

## **Experiment 8**

### **Aim:**

Create a Jenkins CI/CD 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:**

#### Static Application Security Testing (SAST)

SAST is a testing methodology that analyzes source code to find security vulnerabilities, making applications less susceptible to attacks. It scans the application before the code is compiled, also known as white-box testing.

#### Problems SAST Solves:

- **Early Detection:** Identifies vulnerabilities early in the SDLC, allowing developers to resolve issues without breaking builds or passing vulnerabilities to the final release.
- **Real-Time Feedback:** Provides developers with immediate feedback as they code, helping them fix issues before moving to the next phase.
- **Graphical Representations:** Offers visual aids to navigate code, pinpointing exact locations of vulnerabilities and providing guidance on fixes.
- **Regular Scanning:** Should be run regularly, such as during daily/monthly builds, code check-ins, or code releases.

#### Importance of SAST

- **Resource Efficiency:** Developers outnumber security staff, making it challenging to perform manual code reviews. SAST tools can analyze 100% of the codebase quickly.
- **Speed:** Can scan millions of lines of code in minutes, identifying critical vulnerabilities like buffer overflows, SQL injection, and cross-site scripting with high confidence.

#### CI/CD Pipeline

A CI/CD pipeline refers to the Continuous Integration/Continuous Delivery pipeline, which is the backbone of the DevOps approach. It involves a series of tasks connected in sequence to facilitate quick software releases. The pipeline is responsible for building code, running tests, and deploying new software versions.

#### SonarQube

SonarQube is an open-source platform developed by SonarSource for continuous inspection of code quality. It performs static code analysis, providing detailed reports on bugs, code smells, vulnerabilities, and code duplications. It supports over 25 major programming languages and can be extended with various plugins.

Benefits of SonarQube:

- Sustainability: Reduces complexity, vulnerabilities, and code duplications, optimizing application lifespan.
- Increased Productivity: Lowers maintenance costs and risks, reducing the need for extensive code changes.
- Quality Code: Ensures code quality control is an integral part of software development.
- Error Detection: Automatically detects errors and alerts developers to fix them before output submission.
- Consistency: Identifies code criteria breaches, enhancing overall code quality.
- Business Scaling: Supports scaling without restrictions.

Implementation:

Prerequisites

1. Jenkins installed on your machine.
2. Docker installed to run SonarQube.
3. SonarQube installed via Docker

1. Set Up Jenkins

- Open Jenkins Dashboard on localhost:8080 or your configured port.
- Install the necessary plugins:
  - SonarQube Scanner Plugin

2. Run SonarQube in a Docker Container

- Open a terminal and run the following command to start SonarQube in a Docker container

Command -

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

```
C:\Users\272241>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: sha256:72e9feec71242af83faf65f95a40d5e3bb2822a6c3b2cda8568790f3d31ae
cde
Status: Downloaded newer image for sonarqube:latest
8b62aeca4d09887a0db32d349116529581a639b59222a8b20987b42d8cec6ef3
```

### 3. Check SonarQube Status

- Once the container is up and running, check the status of SonarQube by navigating to <http://localhost:9000>

### 4. Login to SonarQube

- Use the credentials to log in:

sonarqube

ProjectsIssuesRulesQuality ProfilesQuality GatesAd

1 of 2

Create a local project

Project display name \*

Pipeline

Project key \*

Pipeline

Main branch name \*

main

The name of your project's default branch [Learn More](#)

CancelNext

5. You can view the project:

sonarqube

ProjectsIssuesRulesQuality ProfilesQuality GatesAdministrationMore

My FavoritesAll

Filters

Quality Gate

Passed1Failed0

Reliability

A1B0C0D0E0

Security

Search for projects...

PerspectiveOverall StatusSort byName

Create Project

2 project(s)

☆ Pipeline PUBLIC

Project's Main Branch is not analyzed yet.

☆ sonarqube PUBLIC

Last analysis: 3 hours ago

The main branch of this project is empty.

Passed

2 of 2 shown

## 6. Generate SonarQube Token

- Go to My Account > Security > Generate Tokens.
- Copy the generated token for later use.

