

Experiment No. 8

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.

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.

What is a CI/CD Pipeline?

CI/CD pipeline refers to the Continuous Integration/Continuous Delivery pipeline. Before we dive deep into this segment, let's first understand what is meant by the term 'pipeline'?

A pipeline is a concept that introduces a series of events or tasks that are connected in a sequence to make quick software releases. For example, there is a task, that task has got five different stages, and each stage has got some steps. All the steps in phase one have to be completed, to mark the latter stage to be complete.



Now, consider the CI/CD pipeline as the backbone of the DevOps approach. This Pipeline is responsible for building codes, running tests, and deploying new software versions. The Pipeline executes the job in a defined manner by first coding it and then structuring it inside several blocks that may include several steps or tasks.

What is SonarQube?

SonarQube is an open-source platform developed by SonarSource for continuous inspection of code quality. Sonar does static code analysis, which provides a detailed report of bugs, code smells, vulnerabilities, code duplications. It supports 25+ major programming languages through built-in rulesets and can also be extended with various plugins.

Benefits of SonarQube

- **Sustainability** - Reduces complexity, possible vulnerabilities, and code duplications, optimising the life of applications.
- **Increase productivity** - Reduces the scale, cost of maintenance, and risk of the application; as such, it removes the need to spend more time changing the code
- **Quality code** - Code quality control is an inseparable part of the process of software development.
- **Detect Errors** - Detects errors in the code and alerts developers to fix them automatically before submitting them for output.
- **Increase consistency** - Determines where the code criteria are breached and enhances the quality
- **Business scaling** - No restriction on the number of projects to be evaluated
- **Enhance developer skills** - Regular feedback on quality problems helps developers to improve their coding skills

Integrating Jenkins with SonarQube:

Prerequisites:

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

Steps to create a Jenkins CI/CD Pipeline and use SonarQube to perform SAST

1. Download sonar scanner and extract the file.

<https://docs.sonarsource.com/sonarqube/latest/analyzing-source-code/scanners/sonarscanner/>

Latest | Analyzing source code | Scanners | SonarScanner CLI

SonarScanner CLI

SonarScanner	Issue Tracker	Show more ▾
6.2		2024-09-17
Support PKCS12 truststore generated with OpenSSL		
Download scanner for: Linux x64 Linux AArch64 Windows x64 macOS x64 macOS AArch64 Docker		
Any (Requires a pre-installed JVM)		
Release notes		

2. Install sonarqube image

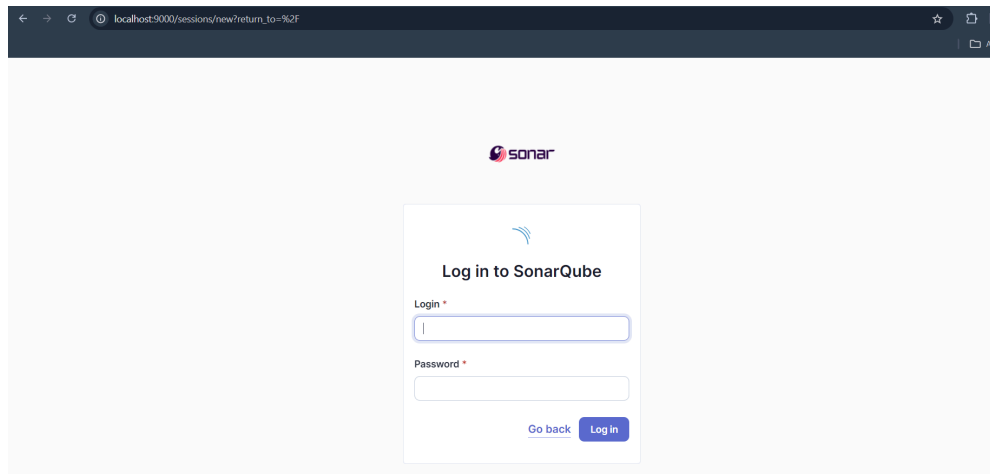
Command:

docker pull sonarqube (no need to do it again as we already did it in Exp 7)

