**How to Integrate SonarQube with Jenkins Pipeline**

In today’s fast-paced software development landscape, maintaining code quality is crucial to ensure reliability, security, and maintainability. Tools like SonarQube and Jenkins play a vital role in enabling teams to automate and enhance their development workflows. SonarQube provides static code analysis, helping identify bugs, vulnerabilities, and code smells, while Jenkins automates the build and deployment processes, making CI/CD pipelines efficient and consistent. Integrating SonarQube with Jenkins enables seamless code quality checks within the CI/CD pipeline, allowing developers to detect and resolve issues early in the development process. This article will guide you through the steps to integrate SonarQube with Jenkins, empowering your development pipeline with automated code quality analysis.

**Prerequisites**

* SonarQube and Jenkins should be installed

**Download SonarQube Plugins in Jenkins**

Download all the necessary plugins in the Jenkins which are mentioned below

Maven Integration plugin

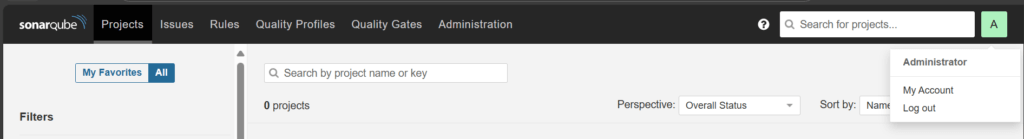
OWASP\_Dependency-checkin plugin

Sonar Quality Gates plugin

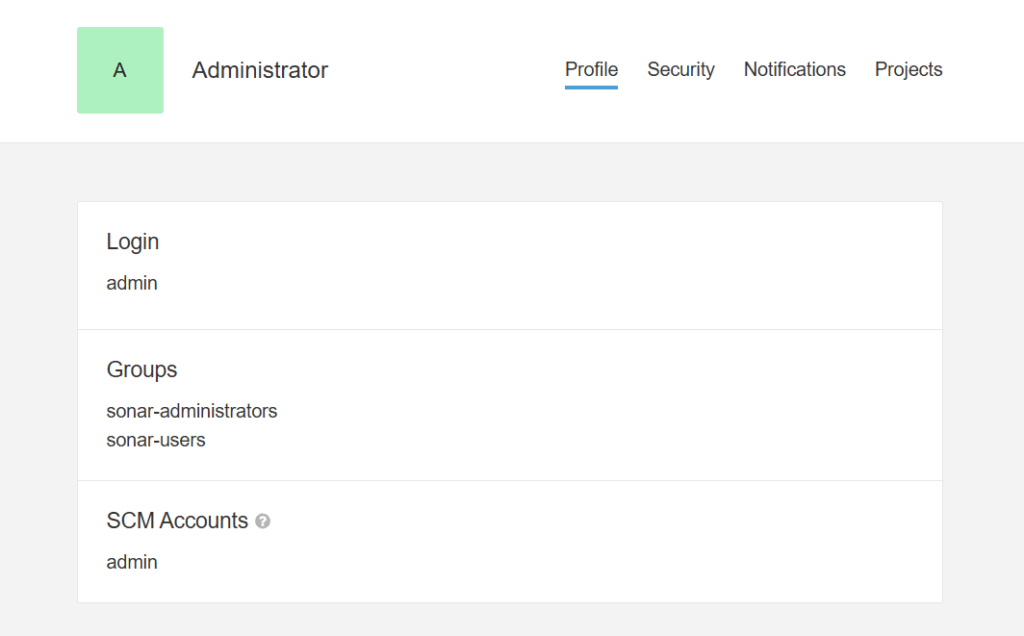
Sonarqube scanner for Jenkins

**Integrate SonarQube with Jenkins Pipeline**

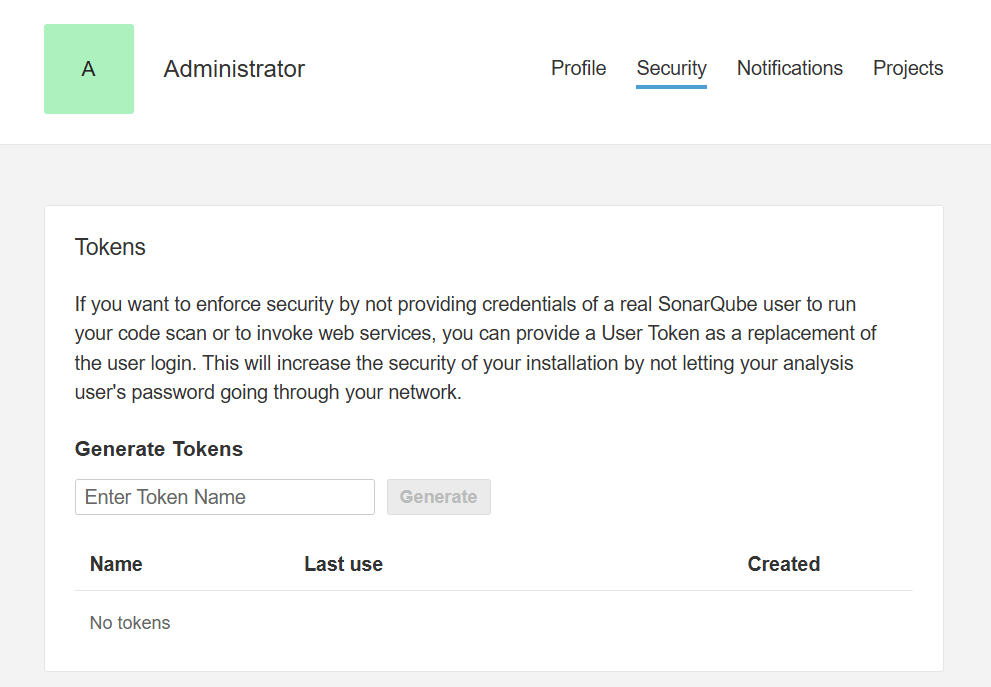
Create a token in SonarQube by logging into SonarQube. Select the Profile and choose MyAccount



