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.

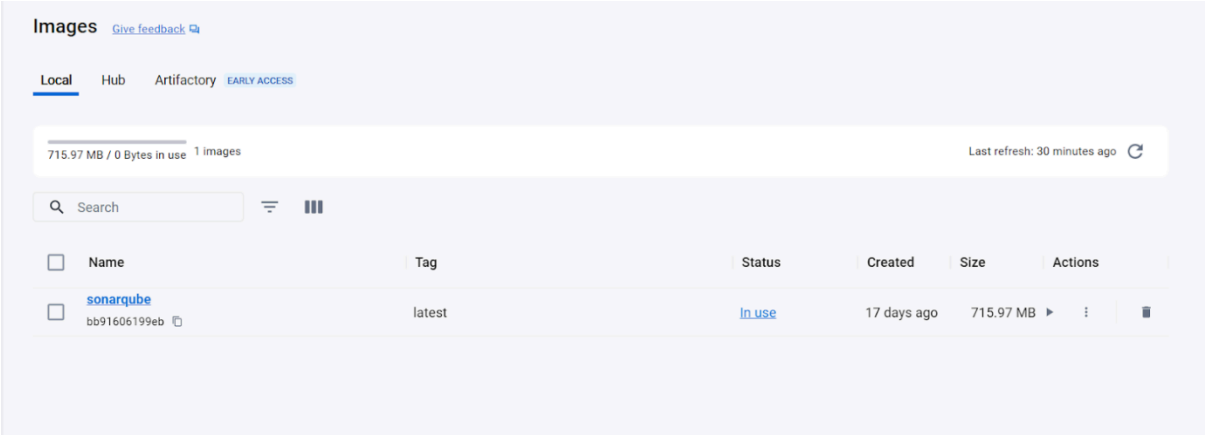
4. **Prioritize and onboard applications.** Once the tool is ready, onboard your applications. If you have a large number of applications, prioritize the high-risk applications to scan first. Eventually, all your applications should be onboarded and scanned regularly, with
application scans synced with 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.

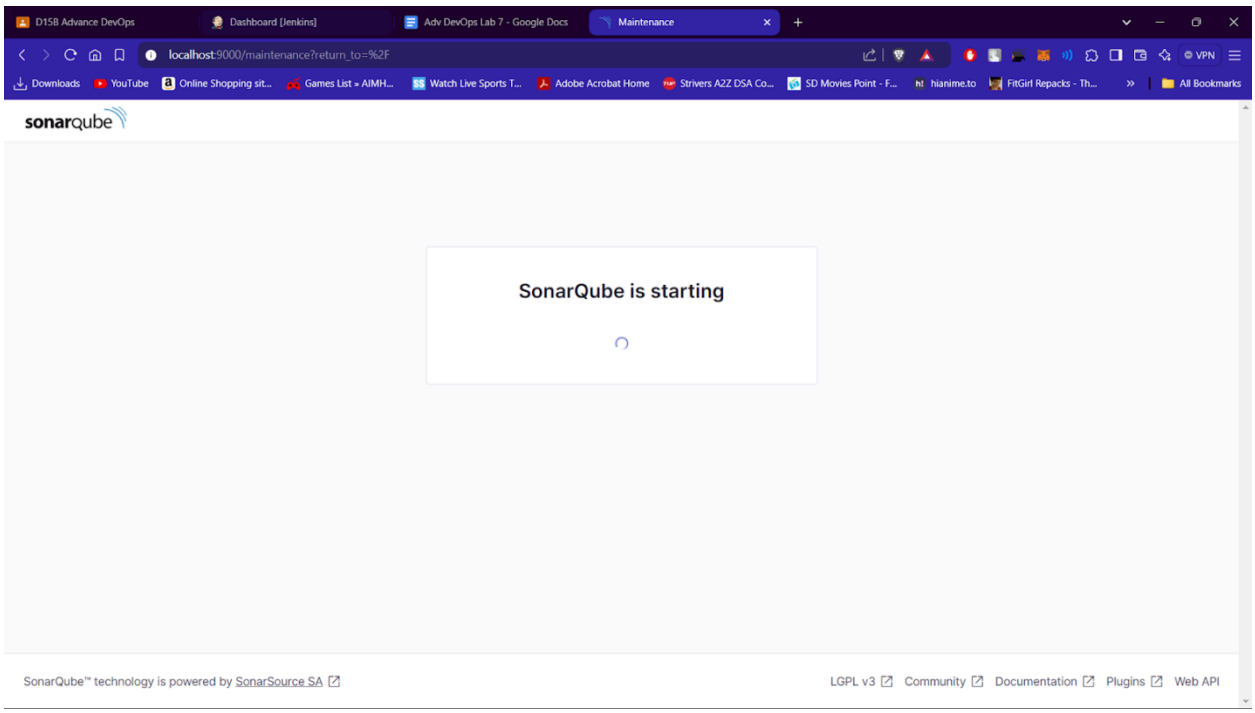
6. **Provide governance and training.** Proper governance ensures that your development teams are employing the scanning tools properly. The software security touchpoints

