

Project Report

on

CI/CD Pipeline using Jenkins and infusing security by performing code quality analysis.



Submitted in partial fulfillment for the award of Post Graduate
Diploma in High Performance Computing System
Administration from C-DAC ACTS (Pune)

Under the Guidance of

Mr. Ashutosh Das

Submitted by:

Anita A Khamitkar PRN: 220940127003

Ankush Kapoor PRN: 220940127032

Kamble Rohini PRN: 220940127038

Mahendra Kumar Pankaj PRN: 220940127044

Numesh Kumar Sahare PRN: 220940127047

TABLE OF CONTENTS

1. ABSTRACT
2. OVERVIEW OF THE PROJECT
3. JENKINS
4. SONARQUBE
5. Procedure
 - 5.1 Cloning GIT Repository.
 - 5.2 Maven Build.
 - 5.3 Code Review using SonarQube.
 - 5.4 Uploading Artifacts using Nexus Server.
 - 5.5 Deployment using Tomcat Server.
6. Results
7. Conclusion

1. ABSTRACT

Continuous Integration and Continuous Delivery (CICD) pipeline approach has increased the efficiency of projects. In agile, new features are introduced to the system in each sprint delivery, and although it may be well developed, the delivery failures are possible due to various security issues. This project aims to implement a Continuous Integration/Continuous Deployment (CI/CD) pipeline with a strong focus on security. The project will utilize best practices and industry standards to ensure that code is continuously integrated, tested, and deployed in a secure manner. The pipeline will consist of several stages, including source code management, build, automated security testing, and deployment. Additionally, security measures such as vulnerability scanning, code analysis, and monitoring will be implemented throughout the pipeline to ensure that the application is secure at every stage of development. The outcome of this project will be a robust and secure CI/CD pipeline that can be used to rapidly deploy new features and updates while maintaining the highest level of security.

2. OVERVIEW OF THE PROJECT

The Project involves the creation of a complete pipeline solution using Jenkins and other related tools such as Git, SonarQube, Nexus and Tomcat server. The Project will automate the entire development process from code development, build, to code quality analysis and deployment.

The pipeline will be created using Jenkins pipeline as it will include necessary stages namely cloning the repo, build, code review and analysis, artifact creation and deployment.

The Project will utilize Git as a version control system, SonarQube for static code analysis, Nexus for artifact management and Tomcat server for application deployment.

The goal of the project is to improve the quality of code, increase development efficiency and deploy the final product with confidence.

3. JENKINS

Jenkins is a popular open-source automation server that can be used to implement Continuous Integration/Continuous Deployment (CI/CD) pipelines in software development projects. It provides a wide range of plugins and integrations with various tools and technologies, making it a flexible and powerful tool for implementing a CI/CD pipeline.

In a typical CI/CD pipeline, Jenkins can be used to automate the build, test, and deployment processes. It can integrate with source code repositories, such as Git, to automatically trigger a build when new code is pushed to the repository. Jenkins can then compile the code, run unit tests, and generate reports on the test results.

One of the key advantages of using Jenkins in a CI/CD pipeline is its ability to provide visibility and control over the entire pipeline. Jenkins can provide real-time feedback on the status of each stage in the pipeline, making it easier to identify and address any issues that arise. It can also be configured to send notifications and alerts to team members when a build fails or when an issue is detected. Overall, Jenkins is a valuable tool for implementing a robust and efficient CI/CD pipeline in software development projects. Its flexibility and extensive plugin library make it an ideal choice for teams looking to automate their build, test, and deployment processes while maintaining control and visibility over the pipeline.



Fig 3.1 Jenkins Logo

4. SONARQUBE

SonarQube is a popular open-source tool for code analysis and quality control in software development projects. It can be integrated into a Continuous Integration/Continuous Deployment (CI/CD) pipeline to ensure that code is thoroughly tested and meets certain quality standards before being deployed to production.

In a typical CI/CD pipeline, SonarQube can be used to analyse code for bugs, vulnerabilities, and code smells. It can also measure code complexity and maintainability, providing developers with insights on how to improve the overall quality of their code.

By integrating SonarQube into a CI/CD pipeline, developers can get real-time feedback on the quality of their code. This allows them to address issues and improve code quality before it is deployed to production, reducing the likelihood of bugs and security vulnerabilities.

SonarQube can also be configured to enforce certain quality standards and best practices, such as adherence to coding conventions or specific security guidelines. This helps ensure that code is consistent and maintains a certain level of quality across the entire project.

Overall, SonarQube is a valuable tool for implementing code analysis and quality control in a CI/CD pipeline. Its ability to provide real-time feedback on code quality and enforce certain standards helps ensure that code is thoroughly tested and meets certain quality standards before being deployed to production.



Fig 4.1 SonarQube Logo

5. PROCEDURE

The CI/CD pipeline will be created using Jenkins pipeline as it will include necessary stages namely cloning the repo, build, code review and analysis, artifact creation and deployment.

The Project will utilize Git as a version control system, SonarQube for static code analysis, Nexus for artifact management and Tomcat server for application deployment.

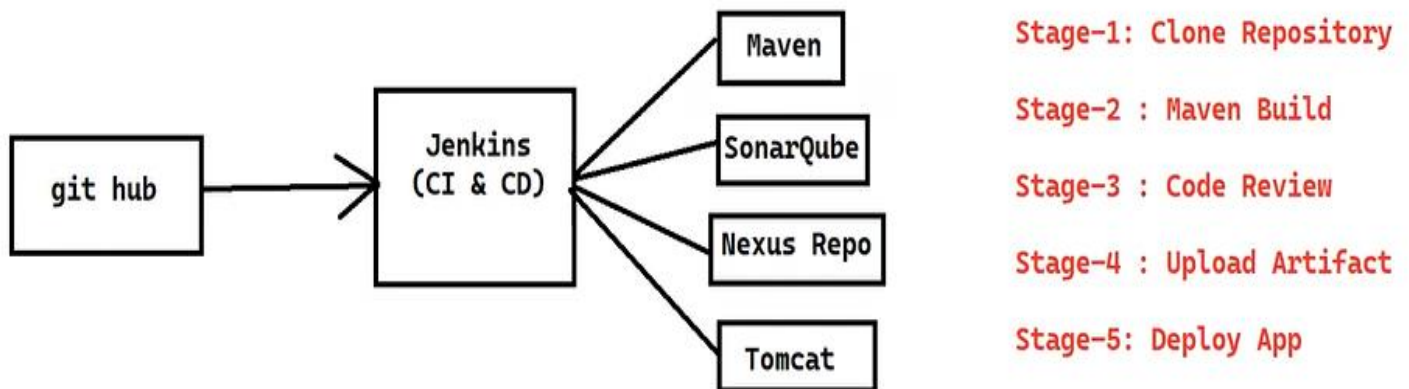


Fig 5.1 Flow of the Pipeline

5.1 Cloning GIT Repository.

First step involves logging-in with GitHub account, then create new repository for the project.

Now, copy the URL of the repository.

