Ce Jenkinsfile utilise une pipeline Jenkins pour contrôler la construction du code. Le pipeline comporte quatre étapes :

1. Checkout : Cette étape vérifie le code source à partir de votre système de contrôle de version. Dans cet exemple, j'ai utilisé **scm**, qui est une variable réservée qui représente le système de contrôle de version configuré pour le projet.
2. Build : Cette étape compile le code et crée des artefacts.
3. Test : Cette étape exécute les tests pour s'assurer que le code fonctionne correctement.
4. Deploy : Cette étape déploie le code sur un serveur. Dans cet exemple, le code est déployé uniquement si la branche est master.

On peut personnaliser ce Jenkinsfile en fonction de nos besoins spécifiques

On a ajouté une nouvelle étape nommée 'Package' avant l'étape 'Deploy'. Cette étape utilise la commande **mvn package** pour générer un fichier JAR ou WAR pour notre application. On doitvous assurer que la commande **mvn package** est correcte pour le projet.

Après la génération du package, nous avons ajouté une section "post" qui archive les fichiers de package enregistrés dans le répertoire **target**. Cela permet de stocker les packages générés en tant qu'artefacts Jenkins, ce qui peut être utile pour les tests ultérieurs et la distribution.

Avec cette étape ajoutée, Jenkins générera maintenant un package pour notre application chaque fois qu'une build est déclenchée, en plus d'exécuter les tests unitaires. Cela permet de distribuer facilement l’application aux autres membres de l’équipe

Nous avons ajouté une nouvelle étape nommée 'SonarQube Analysis' avant l'étape 'Publish'. Cette étape utilise la commande **mvn sonar:sonar** pour analyser le code source à l'aide de SonarQube et envoyer les résultats au serveur SonarQube. on doit d'abord configurer notre serveur SonarQube et installer SonarScanner sur notre machine Jenkins avant de pouvoir exécuter cette étape.

Dans cette étape, on a également utilisé la fonction **withSonarQubeEnv** pour configurer les variables d'environnement SonarQube. Cela permet à SonarQube de se connecter automatiquement auserveur SonarQube en utilisant les informations de connexion stockées dans Jenkins. On doit remplacer "SonarQube Server" par le nom de notre configuration de serveur SonarQube dans Jenkins.

Avec cette étape ajoutée, Jenkins exécutera maintenant automatiquement l'analyse du code source de notre application à chaque build réussie et enverra les résultats à notre serveur SonarQube pour une analyse plus approfondie de la qualité du code. Cela aide à identifier les problèmes de qualité de code et à améliorer la fiabilité et la sécurité de l’application.

La première difficulté qu’on a trouvé c’est l’installation et la configuration de Jenkins

Aussi pour la compilation on avait trouvé beaucoup d’erreur affiché dans le terminal