

Below the inputs, there is a link: "The name of your project's default branch [Learn More](#)". At the bottom are two buttons: "Cancel" and "Next".

5. Under Jenkins 'Configure System', look for SonarQube Servers and enter the details. Enter the Server Authentication token if needed.

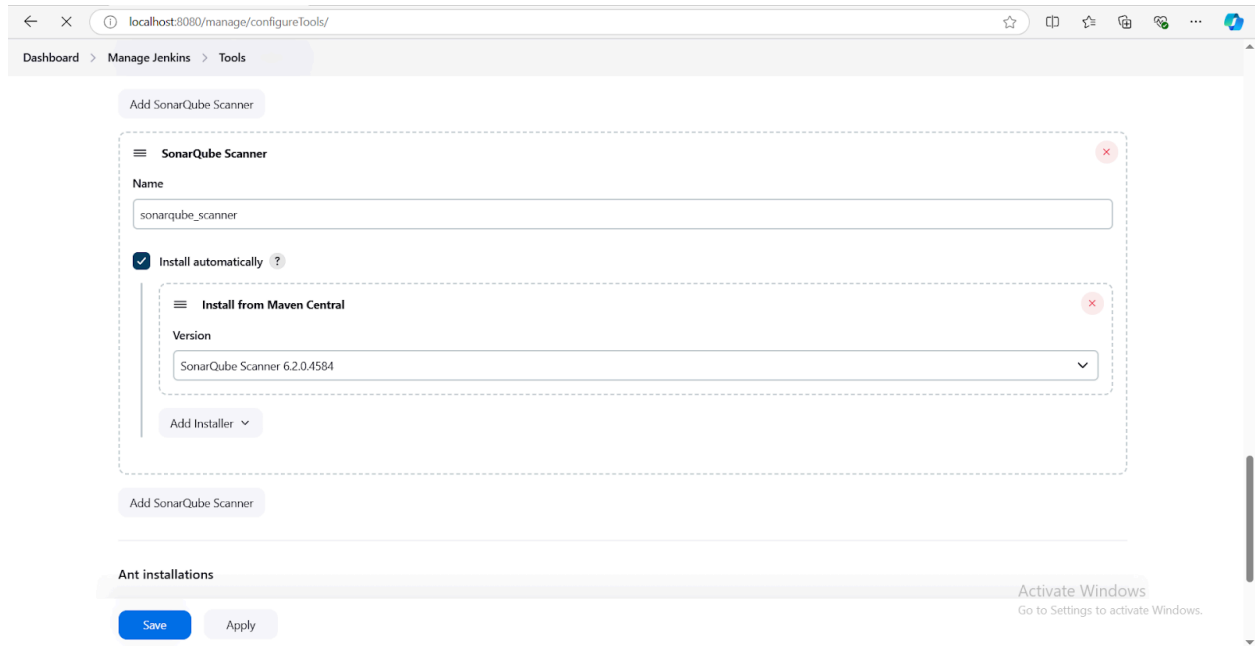


The screenshot shows the Jenkins "Configure System" page under the "System" tab. It features a section for "SonarQube installations" with a list of configurations. The configuration shown has the following details:

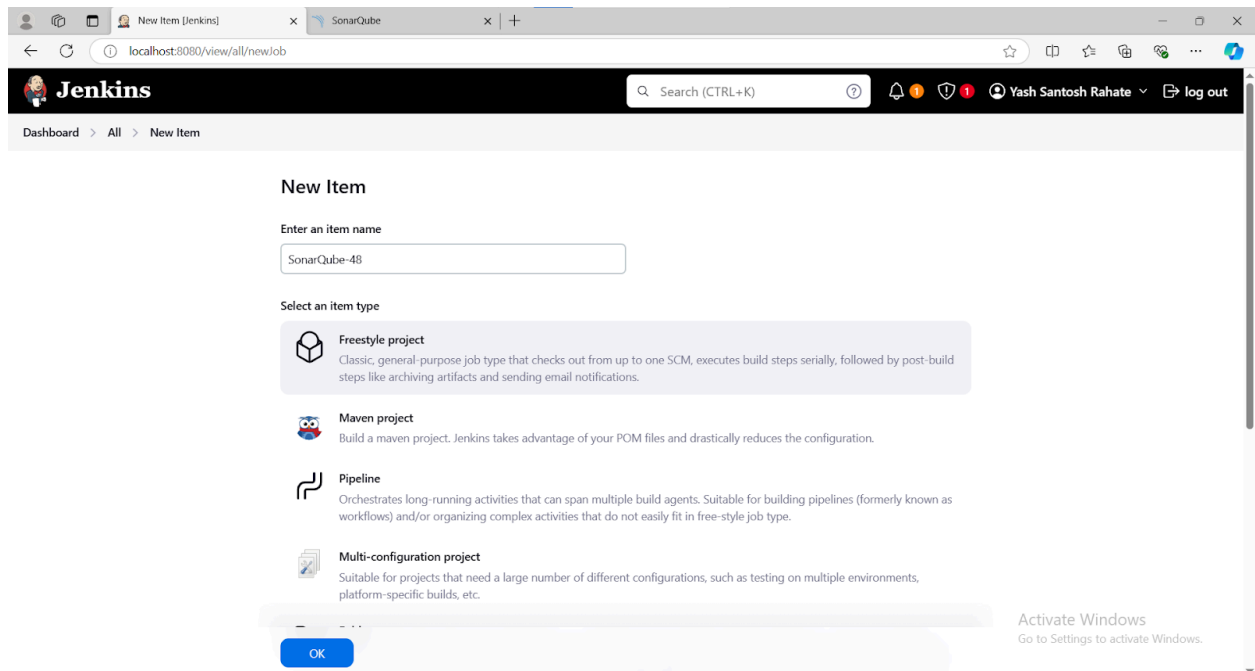
- Name**: `sonarqube`
- Server URL**: `http://localhost:9000/` (Default is `http://localhost:9000`)
- Server authentication token**: `- none -` (Mandatory when anonymous access is disabled)

There is an "+ Add" button at the bottom of the configuration list.

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



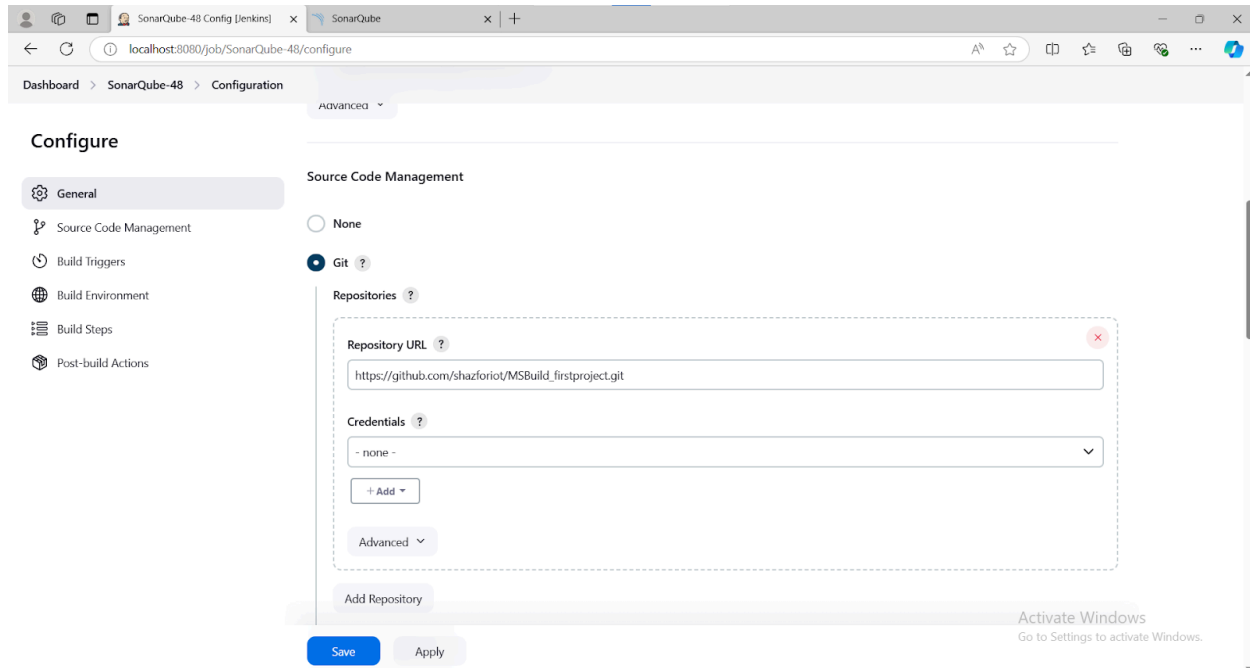
7. After the configuration, create a New Item in Jenkins, choose a freestyle project.



