Coming to the CI CD pipeline, we are using GitHub as a source Code Management Tool, where we are using **git flow** as a branching strategy having branches like Feature, Dev, QA and PROD and bugfix.

And Continuous Integration server as a Jenkins. I will be writing the Jenkins file. There are two types of Jenkins files are available. One is a **declarative pipeline** and another one is a **scripted pipeline**. So, we are using a **declarative pipeline** here. So completely, it was a **groovy script.** Initially, we will be integrating **Jenkins server** with **GitHub**, **SonarQube** and **Nexus antifactory**. So, whenever commit happens in **source code repository**, using **webhook method** it will automatically a **Jenkins maven build**.

**Maven** is kind of a build tool which is used to generate the **Package** by resolving the dependencies using **POM.xml file**. Once the **WAR file** is Built all the packaged **WAR** files will be moved to the **Jenkins target directory**.

In **Jenkins file** there are different stages, the first stage is like **source code checkout stage** where I need to declare my **GitHub credentials**. So, for **authentication** I have to configured my **GitHub username** and **password** in a **Jenkins** add credentials area, so that it will make the **authentication**.

Next stage is a **Maven build.** So, Maven is used to build **Java based** source code as **WAR file**. So, where I have to declaring some couple of **Maven goals**, like a clean package(It cleans all packages), **compile package**.

And the next stage is like a **SonarQube** stage. So, where I'm doing the **quality code check** for that one. So, some **threshold limits** we used to configure and also, we need to attain some plugins for this one. So, the plugins will be like a **PMD**, **Feinberg static code analysis collecto**r. So, these are the plugins I need to install into Jenkins for the **sonar cube testing**(**SonarQube** is an open-source platform developed by **Sonar Source** for **continuous inspection** **of code quality**. Sonar does **static code analysis**, which provides a detailed report of bugs, code smells, vulnerabilities, code duplications).

And another stage is like, **Nexus stage**, so where I can **store all my artifacts**. so, one more stage we are having that is a **Docker image stage** where I'm building my Docker images. So once the Docker image is built, we are storing all the images to the **ECR registry**, so we are using **AWS ECR**. So already I have integrated my **AWS ECR** with the Jenkins so that my Jenkins can able to find the exact **ECR repo** and it will be pushing the images to the **ECR**. another stage is like, it's completely a Docker container stage. So, it will be launching the container. And one more stage I'm having like, it will be removing the existing container. So, this is how my CICD pipeline works.

**Feinberg** is a **static code analysis collector** that is used to collect and analyse **code quality metrics** across an organization's codebase. It can help identify issues such as **security vulnerabilities**, code smells, and design flaws.

The main benefits of using Feinberg include:

**Consistent code quality:** Feinberg helps ensure consistent code quality across an organization's codebase. By collecting and analysing **code quality metrics**, it can help identify areas of improvement and provide feedback to developers on how to improve their code.

**Security:** Feinberg can identify **security vulnerabilities** in code, such as SQL injection, cross-site scripting (XSS), and other common security issues.

**Compliance:** Feinberg can help ensure compliance with industry standards, such as the **OWASP** Top 10 or **PCI DSS**, by identifying potential security issues that could cause a compliance violation.

**Cost savings:** By identifying and fixing issues early in the development cycle, **Feinberg** can help save costs associated with fixing issues later in the development process or after the software has been released.

Overall, **Feinberg** is a useful tool for organizations looking to improve the quality and security of their codebase, and for developers looking to improve their coding skills.

This plugin provides utilities for the **static code analysis plugins**. Jenkins can **parse** the results file from various **Code Analysis tools** such as **CheckStyle**, **FindBugs**, **PMD** etc. For each corresponding **code analysis tool**, a plugin in Jenkins needs to be installed.

Additionally the add-on plugin **Static Analysis Collector** is available that combines the individual results of these plugins into a single trend graph and view.

The plugins can provide information such as

The total number of warnings in a job

A showing of the new and fixed warnings of a build

Trend Reports showing the number of warnings per build

Overview of the found warnings per module, package, category, or type

Detailed reports of the found warnings optionally filtered by severity (or new and fixed)

Used mostly in Build and Post-Build actions in configuration!

**OWASP** is the **Open Web Application Security Project** helping website owners and security experts protect web applications from cyber-attacks.

The **OWASP DevSecOps** Guideline explains how we can implement a secure pipeline and use best practices to

�detect security issues (by design or application vulnerability) as fast as possible.

Steps to Implement OWASP in Basic Pipeline.

Scan git repositories for finding potential credentials leakage.

Static Application Security Test)

Interactive Application Security Testing)

DAST (Dynamic Application Security Test)

IaC Scanning (Scanning Terraform, HelmChart code to find misconfiguration)

Infrastructure scanning

Compliance check

Text

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated



In blue green deployment you already have an application version is 34 and your deploying a new version of 35. So firstly what you do is you point your load balancer to the application 34 which is deployed on your cluster then you install application 35 on the same cluster the new version on your CI/CD and what you do is shift your load balancer to the application certified so that no traffic is sent to application 34. SO you provide some grace time for the users that are on the application 34 and once the grace period is over, you just delete application 34.

Deploying your application is same as 34 and 35 but instead of directly pointing the load balancer to 35 what you do is limit your traffic to application , For 35 application you only send 10 % of traffic so like the modern day balancer like f5 or nginx or any load balancer using nginx, they call it as ratio based load balancing or weight based load balancing initially route only 10 % and probably switch 66.33 % and then finally 100 %. All the popular load balancers can do this..

Chart

Description automatically generated