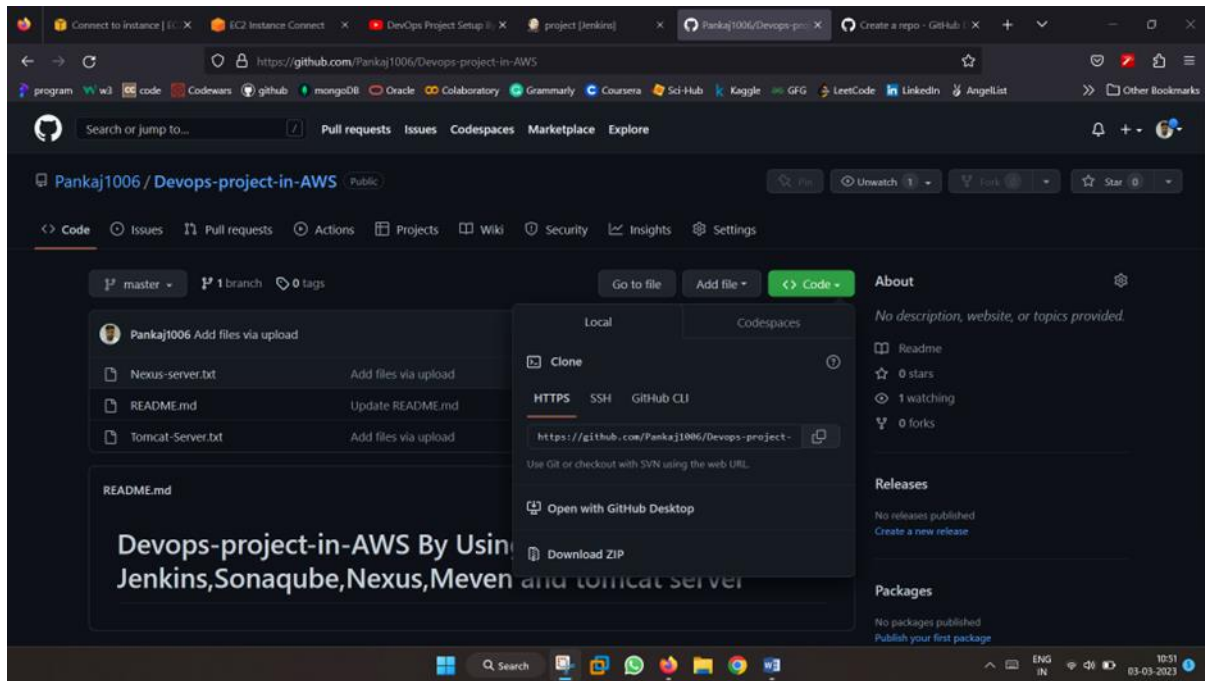


Fig 5.2 GitHub Repository

To configure(clone) GitHub repository you need to click pipeline syntax option given below on Jenkins pipeline script.

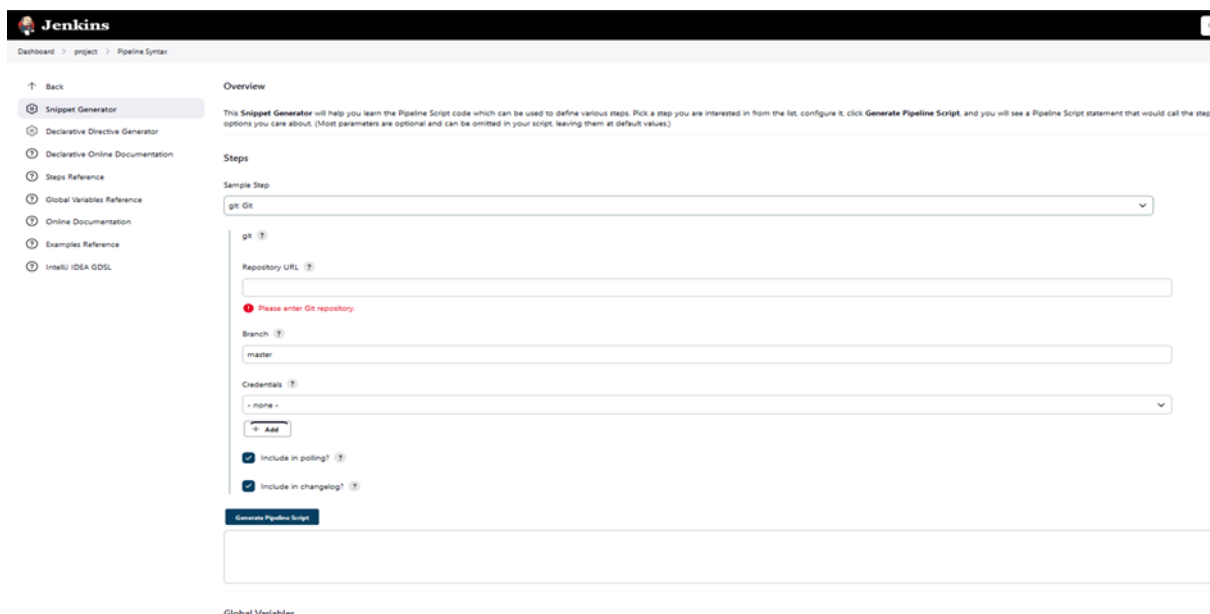
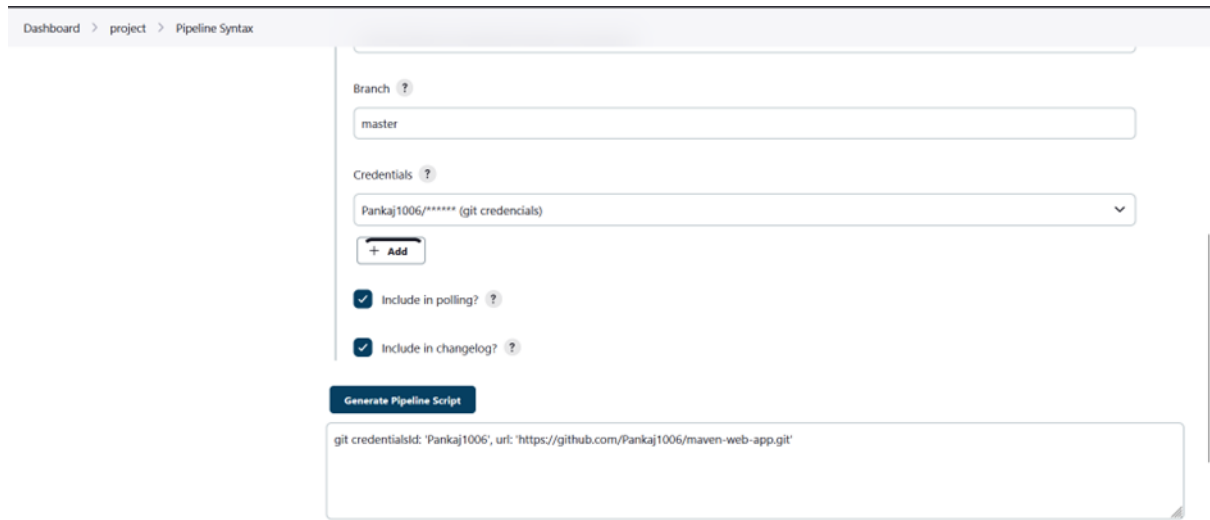


Fig 5.3 Snippet Generator for Pipeline Script code

Select Steps as Git, paste GitHub repository URL here and then click on the add credentials give username and password of your GitHub account.