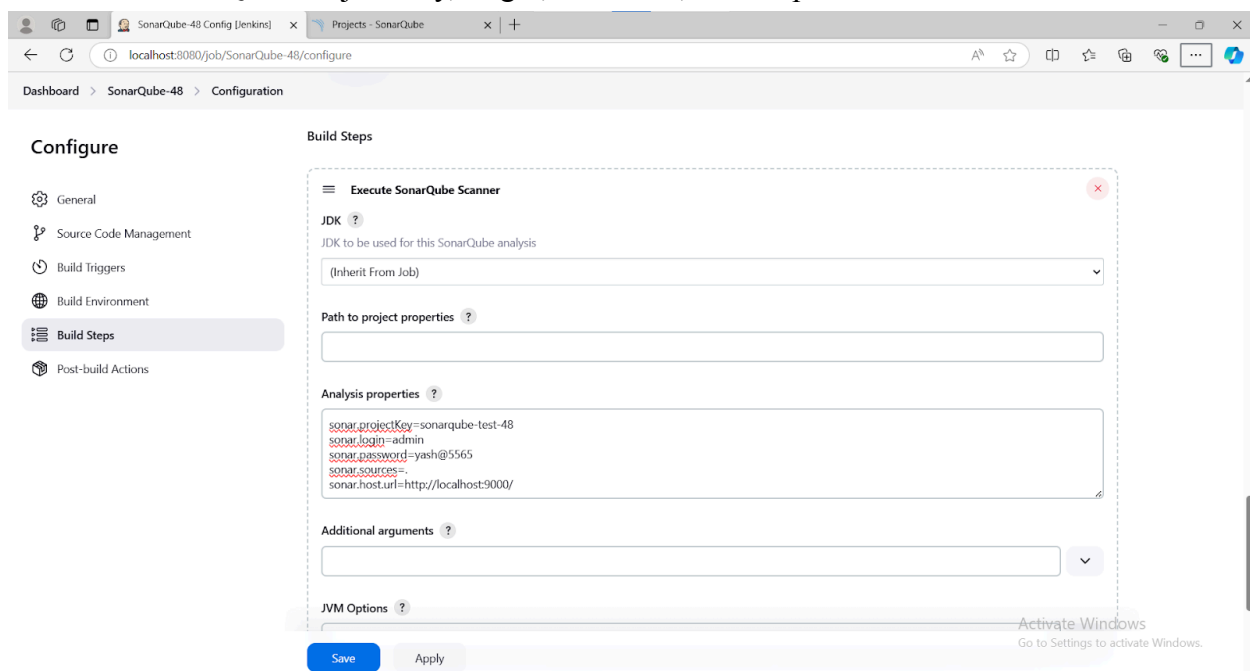
8. Choose this GitHub repository in Source Code Management.

