

EXPERIMENT NO. 8

NAME : PRANAV POL

CLASS : D15A

ROLL NO. : 42

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:

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 Docker

Run the following command to start SonarQube in a Docker container:

command :

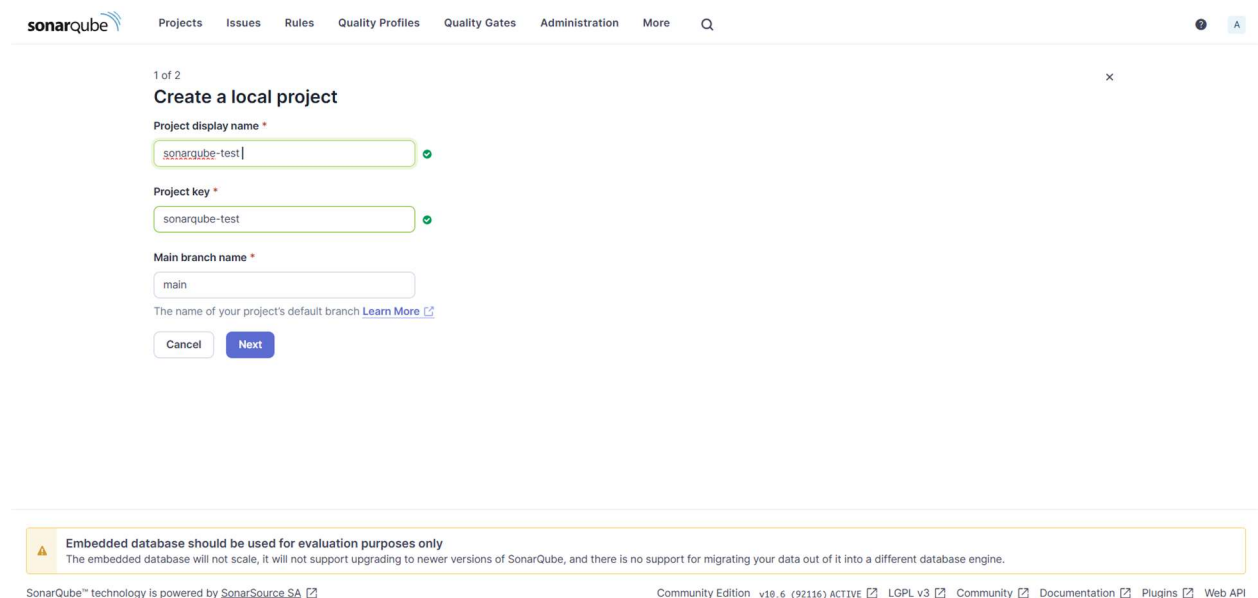
```
docker run -d --name sonarqube -e SONAR_ES_BOOTSTRAP_CHECKS_DISABLE=true -p 9000:9000 sonarqube:latest
```

- Check SonarQube status at <http://localhost:9000>.
- Login with your credentials:

```
C:\Users\sbpol>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: sha256:72e9feec71242af83faf65f95a40d5e3bb2822a6c3b2cda8568790f3d31aecde
Status: Downloaded newer image for sonarqube:latest
2f213117ce50f08304d681a60dc0e2a4dd6c3c8e46f5725be7fb40fd0d48bb5d
```

3. Create a Project in SonarQube

- Go to **Projects > Create Project**.
- Name the project (e.g., sonarqube-test).



The screenshot shows the SonarQube web interface. At the top, there is a navigation bar with links: Projects, Issues, Rules, Quality Profiles, Quality Gates, Administration, More, and a search icon. Below the navigation bar, the main content area displays a form titled "Create a local project". The form has three input fields: "Project display name *" with the value "sonarqube-test", "Project key *" with the value "sonarqube-test", and "Main branch name *" with the value "main". Below these fields, there is a link "The name of your project's default branch [Learn More](#)". At the bottom of the form, there are two buttons: "Cancel" and "Next". Below the form, there is a yellow warning box with a triangle icon and the text: "Embedded database should be used for evaluation purposes only. The embedded database will not scale, it will not support upgrading to newer versions of SonarQube, and there is no support for migrating your data out of it into a different database engine." At the very bottom of the page, there is a footer with the text: "SonarQube™ technology is powered by SonarSource.SA" and a row of links: "Community Edition v10.6 (92116) ACTIVE", "LGPL v3", "Community", "Documentation", "Plugins", and "Web API".

