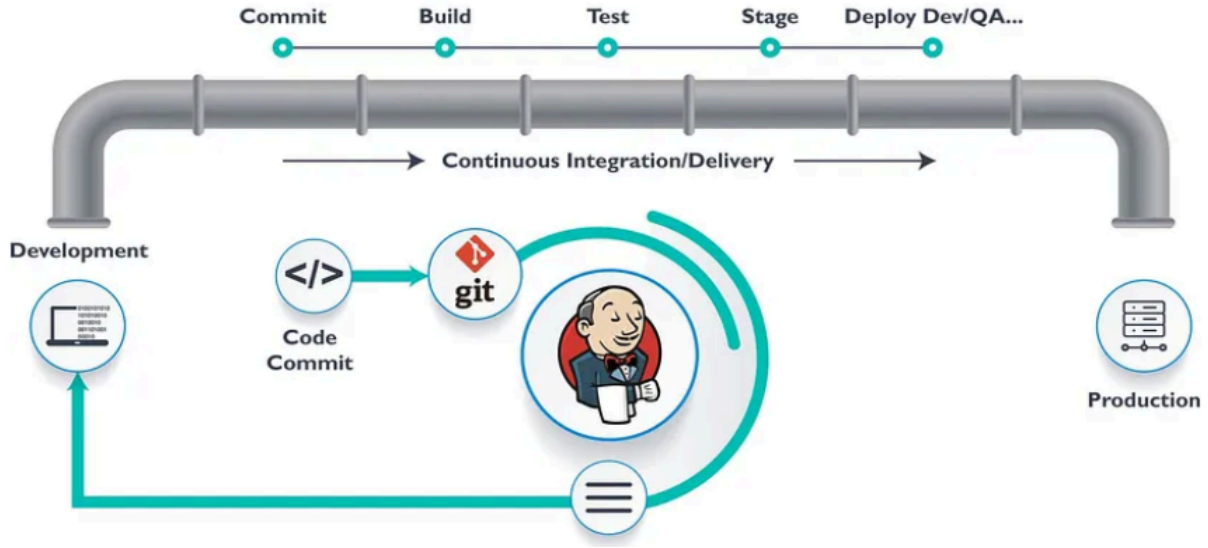
Name : Nidhi Rijhwani
Subject : Adv Devops Exp No 08

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 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

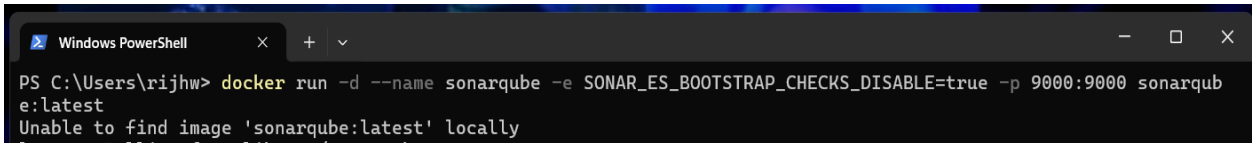
- Sustainability - Reduces complexity, possible vulnerabilities, and code duplications, optimizing the life of applications.
- Increase productivity - Reduces the scale, cost of maintenance, and risk of the application; as such, it removes the need to spend more time changing the code.
- Quality code - 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:

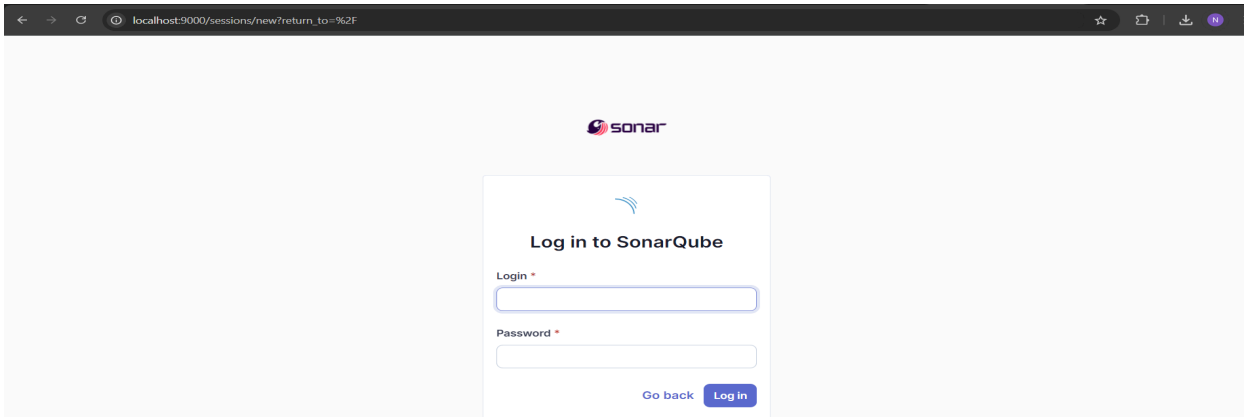
- Prerequisites:
    - Jenkins installed
    - Docker Installed (for SonarQube)
    - SonarQube Docker Image

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

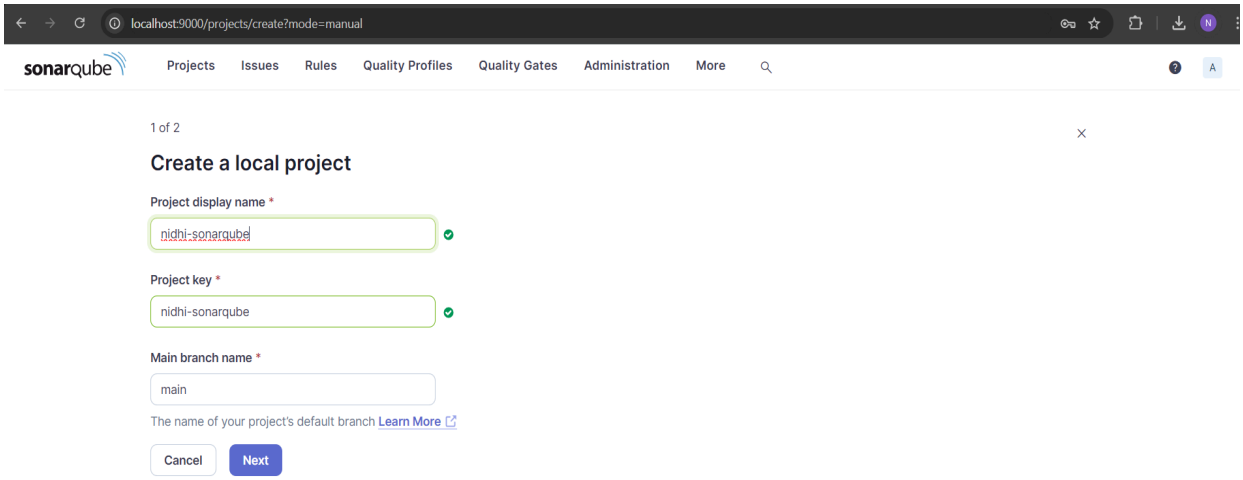
1. Open up Jenkins Dashboard on localhost, port 8080 or whichever port it is at for you.
2. Run SonarQube in a Docker container using this command.



