

Exp 8: 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:

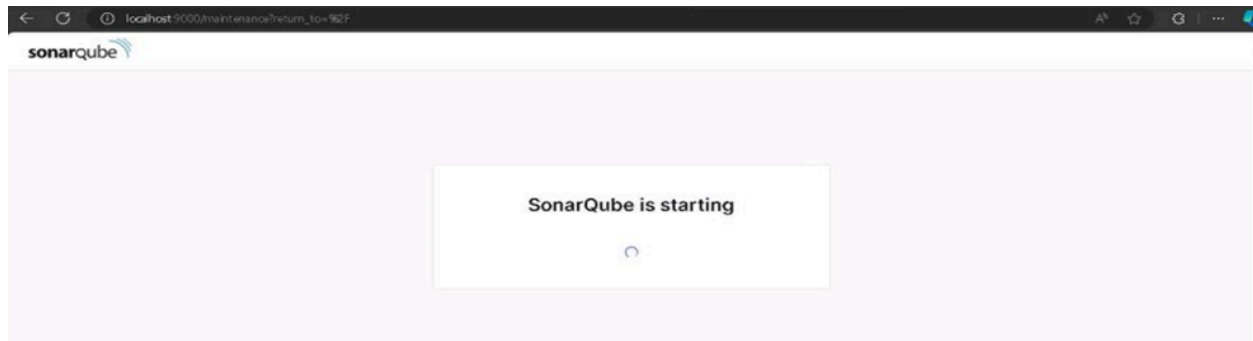
- Jenkins installed
- Docker Installed (for SonarQube)
- SonarQube Docker Image

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

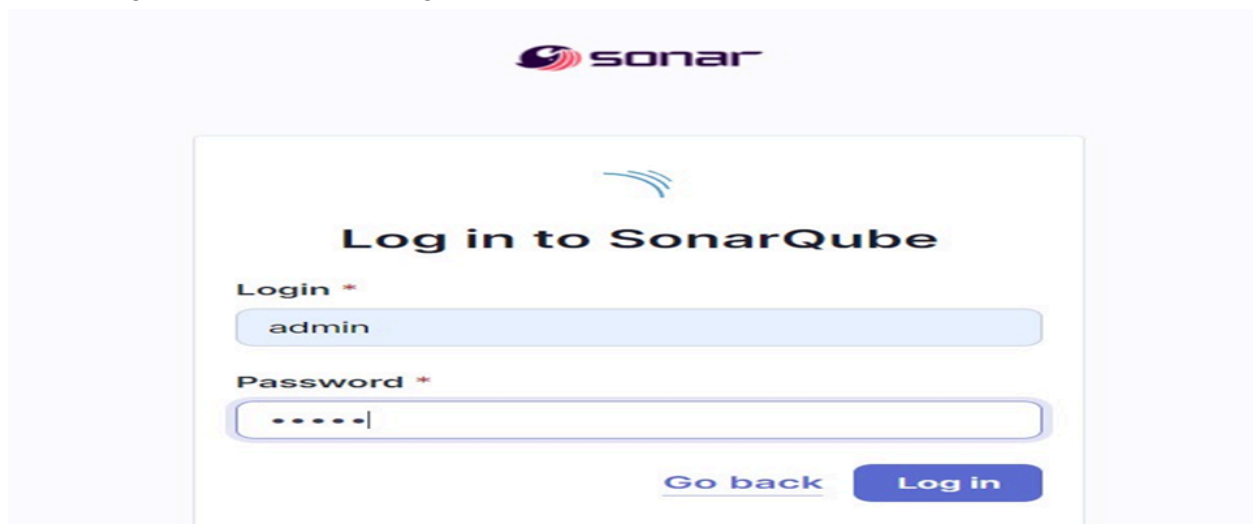
1. Open up Jenkins Dashboard on localhost, port 8080 or whichever port it is at for you.
2. Run SonarQube in a Docker container using this command -

```
PS C:\Users\91773\Desktop\College Resources\Advdevops Exp8> docker run -d --name sonarqube2 -e SONAR_ES_BOOTSTRAP_CHECKS_DISABLE=true -p 9000:9000 sonarqube:latest  
71fc67f0b15baa5be5bdcdd66966938e18682683d020beadc909dd027cfe7a  
PS C:\Users\91773\Desktop\College Resources\Advdevops Exp8>
```