4. Generate SonarQube Token

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

5. Create a Jenkins Pipeline

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

The screenshot shows the Jenkins 'New Item' form. The 'Enter an item name' field is filled with 'sonarpipe'. Below this field, there are five project type options: 'Freestyle project', 'Maven project', 'Pipeline', 'Multi-configuration project', and 'Folder'. The 'Pipeline' option is selected. At the bottom of the form, there is a 'Create Pipeline' button.

6. 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') {
      sh """
        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>
=====
```

```
}
}
}
```

Dashboard > sonarpipe > Configuration

Configure

- General
- Advanced Project Options**
- Pipeline

Advanced Project Options

Advanced ▾

Pipeline

Definition

Pipeline script ▾

```
Script ?
2 = stage('Cloning the GitHub Repo') {
3   git 'https://github.com/shazforiot/GOL.git'
4 }
5 = stage('SonarQube analysis') {
6   withSonarQubeEnv('sonarqube') {
7     sh """
8       docker run --rm --network host \
9         -e SONAR_HOST_URL=http://192.168.1.103:9000 \
10        -e SONAR_LOGIN=admin \
11        -e SONAR_PASSWORD=Pranav@144 \
12        -e SONAR_PROJECT_KEY=sonarqube-test \
13        -v $(WORKSPACE.replace('\ ', '/')):/usr/src \
14        sonarsource/sonar-scanner-cli \
15        -Dsonar.projectKey=sonarqube-test \
16        -Dsonar.exclusions=vendor/**,resources/**,**/*.java \
17        -Dsonar.login=admin \
18        -Dsonar.password=Pranav@144
19     """
20   }
21 }
```

☒ Use Groovy Sandbox ?

[Pipeline Syntax](#)

Save **Apply**

7. Run the Pipeline

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

Jenkins

Search (CTRL+K)

Pranav Santosh Pol

Dashboard > sonarpipe >

Status

sonarpipe

Changes

Build Now

Configure

Delete Pipeline

Full Stage View

Stages

Rename

Pipeline Syntax

Stage View

Stage	Cloning the GitHub Repo	SonarQube analysis
Average stage times:	2s	5min 25s
Average full run time: ~10min		
#11	Sep 22 01:27 No Changes	1s
#10	Sep 22 01:26 No Changes	10min 37s
		2s
		12s failed

Permalinks

- Last build (#11), 11 min ago
- Last stable build (#11), 11 min ago
- Last successful build (#11), 11 min ago
- Last failed build (#10), 12 min ago
- Last unsuccessful build (#10), 12 min ago

Build History