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

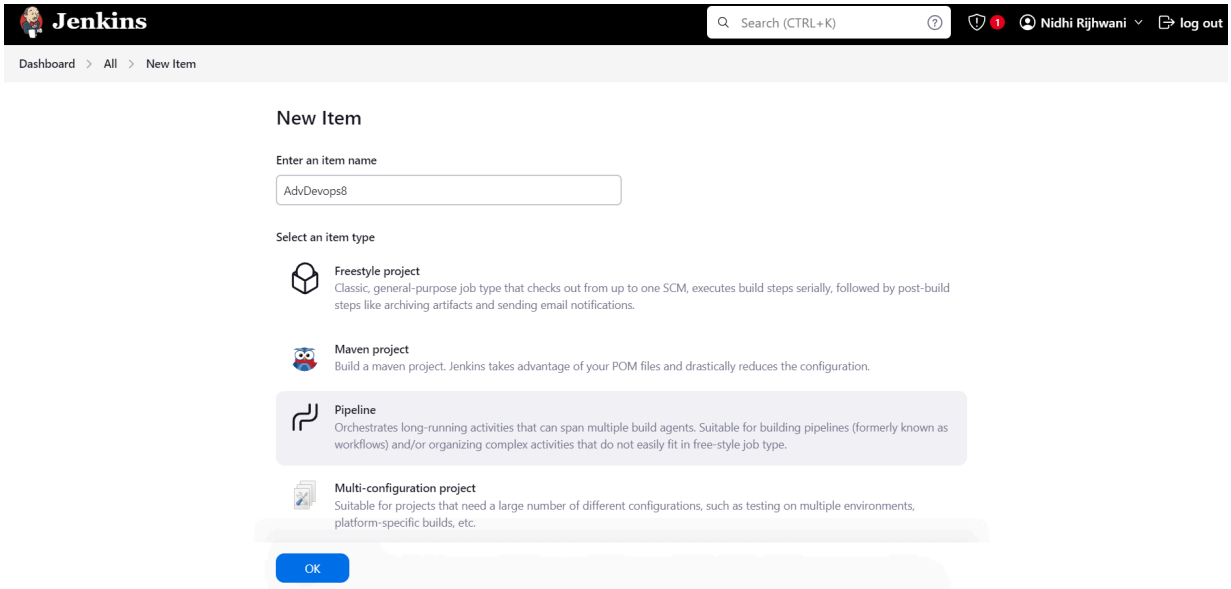


4. Login to SonarQube using username admin and password admin.
5. Create a manual project in SonarQube with the name nidhi-sonarqube.



Setup the project and come back to Jenkins Dashboard.

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



7. Under Pipeline Script, enter the following : make changes according to your project.

```
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 > AdvDevops8 > Configuration

Configure

- General
- Advanced Project Options
- Pipeline

Pipeline

Definition

Pipeline script

Script

try sample Pipeline...

```
node {
    stage('Cloning the GitHub Repo') {
        git 'https://github.com/shazforiot/GOL.git'
    }

    stage('SonarQube analysis') {
        withSonarQubeEnv('sonarqube') { // 'sonarqube' is the name of the SonarQube server configured in Jenkins
            sh "sonar-scanner \
                -D sonar.projectKey=nidhi-sonarqube \
                -D sonar.sources=. \
                -D sonar.exclusions=vendor/**,resources/**,**/*.java \
                -D sonar.host.url=http://localhost:9000/ \
                -D sonar.login=admin \
                -D sonar.password=Nidhi@123456"
        }
    }
}
```

☑ Use Groovy Sandbox

Pipeline Syntax

Save Apply

```
17:52:34.989 INFO  ANALYSIS SUCCESSFUL, you can find the results at: http://127.0.0.1:9000/dashboard?id=sonarqube
17:52:34.989 INFO  Note that you will be able to access the updated dashboard once the server has processed the submitted analysis report
17:52:34.989 INFO  More about the report processing at http://127.0.0.1:9000/api/ce/task?id=8c04859e-050f-41e5-b320-ee32d44d4259
17:52:43.609 INFO  Analysis total time: 16:49.875 s
17:52:43.612 INFO  SonarScanner Engine completed successfully
17:52:44.416 INFO  EXECUTION SUCCESS
17:52:44.419 INFO  Total time: 17:04.839s
[Pipeline] }
[Pipeline] // withSonarQubeEnv
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Declarative: Post Actions)
[Pipeline] echo
Pipeline completed successfully!
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