The screenshot shows the SonarQube Administrator interface. The top navigation bar includes links for Projects, Issues, Rules, Quality Profiles, Quality Gates, Administration, and More. The user is logged in as 'Administrator'. The 'Security' tab is selected in the sub-navigation. The 'Generate Tokens' section is active, displaying a message: 'New token "ronak1" has been created. Make sure you copy it now, you won't be able to see it again!'. Below the message, the token 'squ\_ff29e593687aefbaacce04d8429d2b640d810ee9' is shown with a copy icon. A table below lists the generated tokens:

Name	Type	Project	Last use	Created	Expiration	
ronak1	User		Never	September 23, 2024	October 23, 2024	<a href="#">Revoke</a>

## 7. Create a Jenkins Pipeline

- Go to Jenkins Dashboard, click New Item, and select Pipeline.

The screenshot shows the Jenkins 'Enter an item name' dialog. The 'sonarpipe' text is entered in the input field. Below the input field, a list of item types is displayed: Freestyle project, Maven project, Pipeline, Multi-configuration project, and Folder. The 'Pipeline' option is highlighted, indicating it is the selected item type. At the bottom, there is an 'OK' button and a 'branch Pipeline' link.

8. Under Pipeline Script, enter the following script:

```
docker network create sonarnet
node {
  stage('Cloning the GitHub Repo') {
    git 'https://github.com/shazforiot/GOL.git'
  }
  stage('SonarQube analysis') {
    withSonarQubeEnv('sonarqube') {
      bat """
      docker run --rm --network host \
      -e SONAR_HOST_URL=http://<ip_address>:9000 \
      -e SONAR_LOGIN=admin \
      -e SONAR_PASSWORD=<Sonarqube_password> \
      -e SONAR_PROJECT_KEY=sonarqube-test \
      -v ${WORKSPACE.replace('\', '/')}:/usr/src \
      sonarsource/sonar-scanner-cli \
      -Dsonar.projectKey=sonarqube-test \
      -Dsonar.exclusions=vendor/**,resources/**,*/*.java \
      -Dsonar.login=admin \
      -Dsonar.password=<Sonarqube_password>
      """
    }
  }
}
```

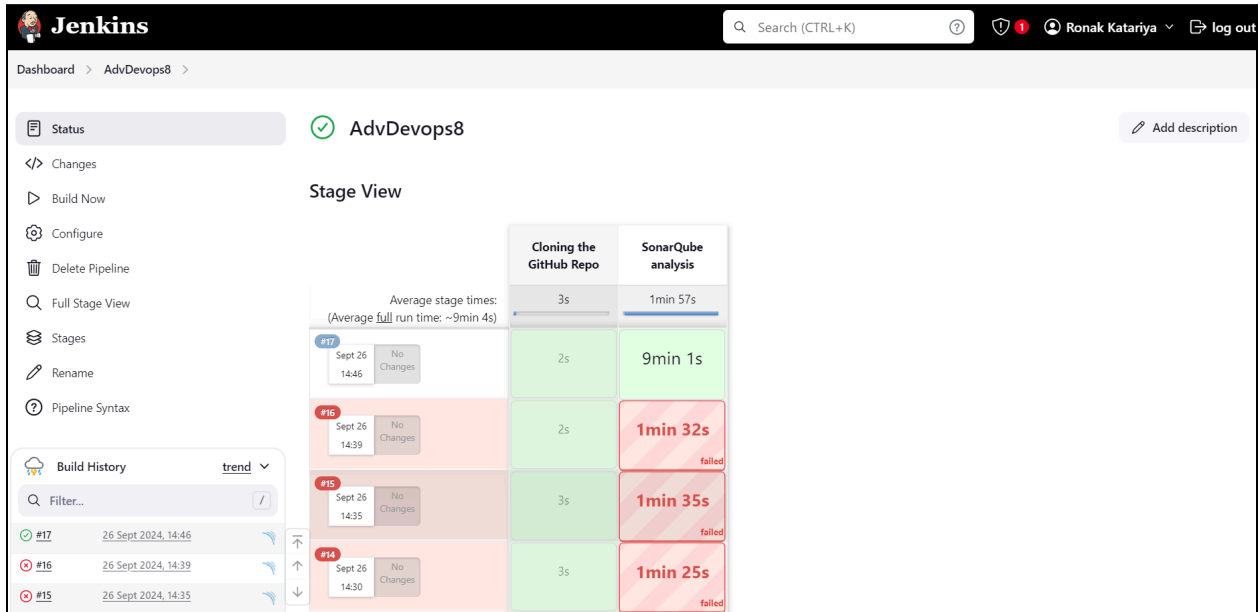
The screenshot shows the Jenkins 'Configure' page for a pipeline named 'AdvDevops8'. The 'Pipeline' tab is selected, and the 'Definition' is set to 'Pipeline script'. The script is displayed in a text area with line numbers 1 through 17. The script content is as follows:

```
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       docker run --rm ^
9       -e SONAR_HOST_URL=http://192.168.23.8:9000 ^
10      -v ${WORKSPACE.replace('\', '/')}:/usr/src ^
11      sonarsource/sonar-scanner-cli ^
12      -Dsonar.projectKey=Pipeline ^
13      -Dsonar.sources=. ^
14      -Dsonar.exclusions=**/*.java,vendor/**/*.java,resources/ ^
15      -Dsonar.login=admin ^
16      -Dsonar.password=Sonar4u
17      """
6     }
5   }
2 }
```