Now Generate the Pipeline Script



The screenshot shows a web interface for configuring a pipeline step. At the top, there is a breadcrumb navigation: Dashboard > project > Pipeline Syntax. The main form has a 'Branch' dropdown set to 'master'. Below it is a 'Credentials' dropdown set to 'Pankaj1006/***** (git credencials)'. There is an '+ Add' button for credentials. Two checkboxes are checked: 'Include in polling?' and 'Include in changelog?'. A blue button labeled 'Generate Pipeline Script' is present. Below the button, a text box displays the generated script: 'git credentialsId: 'Pankaj1006', url: 'https://github.com/Pankaj1006/maven-web-app.git''.

Fig 5.4 Generating Pipeline Script

Copy the generated script and paste in pipeline script.

As the script shown below:

```
node{
    stage('Clone Repo'){
        git credentialsId: 'Pankaj1006', url: 'https://github.com/Pankaj1006/maven-
web-app.git'
    }
}
```

Apply & SAVE

5.2 Maven Build.

For setting up Maven as a global tool; Go to the Jenkins Dashboard Click on manage Jenkins here you have option called Global tool configuration.

Scroll down then select maven installations.

Enter name of the maven after that select required version of maven.

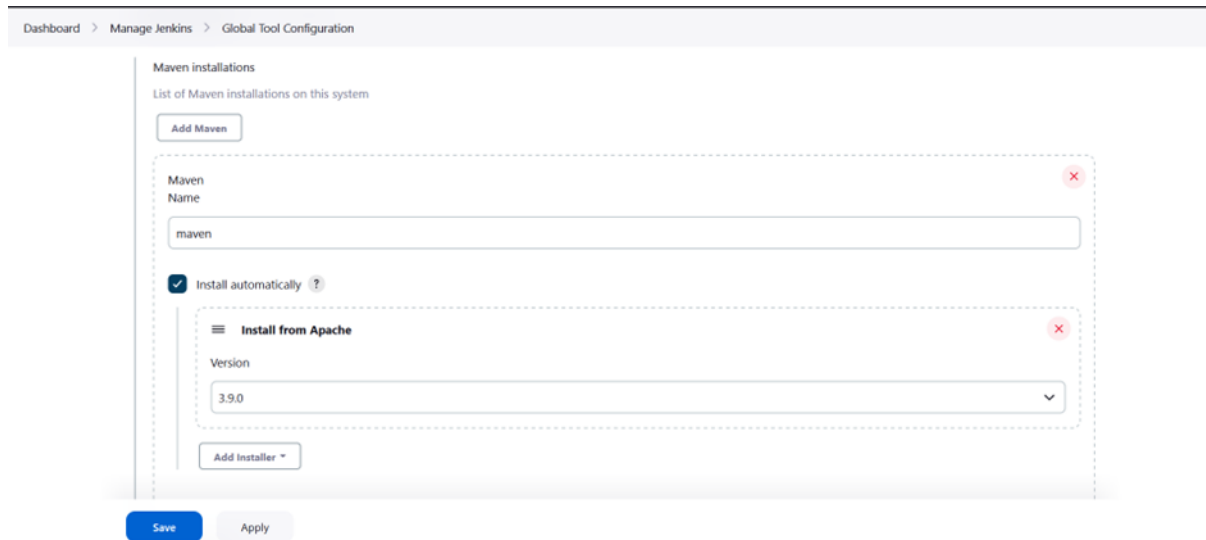
The screenshot shows the Jenkins 'Global Tool Configuration' page for 'Maven installations'. At the top, there's a breadcrumb trail: 'Dashboard > Manage Jenkins > Global Tool Configuration'. Below this, the section is titled 'Maven installations' with a subtitle 'List of Maven installations on this system'. There is an 'Add Maven' button. A dashed box contains a form for adding a new installation. The 'Maven Name' field is filled with 'maven'. The 'Install automatically' checkbox is checked. Below this, there's a section titled 'Install from Apache' with a 'Version' dropdown menu set to '3.9.0'. At the bottom of the dashed box is an 'Add Installer' button. At the bottom of the entire configuration area are 'Save' and 'Apply' buttons.

Fig 5.5 Maven Installation

Apply & SAVE

Now from the pipeline script at second stage we need to update in the Jenkinsfile that maven has been configured as a global tool.

```
stage('Maven Build'){  
    def mavenHome = tool name: "maven", type: "maven"  
    def mavenCMD = "${mavenHome}/bin/mvn"  
    sh "${mavenCMD} clean package"  
}
```

Apply & SAVE

5.3 Code Review using SonarQube.

In order to integrate SonarQube in Jenkins we have to configure plugin called “SonarQube Scanner Plugin”.

Go to manage Jenkins -> manage plugin -> Available plugin -> Install.

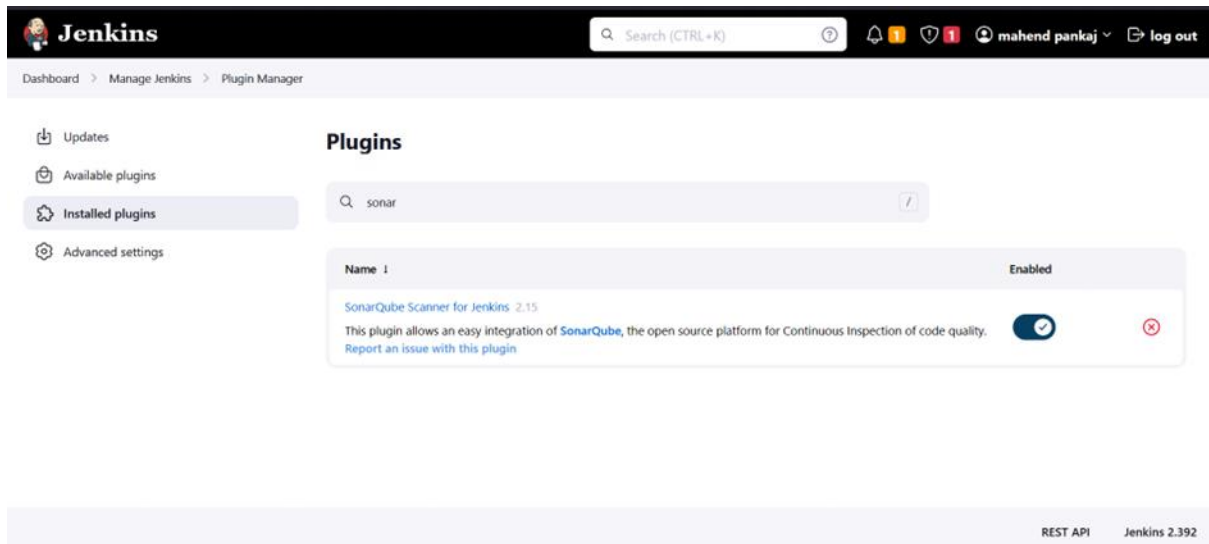


Fig 5.6 SonarQube Plugin installation in Jenkins

Now we need to configure sonar with our Jenkins. for that

Manage Jenkins -> configure systems -> SonarQube Server (Add SonarQube)



Fig 5.7 SonarQube Configuration

We can give any name <sonar_server>

Copy URL of the sonar server from the browser and paste in server URL field.

Now for Server authentication token click add.

