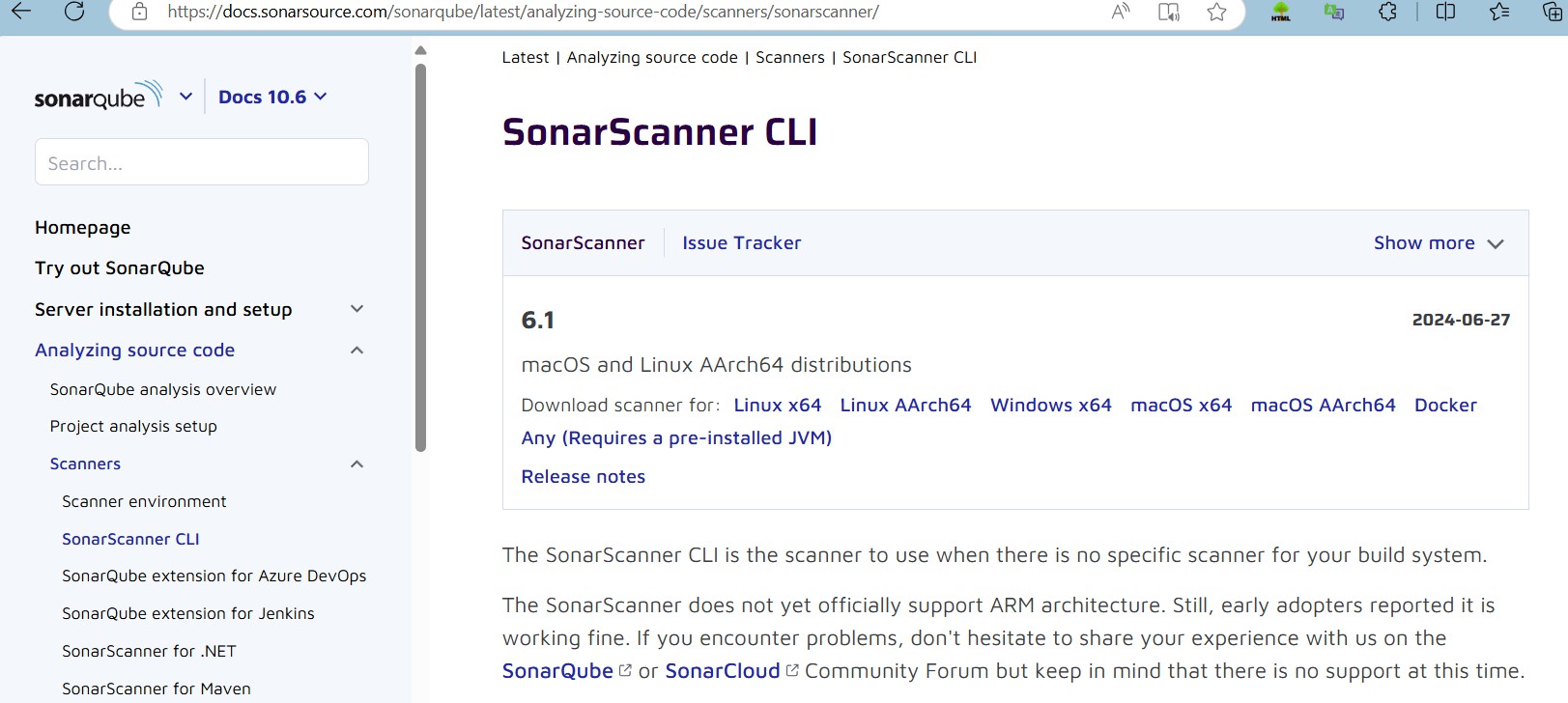
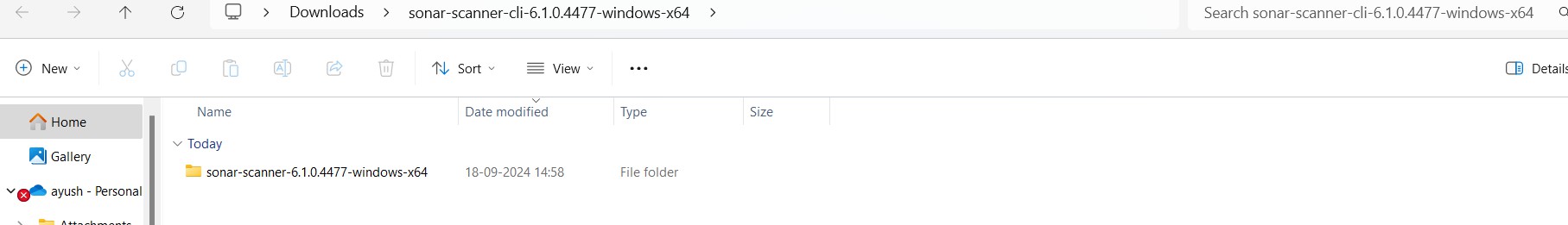
08 Advanced DevOps Lab

Aim: Create a Jenkins CICD Pipeline with SonarQube / GitLab Integration to perform a static analysis of the code to detect bugs, code smells, and security vulnerabilities on a sample Web / Java / Python application.