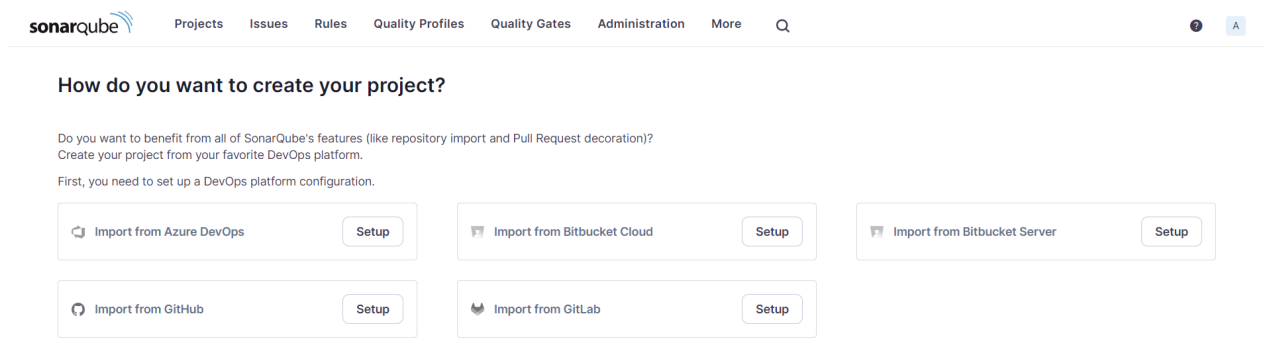
Then run the image by running this command

docker run -d -p 9000:9000 sonarqube

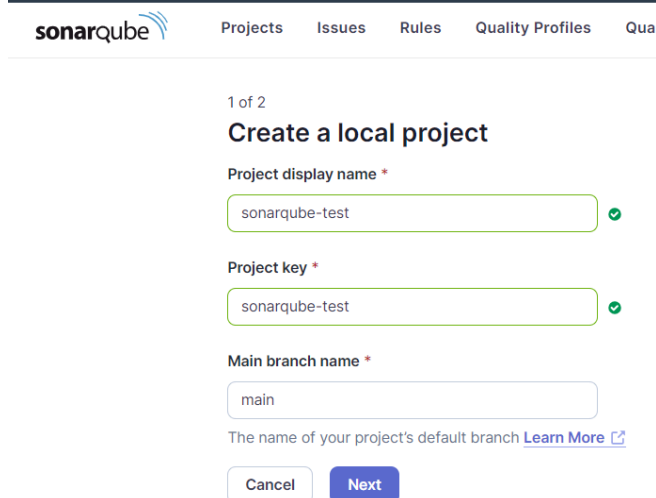
3. Once the container is up and running, you can check the status of SonarQube at localhost port 9000.



4. Login to SonarQube using username and password.



5. Create a manual project in SonarQube with the name **sonarqube-test**



Setup the project and come back to Jenkins Dashboard.

6. Create a New Item in Jenkins, choose **Pipeline**.

Enter an item name

sonarqubeExp8

» Required field

Freestyle project
Classic, general-purpose job type that checks out from up to one SCM, executes build steps serially, archiving artifacts and sending email notifications.

Maven project
Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the need for build steps.

Pipeline
Orchestrates long-running activities that can span multiple build agents. Suitable for organizing complex activities that do not easily fit in free-style job types.

Multi-configuration project
Suitable for projects that need a large number of different configurations, such as different build environments, builds, etc.

Folder
Creates a container that stores nested items in it. Useful for grouping things together in the same namespace, so you can have multiple things of the same name as long as they are in different folders.

OK

7. Under Pipeline Script, enter the following -

```
node {
  stage('Cloning the GitHub Repo') {
    git 'https://github.com/shazforiot/GOL.git'
  }
  stage('SonarQube analysis') {
    withSonarQubeEnv('sonarqube') {
      sh "<PATH_TO_SONARQUBE_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=http://127.0.0.1:9000/"
    }
  }
}
```

```
}  
}
```

Pipeline

Definition

Pipeline script

Script ?

```
1 node {  
2   stage('Cloning the GitHub Repo') {  
3     git 'https://github.com/shazforiot/GOL.git'  
4   }  
5  
6   stage('SonarQube Analysis') {  
7     withSonarQubeEnv('sonarqubeExp7') {  
8       bat ""  
9       "C:\\Users\\akank\\Downloads\\sonar-scanner-6.2.0.4584-windows-x64\\bin\\sonar-scanner.bat" ^  
10      -Dsonar.login=admin ^  
11      -Dsonar.password=admin123 ^  
12      -Dsonar.projectKey=sonarqube-test ^  
13      -Dsonar.exclusions=vendor/**,resources/**/*.java ^  
14      -Dsonar.host.url=http://localhost:9000  
15      ""  
16    }  
17  }  
18 }
```

☒ Use Groovy Sandbox ?