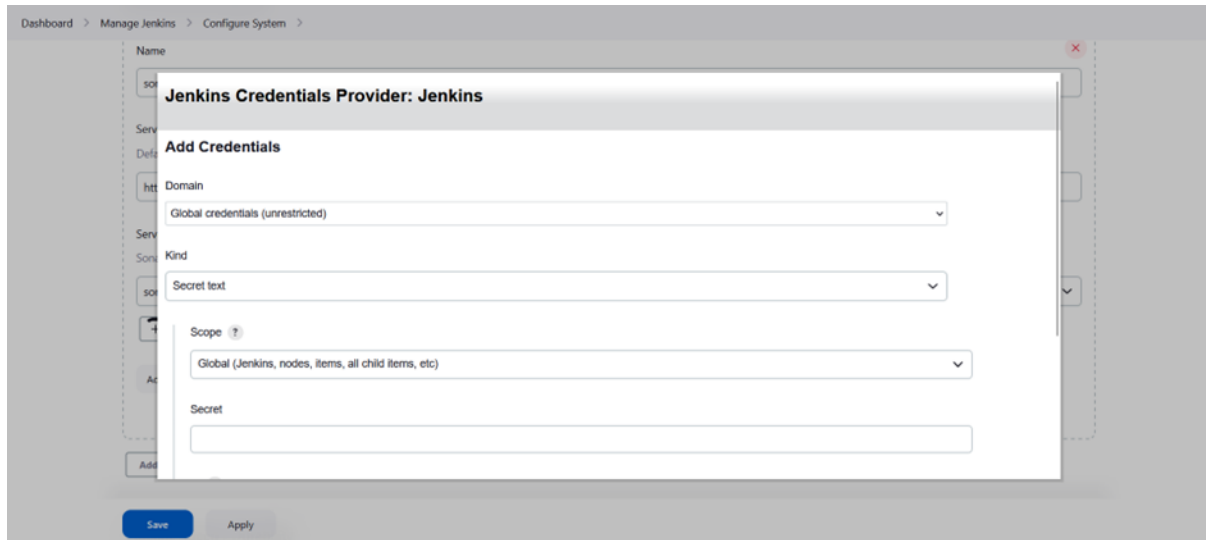


Fig 5.8 Secret Text for SonarQube Server

Note: Secret text we must go to the sonar server and configure token first then we must add here in this field provided.

Login to sonar server -> account -> security -> Generate token (copy)

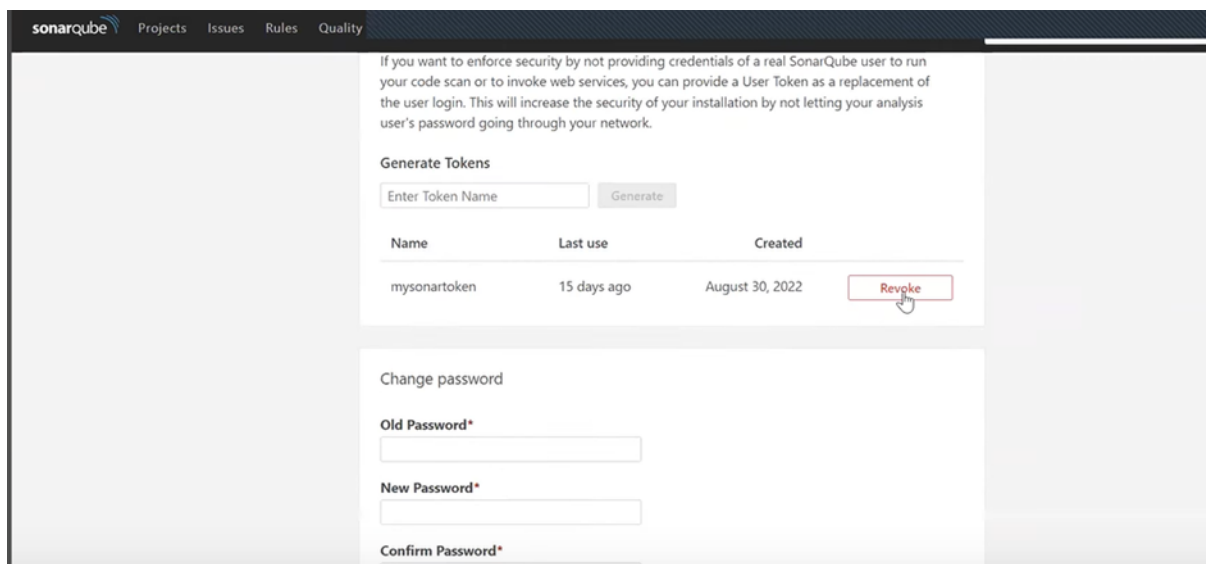


Fig 5.9 Token Generation

Paste as a secret text of credential.

Note: If not able to add credentials here then we need to configure this as a global credential.

(Manage Jenkins → credentials → Global → add credentials)

Now from configure system select the credential at drop-down.

Apply & SAVE

After this we need to add next stage as a 'Code Review' in a pipeline script.

We Click pipeline syntax -> select the option called withSonarQubeEnv: Prepare SonarQube scanner environment -> select authentication token -> Generate Pipeline Script.

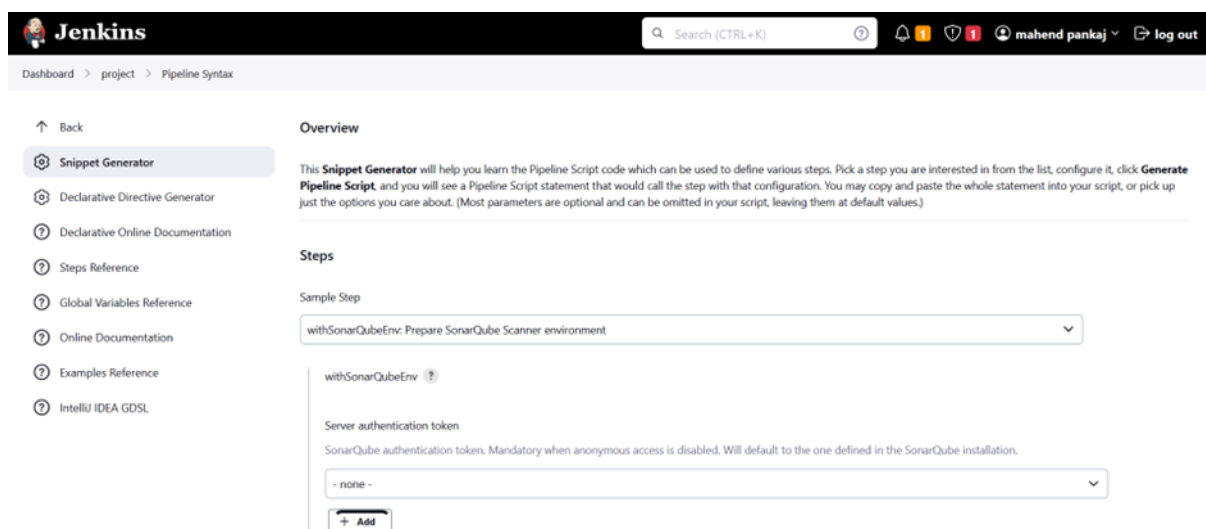


Fig 5.10 Script Generation for Sonar

Copy & paste the pipeline script stage in Jenkinsfile:

```
stage('Code Review'){  
    withSonarQubeEnv(credentialsId: 'sonar-token') {  
        def mavenHome = tool name: "maven", type: "maven"  
        def mavenCMD = "${mavenHome}/bin/mvn"  
        sh "mvn sonar:sonar"  
    }  
}
```

Apply & SAVE

Note: If we build the stages, we get new project report generated in SonarQube server with code review attached in Result Section.