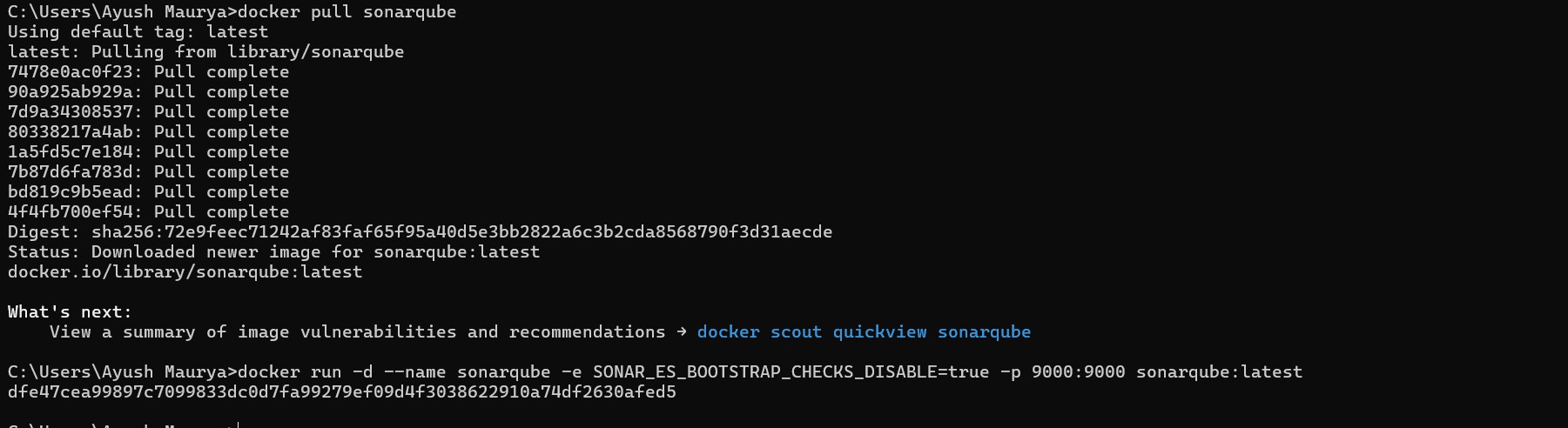
# Step 1: Download sonar scanner

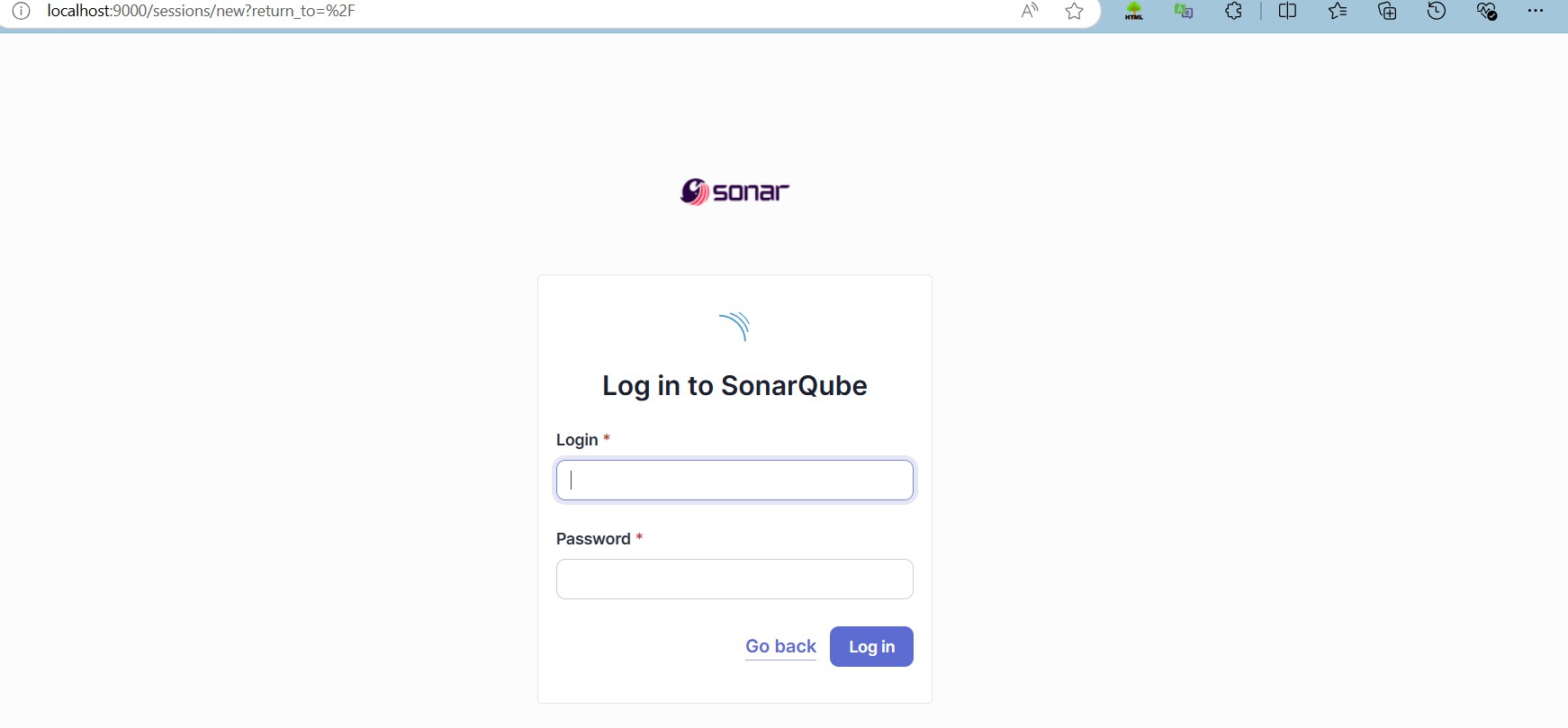
<https://docs.sonarsource.com/sonarqube/latest/analyzing-source-code/scanners/sonarscanner/> Visit this link and download the sonarqube scanner CLI.

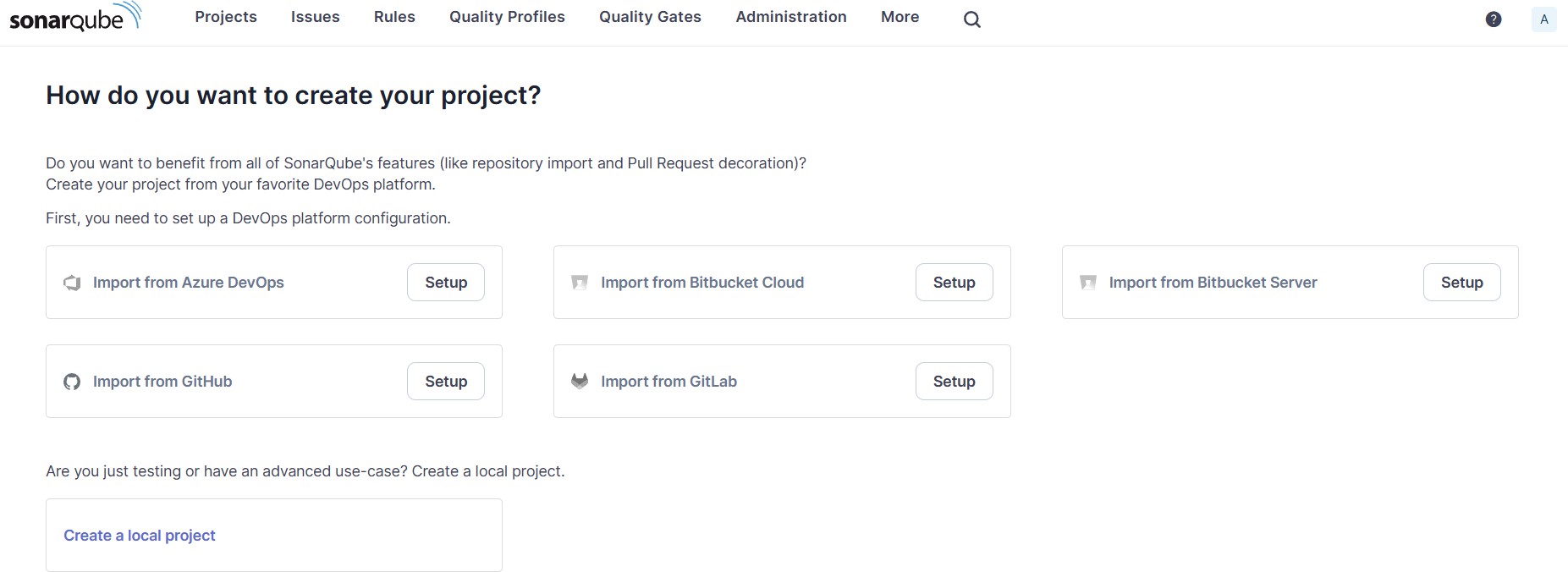
Extract the downloaded zip file in a folder.



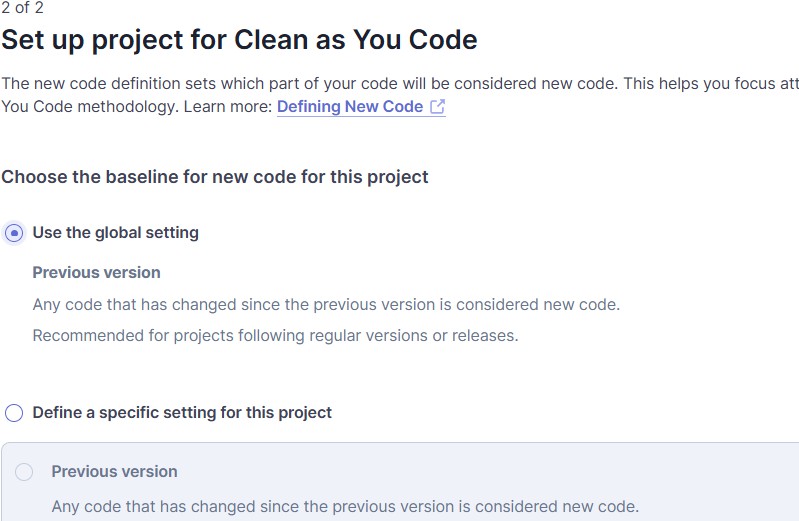
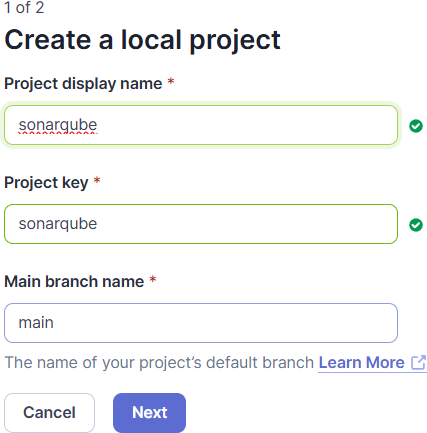
1. Install sonarqube image Command: **docker pull sonarqube**