Steps:

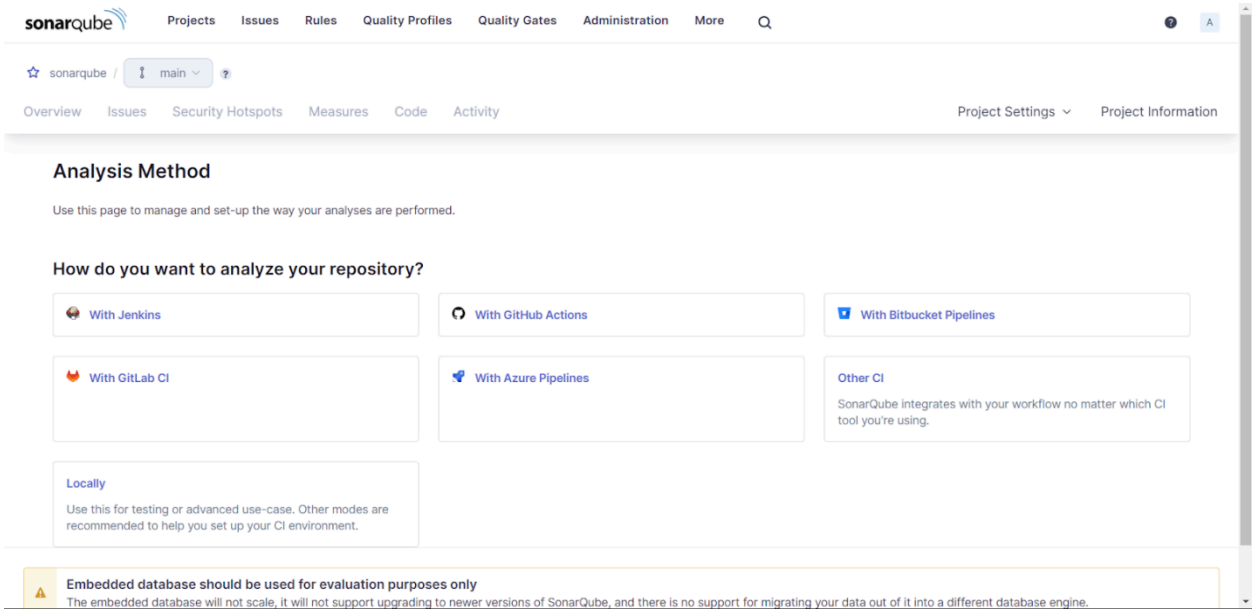
```
$ docker run -d --name sonarqube -e SONAR_ES_BOOTSTRAP_CHECKS_DISABLE=true -p
9000:9000
Sonarqube:latest
```



go to the SonarQube page by typing:
http://localhost:9000/ on your browser.