Below the script area, there is a checkbox labeled 'Use Groovy Sandbox' which is checked. At the bottom, there are 'Save' and 'Apply' buttons.

## 9. Run the Pipeline

- Save the pipeline and click Build Now.
- Monitor the console output for any errors.



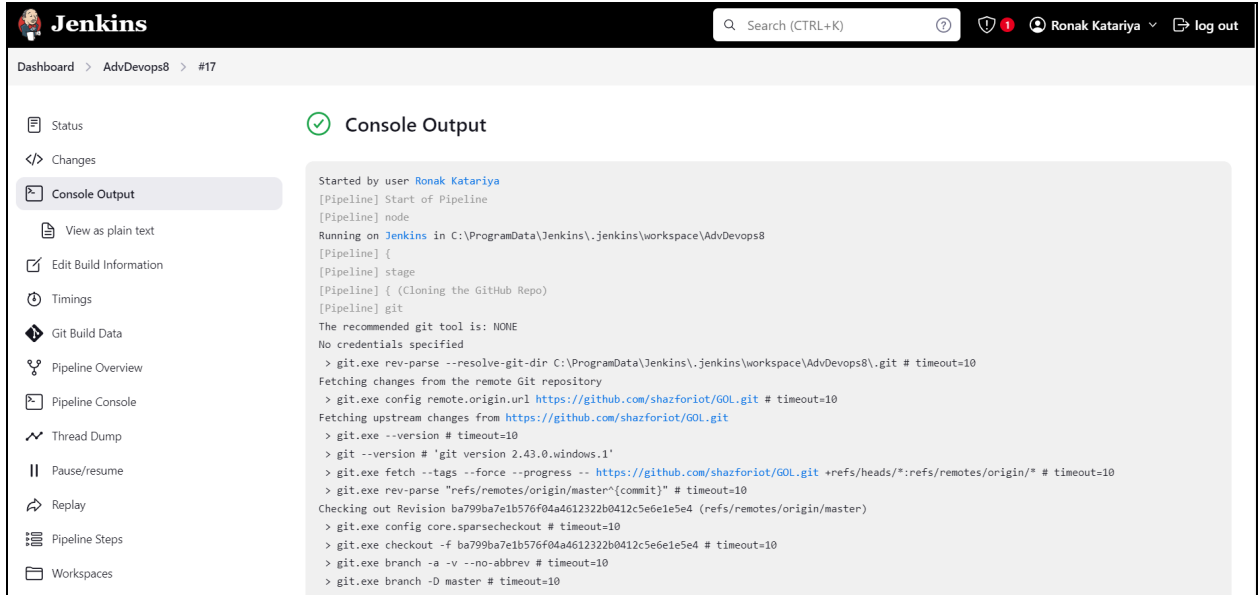
The Jenkins Pipeline Stage View for 'AdvDevops8' shows the following stages and their durations:

Stage	Cloning the GitHub Repo	SonarQube analysis
#17 (Sept 26 14:46)	2s	9min 1s
#16 (Sept 26 14:39)	2s	1min 32s
#15 (Sept 26 14:35)	3s	1min 35s
#14 (Sept 26 14:30)	3s	1min 25s

Average stage times: (Average full run time: ~9min 4s)

Build History:

Build	Time
#17	26 Sept 2024, 14:46
#16	26 Sept 2024, 14:39
#15	26 Sept 2024, 14:35



The Jenkins Pipeline Console Output for 'AdvDevops8' #17 shows the following output:

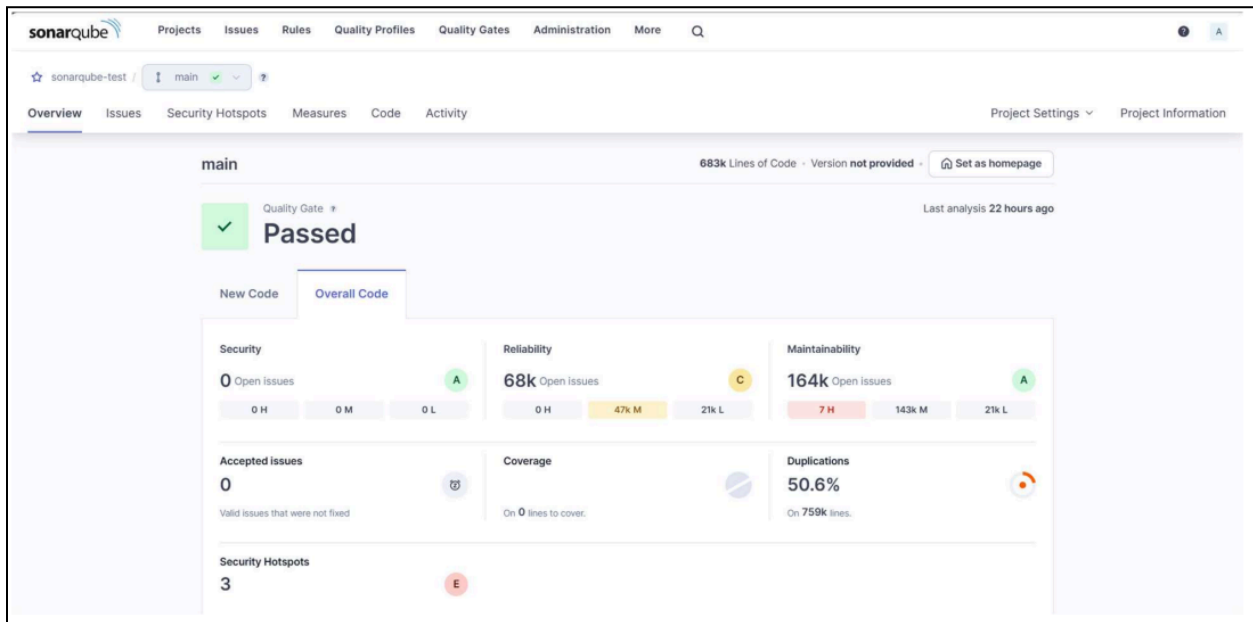
```
Started by user Ronak Katariya
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in C:\ProgramData\Jenkins\jenkins\workspace\AdvDevops8
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Cloning the GitHub Repo)
[Pipeline] git
The recommended git tool is: NONE
No credentials specified
> git.exe rev-parse --resolve-git-dir C:\ProgramData\Jenkins\jenkins\workspace\AdvDevops8\.git # timeout=10
Fetching changes from the remote Git repository
> git.exe config remote.origin.url https://github.com/shazforiot/GOL.git # timeout=10
Fetching upstream changes from https://github.com/shazforiot/GOL.git
> git.exe --version # timeout=10
> git --version # 'git version 2.43.0.windows.1'
> git.exe fetch --tags --force --progress -- https://github.com/shazforiot/GOL.git +refs/heads/*:refs/remotes/origin/* # timeout=10
> git.exe rev-parse "refs/remotes/origin/master^{commit}" # timeout=10
Checking out Revision ba799ba7e1b576f04a4612322b0412c5e6e1e5e4 (refs/remotes/origin/master)
> git.exe config core.sparsecheckout # timeout=10
> git.exe checkout -f ba799ba7e1b576f04a4612322b0412c5e6e1e5e4 # timeout=10
> git.exe branch -a -v --no-abbrev # timeout=10
> git.exe branch -D master # timeout=10
```

```
Dashboard > AdvDevops8 > #17

09:24:33.433 WARN Too many duplication references on file gameoflife-web/tools/jmeter/docs/api/org/apache/jmeter/assertions/BeanShellAssertion.html
for block at line 41. Keep only the first 100 references.
09:24:35.453 WARN Too many duplication references on file gameoflife-web/tools/jmeter/docs/api/org/apache/jmeter/assertions/BeanShellAssertion.html
for block at line 17. Keep only the first 100 references.
09:24:35.453 WARN Too many duplication references on file gameoflife-web/tools/jmeter/docs/api/org/apache/jmeter/assertions/BeanShellAssertion.html
for block at line 698. Keep only the first 100 references.