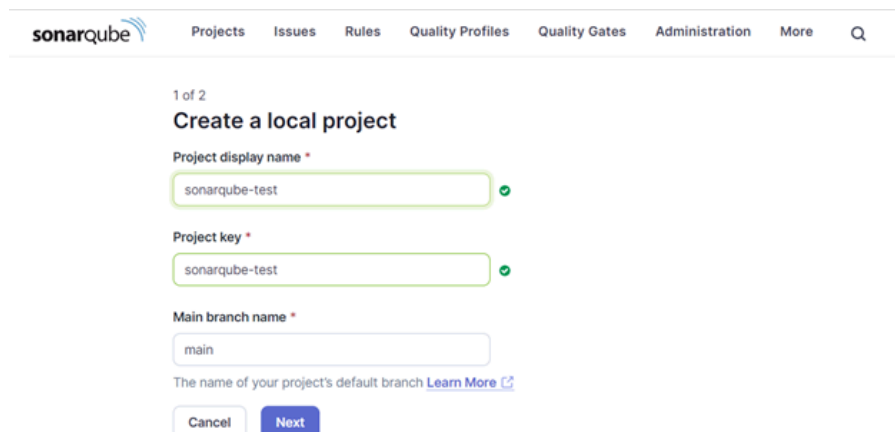
- Once the container is up and running, you can check the status of SonarQube at localhost port 9000.



- Login to SonarQube using username *admin* and password *admin*.



- 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**.

New Item

Enter an item name

SonarQube-2

Select an item type



Freestyle project

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



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

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/" }
    }
  }
}
```

Dashboard > SonarQube-2 > Configuration

Configure

- General
- Advanced Project Options
- Pipeline**

Pipeline

Definition

Pipeline script

```

1 node {
2   stage('Cloning the Github Repo') {
3     git 'https://github.com/shazforiot/GOL.git'
4   }
5   stage('SonarQube analysis') {
6     withSonarQubeEnv('sonarqube') {
7       bat '''
8         sh "C:/ProgramData/Jenkins/.jenkins/tools/hudson.plugins.sonar.SonarRunnerInstallation/sonarqube/bin/sonar-scanner.bat ^
9           -D sonar.login=admin ^
10          -D sonar.password=
11          -D sonar.projectKey=sonarqube-tests ^
12          -D sonar.exclusions=vendor/**/*.resources/**/*.java ^
13          -D sonar.host.url=http://127.0.0.1:9000/
14          """
15     }
16   }
17 }

```

☒ Use Groovy Sandbox


Pipeline Syntax

Save Apply

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

8. Run The Build.

9. Check the console output once the build is complete



Jenkins

Dashboard > SonarQube-2 >

- Status
- Changes
- Build Now
- Configure
- Delete Pipeline
- Full Stage View
- Stages
- Rename
- Pipeline Syntax

SonarQube-2

Stage View

No data available. This Pipeline has not yet run.

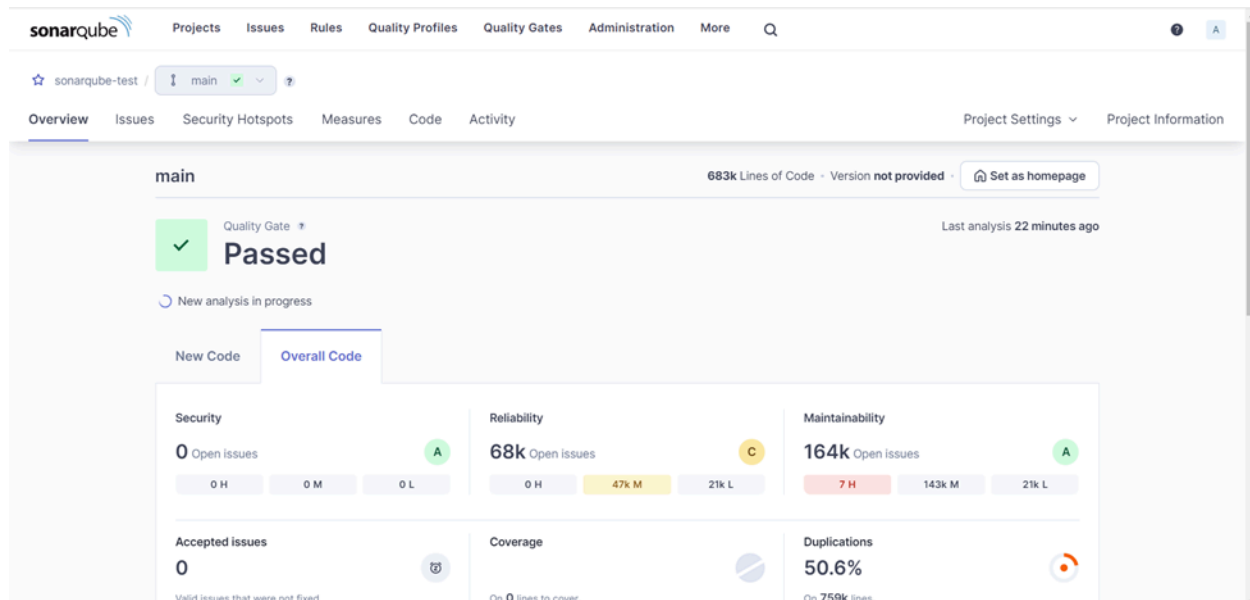
Permalinks