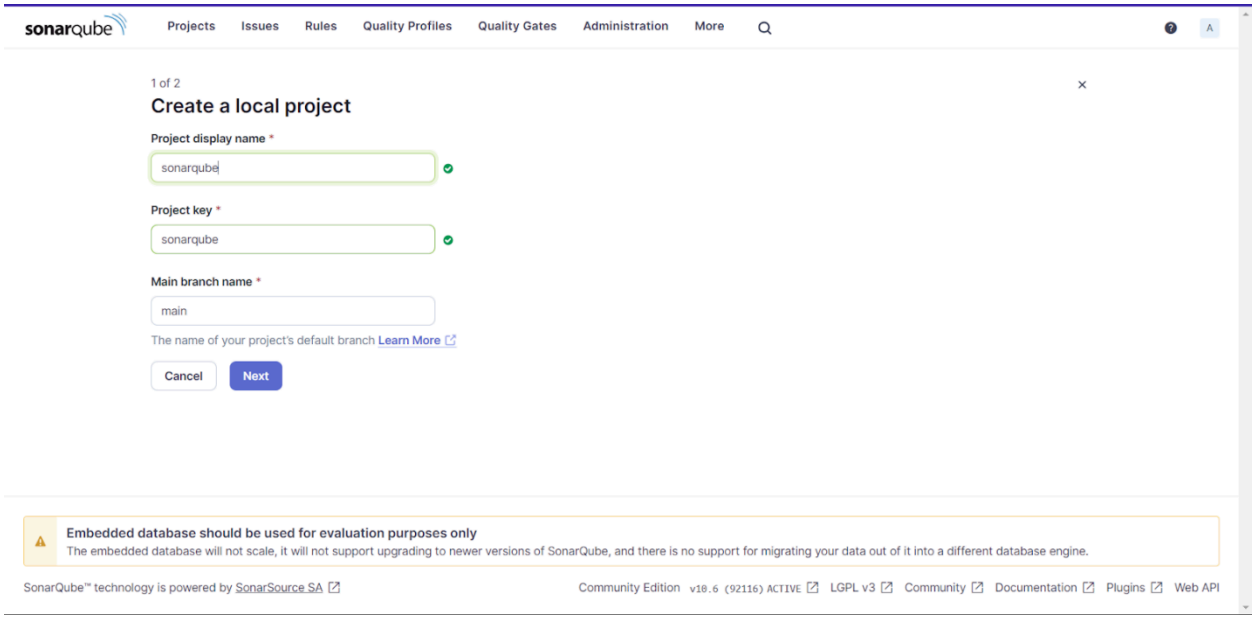


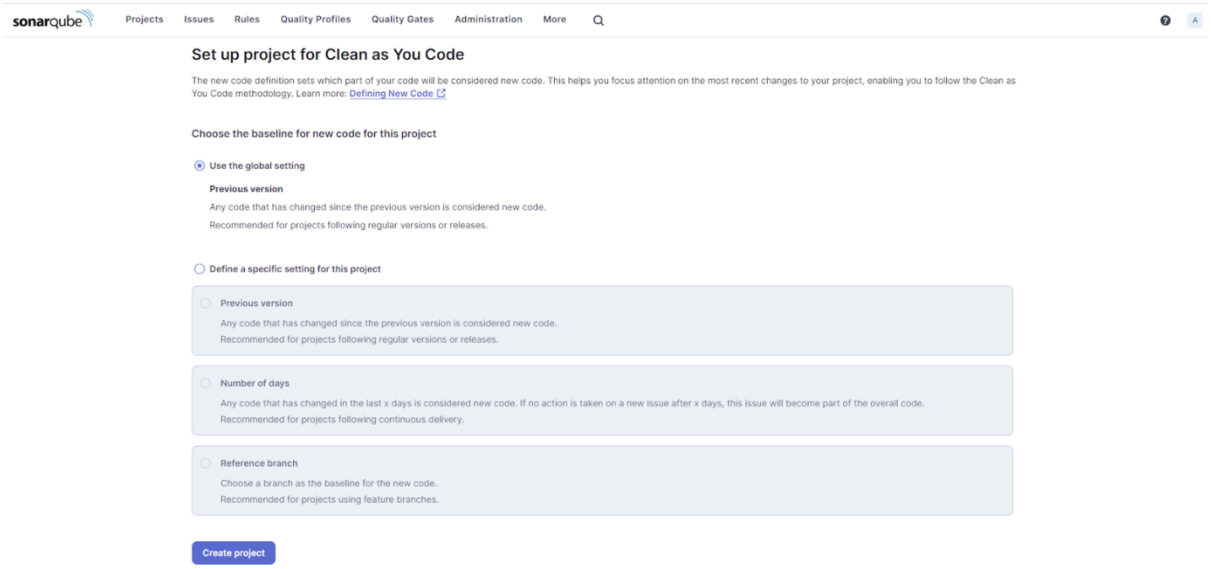
Installation is successful if you see this page
Update to new password



Create project manually:

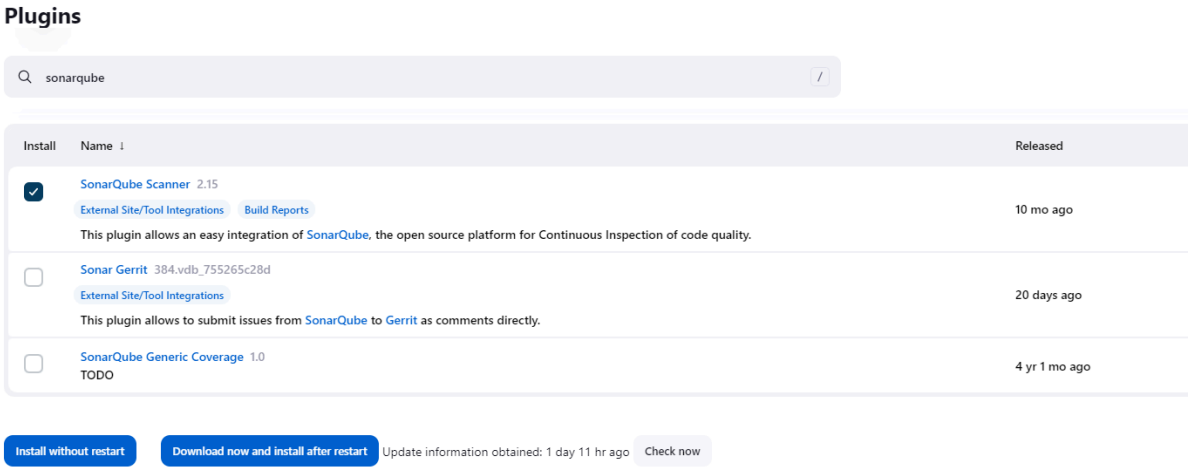
Here project name is “AdDevops”