Filter...

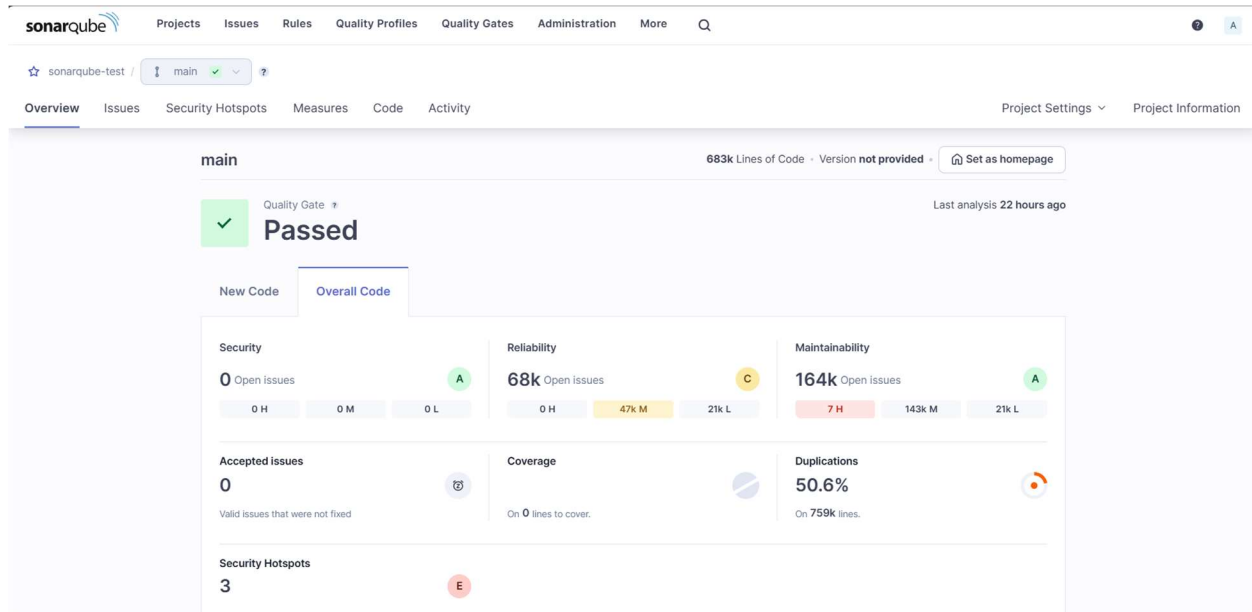
#11 Sep 22, 2024, 1:27 AM

#10 Sep 22, 2024, 1:26 AM

Atom feed for all Atom feed for failures

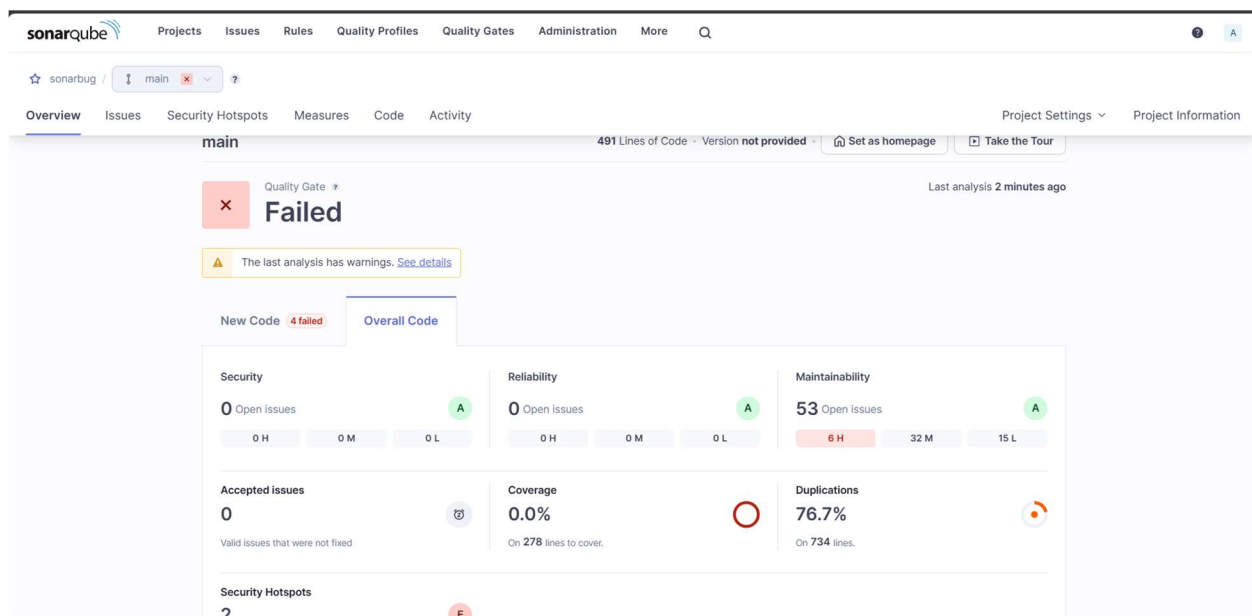
9. Check SonarQube for Analysis Results

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



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

- Overview -



• Issues -

The screenshot shows the SonarQube interface for the 'sonarbug' project. The top navigation bar includes 'sonarqube', 'Projects', 'Issues', 'Rules', 'Quality Profiles', 'Quality Gates', 'Administration', and 'More'. Below this, a breadcrumb shows 'sonarbug / main'. The main navigation tabs are 'Overview', 'Issues', 'Security Hotspots', 'Measures', 'Code', and 'Activity'. The 'Issues' tab is active, showing a list of 54 issues with a total effort of 1d 4h. A sidebar on the left contains filters for 'My Issues' and 'All', and a 'Clean Code Attribute' section with metrics: Consistency (15), Intentionality (21), Adaptability (18), and Responsibility (0). The main content area displays three issues: 1. 'Function 'calculateInvoiceTotal' has too many parameters (8). Maximum allowed is 7.' (Adaptability, brain-overload, L2, 20min effort, 44 minutes ago, @ Code Smell, @ Major). 2. 'Expected a 'for-of' loop instead of a 'for' loop with this simple iteration.' (Consistency, clumsy, L5, 5min effort, 44 minutes ago, @ Code Smell, @ Minor). 3. 'Refactor this function to reduce its Cognitive Complexity from 36 to the 15 allowed.' (Adaptability, brain-overload, L41, 26min effort, 44 minutes ago, @ Code Smell, @ Critical). A yellow banner at the bottom states 'Embedded database should be used for evaluation purposes only'.

• Security Hotspot (Security Vulnerabilities) -

The screenshot shows the SonarQube interface for the 'sonarbug' project, specifically the 'Security Hotspots' tab. The top navigation bar is the same as the previous screenshot. The main navigation tabs are 'Overview', 'Issues', 'Security Hotspots', 'Measures', 'Code', and 'Activity'. The 'Security Hotspots' tab is active, showing a progress bar for '0.0% Security Hotspots Reviewed' and a filter for 'To review'. The main content area displays a security hotspot titled 'Authentication' with a review priority of 'High'. The hotspot description is 'Review this potentially hardcoded credential.' The code snippet shows a function 'connectToDatabase()' with hardcoded credentials: 'const username = "admin";' and 'const password = "password123";'. A red box highlights the password, and a message says 'Review this potentially hardcoded credential.' The code also shows a function 'redirectTo(url)'.

