Chapter 4

Workflow Implementation

Plan

1 **Workflow** **30**

2 **Docker** **34**

3 **Setting up the Containers** **36**

4 **Jenkins Integration** **44**

5 **Pipelines** **50**

Introduction

The conception phase is primordial for a clear and more rigorous vision of an optimized architecture's implementation. To identify the functionalities of our pipeline, we will devote this chapter to detailing the different activities discussed and presenting the diagrams about our DevOps platform, as well as the steps of the realization and the steps to redo the tasks.

4.1 Workflow

4.1.1 Pre-configuration

To set up a CI/CD platform, it is necessary to configure all the tools required. Figure 4.1 illustrates the steps of installation and configuration phases for each tool.

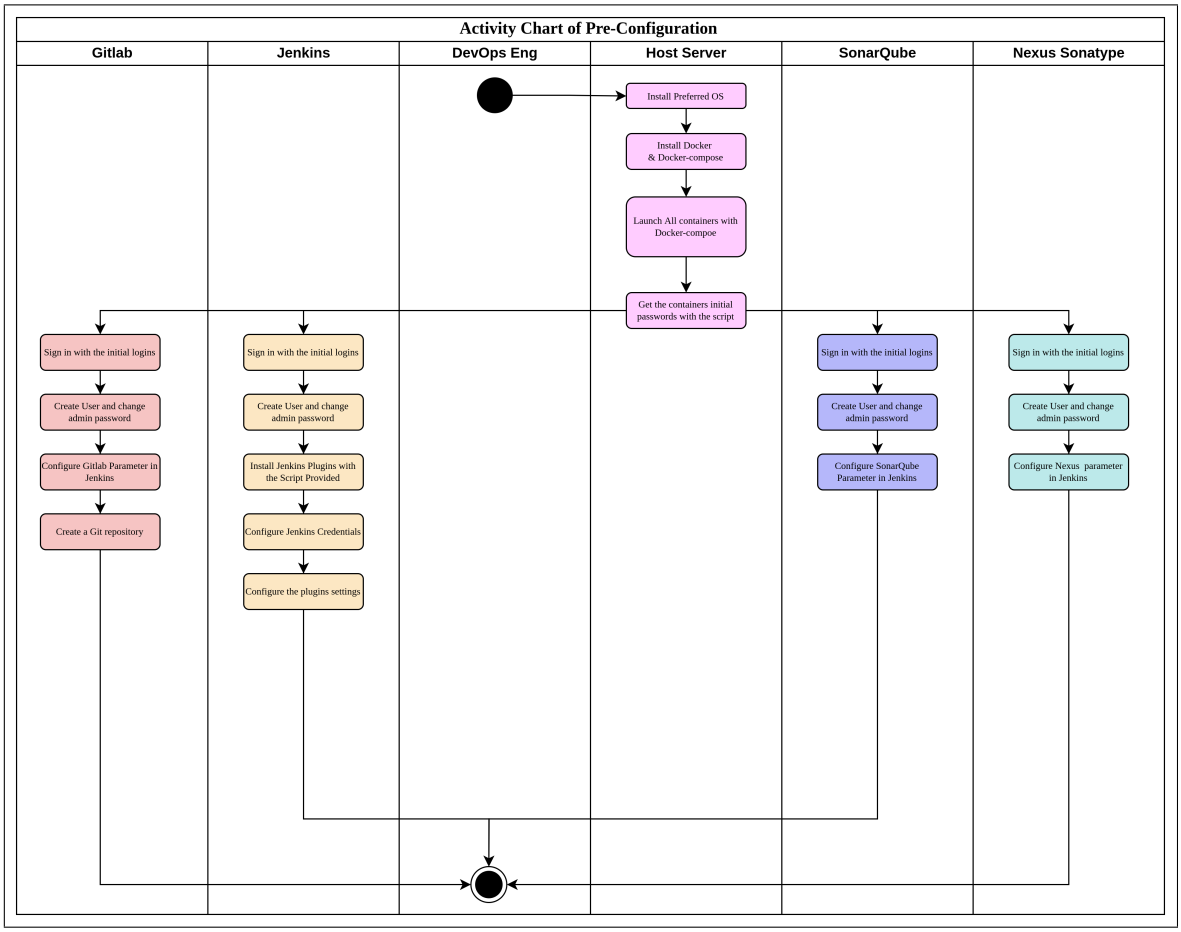

Figure 4.1: Activity Chart of Pre-configuration

4.1.2 Configuration

The activity chart configuration, 4.1, shows the different steps that the DevOps Engineer must follow to set up a continuous integration,daily and deployment pipelines.

Figure 4.1 illustrates the steps to prepare each pipeline.

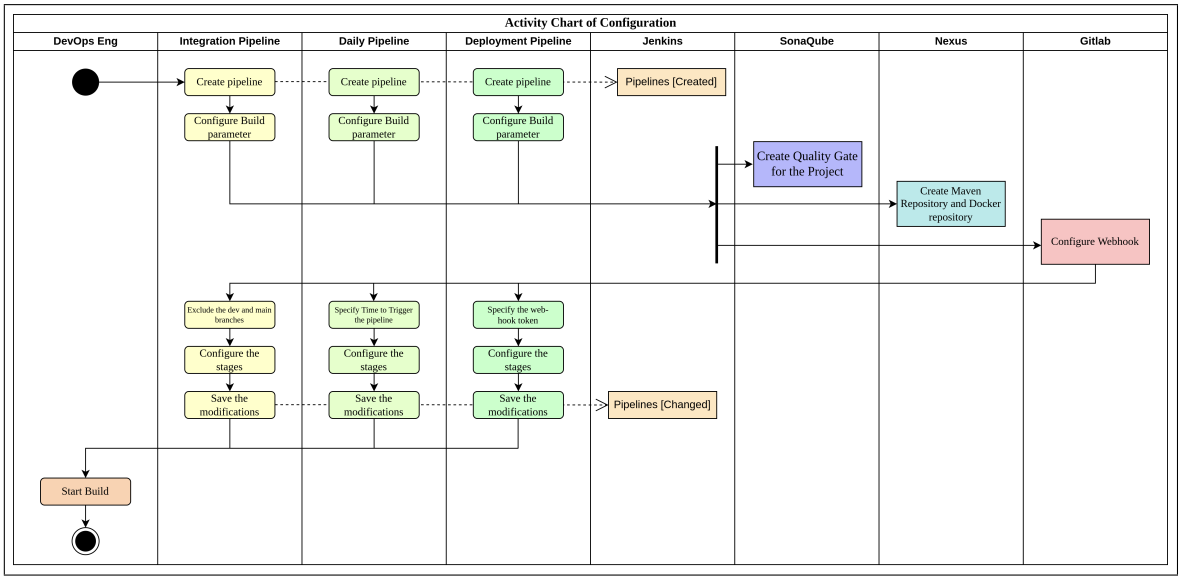

Figure 4.2: Activity Chart of configuration

4.1.3 Continuous Integration pipeline

To Prepare the Continuous Integration Pipeline it is fundamental to prepare what triggers it and it's stages.

Figure 4.3 illustrates the stages and the trigger of the pipeline.

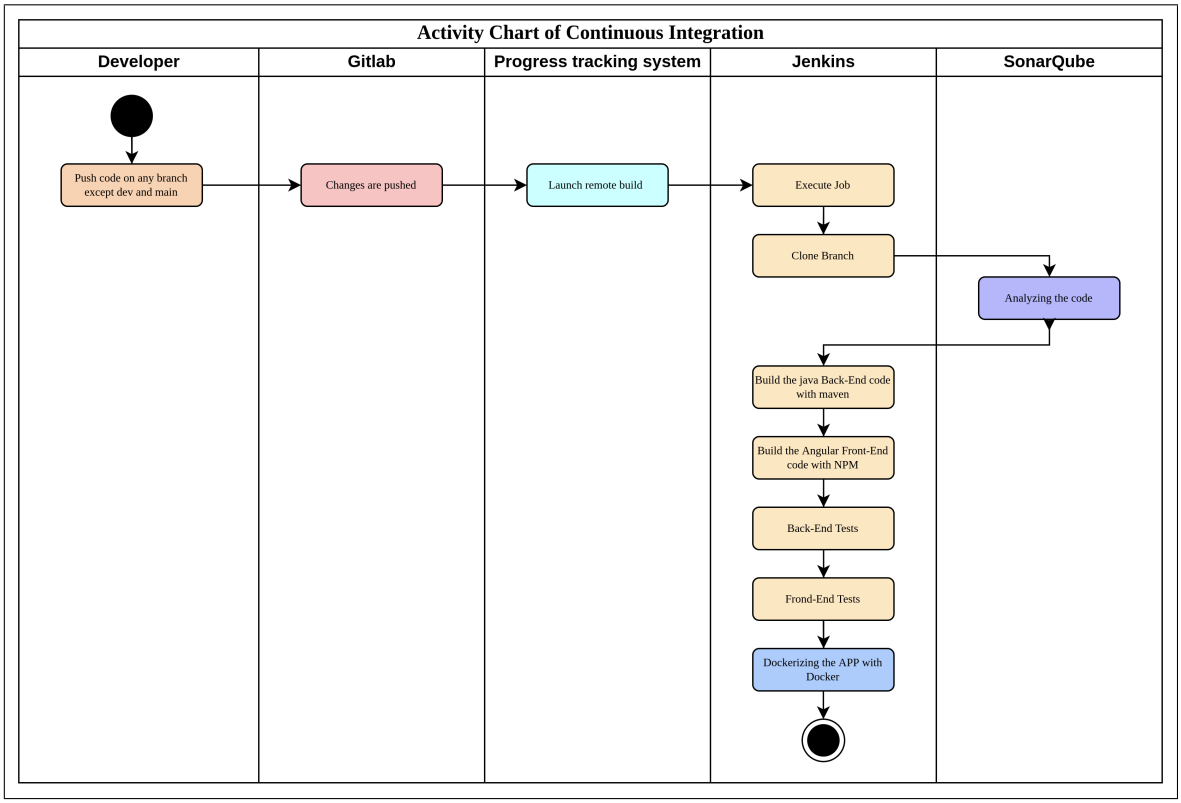

Figure 4.3: Continuous-Integration's Activity Chart

4.1.4 Daily pipeline

To Prepare the Daily Pipeline it is fundamental to prepare what triggers it and it's stages.

Figure 4.4 illustrates the stages and the trigger of the pipeline.

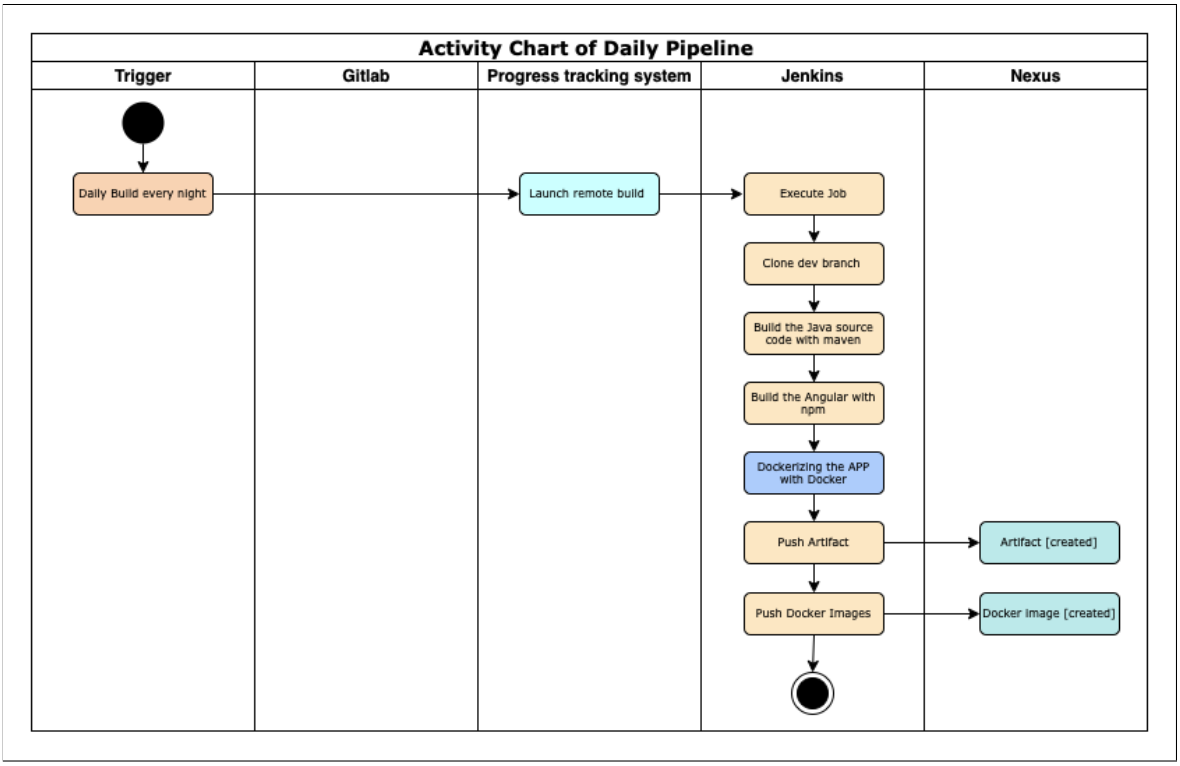

Figure 4.4: Daily pipeline's Activity Chart

4.1.5 Continuous Deployment pipeline

To Prepare the Continuous Deployment Pipeline it is fundamental to prepare what triggers it and it's stages.

Figure 4.5 illustrates the stages and the trigger of the pipeline.