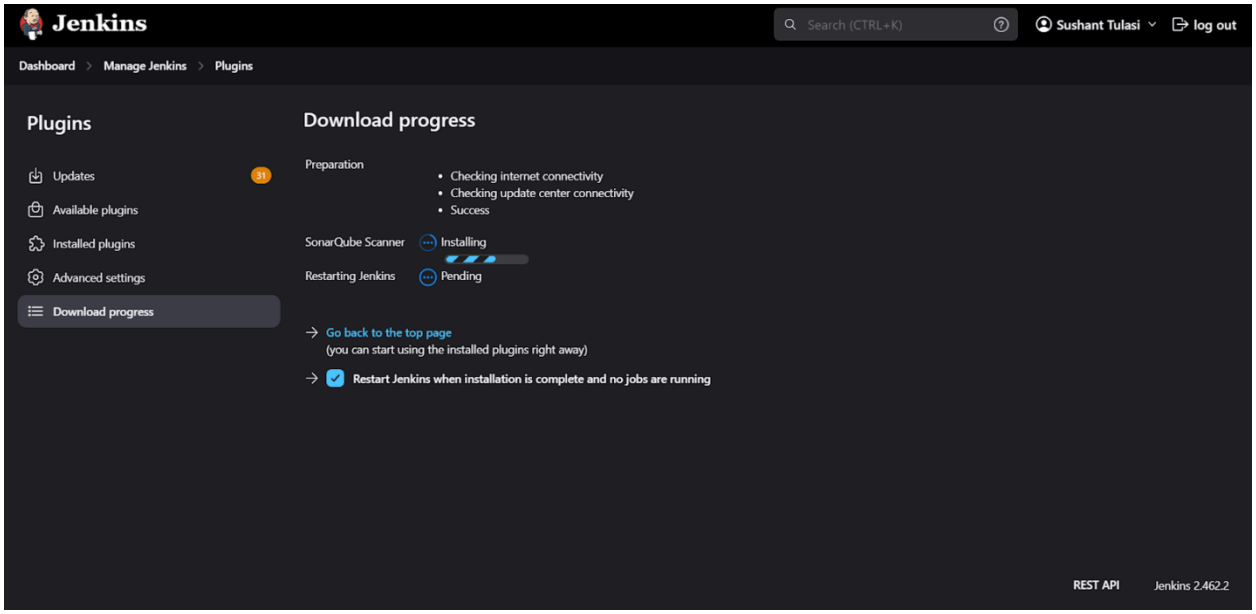




open Jenkins

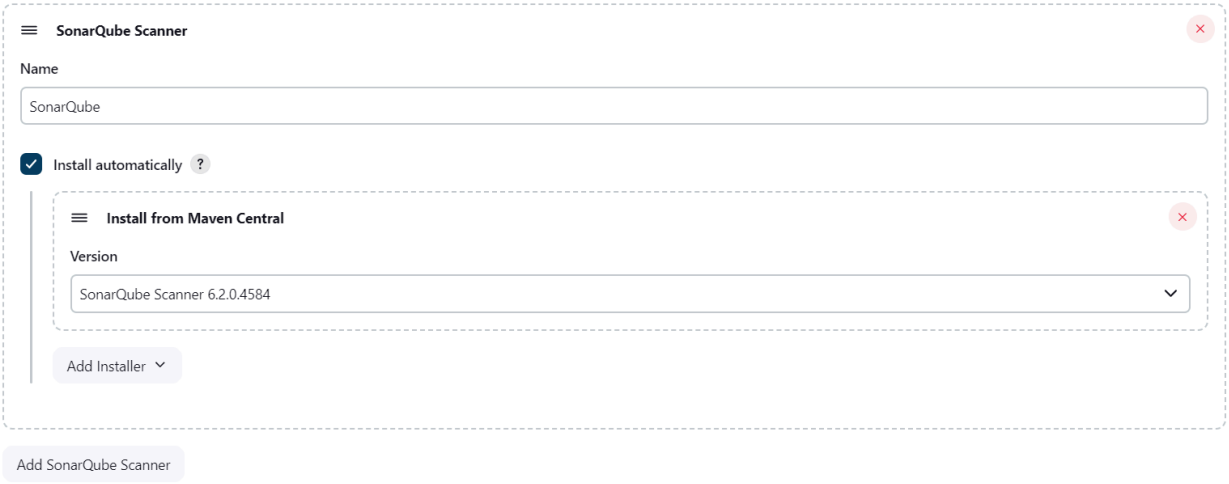
Go to Dashboard ->Manage Jenkins -> Plugin Manager and search for SonarQube Scanner under Available plugins for Jenkins and install without restart.