[https://github.com/shazforiot/MSBuild\\_firstproject.git](https://github.com/shazforiot/MSBuild_firstproject.git)

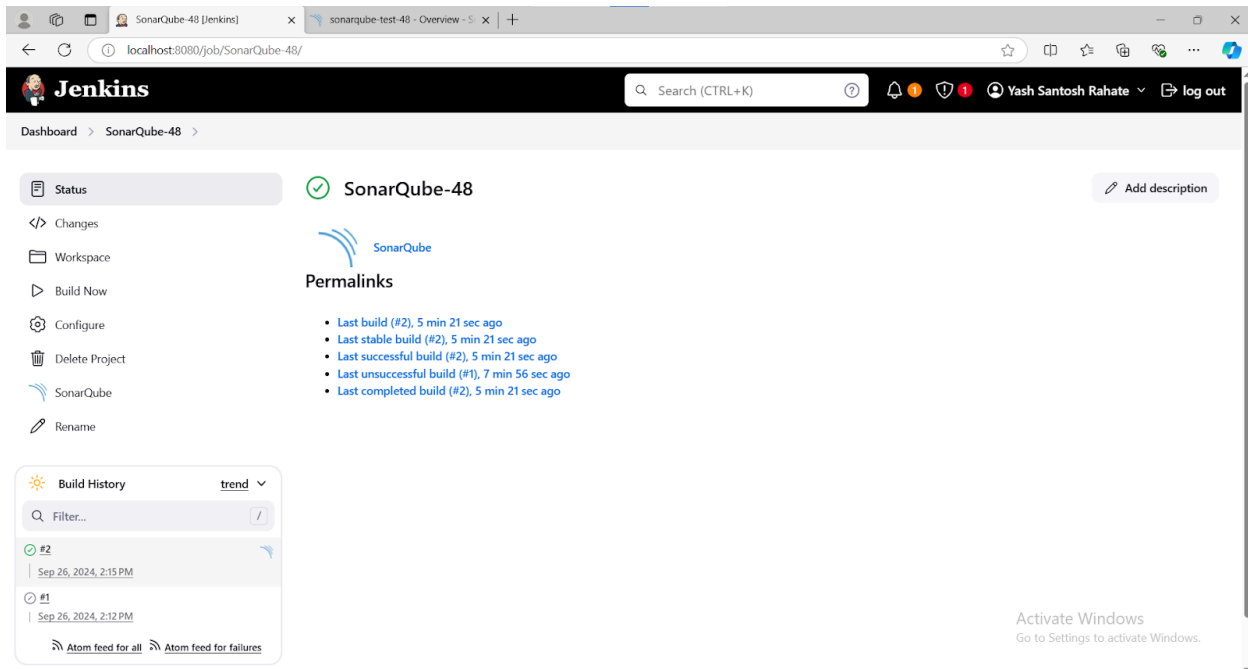
It is a sample hello-world project with no vulnerabilities and issues, just to test