09:24:35.453 WARN Too many duplication references on file gameoflife-web/tools/jmeter/docs/api/org/apache/jmeter/assertions/BeanShellAssertion.html
for block at line 75. Keep only the first 100 references.
09:24:35.454 INFO CPD Executor CPD calculation finished (done) | time=119165ms
09:24:35.488 INFO SCM revision ID 'ba799ba7e1b576f04a4612322b0412c5e6e1e5e4'
09:24:50.535 INFO Analysis report generated in 14994ms, dir size=127.2 MB
09:24:59.094 INFO Analysis report compressed in 8556ms, zip size=29.6 MB
09:25:01.759 INFO Analysis report uploaded in 2664ms
09:25:01.760 INFO ANALYSIS SUCCESSFUL, you can find the results at: http://192.168.23.8:9000/dashboard?id=Pipeline
09:25:01.760 INFO Note that you will be able to access the updated dashboard once the server has processed the submitted analysis report
09:25:01.761 INFO More about the report processing at http://192.168.23.8:9000/api/ce/task?id=f8f82a37-2582-451e-8839-ef19560deada
09:25:03.386 INFO Analysis total time: 8:41.698 s
09:25:03.389 INFO SonarScanner Engine completed successfully
09:25:03.783 INFO EXECUTION SUCCESS
09:25:03.785 INFO Total time: 8:57.529s
[Pipeline] }
WARN: Unable to locate 'report-task.txt' in the workspace. Did the SonarScanner succeed?
[Pipeline] // withSonarQubeEnv
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

## 10. Check SonarQube for Analysis Results

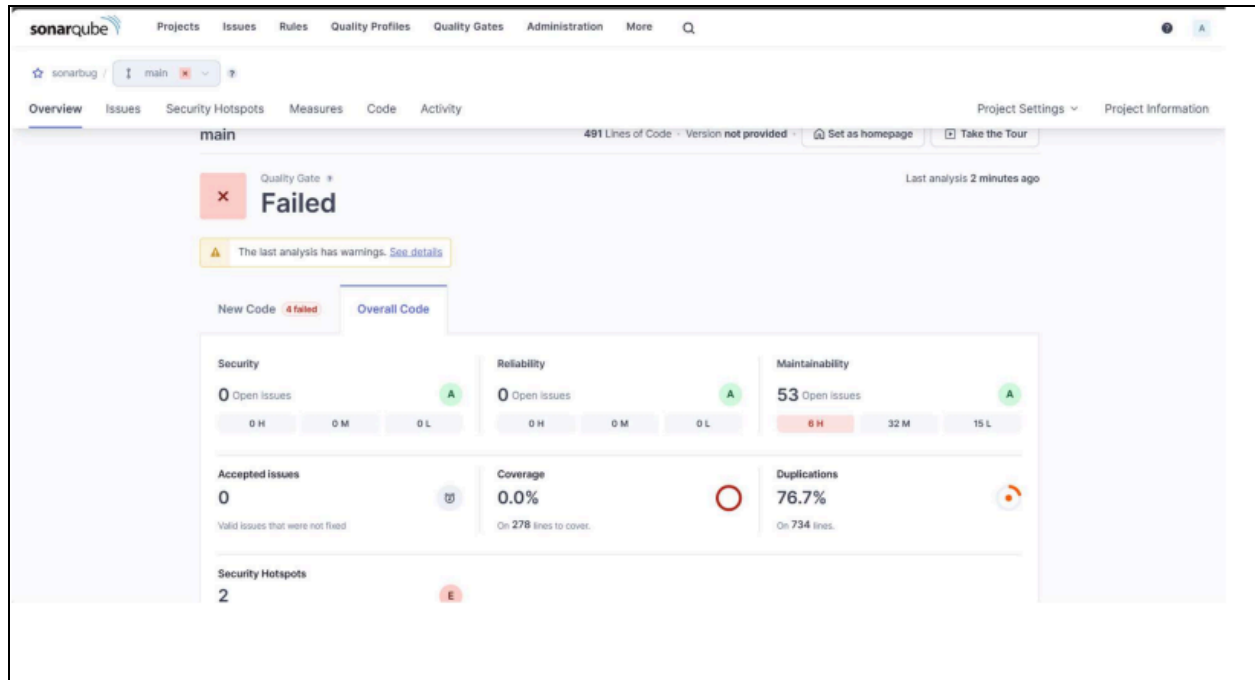
- Go to your SonarQube dashboard and check the project for issues such as bugs, code smells, and security vulnerabilities.



## 11. Checking SonarQube for Analysis Results of a Code File with Bugs , Code Smells, Security Vulnerabilities, Cyclomatic Complexities and Duplicates .

- Overview -





## Conclusion:

In this experiment, we performed a static analysis of the code to detect bugs, code smells, and security vulnerabilities on our sample codes.