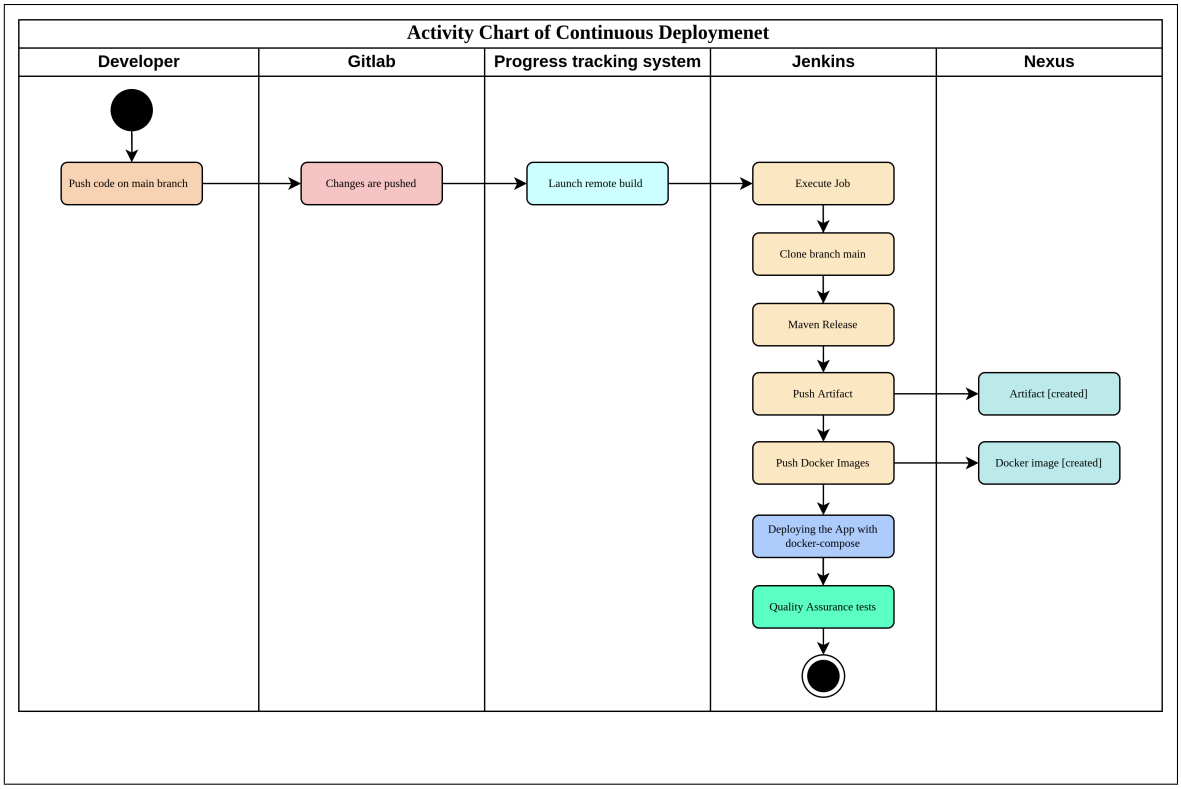

Figure 4.5: Continuous Deployment's Activity Chart

4.2 Docker

4.2.1 Installing Docker

To install Docker and Docker-compose we provided a bash script as it is shown in the annex 4.5.1, to run it type:

```
$ bash installdocker.bash
```

Disclaimer: for Windows or macOS users download the Docker Desktop from the official documentation of Docker. You might need to change the 'apt' command in the script to yum or something else depending of the Linux distribution.

4.2.2 Docker-compose file

Docker-compose file allows us to define and run multi-container Docker applications with YAML file to configure our application's services.Using this one file you can configure multiple containers giving them their base images, names, IP addresses and more.

4.2.3 Docker Compose Configuration

To prepare we need to configure each service as it is shown in the annex 4.5.2:

- The Docker-Compose YAML file version as shown in the 1st line.
- The Services which represents the containers we are going t run as shown in the 2nd line.
- The Base Docker Image for each container as shown in the lines 4-18-31-45.
- The Container Name as shown in the lines 5-19-32-46.
- Privileges: as shown in the lines 6, 7-20, 21-33, 34-47, 48. In this project some containers need root access to run so we give the container root privileges to avoid a couple of issues we faced.
- Volumes: as shown in the lines 8-22-35-49, the container need persistent volumes to keep the data even when we delete the containers and even the images so we mount the appropriate volumes for each of the services.
- Ports: as shown in the lines 11-25-40-55, we maps a host port to a container port.
- Networks: as shown in the lines 14-28-43-58, We give each container a static IP Address along with a custom network and it's driver, we defined the network (network name, driver and subnet) in line 67.

4.2.4 Running Docker-compose

To run Docker-compose on any OS type in the terminal/CMD/PowerShell:

```
$ docker-compose up -d
```

and «-d» stand for detach so the process run in the background, it starts with pulling the docker images from docker hub if they are not available on the local machine, then it runs all the containers and creates the shared network.

Disclaimer: Root access might be need, you just type:

```
$ sudo docker-compose up -d
```

Or change to root user using "sudo su". Machine password is needed.

4.2.4.1 Running Docker Containers

To check all running Docker container we use the command in Linux or macOS terminal or Windows CMD/PowerShell:

```
$ docker ps
```

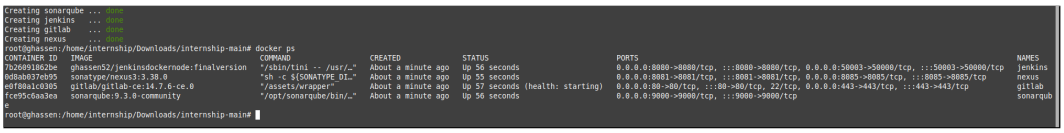

Figure 4.6: Running Docker Containers

4.3 Setting up the Containers

4.3.1 Getting the initial passwords

This command will run the script that will have output the initial passwords of Gitlab, Jenkins, Nexus and as for SonarQube initial password is "admin".

```
$ bash initialpassword.bash
```

4.3.2 Jenkins Containers

You can access the Jenkins container user interface using its IP address used in the docker-compose file and the first thing you need to do is write the Jenkins initial password which was generated by the last script as shown in the figure 4.7.

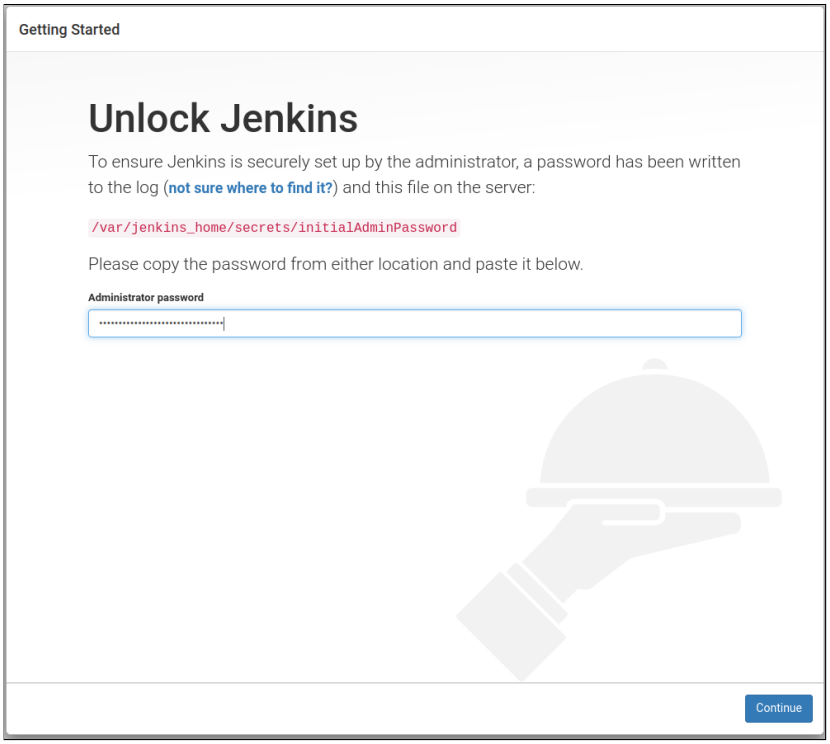

Figure 4.7: Entering Jenkins initial password

After Entering the default password click on "Select Plugins to install as shown in the figure 4.8 :

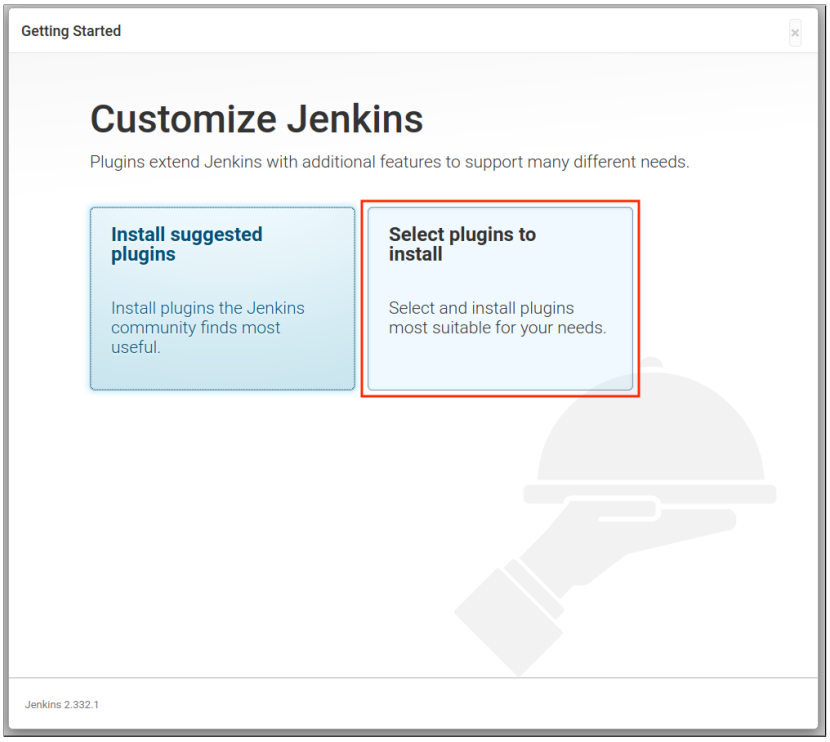

Figure 4.8: Jenkins Plugins Choice Interface

Click on the "X" button on the top right of Interface of the Jenkins Plugins Download Interface will appear as shown in the figure 4.9:

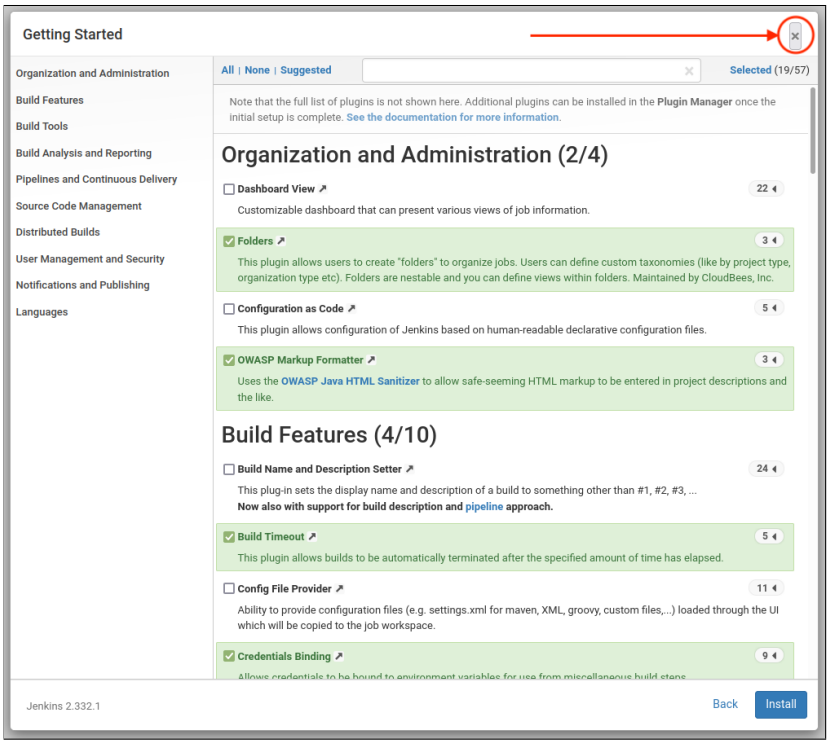

Figure 4.9: Jenkins User Interface

Now Jenkins is ready, click on "Start using Jenkins" as shown in the figure 4.10

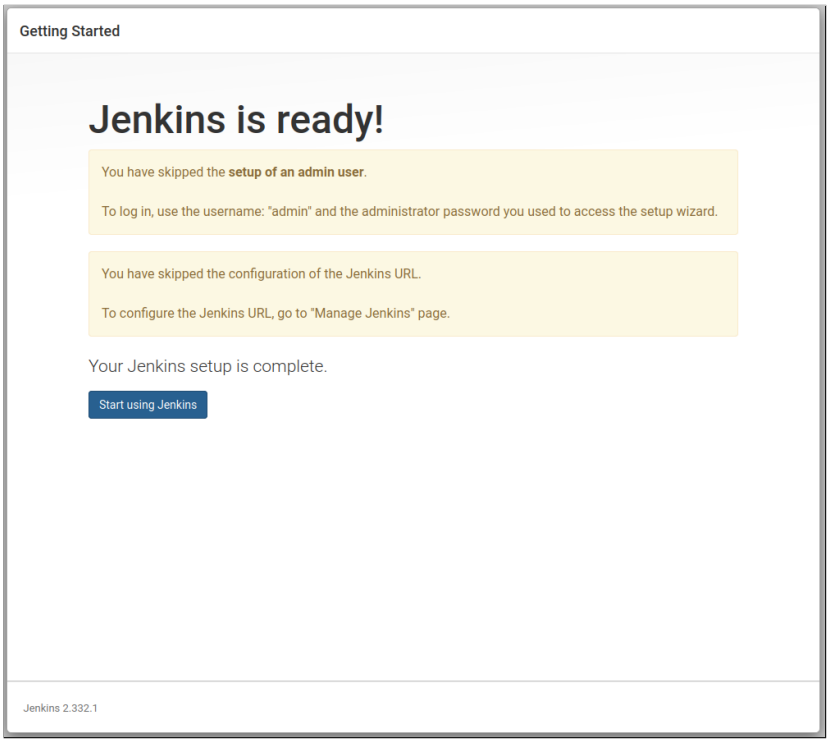

Figure 4.10: Jenkins User Interface

The figures 4.11 and 4.12 show the steps needed to change the Jenkins password.