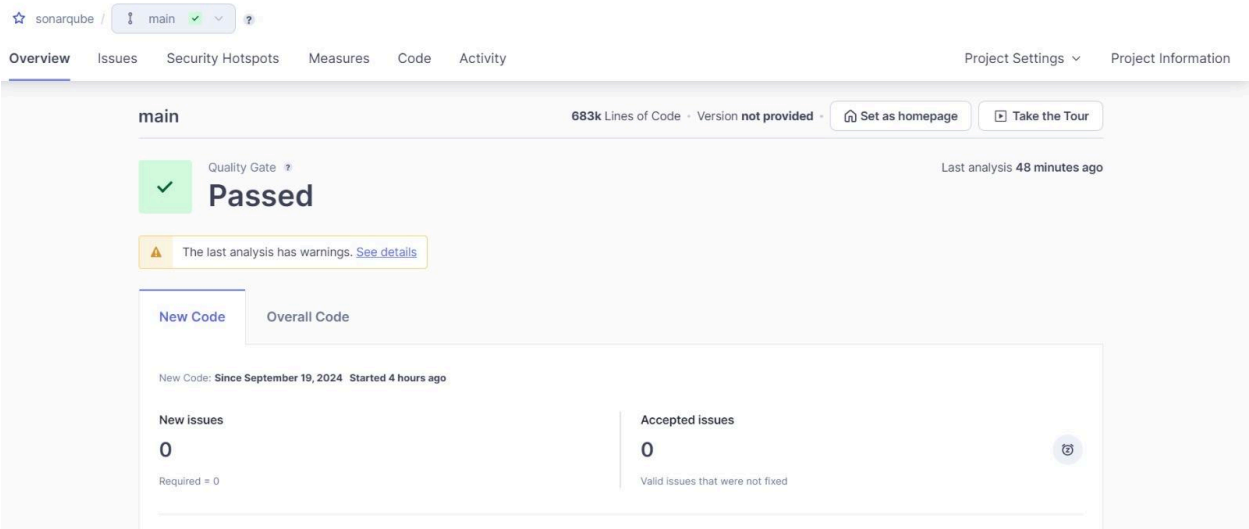
Script

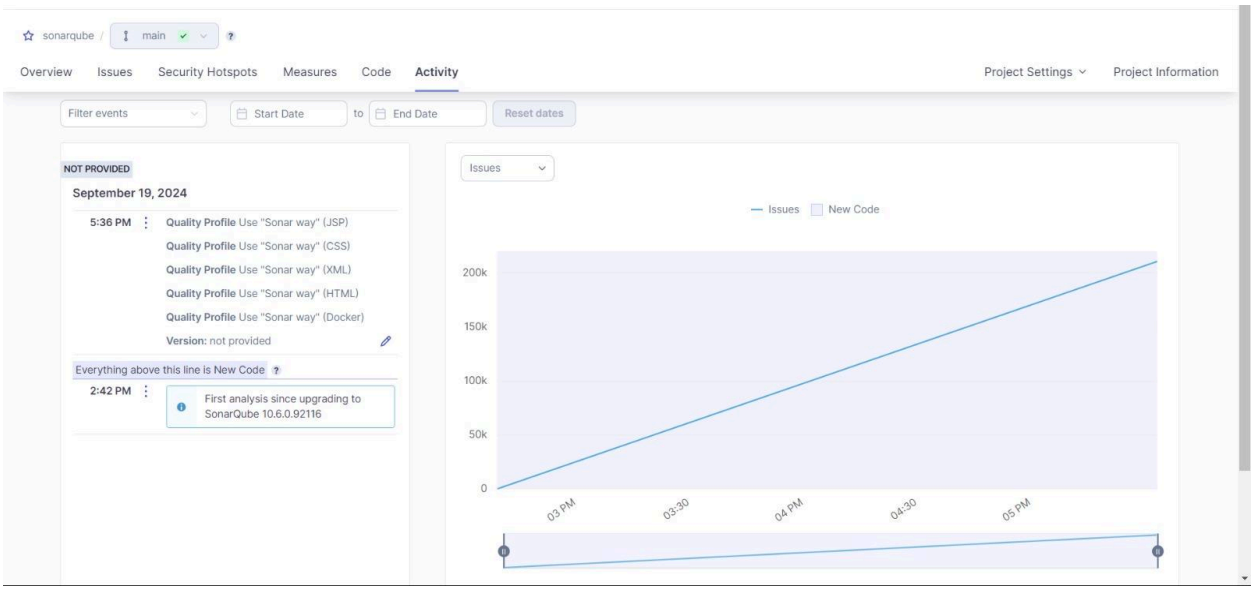
try sample Pipeline...