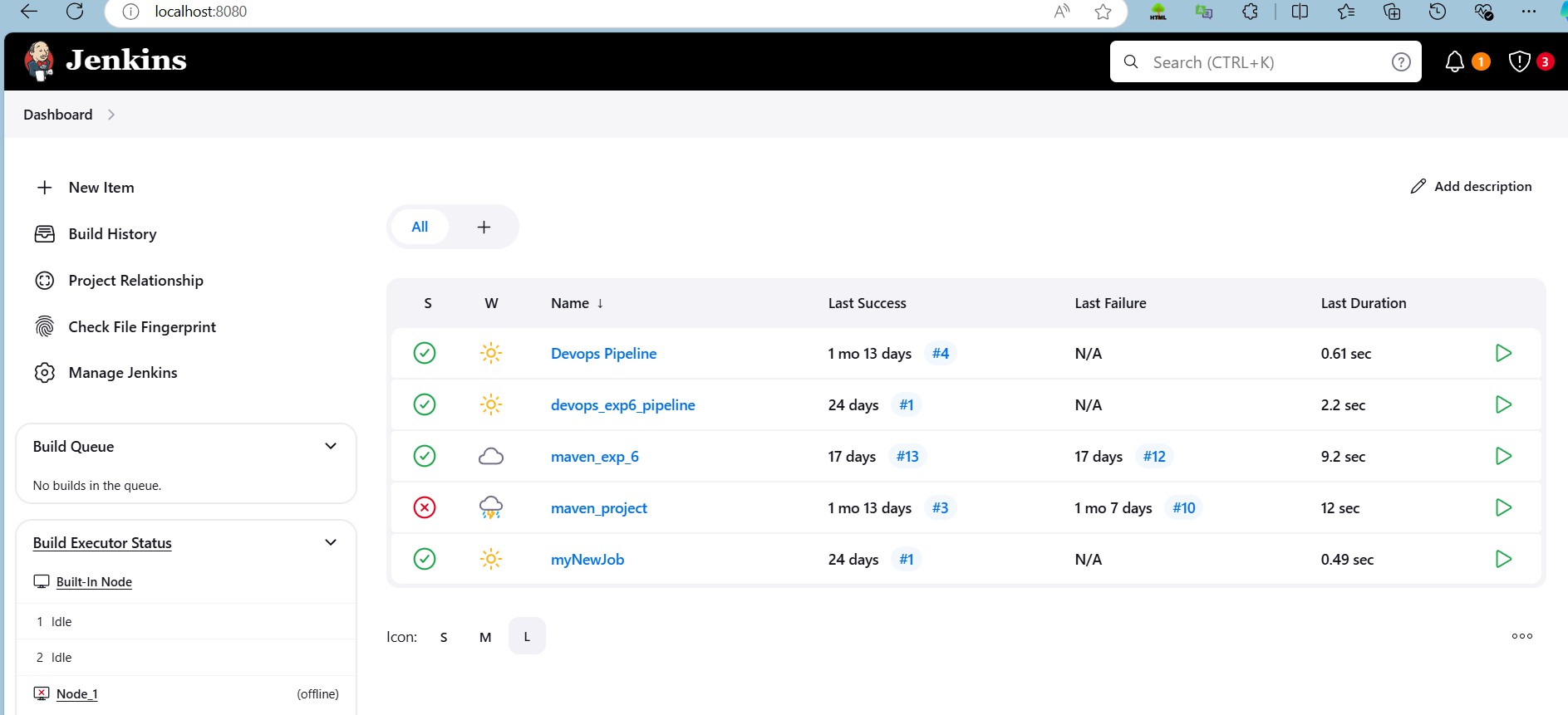
1. Once the container is up and running, you can check the status of SonarQube at localhost port 9000.
2. Login to SonarQube using username admin and password admin.



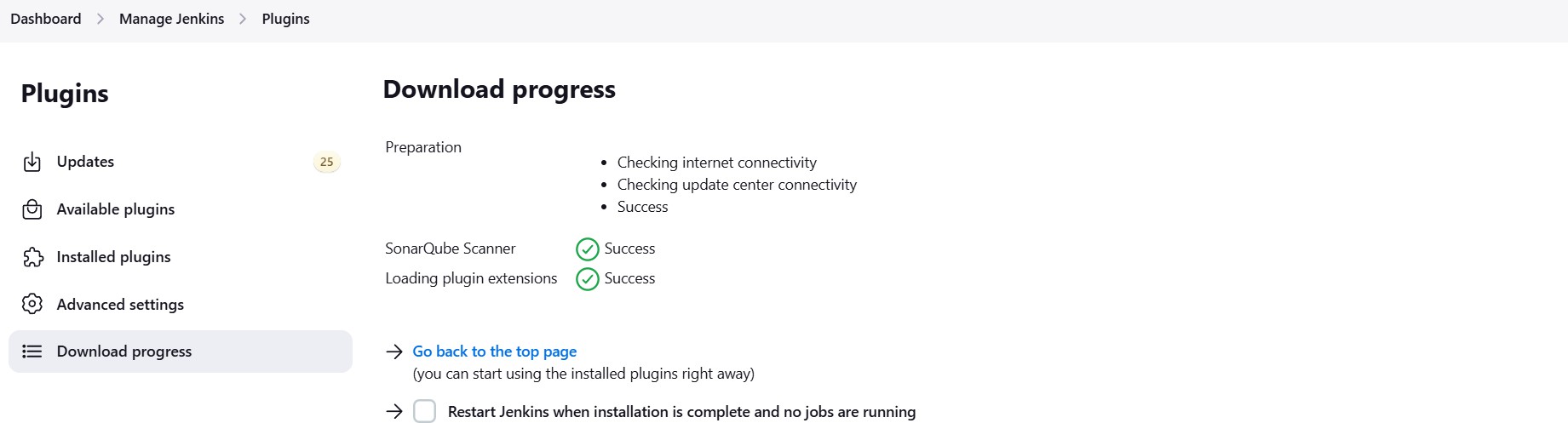
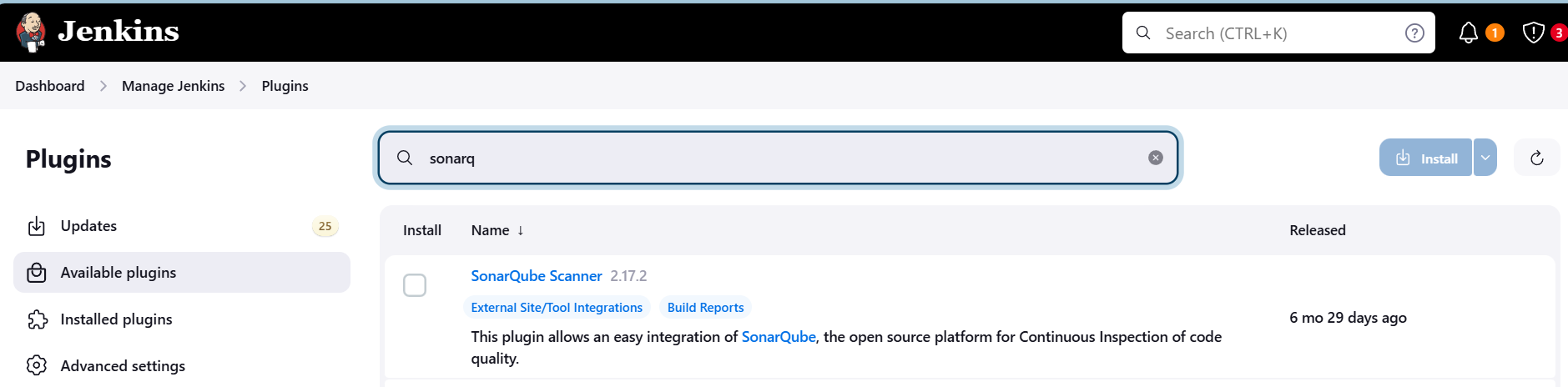
1. Create a manual project in SonarQube with the name sonarqube



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



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



1. Under Jenkins ‘Manage Jenkins’ then go to ‘system’, scroll and look for **SonarQube Servers**

and enter the details.