[Pipeline Syntax](#)


Save

Apply

Above put your login, password, projectKey just like did it in exp7.

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

8. Run The Build.

 **Jenkins**

Dashboard > sonarQubeExp8 >

Status

</> Changes

▶ Build Now

⚙️ Configure

🗑️ Delete Pipeline


📁 Stages

✎ Rename

❓ Pipeline Syntax



sonarQubeExp8

Permalinks


 Build History trend ▾

🟢 #8

Sep 26, 2024, 8:59 PM

 [Atom feed for all](#)  [Atom feed for failures](#)

9. Check the console output once the build is complete.

 **Jenkins**

Search (CTRL+K) 🔍

🔔 🟡 🛡️ 🚫 akanksha ▾ log out

Dashboard > sonarQubeExp8 > #8

Status

</> Changes

📄 Console Output

📄 View as plain text

✔ Edit Build Information

🗑️ Delete build '#8'

🕒 Timings

💎 Git Build Data

🔗 Pipeline Overview

📄 Pipeline Console

↻ Replay

📋 Pipeline Steps

📁 Workspaces

🟢 Console Output

Skipping 4,248 KB. [Full Log](#)

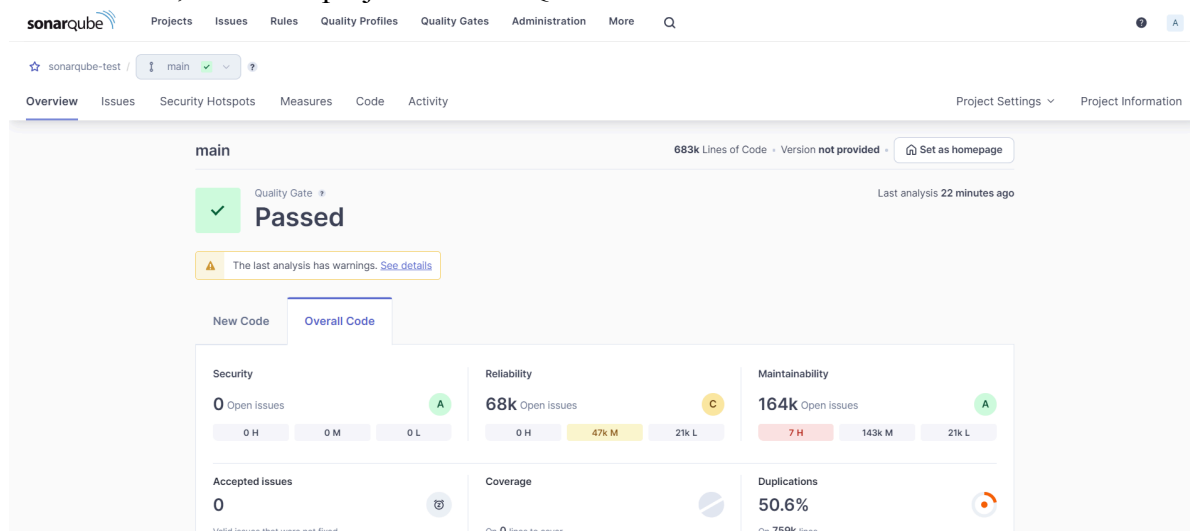
```
21:11:26.170 WARN Too many duplication references on file gameoflife-web/tools/jmeter/docs/api/org/apache/jmeter/protocol/http/gui/HTTPArgumentsPanel.html for block at line 798. Keep only the first 100 references.
21:11:26.170 WARN Too many duplication references on file gameoflife-web/tools/jmeter/docs/api/org/apache/jmeter/protocol/http/gui/HTTPArgumentsPanel.html for block at line 810. Keep only the first 100 references.
21:11:26.170 WARN Too many duplication references on file gameoflife-web/tools/jmeter/docs/api/org/apache/jmeter/protocol/http/gui/HTTPArgumentsPanel.html for block at line 823. Keep only the first 100 references.
21:11:26.170 WARN Too many duplication references on file gameoflife-web/tools/jmeter/docs/api/org/apache/jmeter/protocol/http/gui/HTTPArgumentsPanel.html for block at line 844. Keep only the first 100 references.
21:11:26.171 WARN Too many duplication references on file gameoflife-web/tools/jmeter/docs/api/org/apache/jmeter/protocol/http/gui/HTTPArgumentsPanel.html for block at line 589. Keep only the first 100 references.
21:11:26.171 WARN Too many duplication references on file gameoflife-web/tools/jmeter/docs/api/org/apache/jmeter/protocol/http/gui/HTTPArgumentsPanel.html for block at line 1065. Keep only the first 100 references.