First we start by accessing the Admin Interface as shown in the figure 4.11.

Figure 4.11: Jenkins Admin interface

Then we type in the new Jenkins's password as shown in the figure 4.12

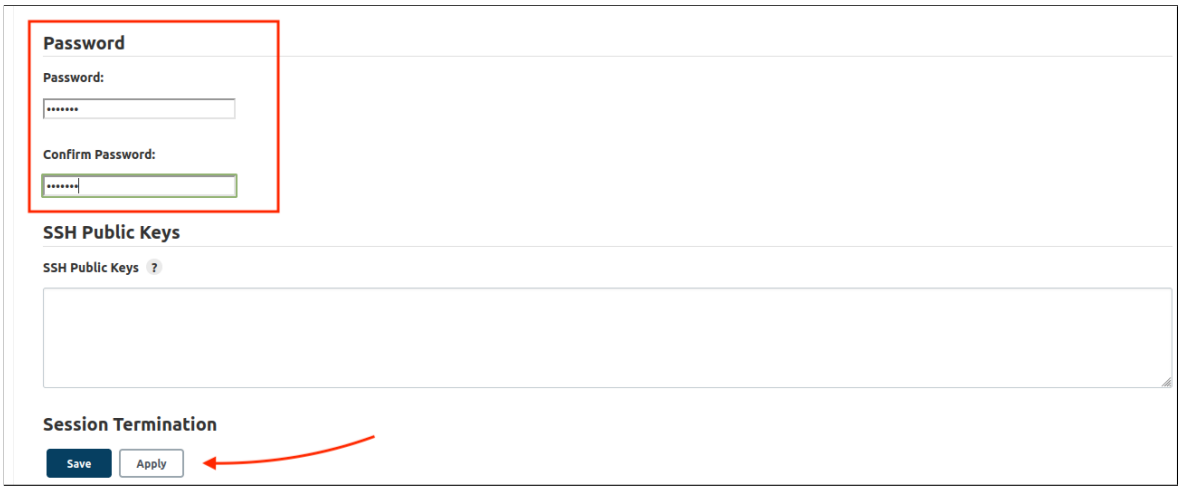

Figure 4.12: Changing Jenkins password

4.3.3 Installing Plugins

We need two bash scripts as shown in annex 4.5.4 to copy the plugins script and execute it inside Jenkins and 4.5.5 which contain the cmd to install the plugins and the name of the plugins, to install Jenkins plugins needed from terminal for linux and macOS based systems, the command used to run this script is :

```
$ bash installplugins.bash
```

Disclaimer: if you are using Windows you can use the command line in the script file named: installplugins.bash and run it in CMD or PowerShell.

4.3.4 Gitlab Container

Sign in Gitlab with user: "root" and it's initial password extracted with the script in the annex 4.5.3 as shown in the figure 4.13

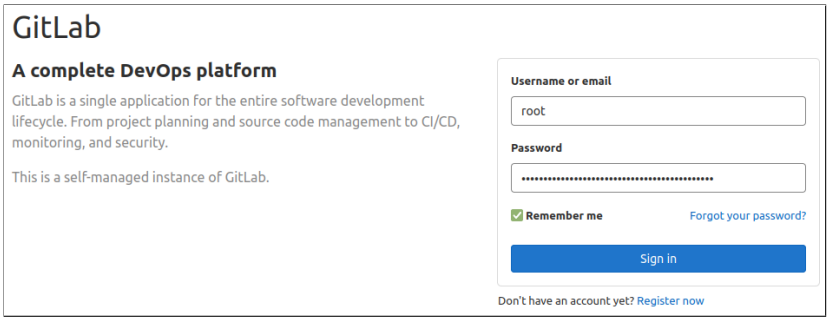

Figure 4.13: Gitlab's Sign in Menu

Gitlab's Home Menu is shown in the figure 4.14

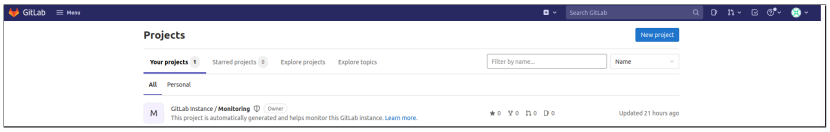

Figure 4.14: Gitlab's Home Menu

We will start by setting up a new password in the account preferences section for Gitlab as shown in the figure 4.15

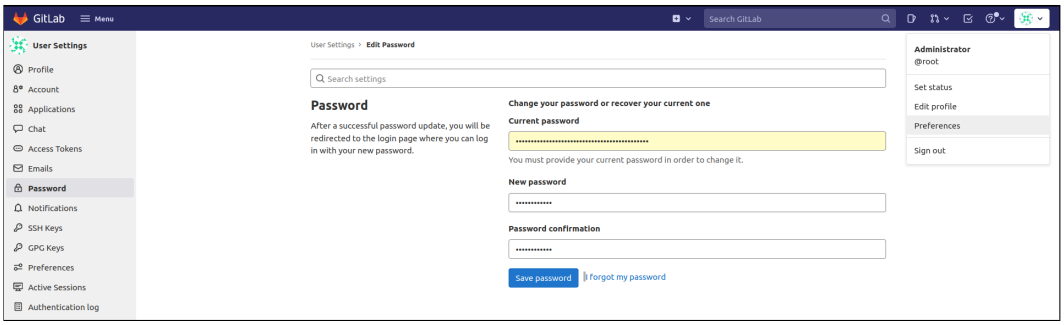

Figure 4.15: Changing Gitlab's password

4.3.5 SonarQube Container

Sign in SonarQube with default user: "admin" and password: "admin" as shown in the figure 4.16.

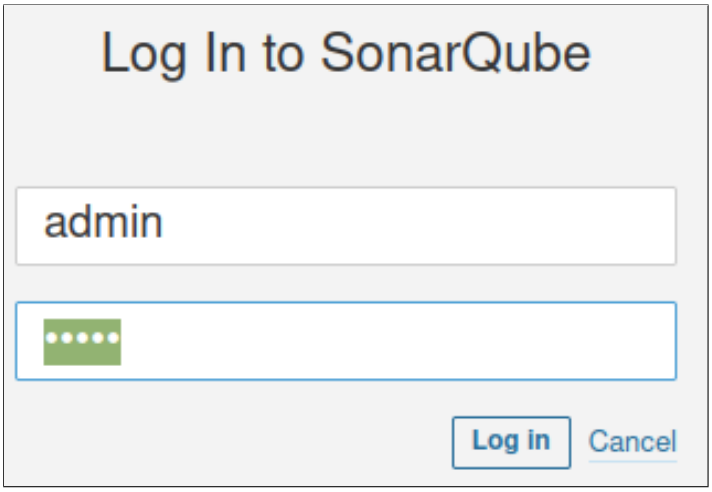

Figure 4.16: SonarQube Sign in

We will start by changing our password by typing in SonarQube's Old and New password as shown in the figure 4.17.

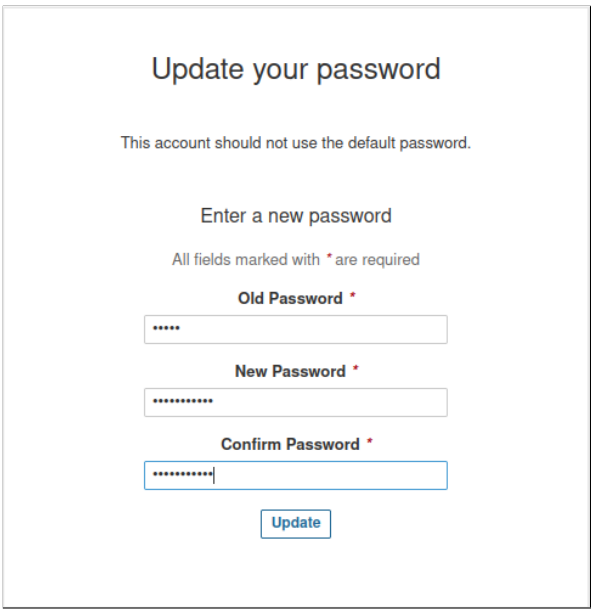

Figure 4.17: Change SonarQube password

The SonarQube Home Menu is shown in the figure 4.18

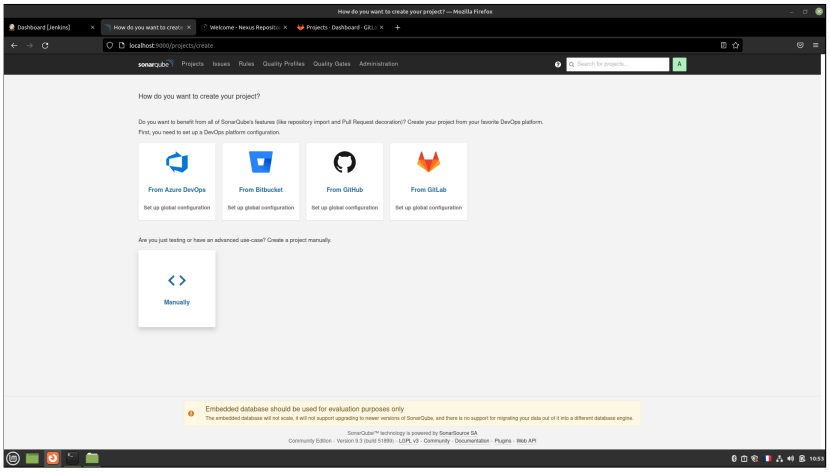

Figure 4.18: SonarQube Home Menu

4.3.6 Nexus Container

First we will start by clicking on the Sign in button in Nexus shown in the figure 4.19 and then sign with username: "admin" and the initial password extracted using the script in the annex 4.5.3 as shown in the figure 4.20

Figure 4.19: Sign In Button

Type in user and initial password as shown in the figure 4.20:

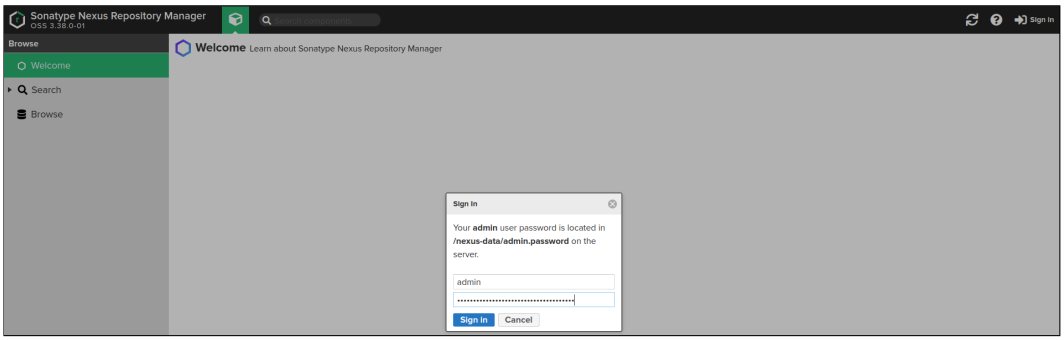

Figure 4.20: Sign In with initial password and admin as username

Then we will start by setting up our new password

First of all we type the new Nexus password as shown in the figure 4.21

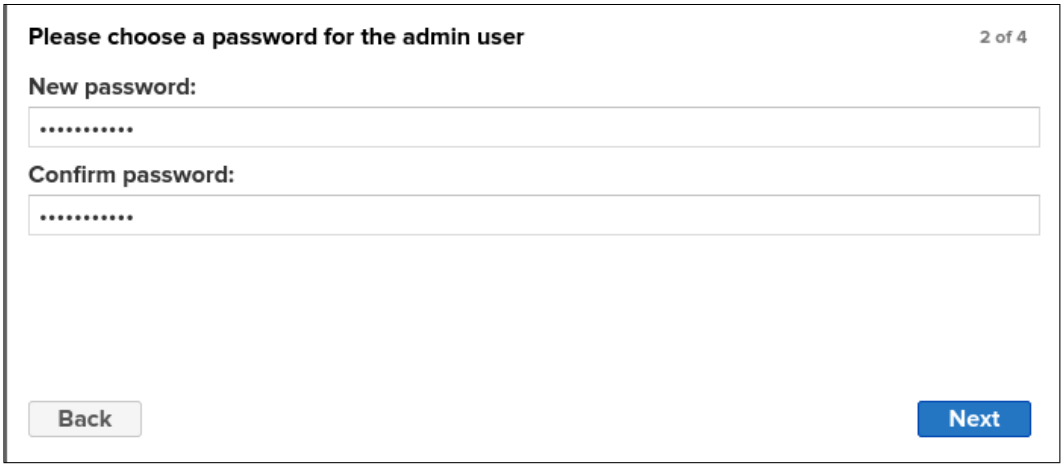

Figure 4.21: Changing Password

Then we will disable Anonymous Access to assure that our nexus repository is only accessed with credentials to assure the security of our artifacts as shown in the figure 4.22