Enter the Server Authentication token if needed.

In SonarQube installations: Under **Name** add <project name of sonarqube> for me

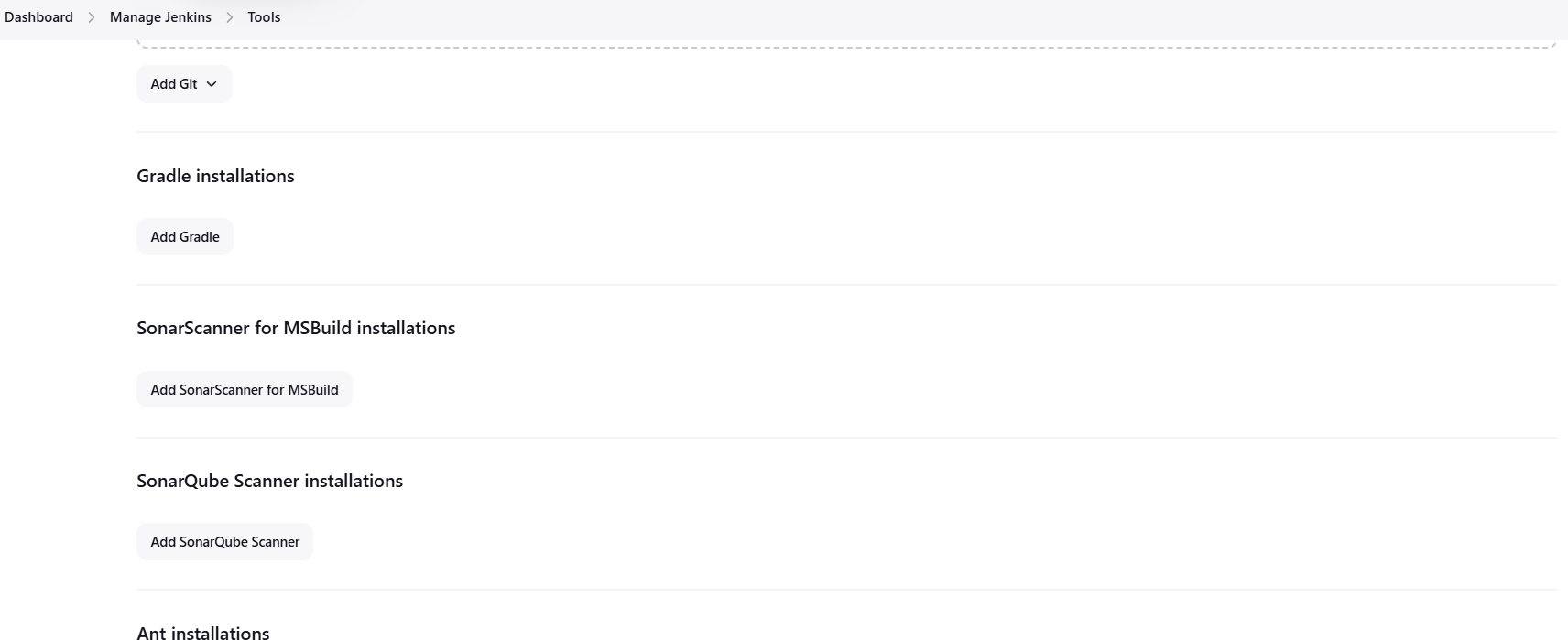
**adv\_devops\_7\_sonarqube**

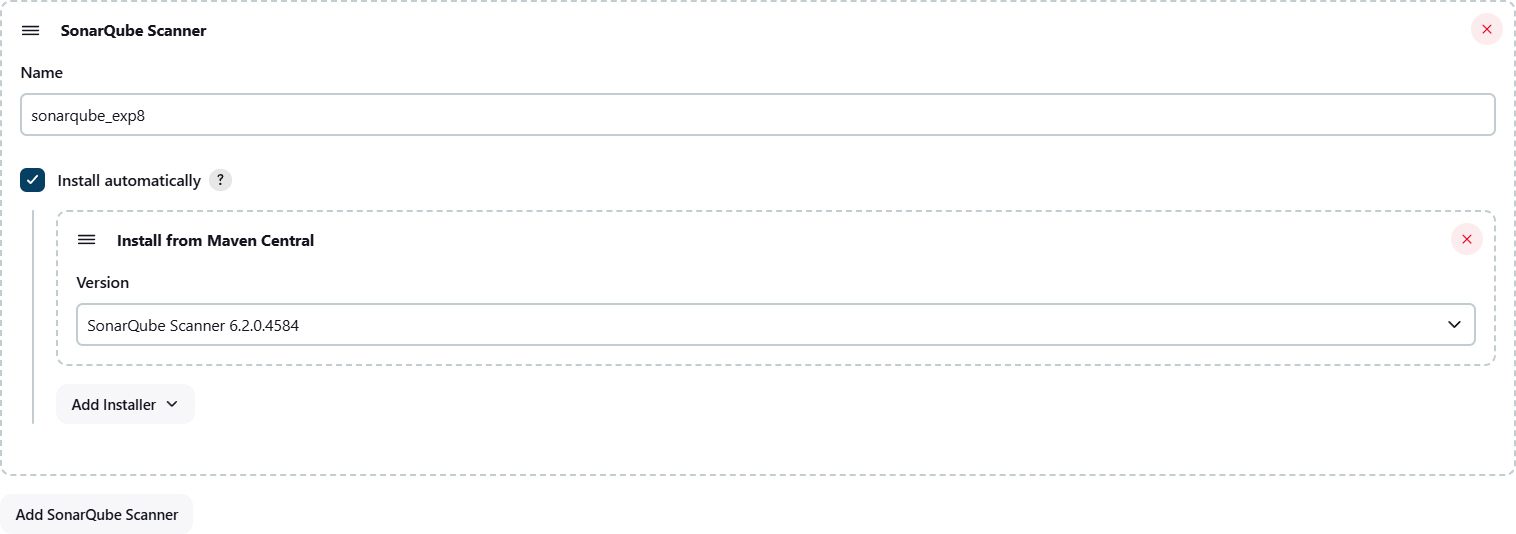
In **Server URL** Default is **http://localhost:9000**



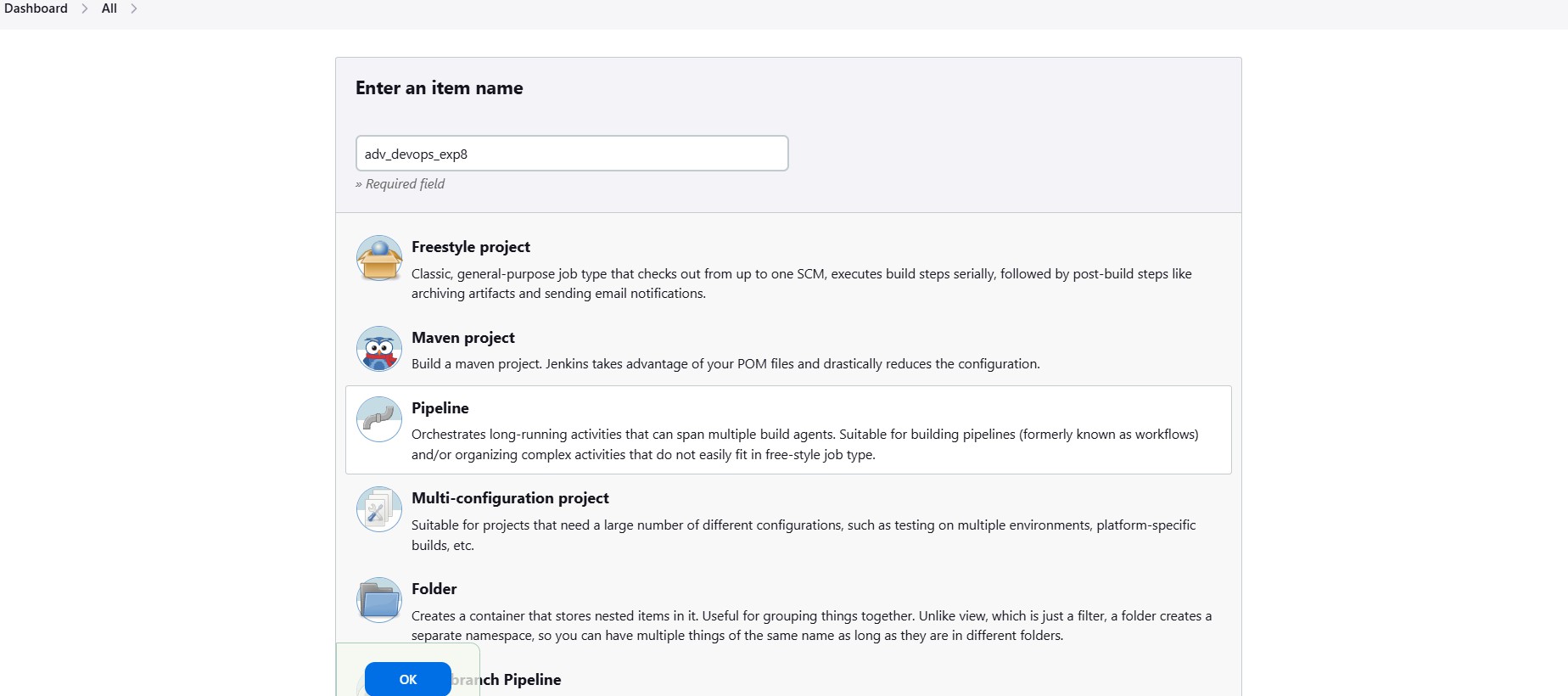
1. Search for SonarQube Scanner under Global Tool Configuration. Choose the latest configuration and choose Install automatically.