```
Dashboard > SonarQube-2 > #4

01:38:50.521 WARN  too many duplication references on file gameoflife-web/tools/jmeter/docs/api/org/apache/jmeter/gui/util/TextAreaEllRenderer.html
for block at line 75. Keep only the first 100 references.
01:38:50.321 INFO  CPD Executor CPD calculation finished (done) | time=238591ms
01:38:50.413 INFO  SCM revision ID 'ba799ba7e1b576f04a4612322b0412c5e6e1e5e4'
01:38:58.184 INFO  Analysis report generated in 5253ms, dir size=127.2 MB
01:39:17.257 INFO  Analysis report compressed in 19881ms, zip size=29.6 MB
01:39:28.472 INFO  Analysis report uploaded in 11215ms
01:39:28.488 INFO  ANALYSIS SUCCESSFUL, you can find the results at: http://127.0.0.1:9000/dashboard?id=sonarqube-test
01:39:28.488 INFO  Note that you will be able to access the updated dashboard once the server has processed the submitted analysis report
01:39:28.488 INFO  More about the report processing at http://127.0.0.1:9000/api/ce/task?id=ad9b398-1d74-4d8a-81ac-9b210f0e0b94
01:39:48.048 INFO  Analysis total time: 11:58.619 s
01:39:48.086 INFO  SonarScanner Engine completed successfully
01:39:49.184 INFO  EXECUTION SUCCESS
01:39:49.314 INFO  Total time: 12:15.607s

[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. Bugs

responsivity

Add to selection **Ctrl** + **click**

Software Quality

Security0

Reliability33k

Maintainability0

Severity

Type

1

Bug33k

Vulnerability0

Code Smell164k

Add to selection **Ctrl** + **click**

Scope

Status

Bulk Change

Select issues

Navigate to issue

32,896 issues

1369d effort

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

Insert a <!DOCTYPE> declaration to before this <html> tag.

Consistency

Reliability

user-experience

Open

Not assigned

L1 • 5min effort • 4 years ago • @ Bug • Major

gameoflife-core/build/reports/tests/allclasses-frame.html

Insert a <!DOCTYPE> declaration to before this <html> tag.

Consistency

Reliability

user-experience

Open

Not assigned

L1 • 5min effort • 4 years ago • @ Bug • Major

gameoflife-core/build/reports/tests/alltests-errors.html

Insert a <!DOCTYPE> declaration to before this <html> tag.