5.4 Uploading Artifacts using Nexus Server.

Nexus Server:

A Nexus Server is a repository manager used to store and manage binary artifacts such as libraries, executables, and documentation. It is commonly used in Continuous Integration/Continuous Delivery (CI/CD) pipelines as a central hub for storing and sharing artifacts between different stages of the pipeline.

Why are we choosing Nexus?

Using a Nexus server in a CI/CD pipeline provides several benefits, including:

1. **Centralized artifact management:** All artifacts produced by the pipeline are stored in a central location, making it easy to manage and share them across different stages of the pipeline.
2. **Improved build performance:** The Nexus server caches artifacts, reducing build times and improving performance.
3. **Greater control and security:** The Nexus server provide access control, ensuring that only authorized users can access and modify artifacts.

Overall, the Nexus server is an essential tool for organizations looking to implement an efficient and streamlined CI/CD pipeline.

We need to integrate nexus in Jenkins to upload the artifacts.

Go to dashboard -> manage Jenkins -> manage plugin -> install Plugin called 'Nexus Artifact Uploader'.

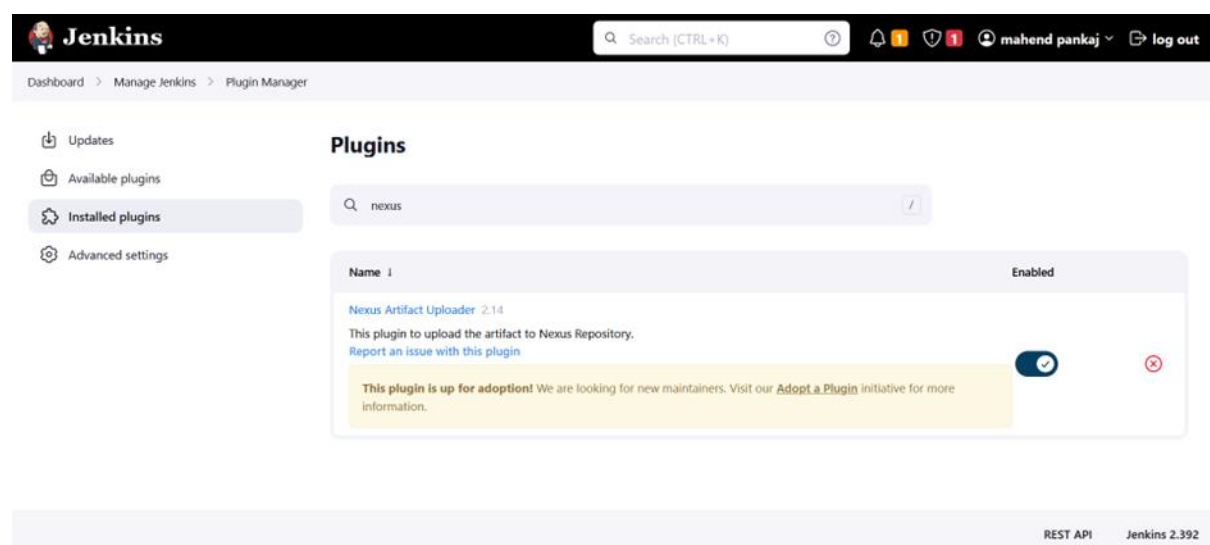


Fig 5.11 Nexus Artifact Uploader

Once the plugin installation is successfully then go to configure pipeline script for next stage.

Click the pipeline syntax to configure pipeline script for Nexus stage.

The screenshot shows the Jenkins Pipeline Syntax configuration page. The breadcrumb navigation at the top reads "Dashboard > project > Pipeline Syntax". On the left sidebar, there are links: "Steps Reference", "Global Variables Reference", "Online Documentation", "Examples Reference", and "IntelliJ IDEA GDSDL". The main content area is titled "Steps" and shows a "Sample Step" dropdown menu with "nexusArtifactUploader: Nexus Artifact Uploader" selected. Below this, the "nexusArtifactUploader" configuration section is visible, containing a "Nexus Details" subsection with three fields: "Nexus Version" (a dropdown menu), "Protocol" (a dropdown menu), and "Nexus URL" (a text input field with a help icon).

Fig. 5.12 Nexus Details in Jenkins

Select step called “nexusArifactUploader: Nexus Artifact Uploader”

Nexus version: we will see on Nexus page which version we have configure (NEXUS2/NUXUS3)

Protocol: http

Nexus URL: [≤public-ip](#) of nexus server>:8081

Credentials: we configured credentials username & password as admin.

Group id: (we can give value)

Version: 01-SNAPSHOT (version of your snapshot)

Repository: make new repository on nexus server.

Login into the Nexus server here we will create a new repository.

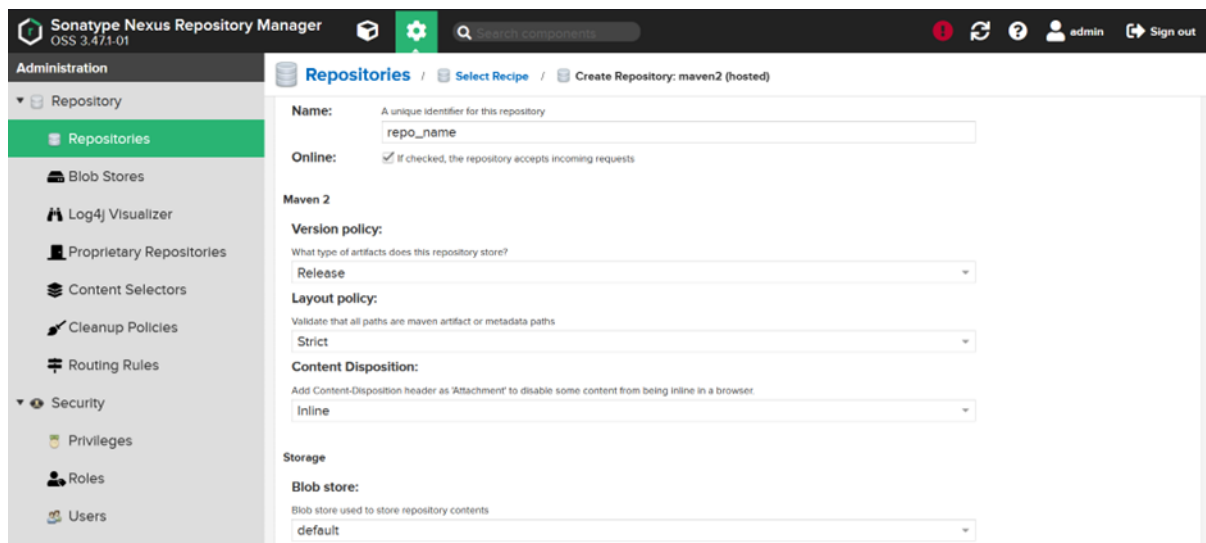


Fig. 5.13 Creation of New Repository in Nexus Server

Select maven2(hosted) and Development policy: Allow redeploy.