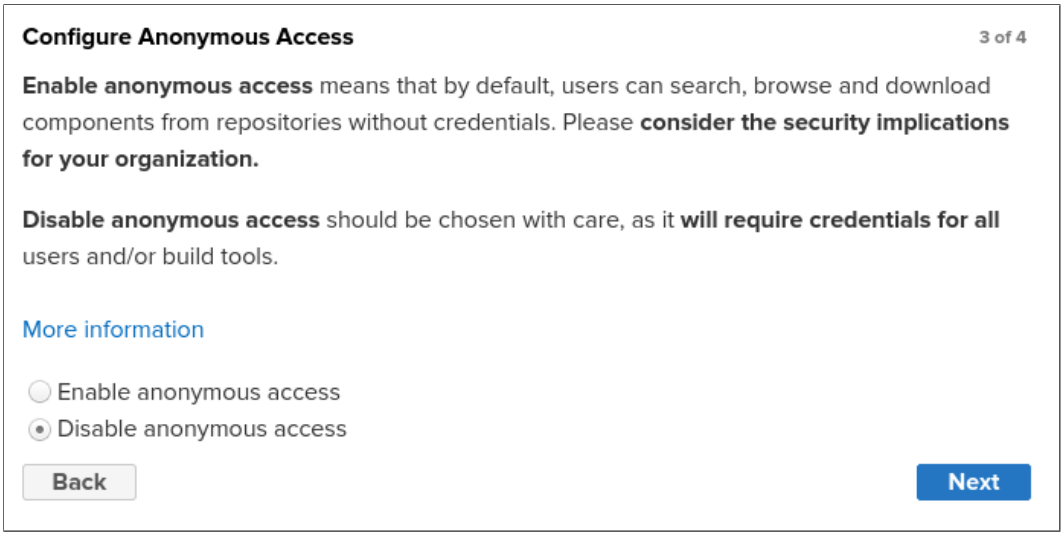

Figure 4.22: Disable anonymous access

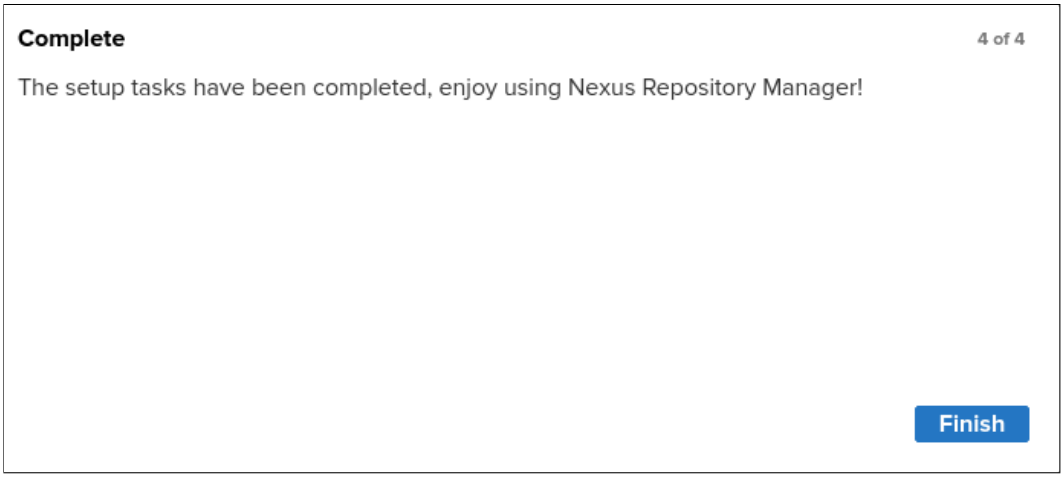

Figure 4.23: Nexus Setup Completed

4.4 Jenkins Integration

One of the most important steps into having a fully functional platform is Integrating Jenkins with the rest of the tools.

4.4.1 Jenkins Credentials

We will start by accessing the credentials section in Jenkins as shown in the figure 4.24

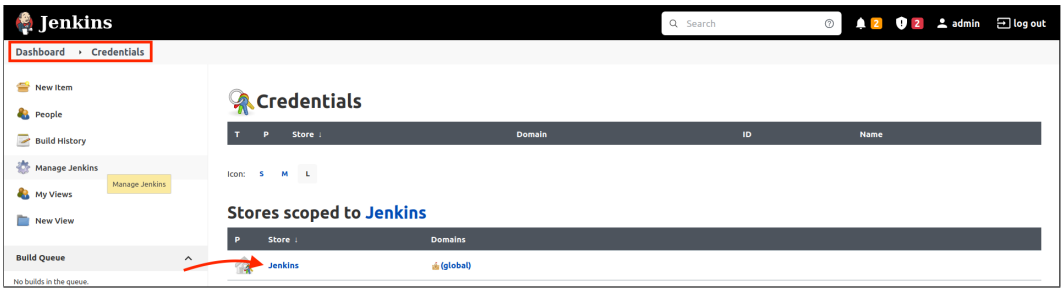

Figure 4.24: Jenkins Credentials Interface

Then we will access the Global credentials unrestricted section in Jenkins as shown in the figure 4.25 to add our credentials.

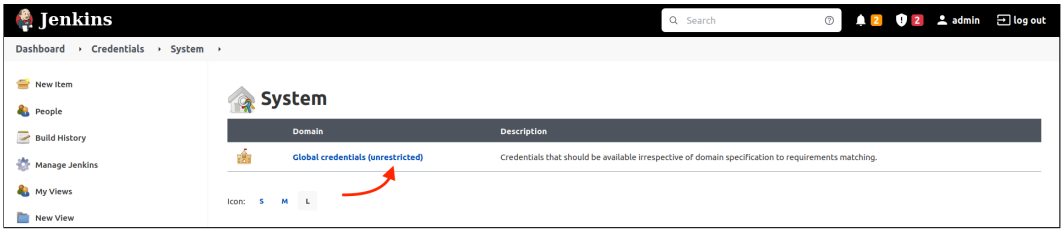

Figure 4.25: Credentials Interface

Then we will click on the "Add Credentials" button as shown in the figure 4.26 to add our credentials.

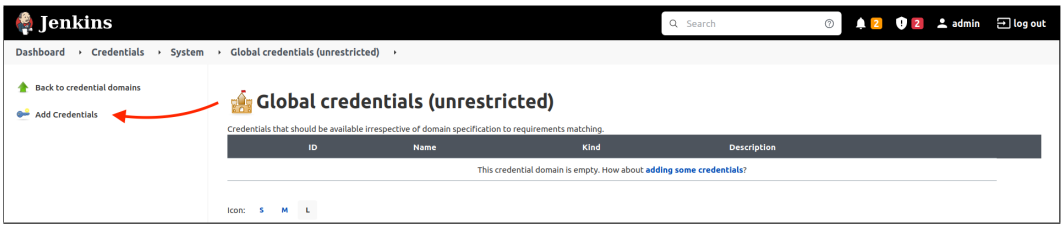

Figure 4.26: Adding Credentials

4.4.2 Integrating Jenkins with Maven

MVN needs to be integrated with Jenkins so that we can fully build and test our application.

We setup the maven plugin in jenkins in the plugins configuration section by naming the installation and choosing the Maven Version as shown in the figure 4.27

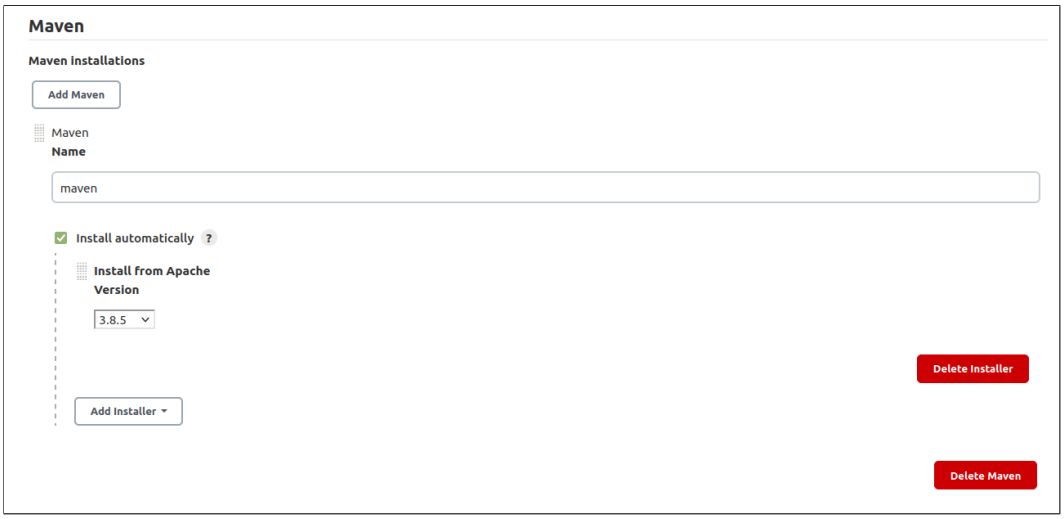

Figure 4.27: Integrating Maven with Jenkins

4.4.3 Integrating Jenkins with Gitlab

To be able to use our Gitlab container as a Code Repository we need to integrate Gitlab in Jenkins by configuring Gitlab's Credentials in Jenkins.

The figures 4.29, 4.30 and 4.30 show the steps followed in order to obtain an access token from Gitlab.

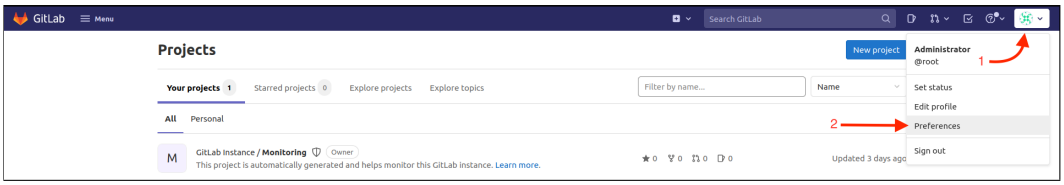

Figure 4.28: Configure Access Token in Gitlab

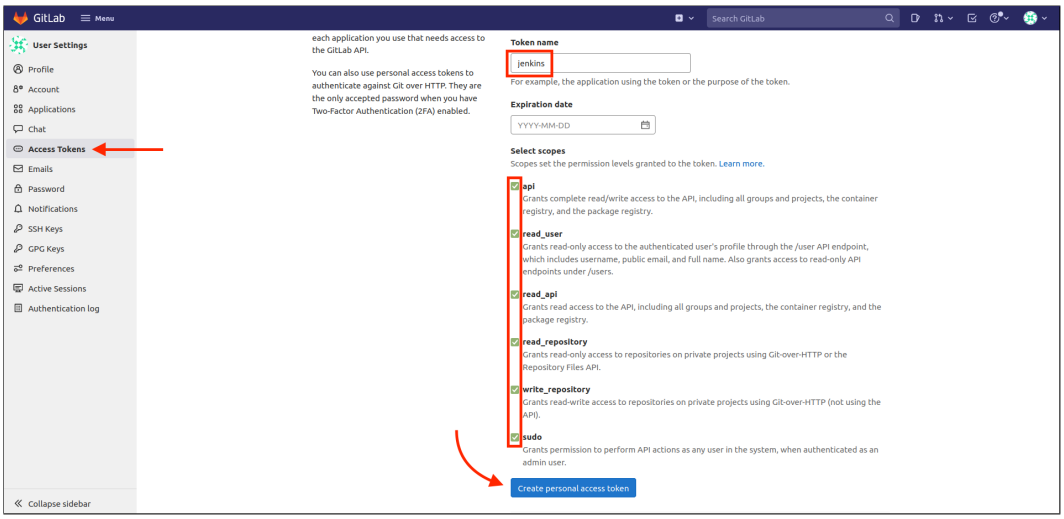

Figure 4.29: Access Token Parameter from Gitlab

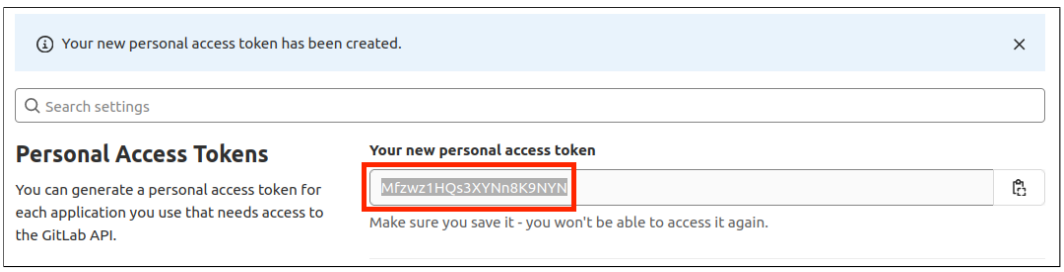

Figure 4.30: Get the Access Token from Gitlab

- After getting the Gitlab Acces Token, we will use it to configure the Gitlab Credentials in Jenkins as shown in the figure 4.31

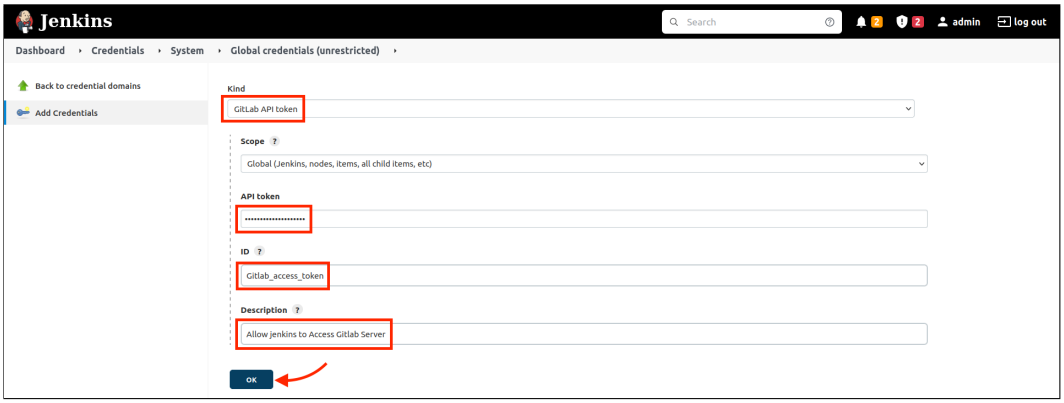

Figure 4.31: Configure Gitlab Credentials in Jenkins

4.4.4 Integrating Jenkins with SonarQube

In order to fully integrate SonarQube with Jenkins we need to configure SonarQube's credentials in Jenkins and Configure SonarQube's jenkins plugin.

4.4.4.1 Credentials

- We will start by creating SonarQube's WebHook as shown in the figures 4.32 and 4.33 .