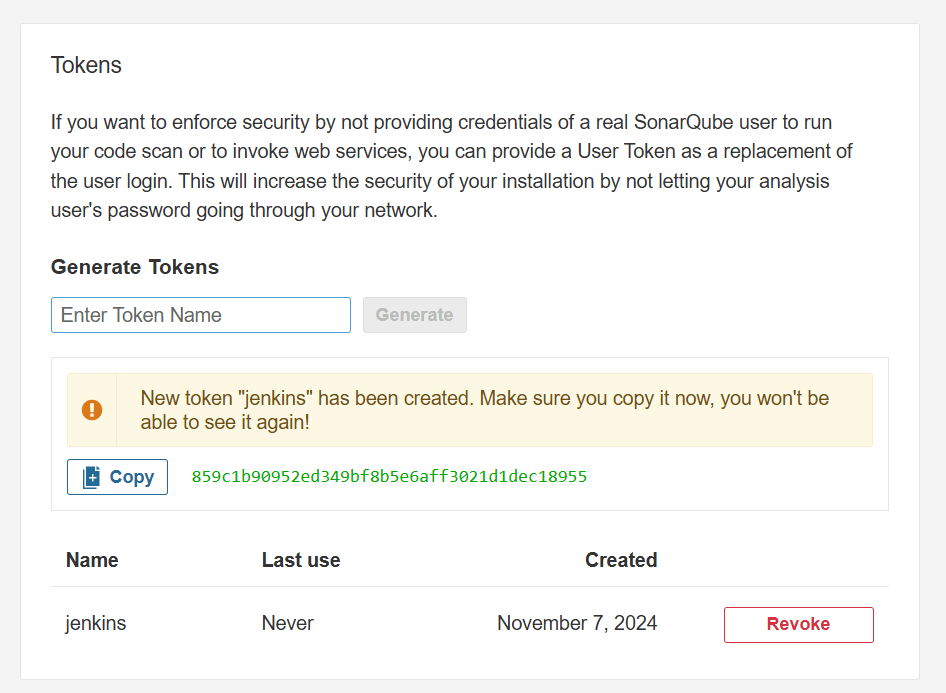
Click on Security



Give the name of the Token and click on generate



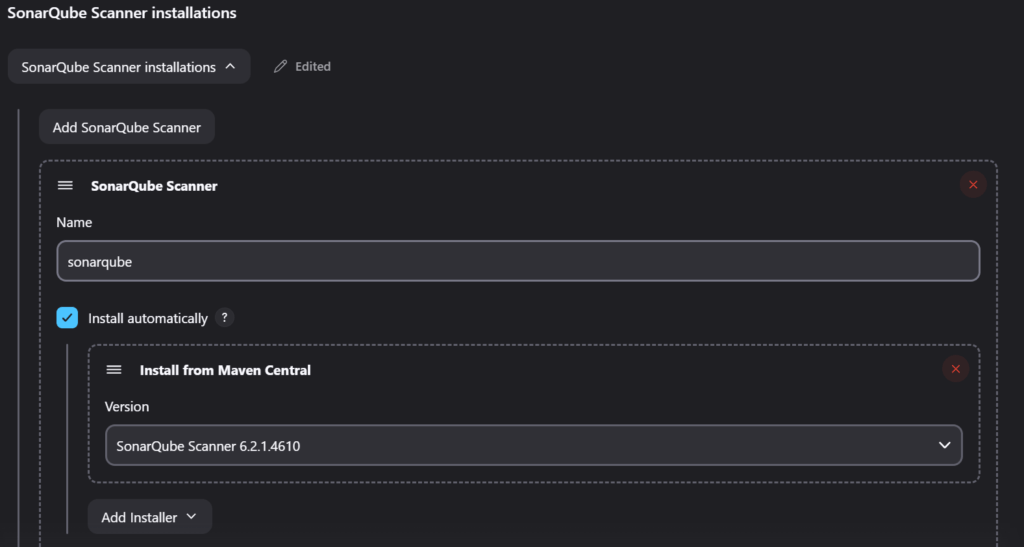
Copy the token



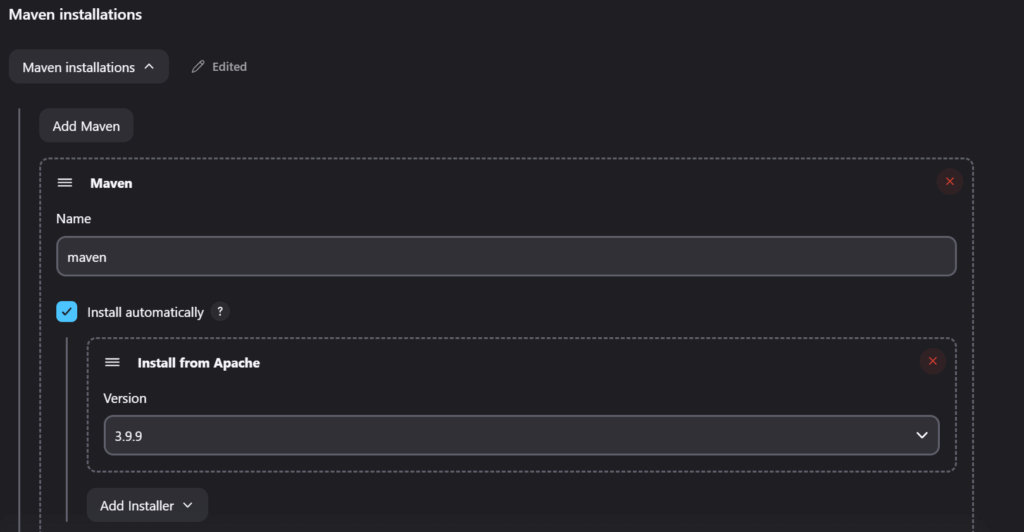
**Add SonarQube in Jenkins**

Go to Manage Jenkins > Tools

Add the SonarQube Scanner installation details:

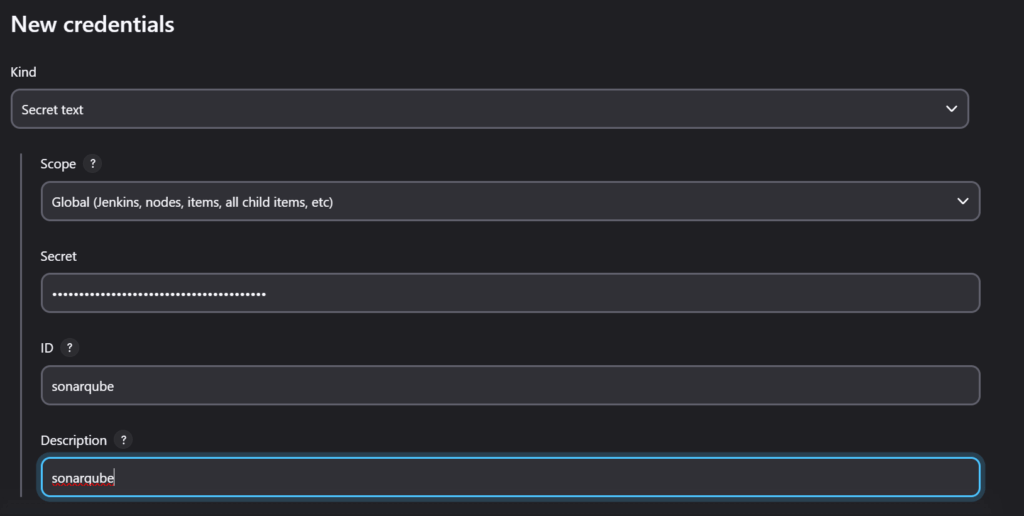


Add the Maven installation details:



Create a credential to connect Jenkins and SonarQube server.

Go to manage jenkins > Credentials > System > Global credentials (unrestricted) > Add Credentials

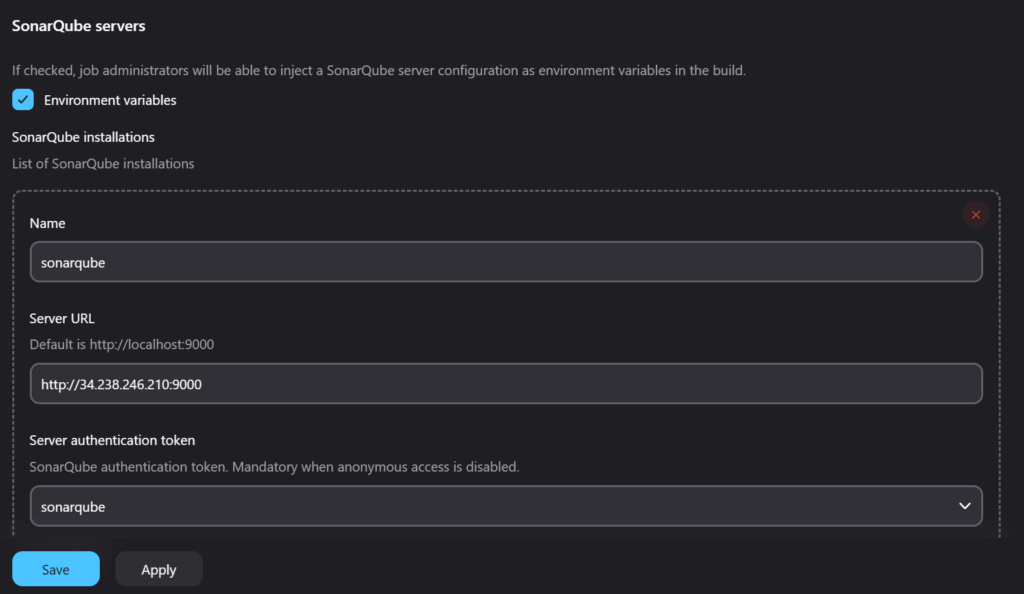


Credentials ID will be used in the Jenkins Pipeline code

**Establish the SonarQube server connections:**

Go to Manage Jenkins > System

Select the check-box of Injecting Environment variables, add the details of the SonarQube Server and also add the server authentication token:



**Write the Jenkins Pipeline code:**

Below is an example Pipeline code:

pipeline {

agent any

tools {

maven 'maven'

}

stages {

stage('Checkout Code') {

steps {

git 'https://<your GitHub repository url>' // Use your Git repository URL

}

}

stage('Build') {

steps {

sh 'mvn compile'

}

}

stage('Test') {

steps {

sh 'mvn test'

}

}

stage('SonarQube Analysis') {

environment {

SONAR\_HOST\_URL = 'http://34.238.246.210:9000' // Replace with your SonarQube URL

SONAR\_AUTH\_TOKEN = credentials('sonarqube') // Store your token in Jenkins credentials

}

steps {

sh 'mvn sonar:sonar -Dsonar.projectKey=sample\_project -Dsonar.host.url=$SONAR\_HOST\_URL -Dsonar.login=$SONAR\_AUTH\_TOKEN'

}

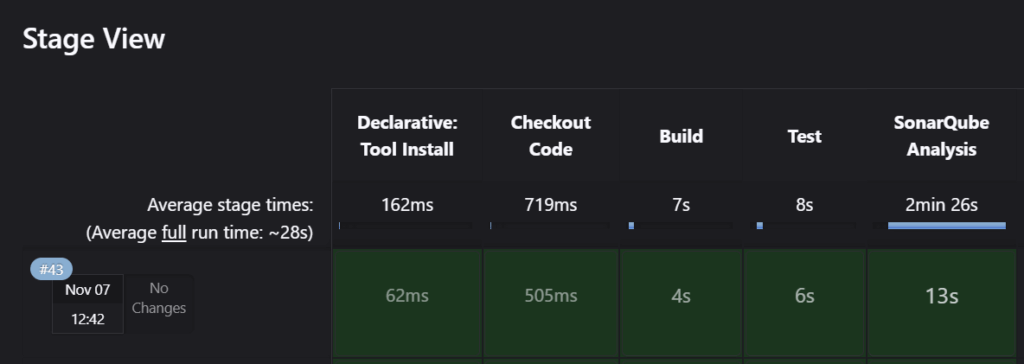
}

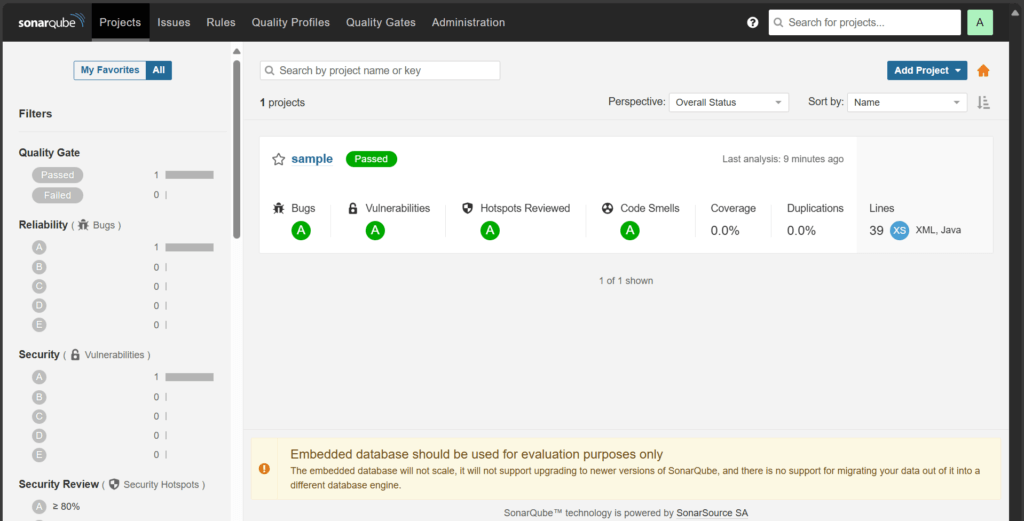
}

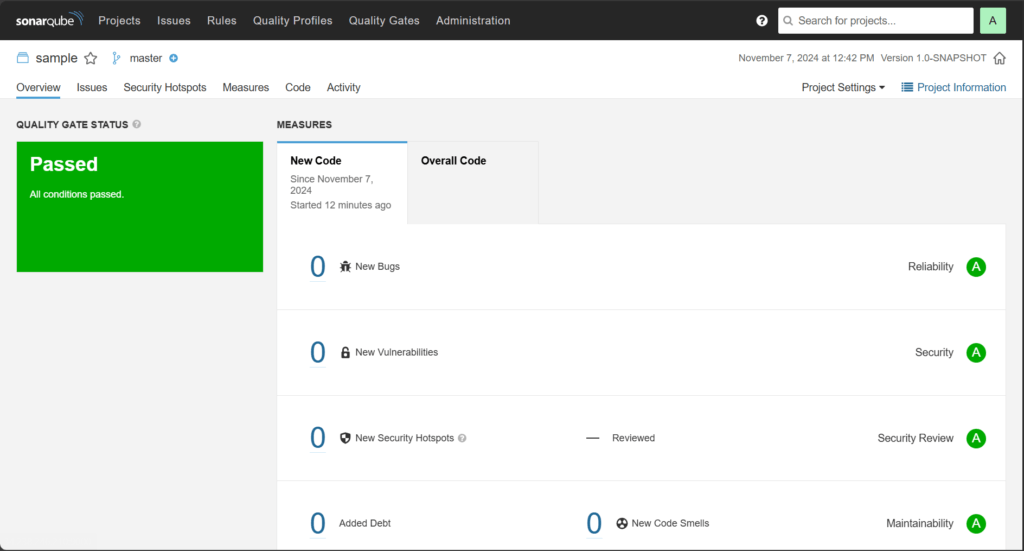
}

In the above Pipeline code modify the GitHub repository link, SONAR\_HOST\_URL and SONAR\_AUTH\_TOKEN with your Repository link, SonarQube url and credentials ID