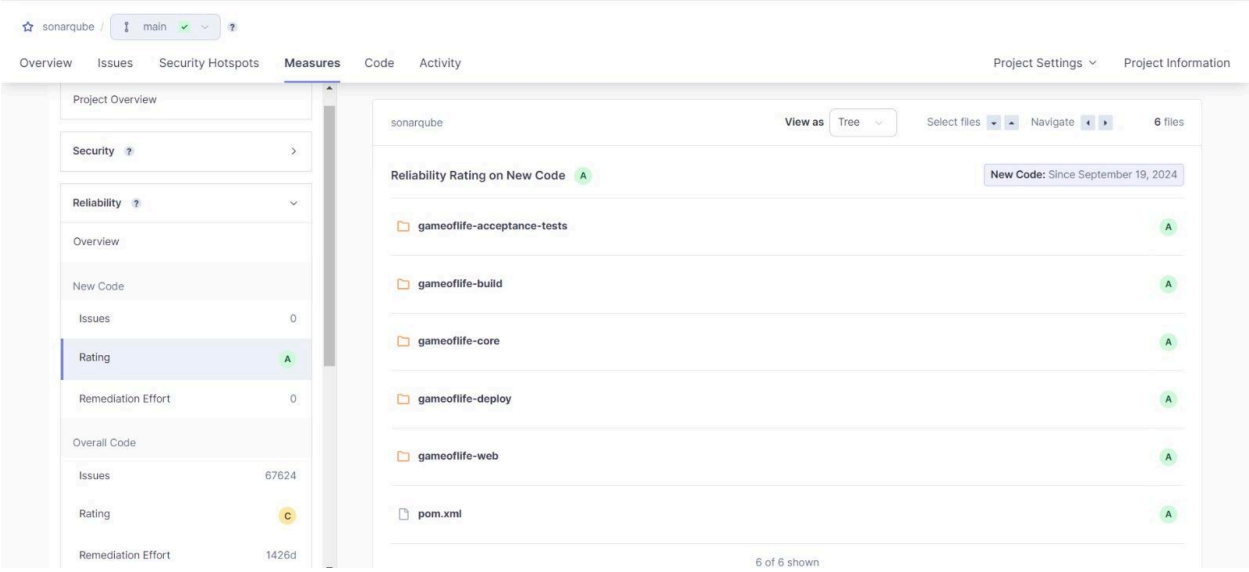
```
stage('SonarQube analysis') {
    steps {
        withSonarQubeEnv('SonarQube') {
            bat "D:\\sonar-scanner-6.1.0.4477-windows-x64\\bin\\sonar-scanner.bat ^
                -Dsonar.login=admin ^
                -Dsonar.password=Nidhi123456- ^
                -Dsonar.projectKey=sonarqube ^
                -Dsonar.exclusions=vendor/*,resources/,/.java ^
                -Dsonar.host.url=http://127.0.0.1:9000/"
        }
    }
}
```

☑ Use Groovy Sandbox

```
17:52:33.696 INFO  Analysis report compressed in 17399ms, zip size=29.6 MB
17:52:34.987 INFO  Analysis report uploaded in 1290ms
17:52:34.989 INFO  ANALYSIS SUCCESSFUL, you can find the results at: http://127.0.0.1:9000/dashboard?id=sonarqube
17:52:34.989 INFO  Note that you will be able to access the updated dashboard once the server has processed the submitted analysis report
17:52:34.989 INFO  More about the report processing at http://127.0.0.1:9000/api/ce/task?id=8c04859e-050f-41e5-b320-ee32d44d4259
17:52:43.609 INFO  Analysis total time: 16:49.875 s
17:52:43.612 INFO  SonarScanner Engine completed successfully
17:52:44.416 INFO  EXECUTION SUCCESS
17:52:44.419 INFO  Total time: 17:04.839s
[Pipeline] }
[Pipeline] // withSonarQubeEnv
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Declarative: Post Actions)
[Pipeline] echo
Pipeline completed successfully!
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```







Conclusion:

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