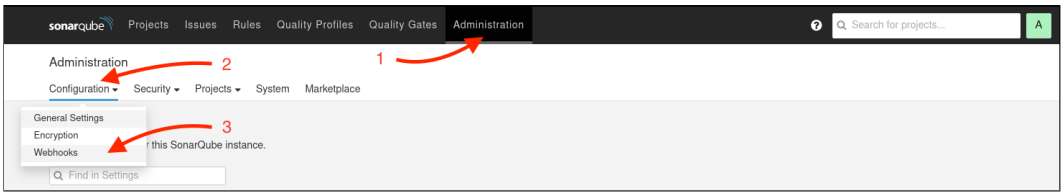

Figure 4.32: Accessing Webhooks settings

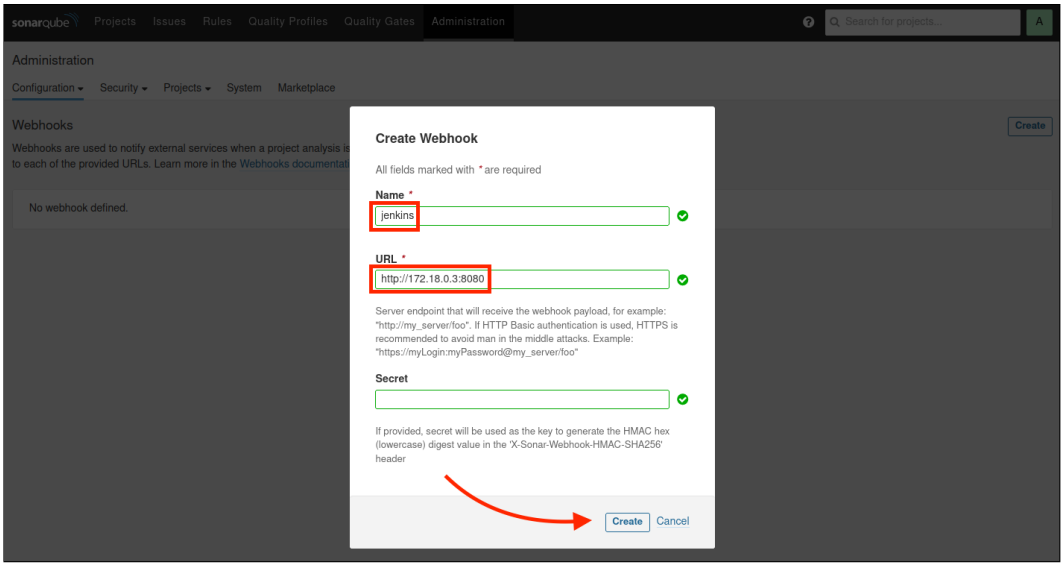

Figure 4.33: Creating the Webhook

- Then we will proceed to get the SonarQube Token4.33 .

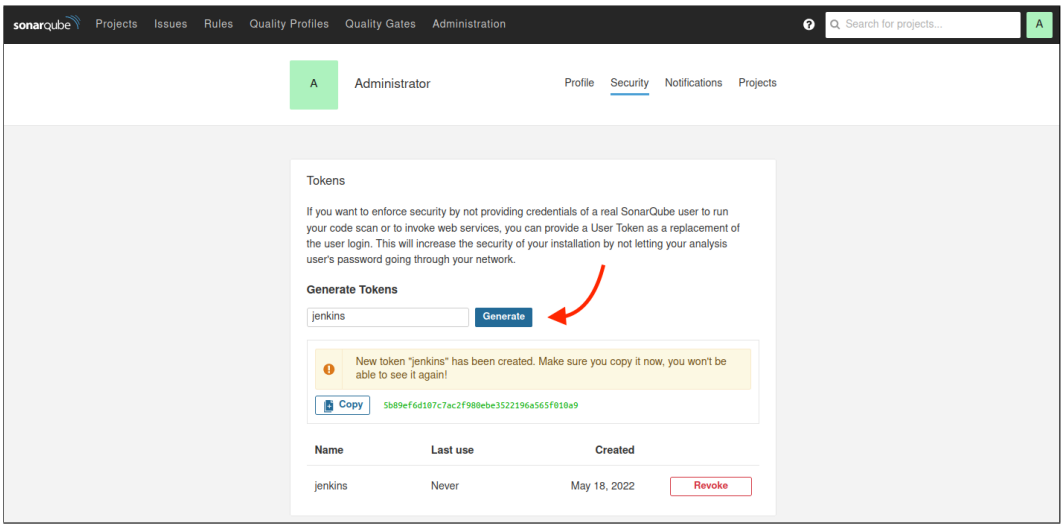

Figure 4.34: Get SonarQube Token

- Finally we will use the Token generated to configure SonarQube's credentials in Jenkins 4.35 .

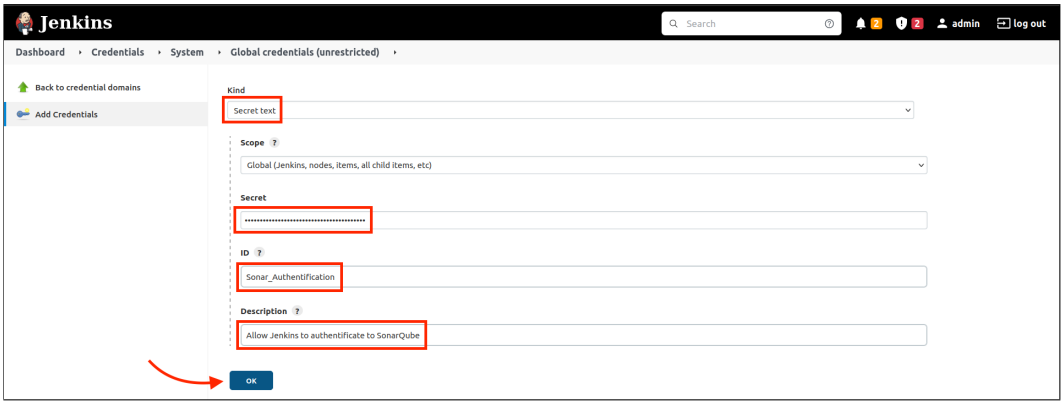

Figure 4.35: Configure SonarQube Credentials in Jenkins

4.4.4.2 Plugins Configuration

- To configure SonarQube Plugin parameter in Jenkins, We start by providing Jenkins with the Installation Name and SonarQube's server URL and then allow Jenkins to authenticate to SonarQube as shown in the figure 4.36 .

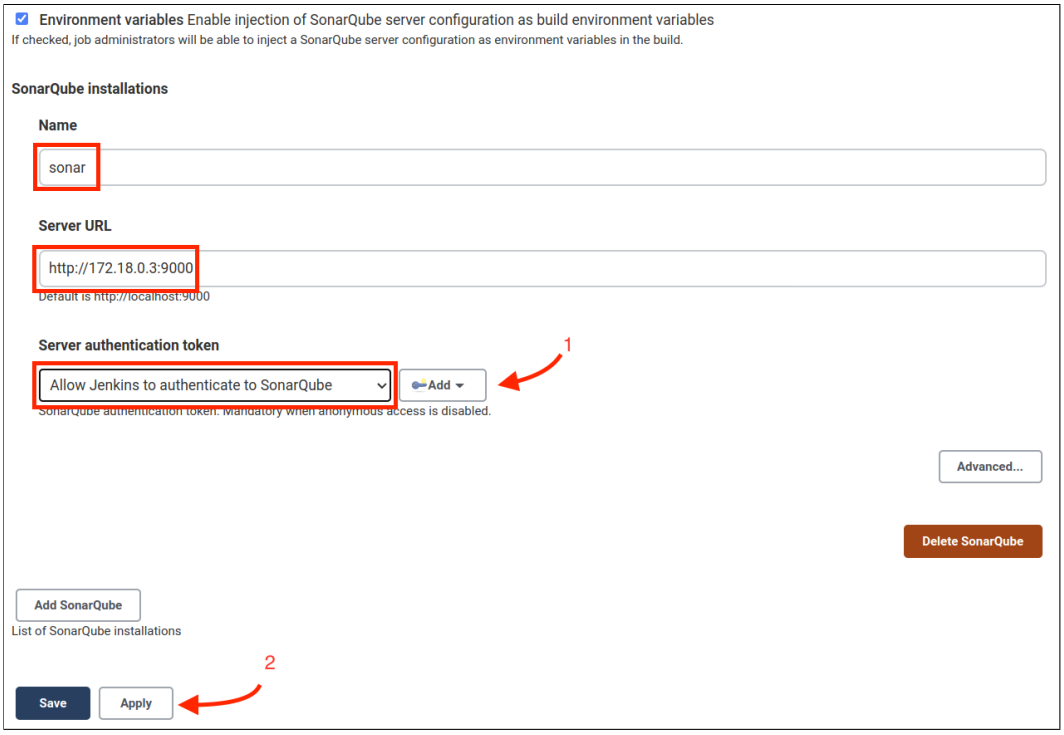

Figure 4.36: SonarQube Plugin in Jenkins

4.4.5 Integrating Jenkins with Nexus

- The figure 4.37 shows the steps to configure nexus credentials in Jenkins using the username "Admin" and the password set previously

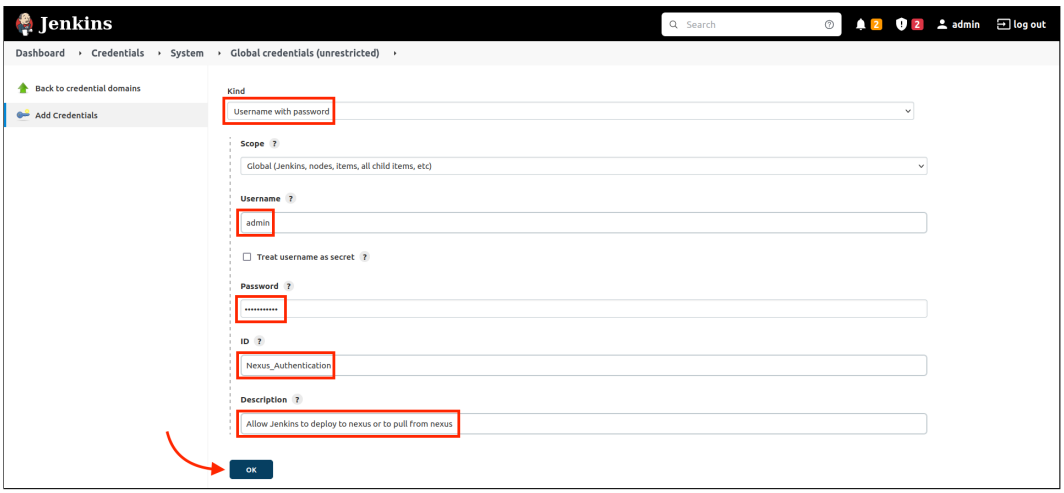

Figure 4.37: Configure Nexus Credentials in Jenkins

4.5 Pipelines

4.5.1 Pipeline's Representation

We will describe briefly our CI/CD pipeline in Figure 4.1 which illustrates the Stage of the Pipeline

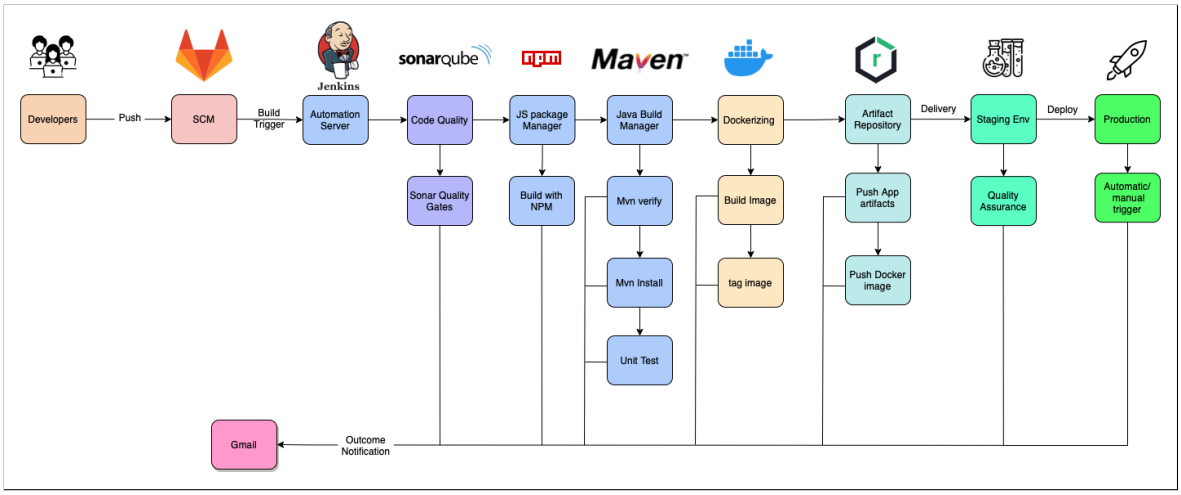

Figure 4.38: DevOps Pipeline

4.5.2 Continuous Integration

Configuring SonarQube Quality Gate Before configuring and running our pipeline we need to start by configuring our SonarQube Quality Gate. As shown in the figure 4.39 we configured the Quality Gate using specific tests provided by our team leader.