Create repository.

Now in repository field fill the repo_name as per created in nexus server.

Artifacts:

Artifactid: <01-maven-web-app> (is the project name that you want to use)

Type: war

Classifier:.....

File: target/<01-maven-web-app.war> (Location where the war file will be created)

Generate pipeline script.

We can just copy the generated syntax and paste in a pipeline script for next stage.

As shown below:

```
stage('Upload Artifact on Nexus'){  
    nexusArtifactUploader artifacts: [[artifactId: '01-maven-web-app',  
classifier: '', file: 'target/01-maven-web-app.war', type: 'war']], credentialsId:  
'nexus-credentials', groupId: 'in.project', nexusUrl: '52.66.204.71:8081',  
nexusVersion: 'nexus3', protocol: 'http', repository: 'project-snapshot', version:  
'2.0-SNAPSHOT'  
}
```

Apply & SAVE

Note: if we build this stage, we can get a snapshot created in Nexus server.

Figure attached in Result Section.

If this build is complete, it means our nexus server is successfully configured in Jenkins server.

5.5 Deployment using Tomcat Server.

To copy the war file generated in nexus server in Tomcat server we need to install plugin in Jenkins called ‘SSH Agent.’

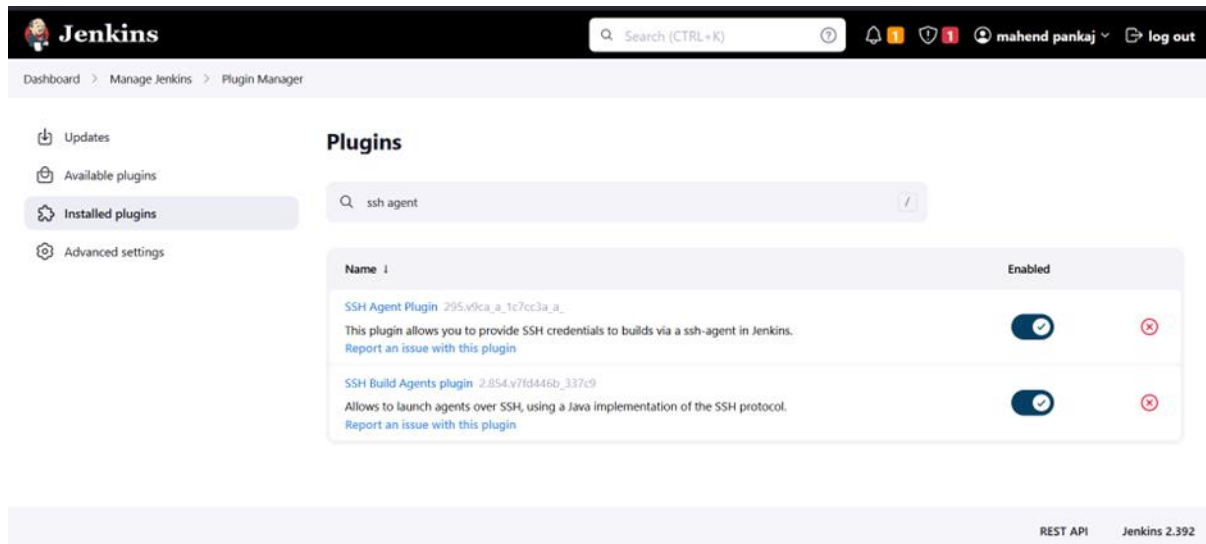


Fig 5.14 SSH Agent Plugin Installation

Now go to our pipeline script create a new stage and for that we need to configure pipeline syntax.

We Select step called “sshagent: SSH Agent” then configure credentials for tomcat server as follows:

Kind: SSH username and private key

Username: ec2-user

Private key: Here you need to copy your private key .pem file content.

Then select credentials.

The screenshot shows the Jenkins Pipeline Syntax configuration page. On the left is a sidebar with navigation links: Declarative Online Documentation, Steps Reference, Global Variables Reference, Online Documentation, Examples Reference, and IntelliJ IDEA GDSDL. The main area is titled 'Steps' and contains a 'Sample Step' dropdown menu set to 'sshagent: SSH Agent'. Below this, there is a configuration section for the 'sshagent' step. It includes a text input field for 'ec2-user (Tomcat-Server-Agent)' with a help icon, an '+ Add' button, and an unchecked checkbox for 'Ignore missing credentials' with a help icon. At the bottom of the configuration section is a blue 'Generate Pipeline Script' button. Below the button is a text area containing the generated pipeline script:

```
sshagent(['Tomcat-Server-Agent']) {
    // some block
}
```

Fig 5.15 Pipeline Script Generation for Tomcat Server

Now SSH Agent got successfully configured now we need to deploy the war file in tomcat server by using SSH agent configured in Jenkins server.

war file location: target/01-maven-web-app.war

(On Jenkins server)

destination location: home/ec2-user/apache-tomcat-9.0.72/webapps

(On tomcat server)

```
stage('Deploy'){
    sshagent(['Tomcat-Server-Agent']) {
        sh 'scp -o StrictHostKeyChecking=no target/01-maven-web-app.war
ec2-user@65.0.32.119:/home/ec2-user/apache-tomcat-9.0.72/webapps'
    }
}
```

Apply & SAVE

At this point our pipeline script is configured and completed successfully now we will click on Build Now in the Dashboard for Pipeline Results.

6. RESULTS

Result 1: Report Generation in SonarQube server Dashboard on successful scanning performed by Sonar Scanner