Consistency

Reliability

user-experience

Open

Not assigned

L1 • 5min effort • 4 years ago • @ Bug • Major

Code Smells

Add to selection **Ctrl** + **click**

Software Quality

Severity

Type

1

Bug33k

Vulnerability0

Code Smell164k

Add to selection **Ctrl** + **click**

Scope

Status

Security Category

Bulk Change

Select issues

Navigate to issue

163,766 issues

1705d effort

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

Remove this deprecated "width" attribute.

Consistency

Maintainability

html5 obsolete

Open

Not assigned

L9 • 5min effort • 4 years ago • @ Code Smell • Major

Remove this deprecated "align" attribute.

Consistency

Maintainability

html5 obsolete

Open

Not assigned

L11 • 5min effort • 4 years ago • @ Code Smell • Major

Remove this deprecated "align" attribute.

Consistency

Maintainability

html5 obsolete

Open

Not assigned

L12 • 5min effort • 4 years ago • @ Code Smell • Major

Remove this deprecated "size" attribute.

Consistency

Maintainability

html5 obsolete

database should be used for pagination purposes only.

Intentional issues

The screenshot displays a software quality tool interface. On the left, a sidebar titled "Issues in new code" contains filters for "Clean Code Attribute" (Consistency: 197k, Intentionality: 14k, Adaptability: 0, Responsibility: 0), "Software Quality", "Severity", and "Type" (Bug: 14k, Vulnerability: 0, Code Smell: 268). The main panel shows a list of issues for the file "gameoflife-acceptance-tests/Dockerfile". The issues are:

- ☐ Use a specific version tag for the image. (Intentionality: No tags, L1 • 5min effort • 4 years ago • @ Code Smell • @ Major)
- ☐ Surround this variable with double quotes; otherwise, it can lead to unexpected behavior. (Intentionality: No tags, L12 • 5min effort • 4 years ago • @ Code Smell • @ Major)
- ☐ Surround this variable with double quotes; otherwise, it can lead to unexpected behavior. (Intentionality: No tags, L12 • 5min effort • 4 years ago • @ Code Smell • @ Major)
- ☐ Surround this variable with double quotes; otherwise, it can lead to unexpected behavior. (Intentionality: No tags, L12 • 5min effort • 4 years ago • @ Code Smell • @ Major)

Reliabilities issue

The screenshot displays a software quality tool interface. On the left, a sidebar titled "Issues in new code" contains filters for "Clean Code Attribute" (Consistency: 54k, Intentionality: 14k, Adaptability: 0, Responsibility: 0), "Software Quality" (Security: 0, Reliability: 54k, Maintainability: 164k), "Severity", and "Type". The main panel shows a list of issues for the file "gameoflife-core/build/reports/tests/all-tests.html". The issues are:

- ☐ Insert a <DOCTYPE> declaration to before this <html> tag. (Consistency: user-experience, L1 • 5min effort • 4 years ago • @ Bug • @ Major)
- ☐ Anchors must have content and the content must be accessible by a screen reader. (Consistency: accessibility, L29 • 5min effort • 4 years ago • @ Code Smell • @ Minor)
- ☐ Anchors must have content and the content must be accessible by a screen reader. (Consistency: accessibility, L38 • 5min effort • 4 years ago • @ Code Smell • @ Minor)
- ☐ Anchors must have content and the content must be accessible by a screen reader. (Consistency: accessibility, L38 • 5min effort • 4 years ago • @ Code Smell • @ Minor)

Duplicates



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:

In this experiment, we successfully integrated Jenkins with SonarQube to automate code quality checks within our CI/CD pipeline. We began by deploying SonarQube using Docker, configuring a project, and setting it up for comprehensive code analysis. Following this, we configured Jenkins by installing the SonarQube Scanner plugin, providing the necessary SonarQube server details, and establishing the scanner tool. We then created a Jenkins pipeline to streamline the process of cloning a GitHub repository and executing SonarQube analysis on the code. This integration facilitates continuous monitoring of code quality, enabling us to identify issues such as bugs, code smells, and security vulnerabilities throughout the development lifecycle.