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

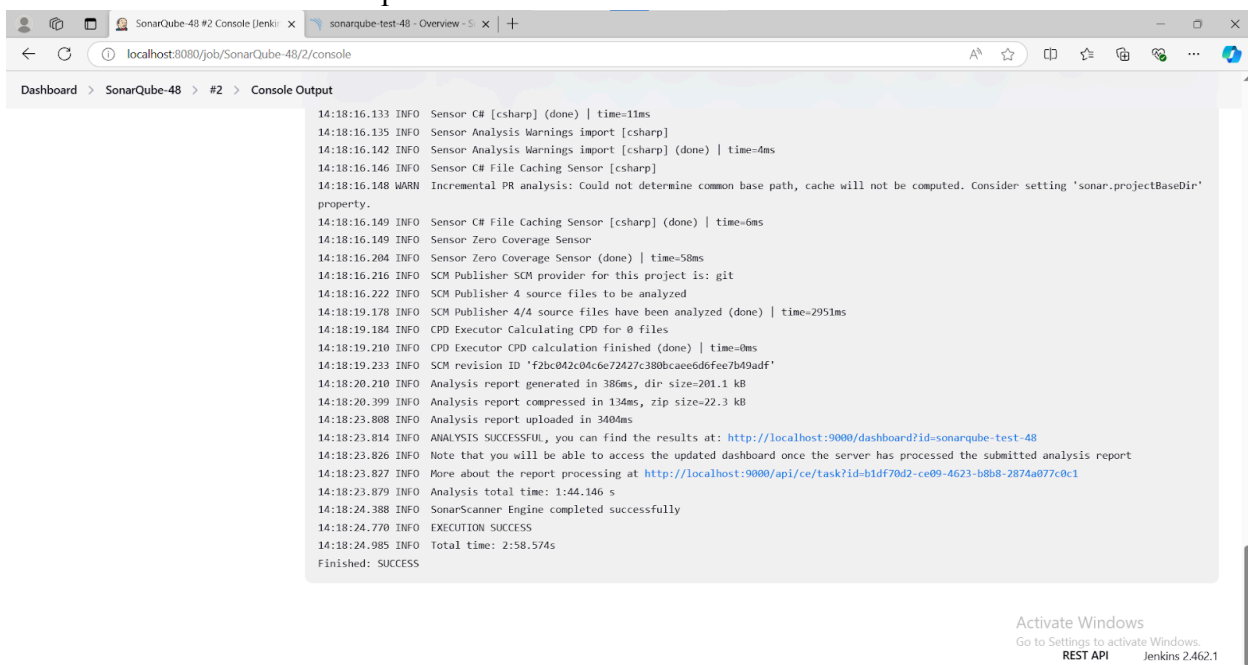


## 10. Run The Build.



The screenshot shows the Jenkins interface for the SonarQube-48 job. The top navigation bar includes the Jenkins logo, a search bar, and user information (Yash Santosh Rahate). The main content area displays the job status as 'Success' with a green checkmark. A sidebar on the left contains links to various job actions: Status, Changes, Workspace, Build Now, Configure, Delete Project, SonarQube, and Rename. The main area also features a 'Permalinks' section with a list of build links, including 'Last build (#2)', 'Last stable build (#2)', 'Last successful build (#2)', 'Last unsuccessful build (#1)', and 'Last completed build (#2)'. A 'Build History' section is visible at the bottom left, showing a list of builds with their status and timestamps. The bottom right corner of the page has an 'Activate Windows' watermark.

## 11. Check the console output.



The screenshot shows the Jenkins console output for the SonarQube-48 job. The top navigation bar is the same as in the previous screenshot. The main content area displays the console output, which is a log of messages from the SonarScanner and the Jenkins server. The log includes information about the sensor execution, analysis warnings, file caching, and the final analysis results. The output shows that the analysis was successful and the report was generated. The bottom right corner of the page has an 'Activate Windows' watermark.

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

The screenshot shows the SonarQube web interface at `localhost:9000/projects`. The 'Projects' tab is active. On the left, there are filters for Quality Gate (1 Passed, 0 Failed), Reliability (1 A, 0 B, 0 C, 0 D, 0 E), and Security (1 A). The main area displays a list of projects. One project, 'sonarqube-test-48', is shown with a 'Passed' status. The last analysis was 7 minutes ago. The main branch of this project is empty. A warning banner at the bottom states: 'Embedded database should be used for evaluation purposes only. The embedded database will not scale. It will not support upgrading to newer versions of SonarQube, and there is no support for migrating your data out of it into a different database engine.'

The screenshot shows the SonarQube web interface at `localhost:9000/dashboard?id=sonarqube-test-48&codeScope=overall`. The 'Overview' tab is active. The project 'sonarqube-test-48' is shown with a 'Passed' status. The last analysis was 6 minutes ago. A warning banner states: 'The last analysis has warnings. See details.' The 'Overall Code' tab is selected, showing the following metrics:

Metric	Value	Quality Gate
Security	0 Open issues	A
Reliability	0 Open issues	A
Maintainability	0 Open issues	A
Accepted issues	0	0
Coverage	0.0%	0
Duplications	0.0%	0
Security Hotspots	0	A

The 'Accepted issues' section shows 'Valid issues that were not fixed'. The 'Coverage' section shows 'On 0 lines to cover'. The 'Duplications' section shows 'On 86 lines'. An 'Activity' section is visible at the bottom.



**Conclusion:**

Integrating SAST (Static Application Security Testing) with Jenkins and SonarQube/GitLab automates the process of identifying security vulnerabilities in the code during development. This improves code quality, detects issues early, and ensures a streamlined, secure development workflow, helping teams to address vulnerabilities quickly and efficiently.