• Codesmells -

The screenshot displays the SonarQube web interface for a project named 'sonarbug'. The top navigation bar includes links for Projects, Issues, Rules, Quality Profiles, Quality Gates, Administration, and More. The left sidebar shows a summary of software quality metrics: Security (0), Reliability (0), and Maintainability (53). Under the 'Severity' section, there are 6 High, 32 Medium, and 15 Low issues. The 'Type' section shows 1 Bug, 0 Vulnerability, and 53 Code Smell. The 'Scope' section is also visible.

The main content area displays a list of code smells for the file 'sonarbug.js'. The first three issues are:

- Function 'calculateInvoiceTotal' has too many parameters (8). Maximum allowed is 7.** (Maintainability, brain-overload, L2, 20min effort, 16 minutes ago, Code Smell, Major)
- Expected a 'for-of' loop instead of a 'for' loop with this simple iteration.** (Maintainability, clumsy, L5, 5min effort, 16 minutes ago, Code Smell, Minor)
- Refactor this function to reduce its Cognitive Complexity from 36 to the 15 allowed.** (Maintainability, brain-overload, L41, 26min effort, 16 minutes ago, Code Smell, Critical)

Below the list, there is a warning message: "Embedded database should be used for evaluation purposes only. The embedded database will not scale, it will not support upgrading to newer versions of SonarQube, and there is no support for migrating your data out of it into a different database engine."

The second part of the screenshot shows a detailed view of the first code smell. It includes the following information:

- Function 'calculateInvoiceTotal' has too many parameters (8). Maximum allowed is 7.** (Adaptability | Not focused, Software qualities impacted: Maintainability)
- Functions should not have too many parameters javascript:S107** (Line affected: L2, Effort: 20min, Introduced: 47 minutes ago, Code Smell, Major)
- Where is the issue? Why is this an issue? Activity** (Buttons: Open in IDE, See all issues in this file, New Code)
- Code Snippet:**