21:11:26.171 WARN Too many duplication references on file gameoflife-web/tools/jmeter/docs/api/org/apache/jmeter/protocol/http/gui/HTTPArgumentsPanel.html for block at line 776. Keep only the first 100 references.
21:11:26.171 WARN Too many duplication references on file gameoflife-web/tools/jmeter/docs/api/org/apache/jmeter/protocol/http/gui/HTTPArgumentsPanel.html for block at line 778. Keep only the first 100 references.
21:11:26.171 WARN Too many duplication references on file gameoflife-web/tools/jmeter/docs/api/org/apache/jmeter/protocol/http/gui/HTTPArgumentsPanel.html for block at line 530. Keep only the first 100 references.
21:11:26.171 WARN Too many duplication references on file gameoflife-web/tools/jmeter/docs/api/org/apache/jmeter/protocol/http/gui/HTTPArgumentsPanel.html for block at line 648. Keep only the first 100 references.
21:11:26.171 WARN Too many duplication references on file gameoflife-web/tools/jmeter/docs/api/org/apache/jmeter/protocol/http/gui/HTTPArgumentsPanel.html for block at line 798. Keep only the first 100 references.
```



```
Dashboard > sonarQubeExp8 > #8

Keep only the first 100 references.
21:11:31.085 WARN Too many duplication references on file gameoflife-web/tools/jmeter/docs/api/org/apac
Keep only the first 100 references.
21:11:31.085 WARN Too many duplication references on file gameoflife-web/tools/jmeter/docs/api/org/apac
Keep only the first 100 references.
21:11:31.085 WARN Too many duplication references on file gameoflife-web/tools/jmeter/docs/api/org/apac
Keep only the first 100 references.
21:11:31.089 INFO CPD Executor CPD calculation finished (done) | time=158006ms
21:11:31.233 INFO SCM revision ID 'ba799ba7e1b576f04a4612322b0412c5e6e1e5e4'
21:13:39.419 INFO Analysis report generated in 5815ms, dir size=127.2 MB
21:14:00.338 INFO Analysis report compressed in 20896ms, zip size=29.6 MB
21:14:03.259 INFO Analysis report uploaded in 2917ms
21:14:03.266 INFO ANALYSIS SUCCESSFUL, you can find the results at: http://localhost:9000/dashboard?id=sonarqube-test
21:14:03.266 INFO Note that you will be able to access the updated dashboard once the server has proces
21:14:03.266 INFO More about the report processing at http://localhost:9000/api/ce/task?id=8c5f9165-d76
21:14:19.391 INFO Analysis total time: 14:16.345 s