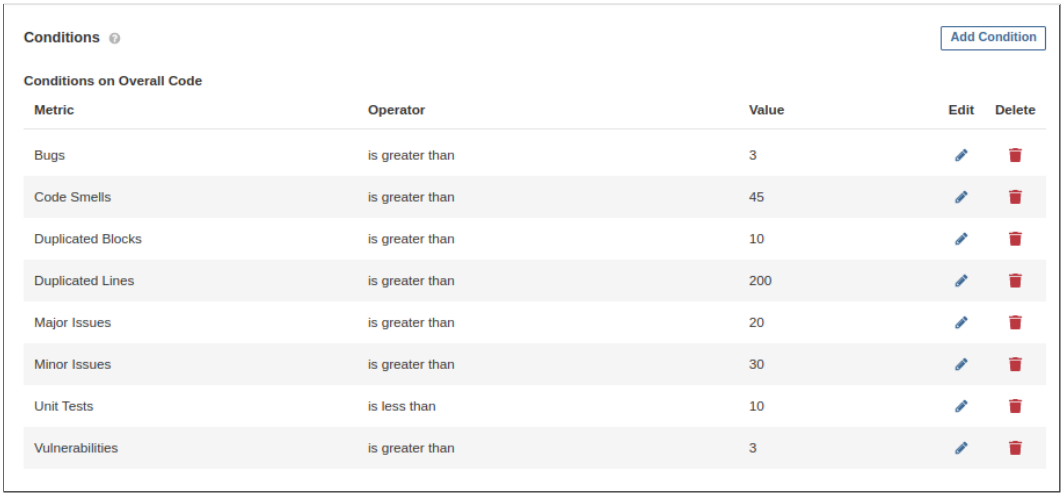

Figure 4.39: Configuring the Quality Gate

4.5.2.1 Triggers

Using a Webhook created in Gitlab like shown in the figure **??** to trigger the Continuous Integration when there's a push on any branch except 'main' and 'dev' branches Using a Webhook to trigger the Release Pipeline/Continuous Deployment when we push on 'main' branch.The figures 4.40 ,4.41 and 4.42 shows the steps followed to create and configure the Integration Webhook like shown in the figures

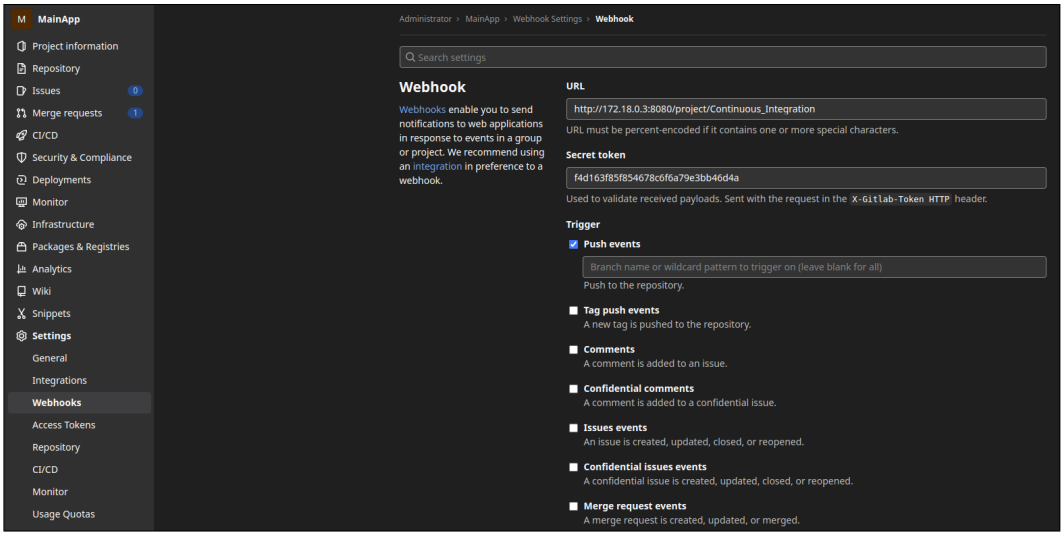

Figure 4.40: Integration Webhook Configuration

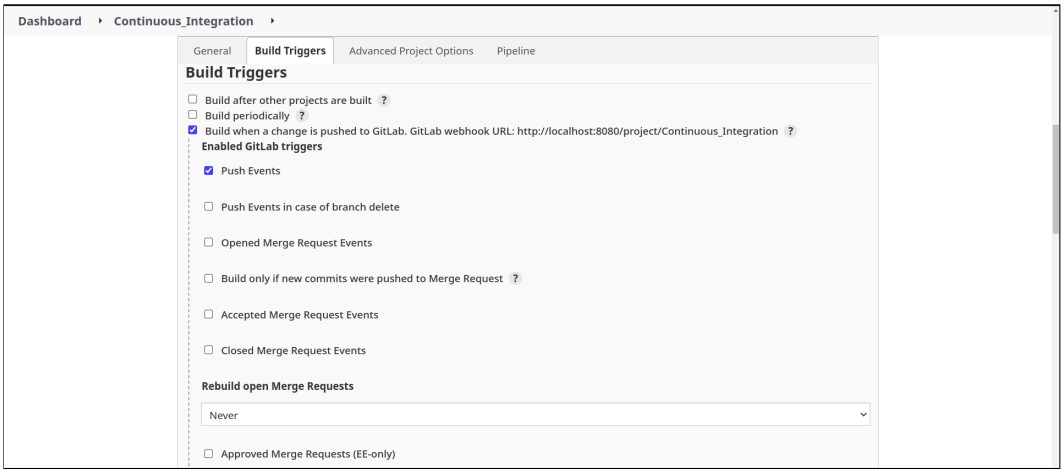

Figure 4.41: Continuous Integration Trigger Configuration

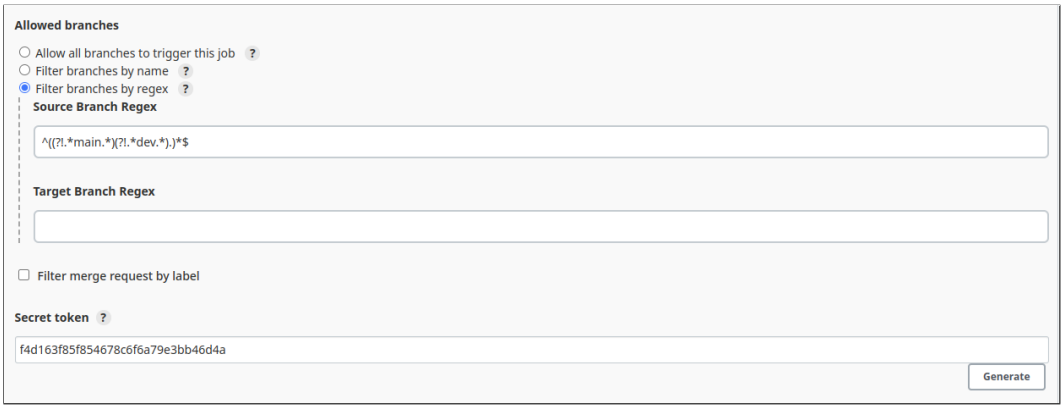

Figure 4.42: Trigger Branch Configuration

4.5.2.2 Pipeline Stages

- Cloning the Code from the Gitlab Repository (From any branch except Main and Dev)

  In this stage we will clone the code from our Gitlab repository. The source code will be cloned from the branch that triggered the pipeline by running the jenkinsfile as illustrated in 4.5.6 from line 6 to 13.

- Analysing the code with SonarQube

  In this stage we will analyse the code cloned in the previous stage using SonarQube by running the jenkinsfile as illustrated in 4.5.6 from line 15 to 26.

- Testing the code with SonarQube Quality Gate

  In this stage we will pass the code through a quality gate which has been already defined in SonarQube to check the quality of the code by running the Jenkinsfile as illustrated in 4.5.6 from line 28 to 32. The figure 4.43 shows that the code passed the Quality Gate successfully.

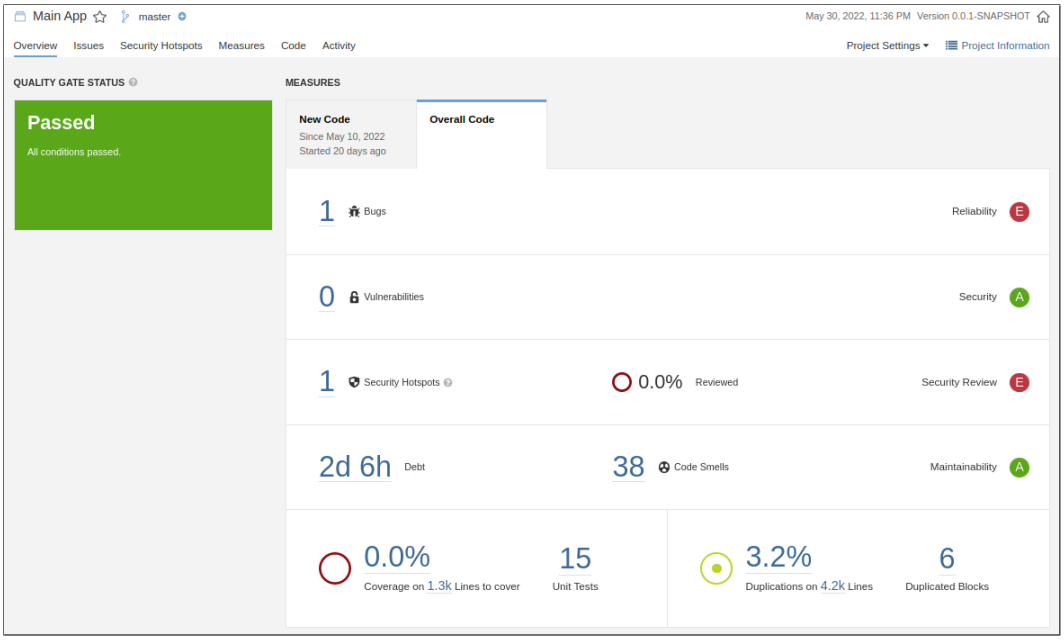

Figure 4.43: Successfully passing SonarQube Quality Gate

- Building the App using NPM for the Front-End and MVN for the Back-End

In this stage we will build the app using MVN to build the SpringBoot Back-End and NPM to build the Angular Front-End by running the jenkinsfile as illustrated in 4.5.6 from line 34 to 43.

- Running the Back-End Tests with MVN

  In this stage we will test the Back-End using MVN by running the jenkinsfile as illustrated in 4.5.6 from line 45 to 57.

- Running the front-end tests with NPM

  In this stage we will test the front-end using NPM by running the jenkinsfile as illustrated in 4.5.6 from line 59 to 72.

- Dockerizing the app

  In this stage we will be creating the docker image of the application by running the jenkinsfile as illustrated in 4.5.6 from line 74 to 79.

- Post Action

  In this stage we will define the actions done automatically after running all the stages, in our case an email will be sent to inform us whether the pipeline succeeded or failed by running the jenkinsfile as illustrated in 4.5.6 from line 80 to 88.

The figure 4.44 shows the Jenkins dashboard output when the pipeline is successful while the figure 4.45

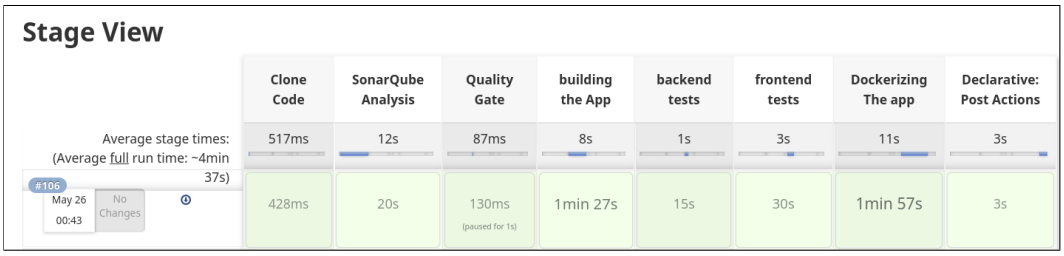

Figure 4.44: Continuous Integration Completed

Figure 4.45: Continuous Integration Mail

4.5.3 Daily

4.5.3.1 Creating the nexus Artifact Repository and Docker Images Repository

The Daily pipelines require the creation of a Snapshot Artifact Repository and a Docker Images Repository

- The figures 4.46, 4.47, 4.48 and 4.49 shows the steps followed in order to create the Docker images Repository