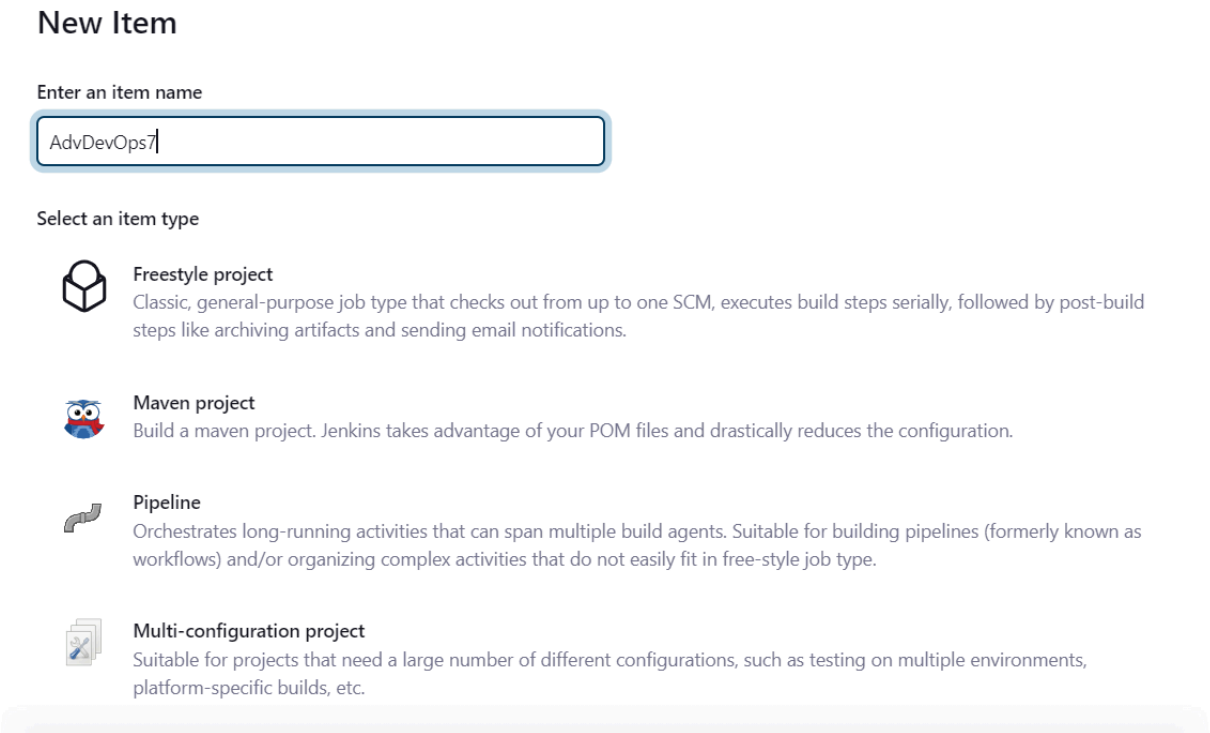


Under Jenkins ,
Dashboard -> Manage Jenkins -> Configure System ,
Look for SonarQube Servers and enter the details. Enter the Server Authentication Token if needed.

Search SonarQube Scanner under Dashboard -> Manage Jenkins -> Global Tool Configuration.
Choose the latest configuration and choose Install Automatically.