```
// Code Smell: Duplicate Code, Large Functions, Unused Variables, and Long Parameter Lists
function calculateInvoiceTotal(items, discount, taxRate, shipping, isGift, useCoupon, couponCode, isInternational) {
  let total = 0;
  for (let i = 0; i < items.length; i++) {
    // Bug: Incorrect price calculation
    total += items[i].price * items[i].quantity * 1.1; // Wrong multiplier
  }
}
```
- Expected a 'for-of' loop instead of a 'for' loop with this simple iteration.**

At the bottom, there is another warning message: "Embedded database should be used for evaluation purposes only. The embedded database will not scale, it will not support upgrading to newer versions of SonarQube, and there is no support for migrating your data out of it into a different database engine."

• Cyclomatic Complexity -

The screenshot shows the SonarQube interface for a project named 'sonarbug'. The 'Measures' tab is selected, displaying a list of metrics on the left: Overall Code, Security Hotspots (2), Rating (E), Security Hotspots Reviewed (0.0%), Coverage, Duplications, Size, Complexity (173), and Cognitive Complexity (0). The 'Cyclomatic Complexity' measure is highlighted, showing a value of 173. The main panel displays the source code for 'sonarbug.js' with various annotations. Comments include 'Code Smell: Duplicate Code, Large Functions, Unused Variables, and Long Parameter Lists', 'Bug: Incorrect price calculation', 'Wrong multiplier', 'Code Smell: Duplicate Code', 'Magic number used, no explanation', 'Adding tax', 'Bug: Shipping is always applied even if it's a gift', 'Extra charge for gift wrapping', and 'Code Smell: This check is overly complicated'. The code is a JavaScript function 'calculateInvoiceTotal' that calculates the total price of an invoice based on items, discount, tax rate, shipping, and gift status.

• Cognitive Complexity -

The screenshot shows the SonarQube interface for a project named 'sonarbug'. The 'Issues' tab is selected, displaying a list of issues on the left. The main panel shows a specific issue titled 'Refactor this function to reduce its Cognitive Complexity from 36 to the 15 allowed.' with a severity of 'Critical'. The issue is associated with the file 'sonarbug.js' and the function 'validateOrder'. The code snippet shows a function 'validateOrder' with multiple nested if statements. The issue description states: 'Cognitive Complexity of functions should not be too high javascript:S3776'. The issue is marked as 'Open' and 'Not assigned'. The right sidebar shows the 'Clean code attribute' as 'Adaptability | Not focused' and 'Software qualities impacted' as 'Maintainability'.

• Duplications -

The screenshot displays the SonarQube web interface for a project named 'sonarbug'. The 'Measures' tab is selected, showing a summary of code quality metrics. On the left, a sidebar lists 'Duplications' with an overview table. The main area shows a code editor for 'sonarbug.js' with highlighted duplicated lines and associated code smells.

Category	Metric	Value
New Code	Density	77.7%
	Duplicated Lines	543
	Duplicated Blocks	2
	Duplicated Files	1
Overall Code	Density	76.7%
	Duplicated Lines	563
	Duplicated Blocks	2
	Duplicated Files	1

Duplicated Lines (%) on New Code 77.7%

```
1  pranav.. // Code Smell: Duplicate Code, Large Functions, Unused Variables, and Long Parameter Lists
2  // function calculateInvoiceTotal(items, discount, taxRate, shipping, isGift, useCoupon, couponCode, isInternational) {
3  //   let total = 0;
4  //   for (let i = 0; i < items.length; i++) {
5  //     // Bug: Incorrect price calculation
6  //     total += items[i].price * items[i].quantity * 1.1; // Wrong multiplier
7  //   }
8  //   // Code Smell: Duplicate Code
9  //   if (discount > 0) {
10 //     total -= total * (discount / 100);
11 //   }
12 //   if (useCoupon && couponCode === 'SAVE20') {
13 //     total -= 20; // Magic number used, no explanation
14 //   }
15 //   total += total * (taxRate / 100); // Adding tax
16 //   // Bug: Shipping is always applied even if it's a gift
17 //   total += shipping;
18 //   if (isGift) {
19 //     total += 5; // Extra charge for gift wrapping
20 //   }
21 // }
22
23
24
25
26
27
```

Conclusion:

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