# Dashboard > Manage Jenkins > Tools



Check the “Install automatically” option. → Under name any name as identifier → Check the “Install automatically” option.

1. After configuration, create a New Item → choose a pipeline project.



1. Under Pipeline script, enter the following:

node {

stage('Cloning the GitHub Repo') {

git 'https://github.com/shazforiot/GOL.git'

}

stage('SonarQube analysis') { withSonarQubeEnv('<Name\_of\_SonarQube\_environment\_on\_Jenkins>') {

sh """

<PATH\_TO\_SONARQUBE\_SCANNER\_FOLDER>/bin/sonar-scanner \

-D sonar.login=<SonarQube\_USERNAME> \

-D sonar.password=<SonarQube\_PASSWORD> \

-D sonar.projectKey=<Project\_KEY> \

-D sonar.exclusions=vendor/\*\*,resources/\*\*,\*\*/\*.java \

-D sonar.host.url=<SonarQube\_URL>(default: http://localhost:9000/)

"""

}

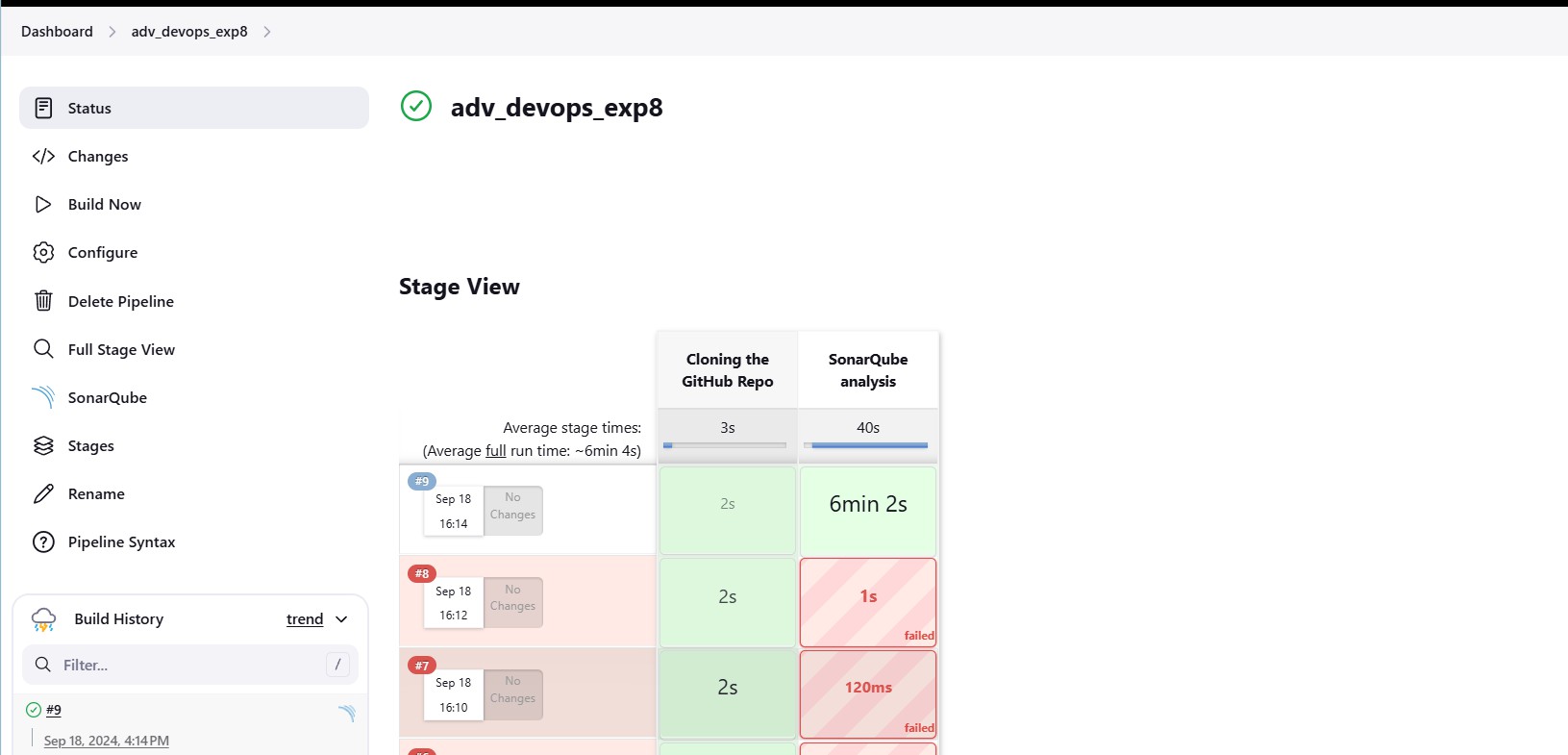
}

}

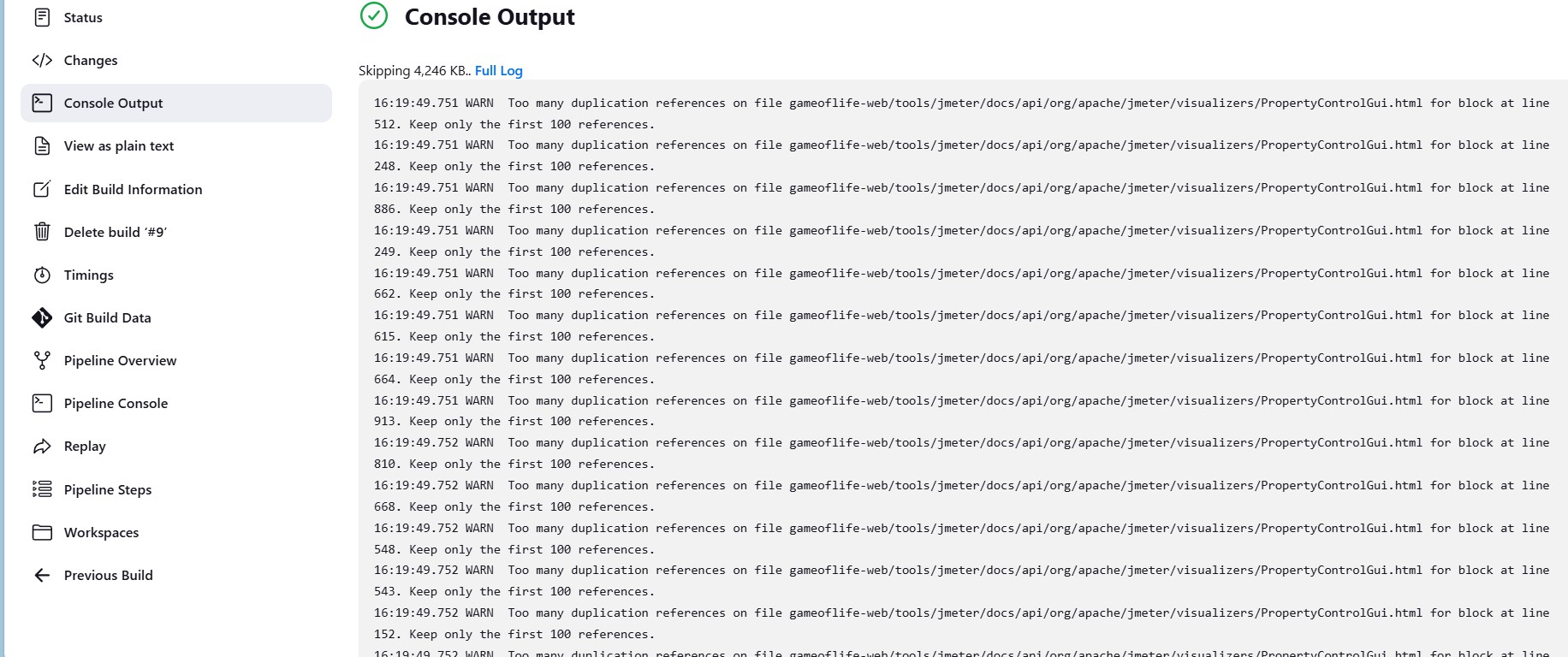
It is a java sample project which has a lot of repetitions and issues that will be detected by SonarQube.