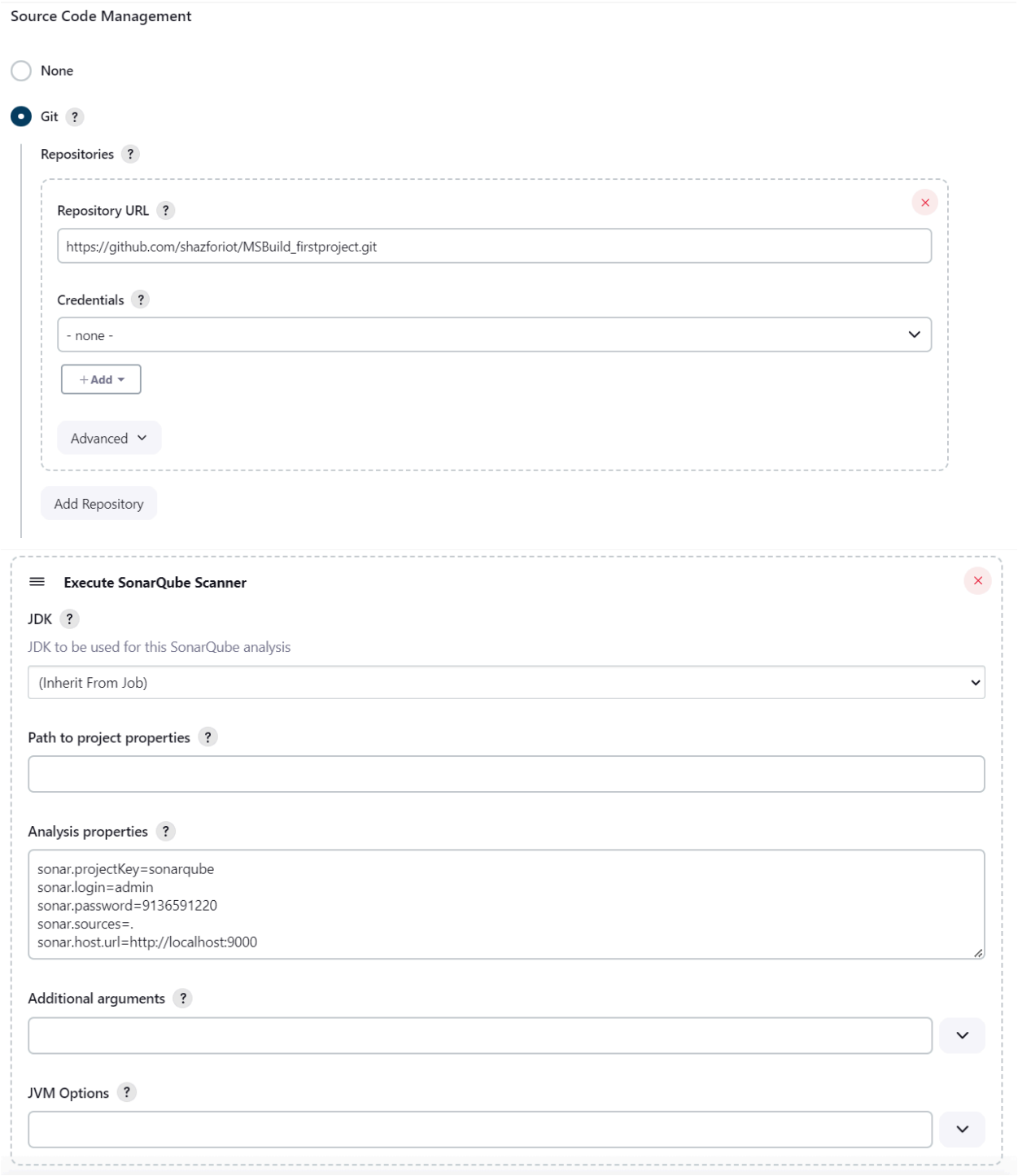


create a New Item in Jenkins, choose a freestyle project.

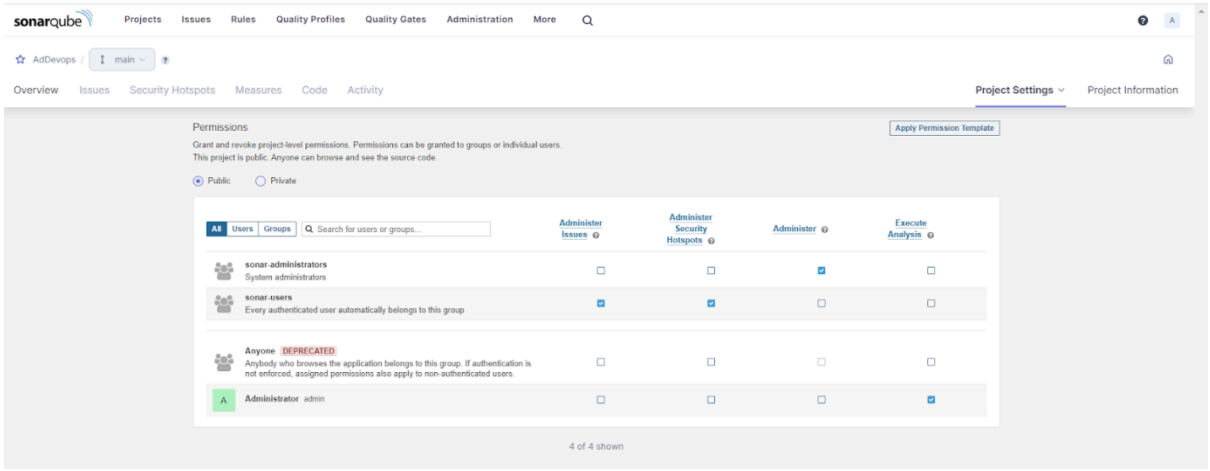


Choose this GitHub repository in Source Code Management.
https://github.com/shazforiot/MSBuild_firstproject.git
Under Build ->Execute SonarQube Scanner, enter these Analysis properties. Mention the SonarQube Project Key, Login, Password, and Host URL.
sonar.projectKey=AdDevops
sonar.login=admin
sonar.password=abc