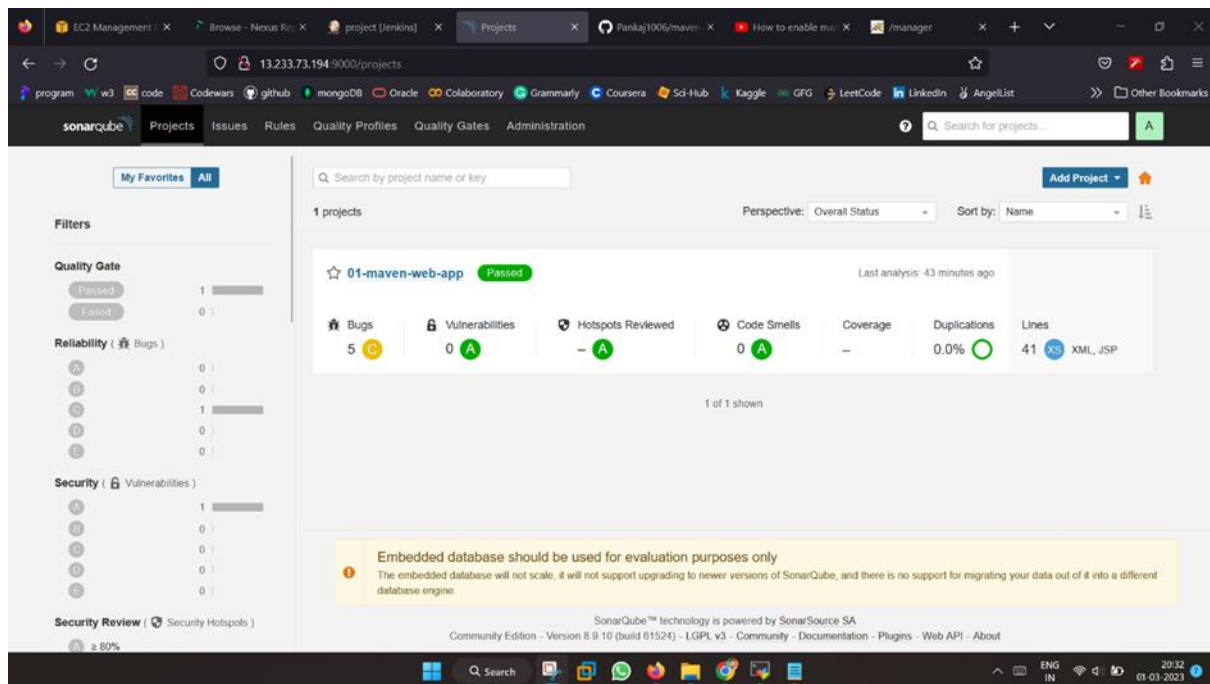


Fig 6.1 SonarQube Report Generated on SonarQube Server

Result 2: Artifacts are uploaded as war file on build and version change.

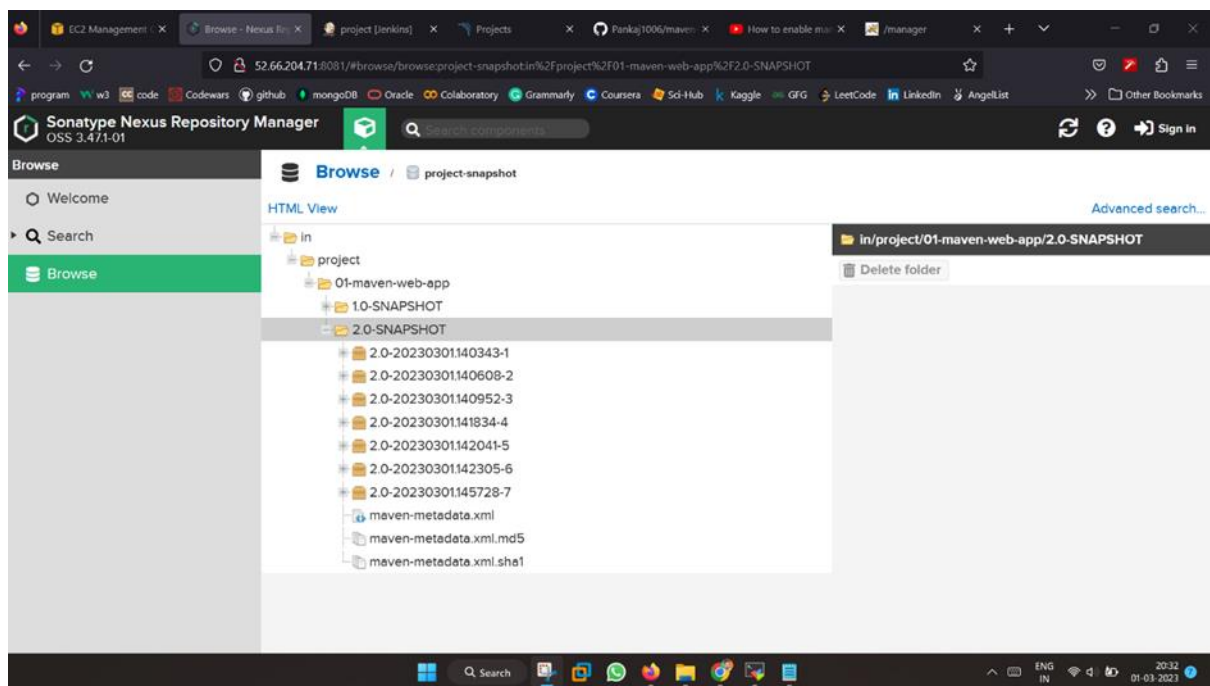


Fig 6.2 Snapshots Created (war file) in Nexus Server

Result 3: Green colour marks and signifies that the build is successful and pipeline is working properly without any interruption and with secure code.