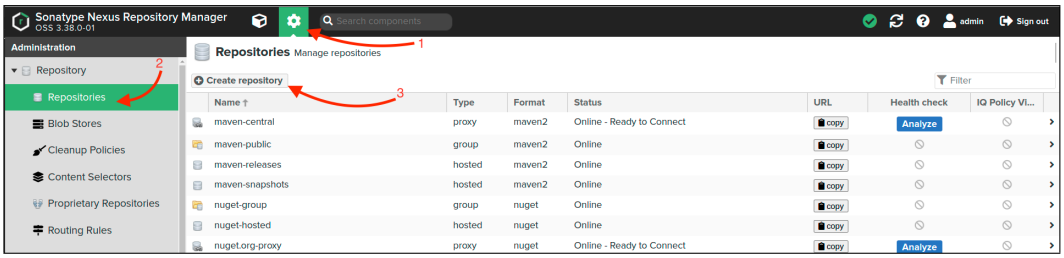

Figure 4.46: Creating the repository

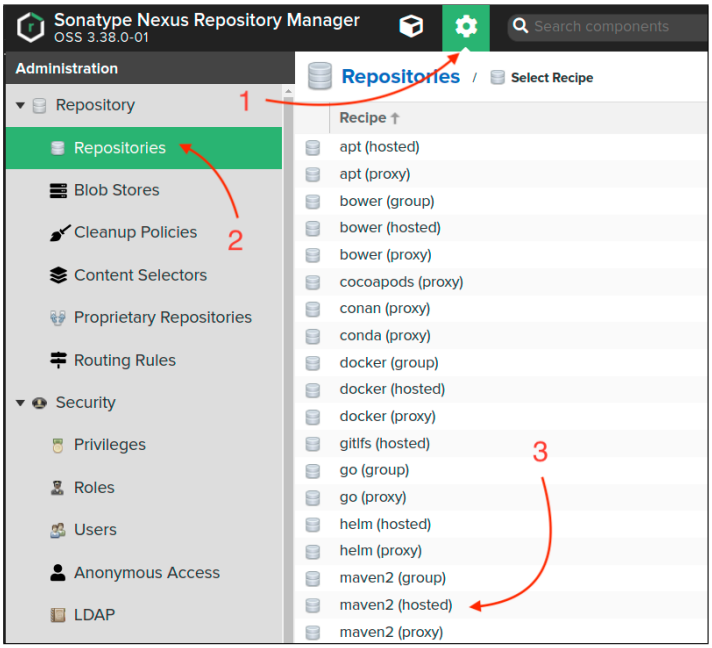

Figure 4.47: Choosing the Repository type

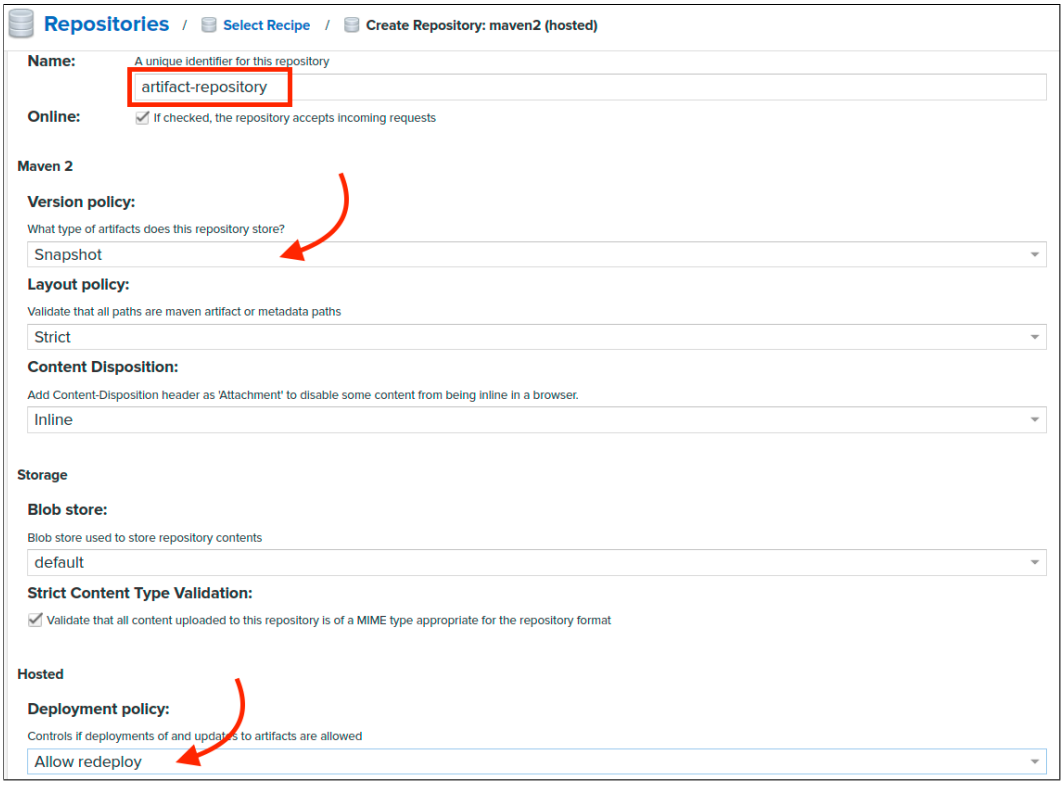

Figure 4.48: Confirming the Creation of the repository

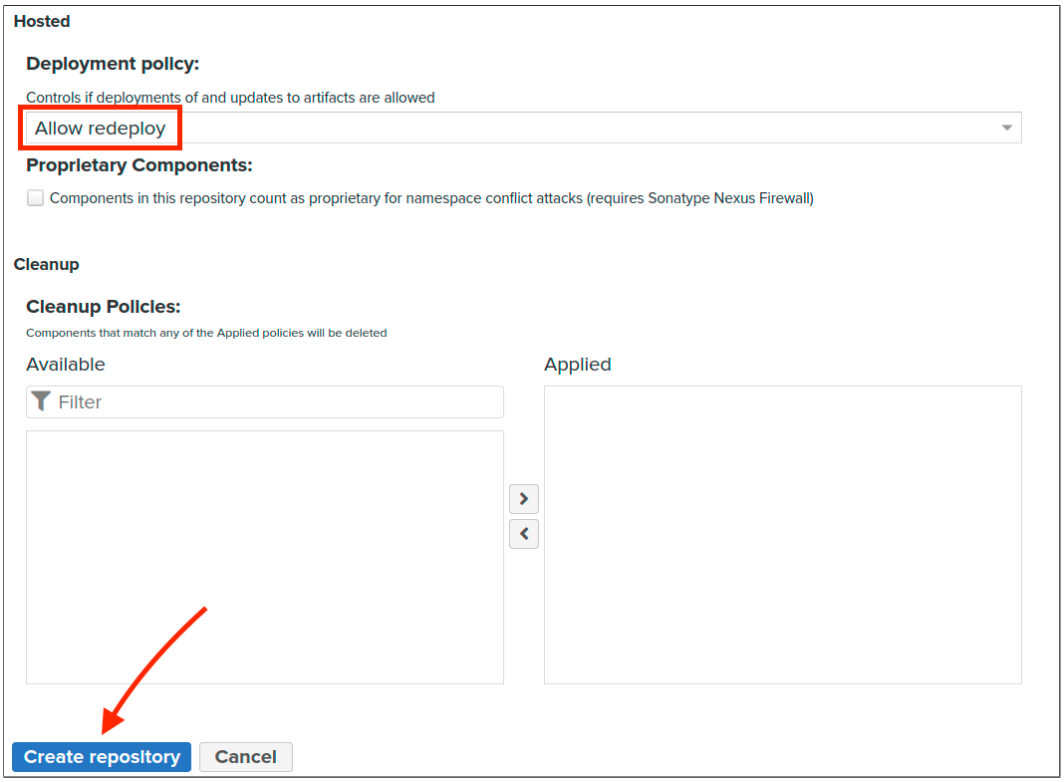

Figure 4.49: Configuring the Artifact Repository

- The figures 4.50, 4.51, 4.52 and 4.53 shows the steps followed in order to create the Docker images Repository

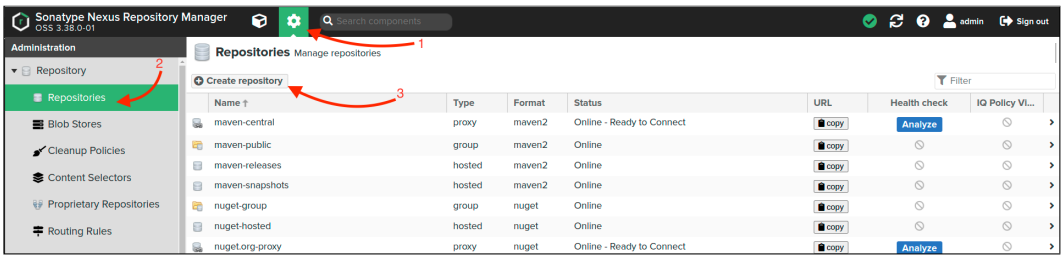

Figure 4.50: Creating the repository

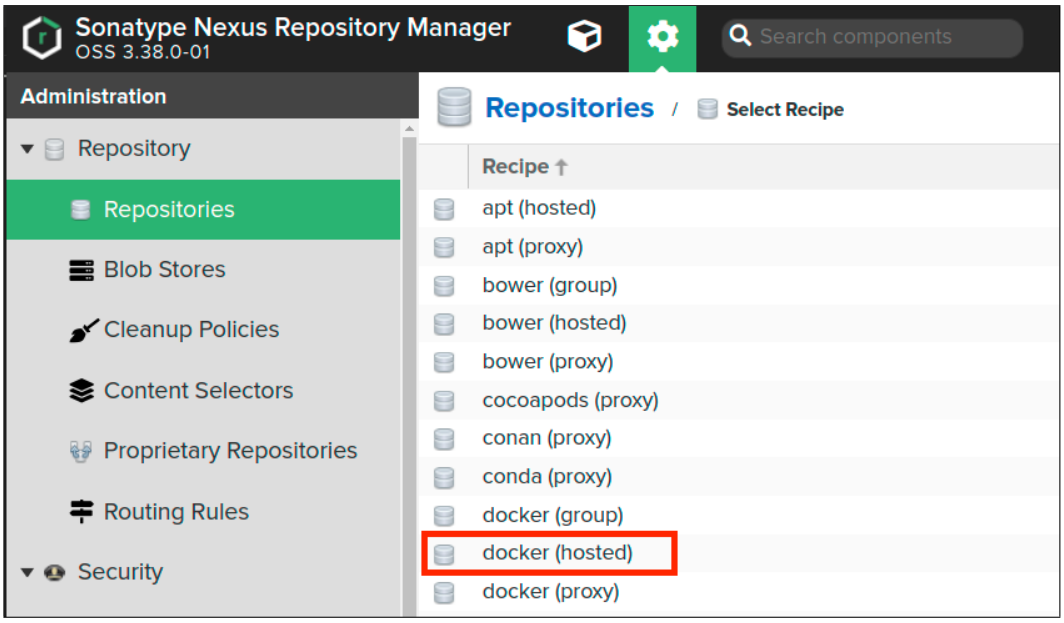

Figure 4.51: Choosing the repository type

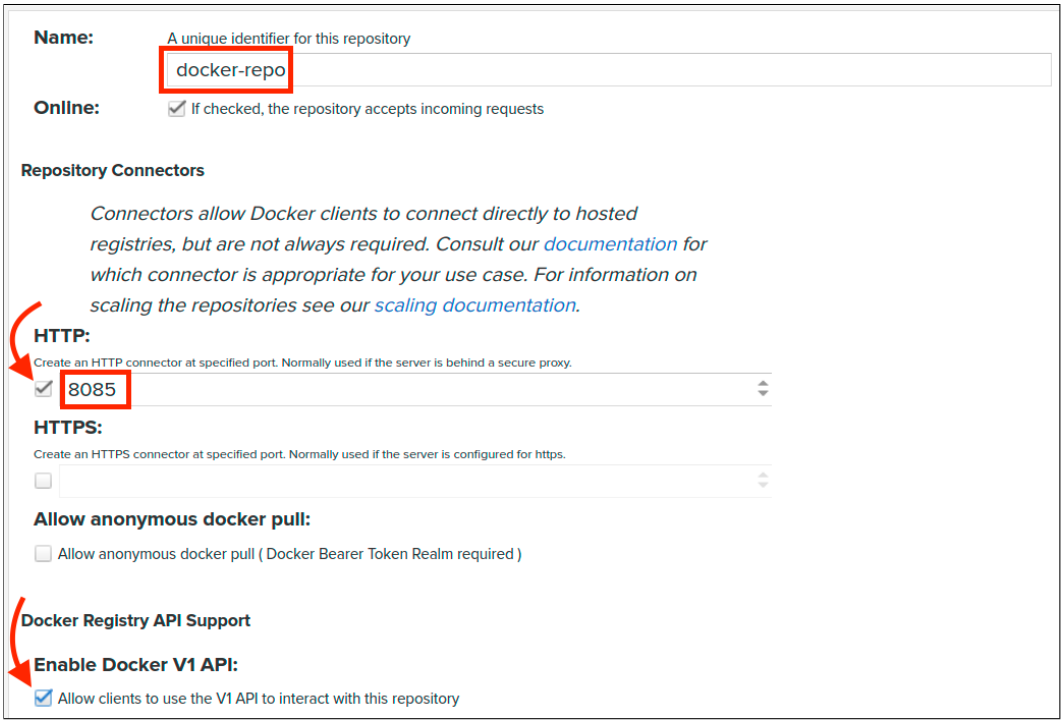

Figure 4.52: Configuring the Docker images repository

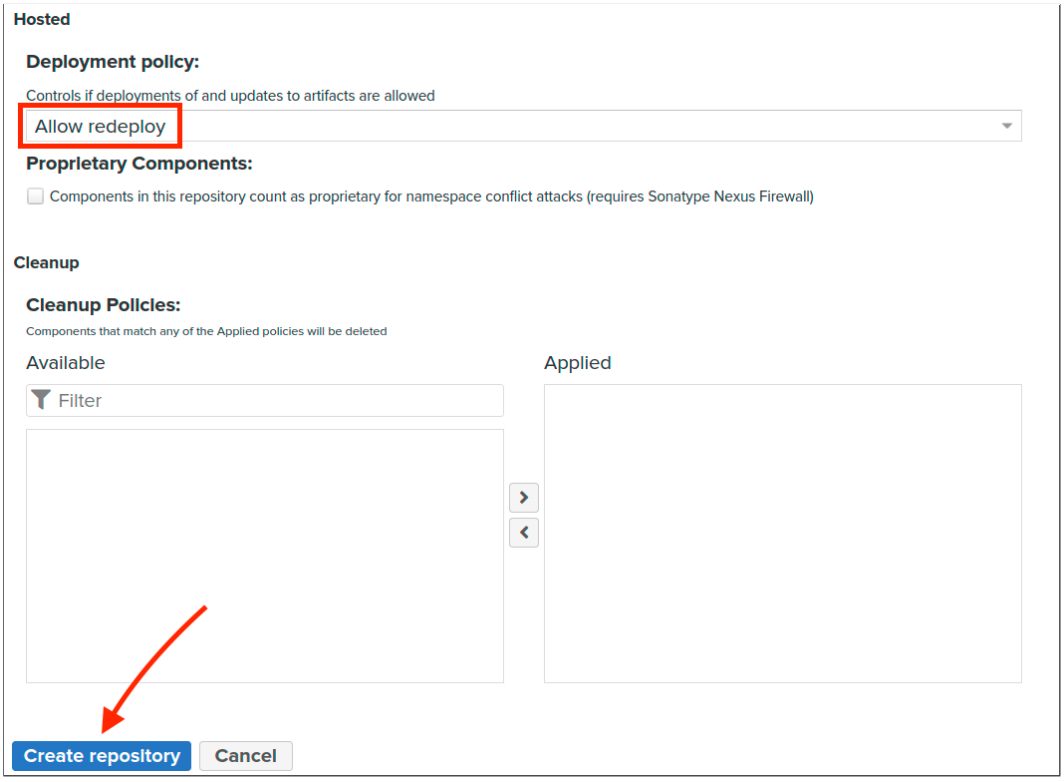

Figure 4.53: Confirming the Creation of the repository

After creating the nexus repositories, to give the pipelines the ability to push docker images to the nexus repositories we can either add a SSC to Jenkins and nexus or, like in our case, we could create "daemon.json" file inside /etc/docker which contains the insecure registries by using the command:

```
$ touch etc/docker/daemon.json
```

then writing this code in the file:

```
{
"insecure-registries" : ["172.18.0.4:8085"]
}
```

4.5.3.2 Triggers

The Daily Pipeline is triggered periodically every night at 9pm which change depends on the company

Figure 4.54: Daily pipeline trigger

4.5.3.3 Pipeline Stages

- Declaring The environment variable

  In this section of the Jenkinsfile we will declare some variables we will need in stage "Publish to Nexus" as illustrated in 4.5.7 from line 4 to 14.

