Experiment 4: Jenkins CI/CD Pipeline Integration with SonarQube for Static Code Analysis.

Objective:
To **install and configure SonarQube** for static code analysis, **set up Jenkins**, and **create a CI/CD pipeline** integrating SonarQube, ensuring **code quality checks** are automatically performed during the software build process.

1. Install Java (Prerequisite)
Why?
Both **SonarQube** and **Jenkins** require Java (JDK) as a runtime environment.
Steps:
```
sudo apt update                  # Updates the package list
sudo apt install openjdk-11-jdk -y   # Installs OpenJDK 11
java -version                    # Verifies Java installation
```
- **Explanation:**
  Java is essential because SonarQube is a Java-based web application, and Jenkins is also a Java-based automation server. **JDK 11** is recommended as it provides stability and compatibility for both tools.

2. Install and Configure SonarQube
Why?
SonarQube is used to **analyze source code quality**. It detects **bugs, vulnerabilities, code smells, and duplications**.
Steps:
a. Download SonarQube:
```
wget
https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-10.2.1.78527.zip
unzip sonarqube-10.2.1.78527.zip
sudo mv sonarqube-10.2.1.78527 /opt/sonarqube
```
- **Explanation:**
  Downloads and extracts SonarQube to the /opt/sonarqube directory for central management.

b. Start SonarQube:

```
cd /opt/sonarqube/bin/linux-x86-64
./sonar.sh start
```
- **Explanation:**
  Starts the SonarQube service in the background.
  Access it via **http://localhost:9000**.

c. Login to SonarQube:
- Open a browser → **http://localhost:9000**.
- Default credentials:
  - **Username:** admin
  - **Password:** admin
- Change the password after the first login.

3. Install and Configure Jenkins
Why?
Jenkins is used to **automate CI/CD processes** such as **building, testing, and deploying applications**. It integrates with SonarQube for static code analysis.
Steps:
a. Install Jenkins:
```
wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key |
sudo apt-key add -
sudo sh -c 'echo deb https://pkg.jenkins.io/debian-stable binary/ >
/etc/apt/sources.list.d/jenkins.list'
sudo apt update
sudo apt install jenkins -y
```
- **Explanation:**
  Adds Jenkins' repository, updates the package list, and installs Jenkins.

b. Start Jenkins Service:
```
sudo systemctl start jenkins
sudo systemctl enable jenkins
```
- **Explanation:**
  Starts Jenkins and enables it to run on system boot.
  Access Jenkins via **http://localhost:8080**.

c. Unlock Jenkins:
```
sudo cat /var/lib/jenkins/secrets/initialAdminPassword
```
- Copy the displayed password → paste it into the Jenkins setup page to unlock.
- **Explanation:**
  This step unlocks Jenkins for initial setup.
- Install **suggested plugins** during setup.

4. Integrate SonarQube with Jenkins

Why?

To ensure **static code analysis** is part of the **CI/CD pipeline**, integrating SonarQube with Jenkins allows **automated code quality checks** every time code is built.

Steps:

a. Install SonarQube Scanner Plugin in Jenkins:

- Jenkins Dashboard → **Manage Jenkins → Manage Plugins → Available** tab.
- Search for **SonarQube Scanner**, install it, and restart Jenkins.

b. Configure SonarQube Server in Jenkins:

- Jenkins Dashboard → **Manage Jenkins → Configure System**.
- Scroll to **SonarQube servers**.
- Add a new server:
  - **Name:** My SonarQube Server
  - **Server URL:** http://localhost:9000
  - **Authentication Token:** Generate it in SonarQube (**My Account → Security → Generate Tokens**).
- **Explanation:**
  This allows Jenkins to connect securely with SonarQube for analysis.

5. Create a CI/CD Pipeline Integrating SonarQube

Two Options:

1. **Freestyle Project**
2. **Pipeline Project (Jenkinsfile)**

For Freestyle Project:

- **New Item → Freestyle project → Source Code Management (Git)** → Enter repository URL.
- **Build Step → Execute SonarQube Scanner**.

Create a **sonar-project.properties** file in the project's root:

```
sonar.projectKey=myproject
sonar.sources=src
sonar.host.url=http://localhost:9000
sonar.login=<Generated_Token>
```

For Pipeline Project (Recommended):

- **New Item → Pipeline → Jenkinsfile:**

```
pipeline {
    agent any
    tools {
        maven 'Maven3'          // Maven tool defined in Jenkins
    }
    stages {
        stage('Build') {
            steps {
                sh 'mvn clean install'
            }
        }
        stage('SonarQube Analysis') {
            steps {
                withSonarQubeEnv('My SonarQube Server') {
                    sh 'mvn sonar:sonar'
                }
            }
        }
    }
}
```

- **Explanation:**
  - **Build Stage:** Compiles and packages the application using Maven.
  - **SonarQube Analysis Stage:** Runs **SonarQube scanner** integrated with Maven to analyze the source code.

6. Run the Pipeline

- Trigger the Jenkins job manually or automatically (via webhooks).
- Check the SonarQube dashboard (**http://localhost:9000**) for analysis results (bugs, code smells, vulnerabilities).

Outcome:

- This setup ensures **automated static code analysis** with **every code build**, improving **code quality** and **security**.
- SonarQube reports help **detect and fix issues early**, reducing **technical debt**.
- Jenkins orchestrates the **continuous integration** process, automating build and analysis.