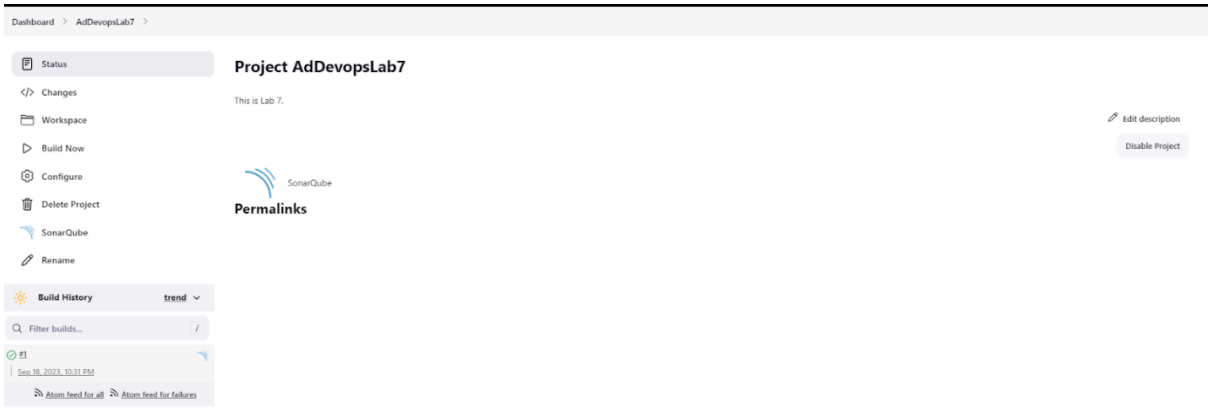
sonar.hosturl=http://localhost:9000/

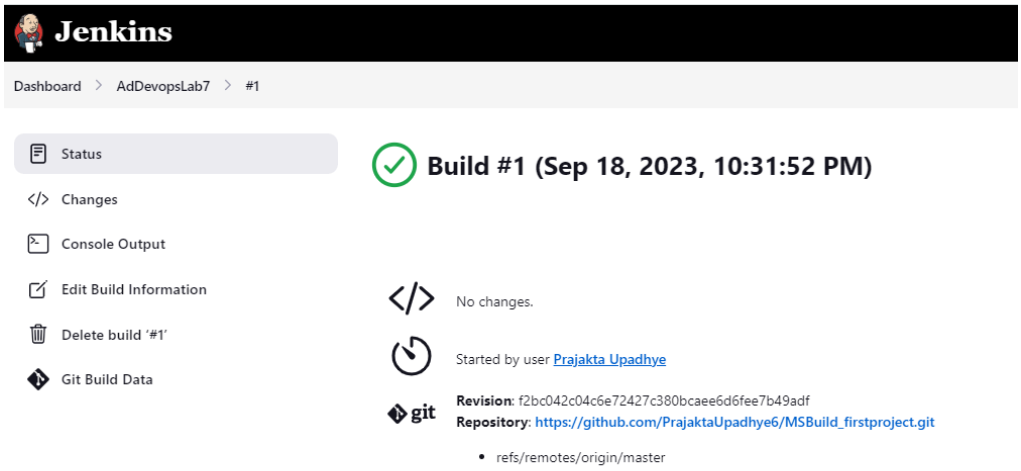


Go to http://localhost:9000/ and enter your previously created username.
Go to Permissions and grant the Admin user Execute Permissions.