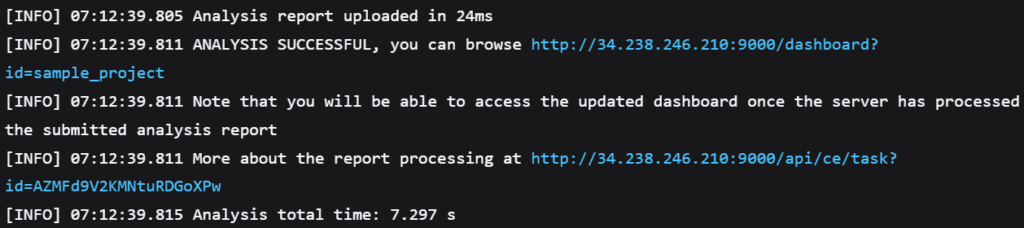
Once the pipeline code is built, the output will appear as shown in the Jenkins Pipeline Stage view:







The Jenkins console output will appear as follows:



**Conclusion:**

Integrating SonarQube with Jenkins is a powerful way to enhance your CI/CD pipeline with automated code quality analysis, enabling faster and more reliable software releases. This integration helps teams catch code quality issues early, maintain cleaner codebases, and improve overall project health. By combining Jenkins’ automation capabilities with SonarQube’s comprehensive code analysis, developers can confidently push updates knowing that quality checks are seamlessly built into the pipeline. With this setup, your team can focus more on innovation and less on resolving bugs and vulnerabilities, ultimately leading to more robust and secure software applications. Embracing this integration is a step toward fostering a culture of continuous improvement and quality within your development process.