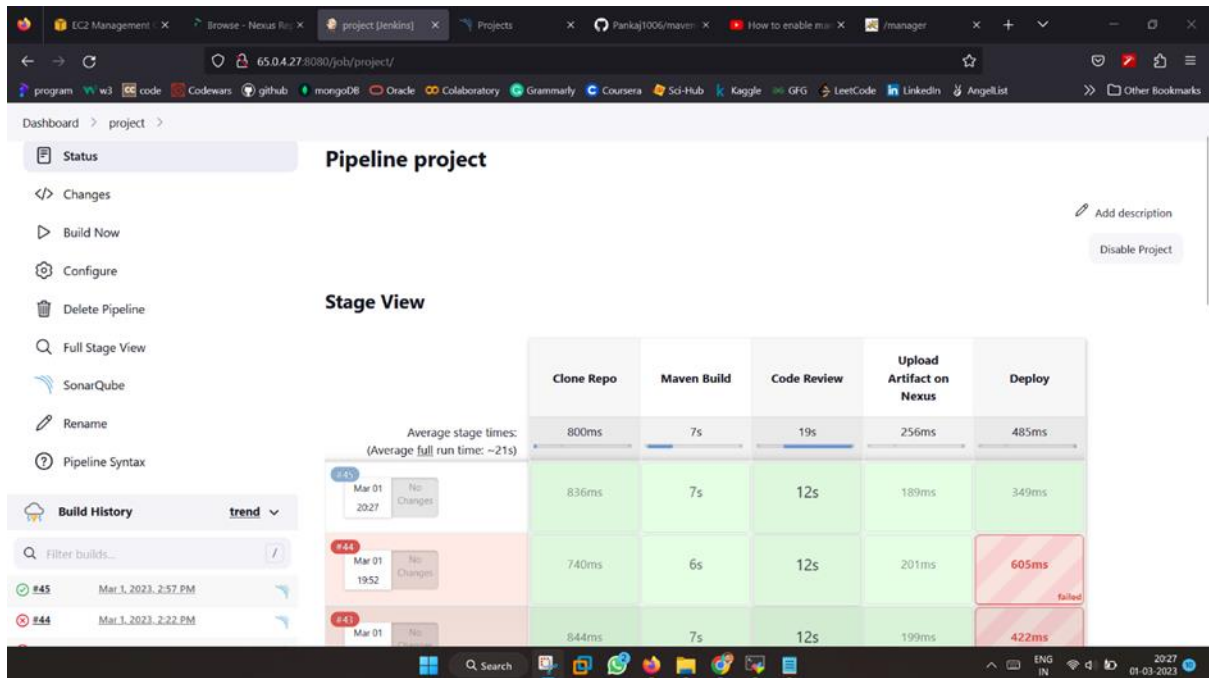


Fig 6.3 Successful Build of the different stages of Jenkins Pipeline

Result 4: Deployment is done on Tomcat server

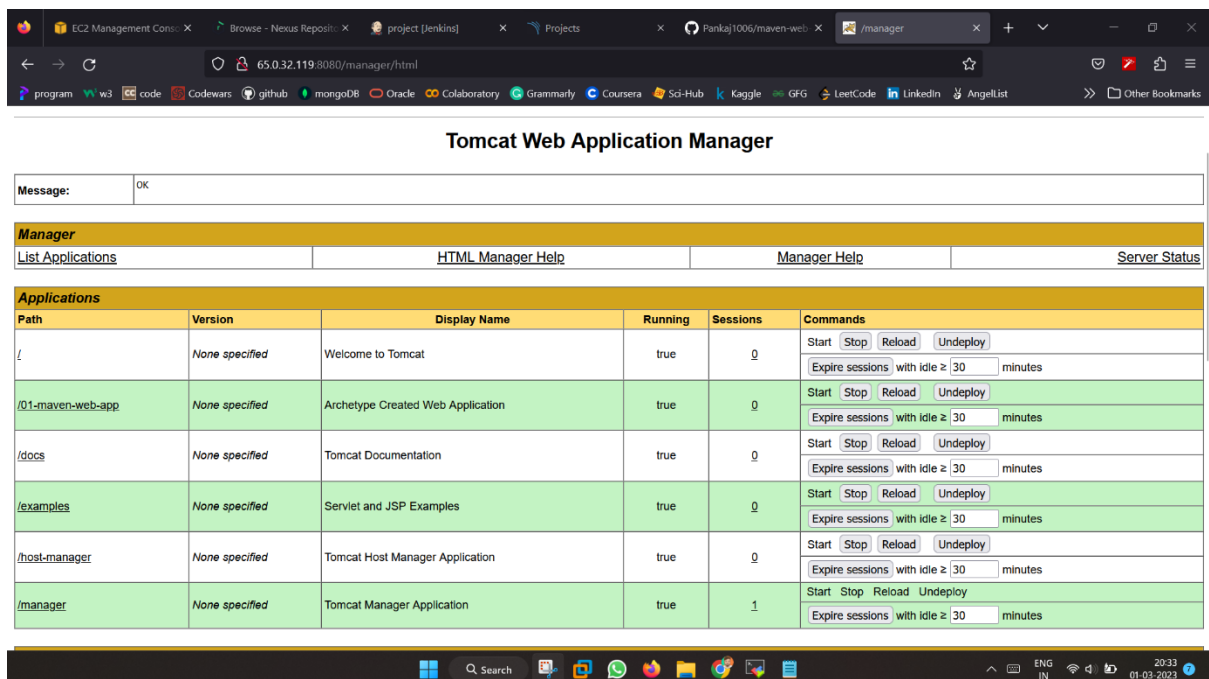


Fig 6.4 Successful Deployment of webpage on Tomcat Server

Result 5: Sample website output hosted on Tomcat Server after the successful build of pipeline. Any change or new update made in the code by developer will be checked for security issues and on successful passing the quality gates the changes will be reflected in the webpage. Thus, symbolizing the Continuous Integration and Continuous Deployment of code with infusion of security.

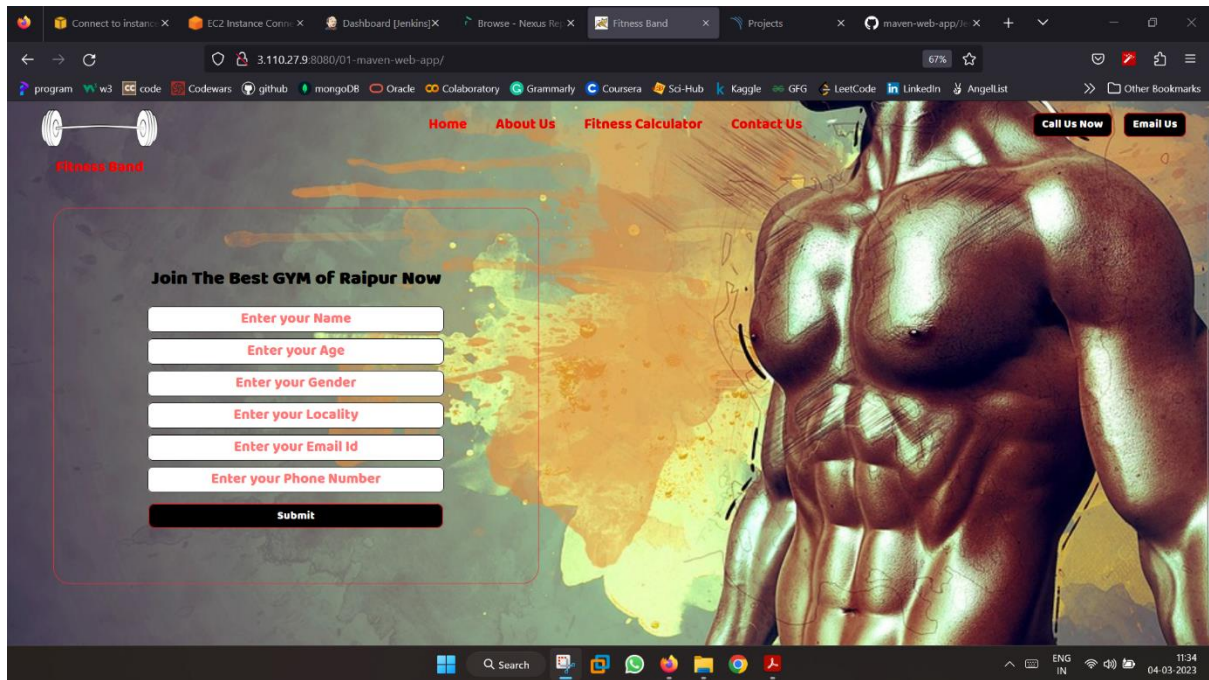


Fig 6.5 Final Display of Webpage on Successful Pipeline Build

7. CONCLUSION

In conclusion, implementing a CI/CD pipeline using Jenkins, SonarQube, Nexus, and Tomcat server can greatly improve software development processes by automating the build, test, and deployment processes, while also providing continuous feedback on code quality and security.

Jenkins is a powerful automation tool that helps to automate the build, test, and deployment processes, while SonarQube can analyse the code quality and security of the application. Nexus is a repository manager that allows for the storage and management of artifacts, while Tomcat server is a web server and servlet container that can be used to deploy the application.

By integrating these tools into a CI/CD pipeline, developers can ensure that code changes are thoroughly tested before being deployed, and that any issues are identified and addressed early in the development process. This can help to improve the quality of the application and reduce the risk of issues arising in production.

In addition, using a CI/CD pipeline can help to speed up the development process, as it allows for faster and more frequent deployments. This can be particularly important in agile development environments, where there is a need to quickly respond to changing customer needs and requirements.

Overall, implementing a CI/CD pipeline using Jenkins, SonarQube, Nexus, and Tomcat server can greatly improve the software development process by automating key tasks, improving code quality and security, and enabling faster and more frequent deployments.

8. REFERENCES

1. Jenkins Documentation for pipeline
<https://www.jenkins.io/doc/tutorials/#pipeline>
2. Reference for SonarQube
<https://www.sonarsource.com/products/sonarqube/>
3. Reference for Nexus
<https://www.sonatype.com/products/nexus-repository>
4. Tomcat Server Documentation
<https://tomcat.apache.org/>
5. References from Git
<https://github.com/AnkushKapoor-97/DevOps.git>
<https://github.com/Pankaj1006/Devops-project-in-AWS.git>
6. AWS Documentation
<https://docs.aws.amazon.com/>