21:14:19.411 INFO SonarScanner Engine completed successfully
21:14:20.182 INFO EXECUTION SUCCESS
21:14:20.245 INFO Total time: 14:20.313s
[Pipeline] }
[Pipeline] // withSonarQubeEnv
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

10. After that, check the project in SonarQube.



Under different tabs, check all different issues with the code.

11. To check the **Code Problems** click on **issues-**

1. Consistency

☆ sonarqube-test / ⓘ main ✓ ▾ ?

Overview **Issues** Security Hotspots Measures Code Activity

My Issues All

Filters Clear All Filters

Issues in new code

Clean Code Attribute 1 ✕

Consistency	197k
Intentionality	14k
Adaptability	0
Responsibility	0

Add to selection Ctrl + click

Software Quality

☐ Bulk Change

gameoflife-core/build/reports/tests/all-tests.html

☐ [Insert a <!DOCTYPE> declaration to before this <html> tag.](#)

Reliability ⬆

☐ Open ▾ Not assigned ▾

☐ [Remove this deprecated "width" attribute.](#)

Maintainability ⬆

☐ Open ▾ Not assigned ▾

☐ [Remove this deprecated "align" attribute.](#)

Maintainability ⬆

2. Intentionality

sonarqube Projects Issues Rules Quality Profiles Quality Gates Administration More Q

☆ sonarqube-test / ⓘ main ✓ ▾ ?

Overview **Issues** Security Hotspots Measures Code Activity

My Issues All

Filters Clear All Filters

Issues in new code

Clean Code Attribute 1 ✕

Consistency	197k
Intentionality	14k
Adaptability	0
Responsibility	0

Add to selection Ctrl + click

Software Quality

☐ Bulk Change

gameoflife-acceptance-tests/Dockerfile

☐ [Use a specific version tag for the image.](#)

Maintainability ⬆

☐ Open ▾ Not assigned ▾

☐ [Surround this variable with double quotes; otherwise, it can lead to unexpected behavior.](#)

Maintainability ⬆

☐ Open ▾ Not assigned ▾

☐ [Surround this variable with double quotes; otherwise, it can lead to unexpected behavior.](#)

Maintainability ⬆

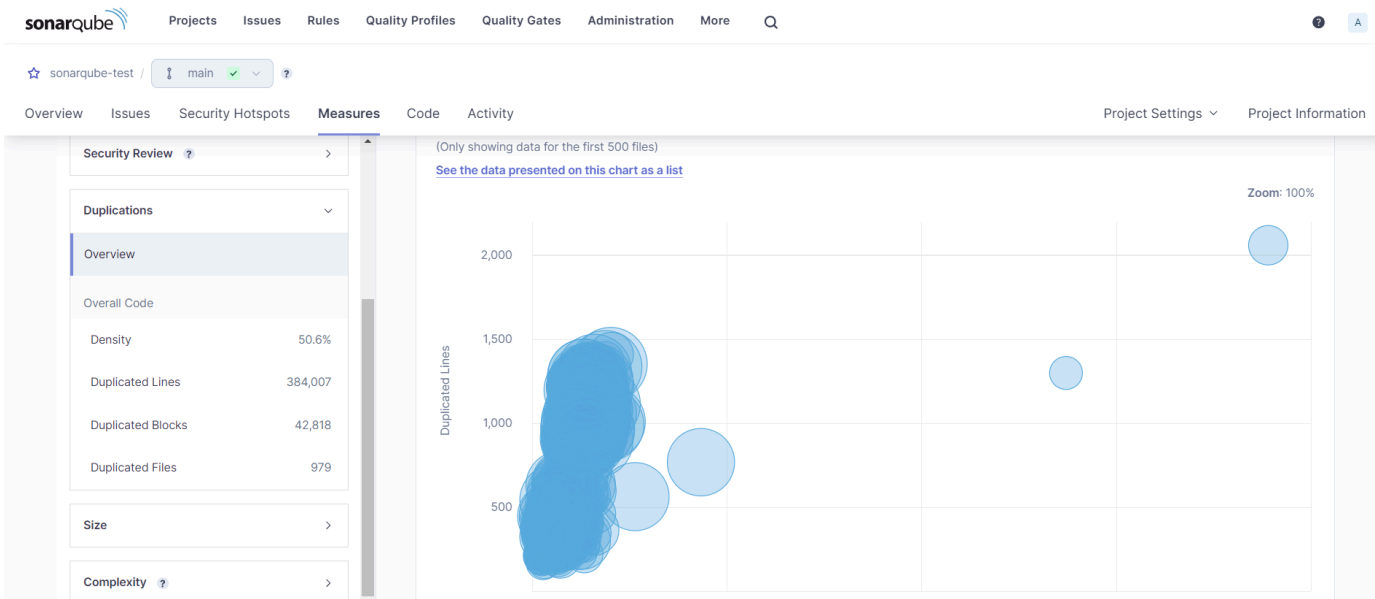
Bugs

The screenshot shows the SonarQube web interface. At the top, there's a navigation bar with 'sonarqube' logo and links for Projects, Issues, Rules, Quality Profiles, Quality Gates, Administration, and More. Below this, a breadcrumb shows 'sonarqube-test / main' with a green checkmark. The 'Issues' tab is selected, showing a list of issue types: Bug (14k), Vulnerability (0), and Code Smell (268). The 'Bug' type is selected. On the right, a list of bugs is shown. The first bug is titled 'Add "lang" and/or "xml:lang" attributes to this "<html>" element' and is categorized under 'Reliability'. The second bug is titled 'Add "<th>" headers to this "<table>"' and is also categorized under 'Reliability'. Both bugs have a status of 'Open' and are assigned to 'Not assigned'.

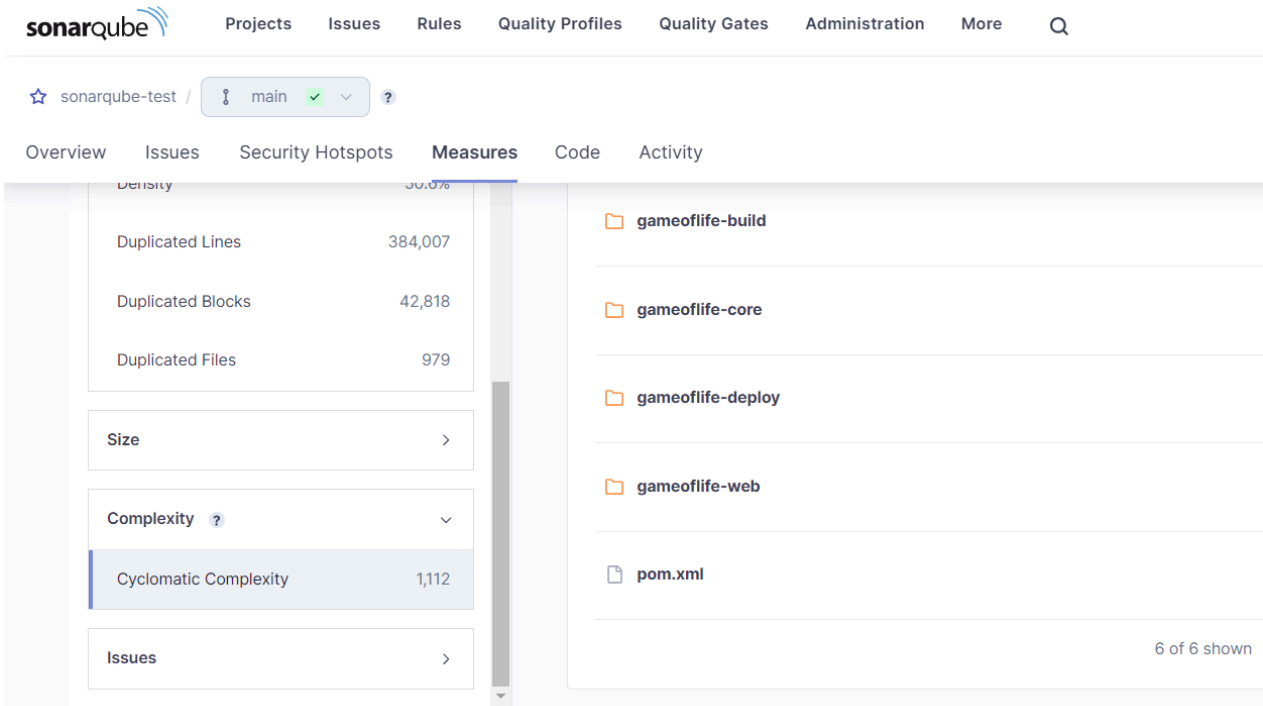
Code Smells

The screenshot shows the SonarQube web interface. At the top, there's a navigation bar with 'sonarqube' logo and links for Projects, Issues, Rules, Quality Profiles, Quality Gates, Administration, and More. Below this, a breadcrumb shows 'sonarqube-test / main' with a green checkmark. The 'Issues' tab is selected, showing a list of issue types: Bug (14k), Vulnerability (0), and Code Smell (268). The 'Code Smell' type is selected. On the right, a list of code smells is shown. The first code smell is titled 'Use a specific version tag for the image.' and is categorized under 'Maintainability'. The second code smell is titled 'Surround this variable with double quotes; otherwise, it can lead to unexpected behavior' and is also categorized under 'Maintainability'. Both code smells have a status of 'Open' and are assigned to 'Not assigned'.

Duplicates



Cyclomatic Complexities



In this way, we have created a CI/CD Pipeline with Jenkins and integrated it with SonarQube to find issues in the code like bugs, code smells, duplicates, cyclomatic complexities, etc.

Conclusion:

We set up Jenkins and SonarQube to help check our code for problems automatically. First, we used Docker to get SonarQube running. Then, we made a project for it and connected Jenkins using a special plugin. We also added the details for the SonarQube server.

Next, we created a Jenkins pipeline that can pull our code from GitHub and check it for mistakes. Here I had faced errors while mentioning the path of the sonar-scanner.bat file and also my login, password credentials were not working so I had to create a token for that. Thus, ultimately the experiment was performed successfully. And every time when we work on our code it can find bugs, bad code, and security issues. This makes our code better and helps work faster.