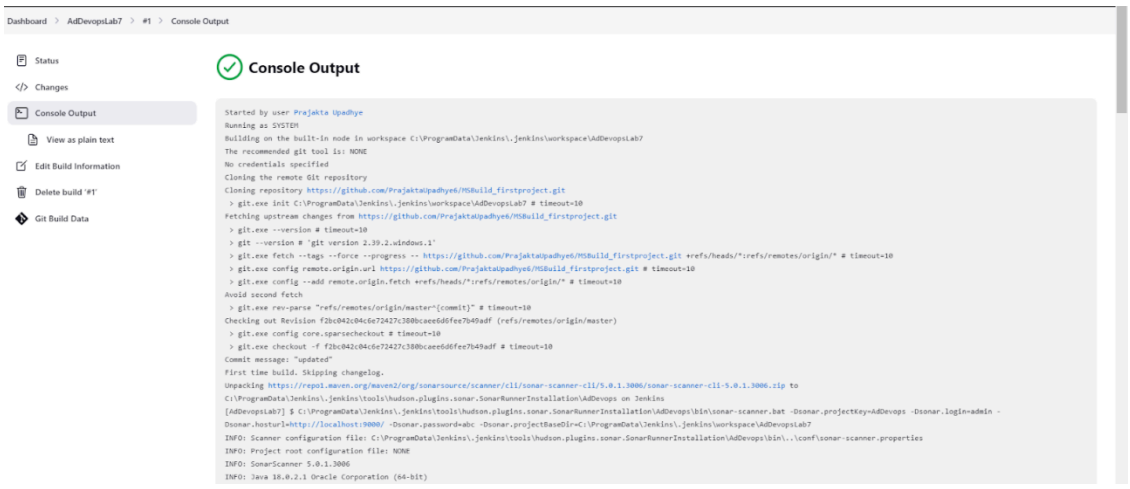


Build and Run:



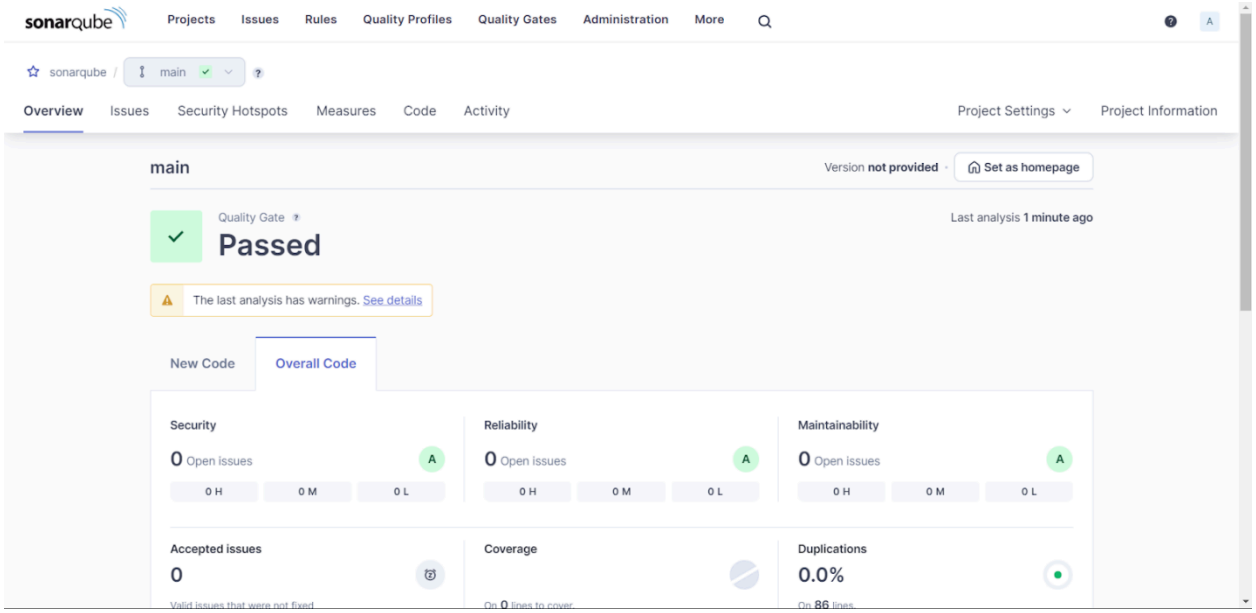


Console Output:




```
14:42:33.145 INFO  CPD Executor CPD calculation finished (done) | time=0ms
14:42:33.151 INFO  SCM revision ID 'f2bc042c04c6e72427c380bcaee6d6fee7b49adf'
14:42:33.466 INFO  Analysis report generated in 95ms, dir size=201.0 kB
14:42:33.546 INFO  Analysis report compressed in 54ms, zip size=22.4 kB
14:42:33.834 INFO  Analysis report uploaded in 285ms
14:42:33.847 INFO  ANALYSIS SUCCESSFUL, you can find the results at: http://localhost:9000/dashboard?id=sonarqube
14:42:33.847 INFO  Note that you will be able to access the updated dashboard once the server has processed the submitted analysis report
14:42:33.847 INFO  More about the report processing at http://localhost:9000/api/ce/task?id=7db13258-220d-47ae-9e4e-dcbe6e3771ea
14:42:33.862 INFO  Analysis total time: 27.320 s
14:42:33.863 INFO  SonarScanner Engine completed successfully
14:42:33.947 INFO  EXECUTION SUCCESS
14:42:33.949 INFO  Total time: 42.241s
Finished: SUCCESS
```

Project on sonarqube:



Conclusion: Thus, we have successfully installed SonarQube from Docker image.