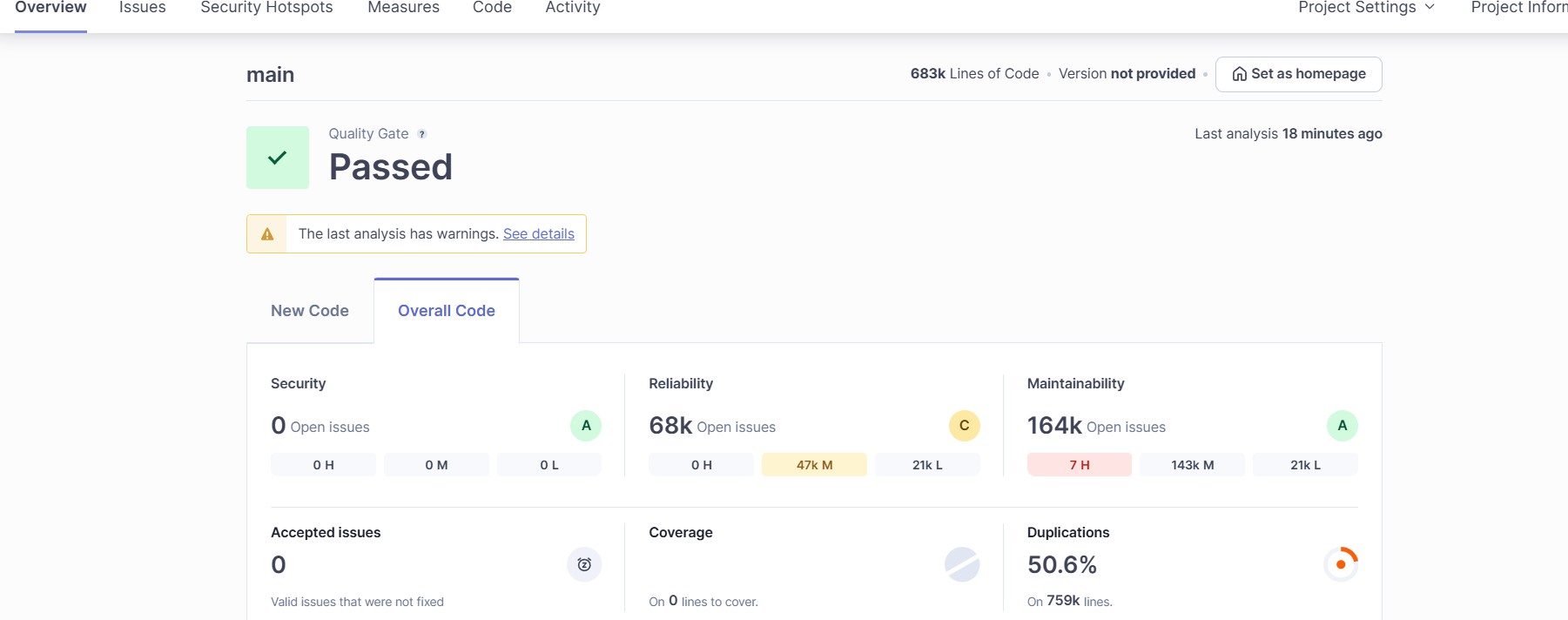
1. Build project



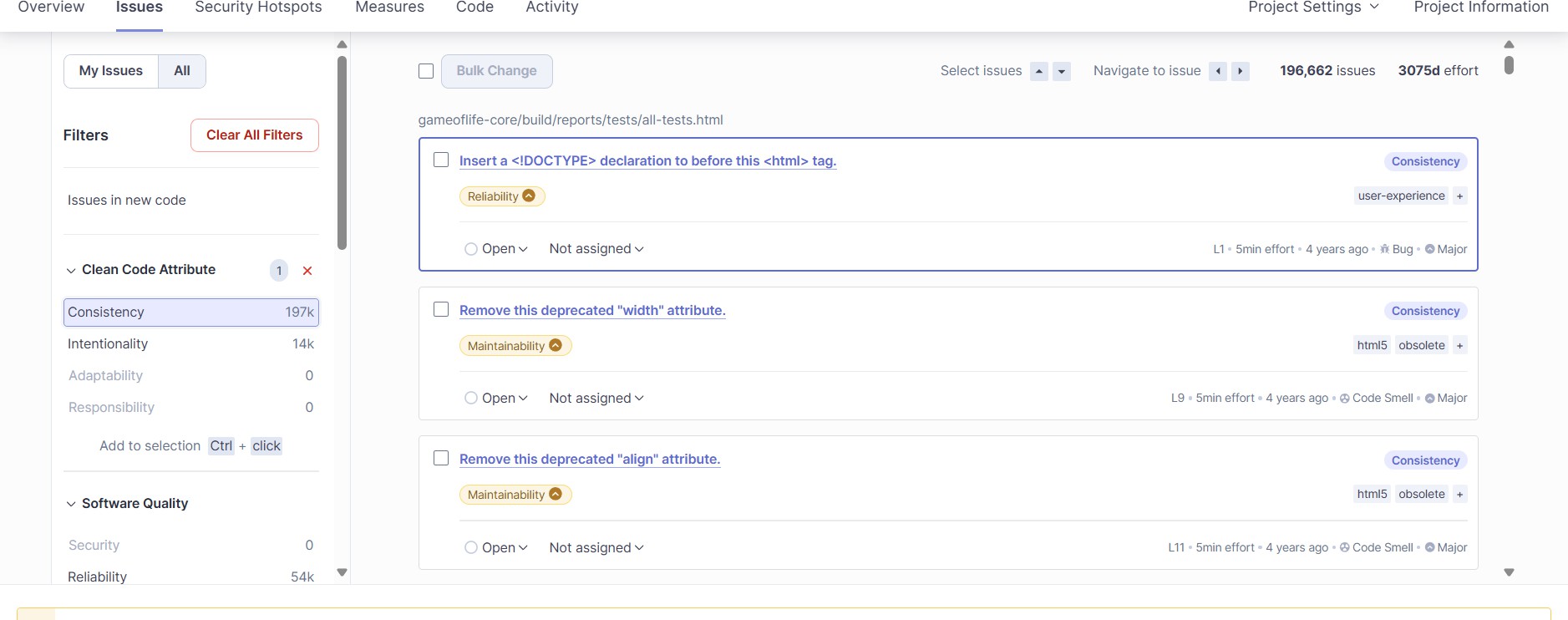
1. Check console



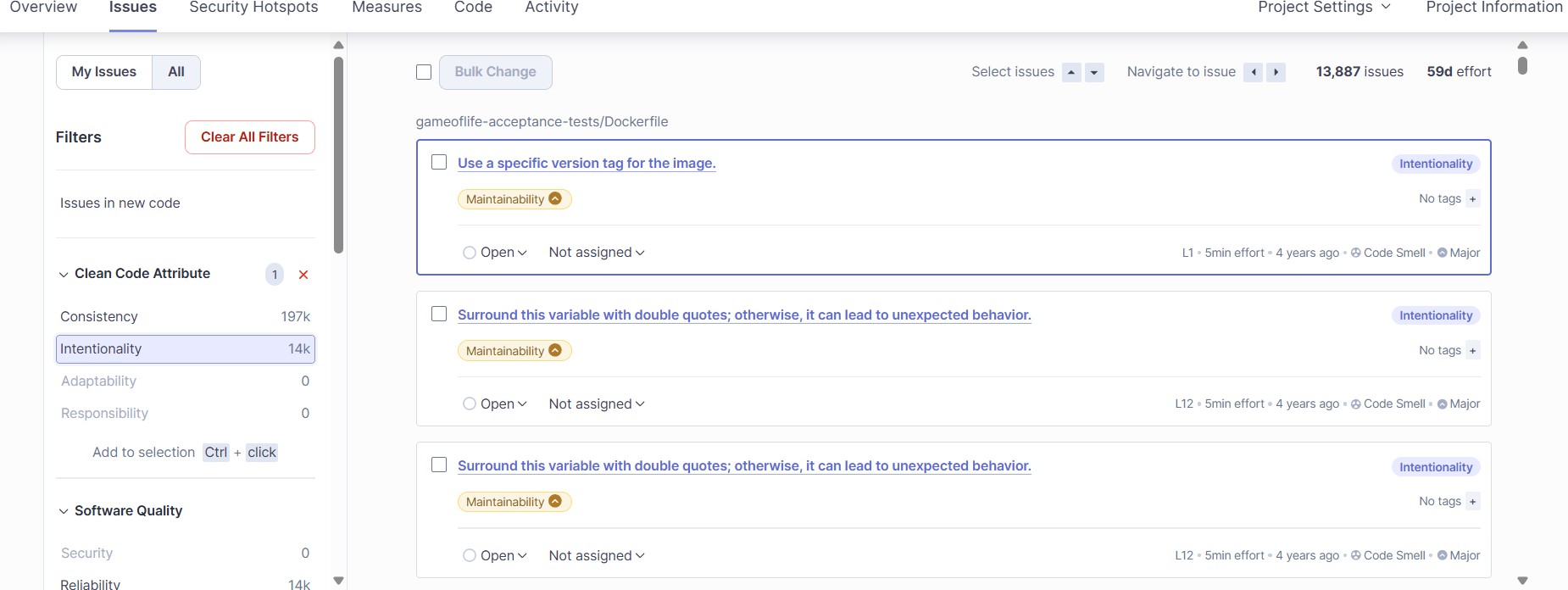
1. Now, check the project in SonarQube



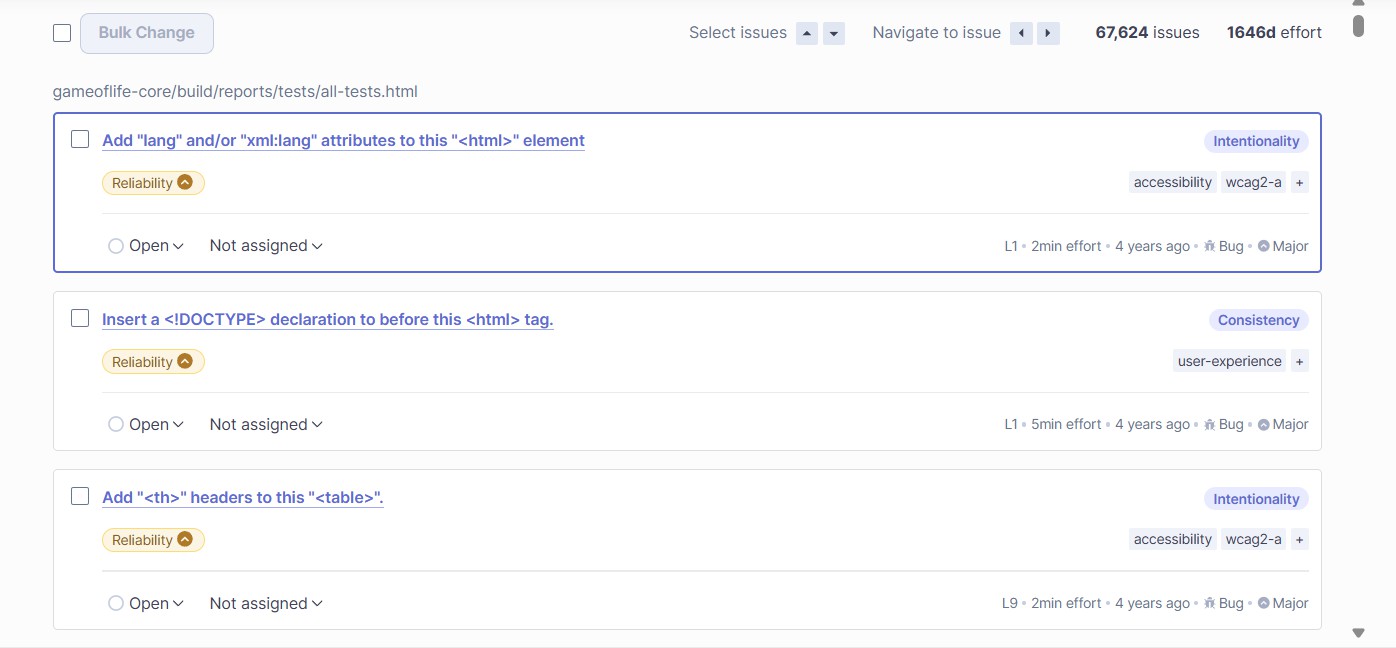
1. Code Problems
   * Consistency



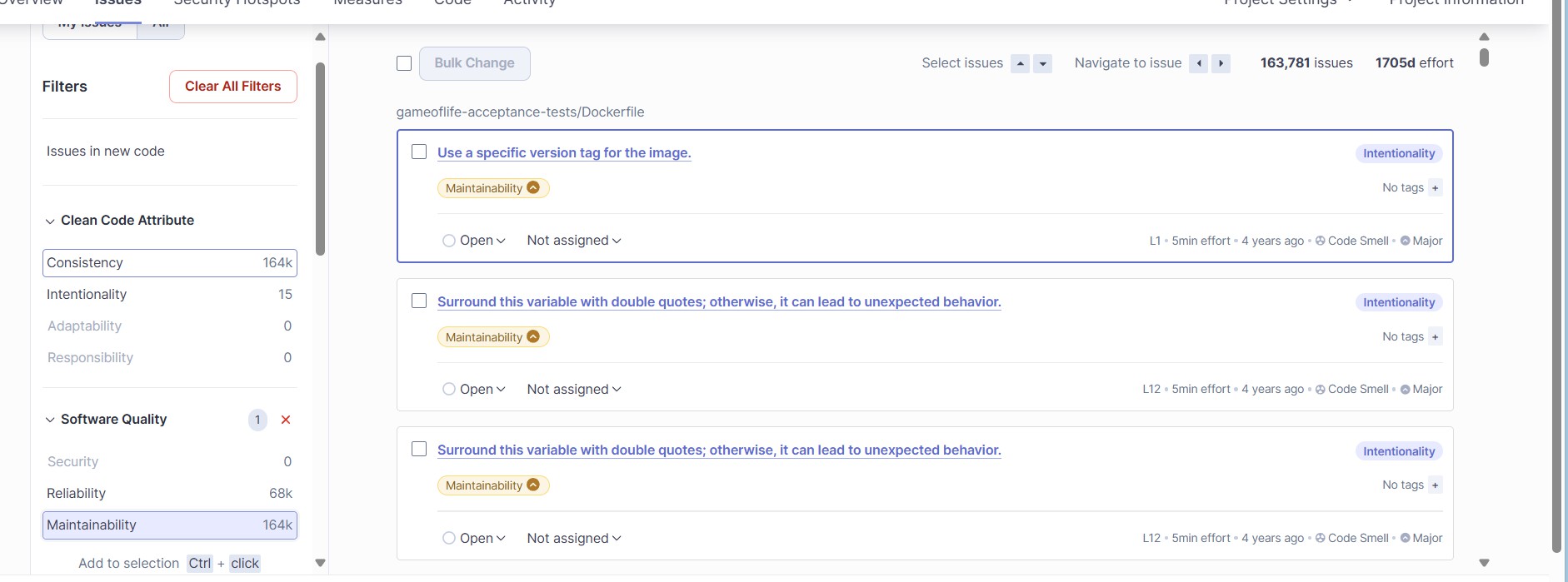
* + Intentionality



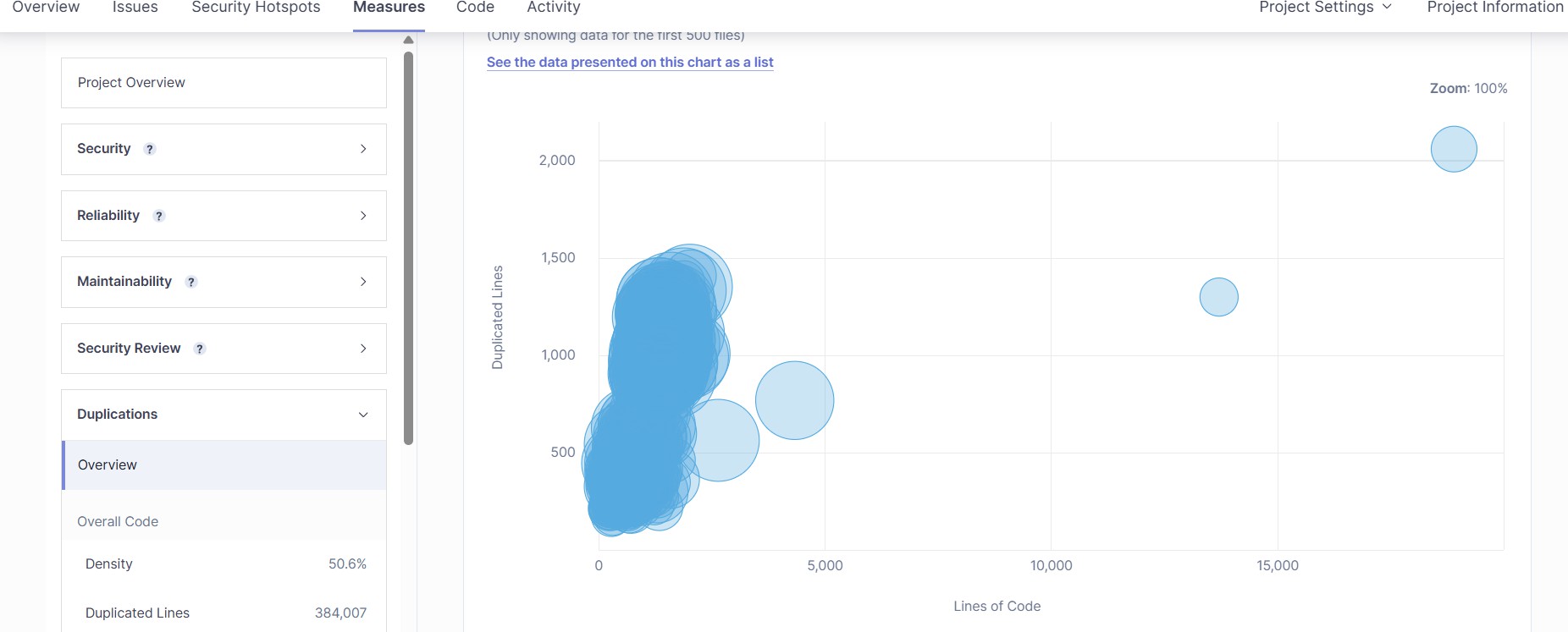
* + Bugs



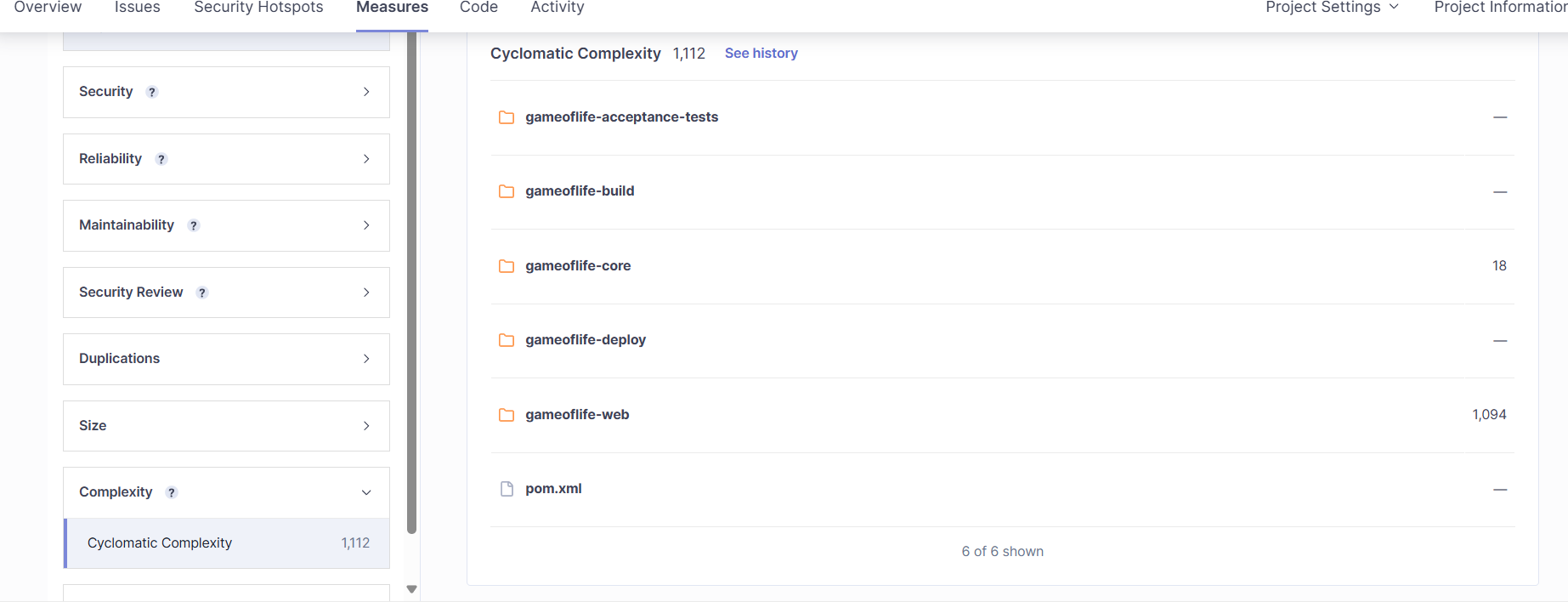
* + Code Smells



* + Duplications



* + Cyclomatic Complexities



In this way, we have integrated Jenkins with SonarQube for SAST.

# Conclusion:

In this experiment, we integrated Jenkins with SonarQube to enable automated code quality checks within our CI/CD pipeline. We started by deploying SonarQube using Docker, setting up a project, and configuring it to analyze code quality. Next, we configured Jenkins by installing the SonarQube Scanner plugin, adding SonarQube server details, and setting up the scanner tool. We then developed a Jenkins pipeline to automate the process of cloning a GitHub repository and running SonarQube analysis on the code. This integration helps ensure continuous monitoring of code quality, detecting issues such as bugs, code smells, and security vulnerabilities throughout the development process.