**SESSION - 80**

--> Connect sonarqube instances

--> create record on sonarqube

--> After connecting instance password will be there in

--> present sonarqube doing like this. Install a agent in sonarqube install plugins we are adding sonarqube scanner to the jenkins and we are configuring sonarqube server in jenkins.

**cicd-tools/readme.MD**

# Jenkins

\* Create Jenkins master, Jenkins agent EC2 instances. Update R53 records.

Install below plugins when you started Jenkins.

Plugins:

\* Pipeline stage view

\* Pipeline Utility Steps

\* AWS Credentials

\* AWS Steps

\* Rebuild

\* Ansi Color

\* SonarScanner

Restart Jenkins once plugins are installed

### Manage Credentials:

\* We need to add ssh credentials for Jenkins to connect to agent. I am using ID as ssh-auth.

\* We need to add aws credentials for Jenkins to connect with AWS for deployments. I am using

\* aws-creds

\* aws-creds-prod

\* aws-creds-dev

\* Connect to SonarQube EC2 instance from EC2 connect in AWS console UI.

\* We need to get security token from my account in sonarqube.

\* We need to add Sonar credentials for Jenkins to connect with Sonar Server. I am using sonar-auth as ID.

### Configure Master and Agent

### Configure SonarScanner and SonarQube Server

### Configure Jenkins Shared Libraries

\* Go to Manage Jenkins -> System

\* Find Global Trusted Pipeline Libraries section

\* Name as jenkins-shared-library, default version main and load implicitly

\* Location is https://github.com/daws-84s/jenkins-shared-library.git

Now Jenkins is ready to use.

### EKS

\* Make sure roboshop namespace exist.

\* make sure EKS control plane allows Jenkins agent to connect on port 443 HTTPS.

### SonarQube Quality Gates

🔴 Bugs

What it means:

Definite coding errors that will cause the program to behave incorrectly at runtime.

🟠 Code Smells

What it means:

Not actual errors, but poor coding practices that make code hard to maintain or read.

🟡 Coverage

What it means:

Percentage of code covered by automated tests (unit/integration tests).

🟣 Issues

What it means:

The total number of all problems detected = Bugs + Code Smells + Vulnerabilities.

🟢 Maintainability Rating

What it means:

A grade (A–E) showing how easy it is to maintain the code.

🔐 Security Rating

What it means:

Grade (A–E) for how secure the code is.

🛡️ Vulnerabilities

What it means:

Parts of the code that could be exploited by attackers.

Bugs = runtime errors.

Code Smells = maintainability issues.

Coverage = test percentage.

Issues = all detected problems.

Maintainability Rating = ease of maintaining code.

Security Rating = safety from attacks.

Vulnerabilities = exploitable weaknesses.

That’s a great achievement 👏 Let me put your experience into a story format (like you would tell in an interview, or write in your resume/LinkedIn post).

🚀 My SonarQube Implementation Journey

When I joined the project, we had no SonarQube integration in place. Code quality, coverage, and vulnerabilities were not being measured at all. I was assigned the responsibility of onboarding all project components into SonarQube and ensuring developers adopted the tool effectively.

Since it was a major shift, I decided to roll this out in stages over 3–4 months:

Phase 1 – Initial Setup (Month 1)

Configured SonarQube server and integrated it with Jenkins pipelines.

Added SonarQube scanner plugin and server configuration in Jenkins.

Created SonarQube projects and requested each team to add sonar-project.properties in the root of their repositories.

Ran the first scans to generate baseline reports of bugs, code smells, vulnerabilities, maintainability, and coverage.

Purpose: Give visibility to developers about the current state of their code without blocking them.

Phase 2 – Developer Adoption (Months 2–3)

Educated developers on how to interpret SonarQube reports.

Set expectations: they had 2.5 months to fix issues (bugs, vulnerabilities, code smells, maintainability, coverage).

Held weekly tracking sessions to monitor progress and help teams clear blockers.

Encouraged best practices (unit testing, removing duplications, refactoring long methods, fixing security flaws).

Phase 3 – Enforcing Quality Gates (Month 3–4)

Introduced SonarQube Quality Gates in the CI pipeline.

Initially configured the pipeline to show warnings but not fail builds.

Gave developers 2 weeks buffer to adapt to the gate requirements.

After 2 weeks, enabled strict enforcement: if the code failed the quality gate (bugs, coverage < threshold, security issues, etc.), the pipeline was aborted automatically.

Outcome

Within 3–4 months, we achieved:

Consistent code quality checks across all project components.

Higher unit test coverage (teams motivated to reach at least 80%).

Reduced vulnerabilities and better security posture.

Improved maintainability of codebase.

Developers became self-reliant in using SonarQube reports to improve their code.

Because of the impact on code quality and team adoption, I was recognized with the Best Performer Award in the project.

✨ Key Skills Demonstrated

CI/CD pipeline integration (Jenkins + SonarQube).

Change management (gradual adoption instead of forcing developers from day one).

Tracking and enforcing quality over time.

Collaboration with developers, ensuring code quality culture.

Security and maintainability improvements in large-scale codebases.

--> Try to connect sonarqube

--> sonar.daws84s.site

--> Login the sonarqube

--> admin and password

--> connect sonarqube server you will get password.

--> after this need to create one security

--> Need to check security token from sonar my account.

--> Connect to sonarqube visit EC2 instances from ec2 connect in AWS console UI.

--> Generate Tokens

--> Analysis Token

--> Create records also

--> if you are doing start and stop you need t update manually.

--> connect jenkins.

--> update sonar auth for new token.

Issues -->

Bugs -->

Maintainability Rating ->

Code Coverage ->

SAST

=====

Vulnerabilities

Code Smells

Security Rating

C1, C2

new code = C2-C1

Overall code = C1+C2

Overall code --> 100% --> 10000 lines

New code --> 0% -> 100 lines

Code Coverage -> 90%

1. Enable SonarScanner and Server in Jenkins.

2. Ask developers to onboard into sonarqube server.

3. We gave them 3 months time to clear all the backlog issues, bugs, vulnerabilities, code smells, mantaintability rating, security rating on overall code and new code.

1st week --> issues

2nd week --> vulnerabilities

3rd week --> bugs

4th week --> mantaintability rating

4. We enabled quality gates and enabled in the pipeline too using webhook but initally not aborting pipeline.

5. After 2 weeks we started aborting pipeline if quality gates.

**Open source library scan**

**=========================**

Nexus IQ --> enterprise level scanning tool. We recently decommissioned this and started using github dependabot

We enabled dependabot in all our repos, we are checking the dependabot alerts in our pipeline, if we see high and critical alerts we are failing the pipeline...

curl -s -H \

-H "Accept: application/vnd.github+json" \

-H "Authorization: token <token>" \

https://api.github.com/repos/daws-84s/catalogue/dependabot/alerts/

--> tokens - jenkins-ci

--> curl -L \

-->

curl -s -H "Accept: application/vnd.github+json" -H "Authorization: token <token>" <https://api.github.com/repos/daws-84s/catalogue/dependabot/alerts>

**Catalogue/package.json**

{

"name": "catalogue",

"version": "1.0.0",

"description": "product catalogue REST API",

"main": "server.js",

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1"

},

"author": "SteveW",

"license": "Apache-2.0",

"dependencies": {

"body-parser": "^1.18.1",

"express": "^4.15.4",

"mongodb": "^4.7.0",

"pino": "^5.10.8",

"redis": "^3.1.1",

"express-pino-logger": "^4.0.0",

"pino-pretty": "^2.5.0",

"@instana/collector": "^1.132.2"

}

}

--> after pushing the code in jenkins 0 console output