- Cloning the Code from the Gitlab Repository(From the Dev Branch)

  In this stage we will clone the code from our Gitlab repository. The source code will be cloned from the branch that triggered the pipeline by running the jenkinsfile as illustrated in 4.5.7 from line 18 to 24.

- Analysing the code with SonarQube

  This stage is similar to the previous pipeline as illustrated in 4.5.7 from line 27 to 37.

- Testing the code with SonarQube Quality Gate

  This stage is similar to the previous jenkinsfile and it is illustrated in 4.5.7 from line 39 to 43.

- Building the App using NPM for the Front-End and MVN for the back-end

  This stage is similar to the previous pipeline and it is illustrated in 4.5.7 from line 46 to 53.

- Building the App using MVN for the Back-End

  This stage is similar to the previous pipeline and it is illustrated in 4.5.7 from line 56 to 64

- Deploying the artifacts to the Nexus repository

  In this stage will deploy the artifact in our case jar file to our nexus maven repository to facilitate version control in the future as illustrated in 4.5.7 from line 66 to 112.The figure 4.55 shows that the artifacts were pushed as a snapshot.

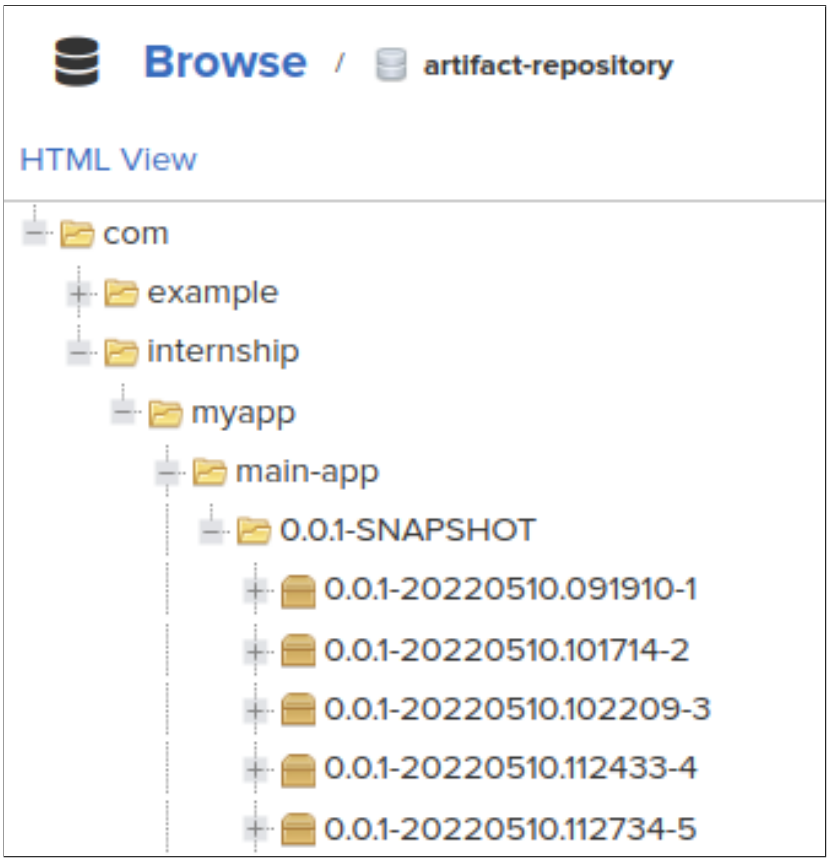

Figure 4.55: Artifacts pushed to nexus snapshot repository

- Dockerizing the App

  In this stage using Docker will build a docker image for our app along with the environment it needs to run as illustrated in 4.5.7 from line 114 to 118.

- Deploying the docker image to the Nexus repository

  In this stage after creating the image we will deploy it to a docker repository in Nexus so we can roll-back directly to a stable older version in case of a problem and deploy it to production server as illustrated in the annex 4.5.7 from line 119 to 125.The figure 4.56 shows that the docker image was pushed to nexus.

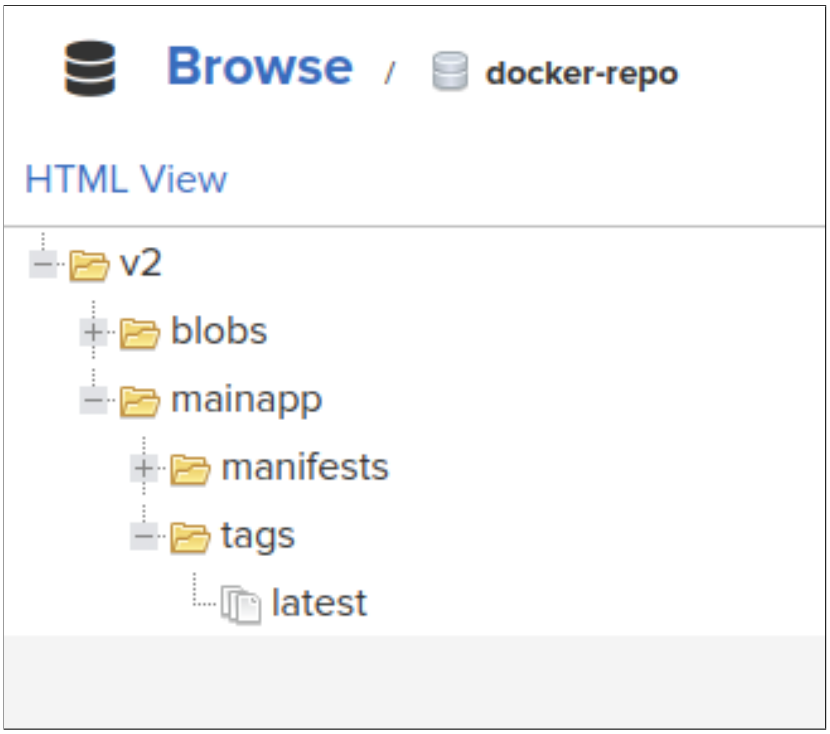

Figure 4.56: Docker image pushed to nexus repository

- Post Action

  This stage is similar to the previous pipeline and it is illustrated in 4.5.7 from line 127 to 134. The figure 4.57 shows that the mail is received when the Daily Pipeline is successful.

Figure 4.57: Daily Mail

The figure 4.58 shows the Jenkins dashboard output when the pipeline is successful.

Figure 4.58: Daily Stages

4.5.4 Deployment Pipeline

4.5.4.1 Creating the nexus Artifact Repository

The Deployment pipelines require the creation of a Release Artifact Repository.

- The figures 4.59, 4.60, 4.61 and 4.62 are the steps followed in order to create the nexus Release Repository

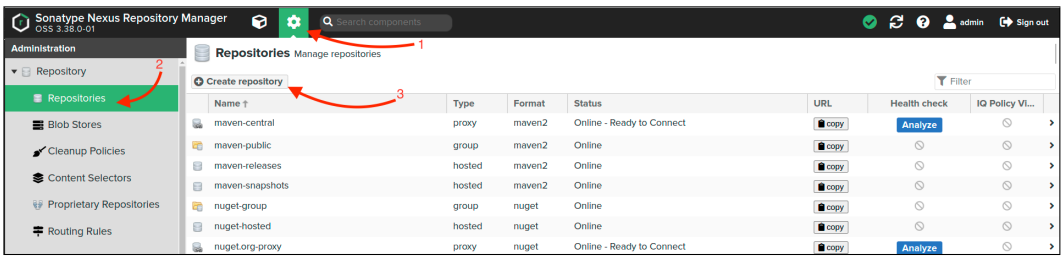

Figure 4.59: Creating the repository

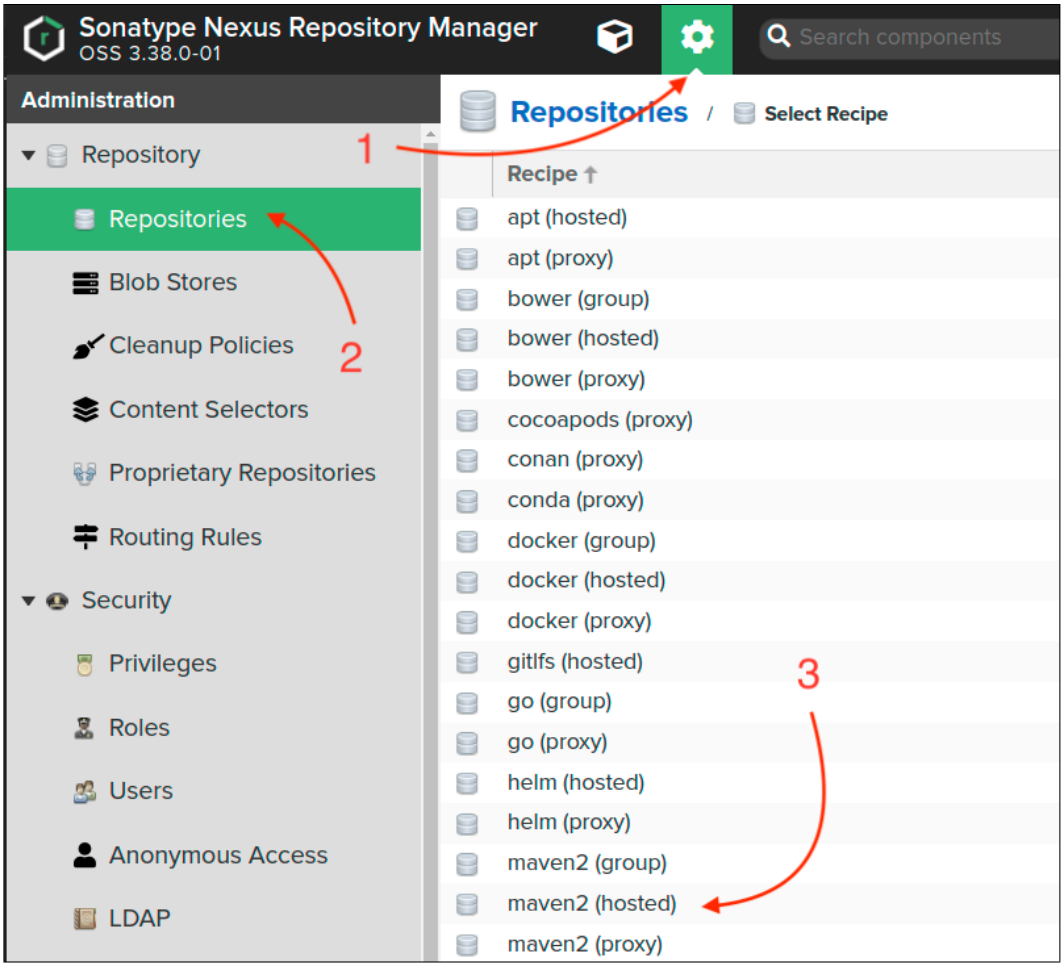

Figure 4.60: Choosing the Repository type

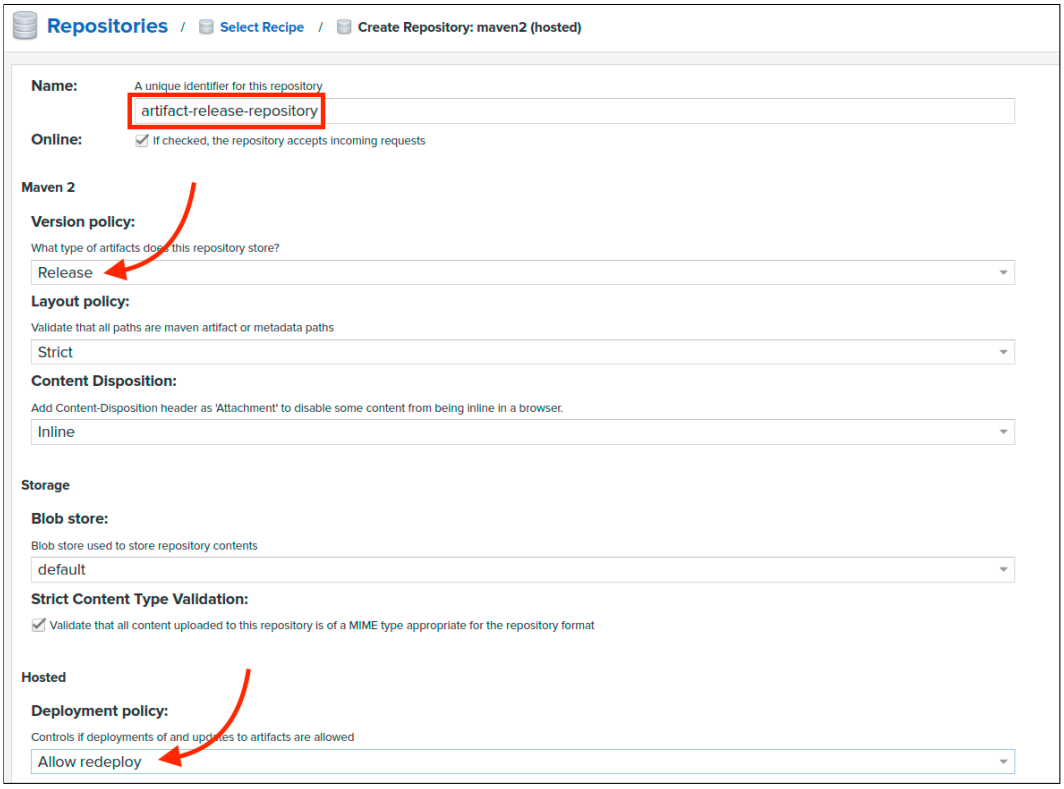

Figure 4.61: Configuring the Artifact Repository

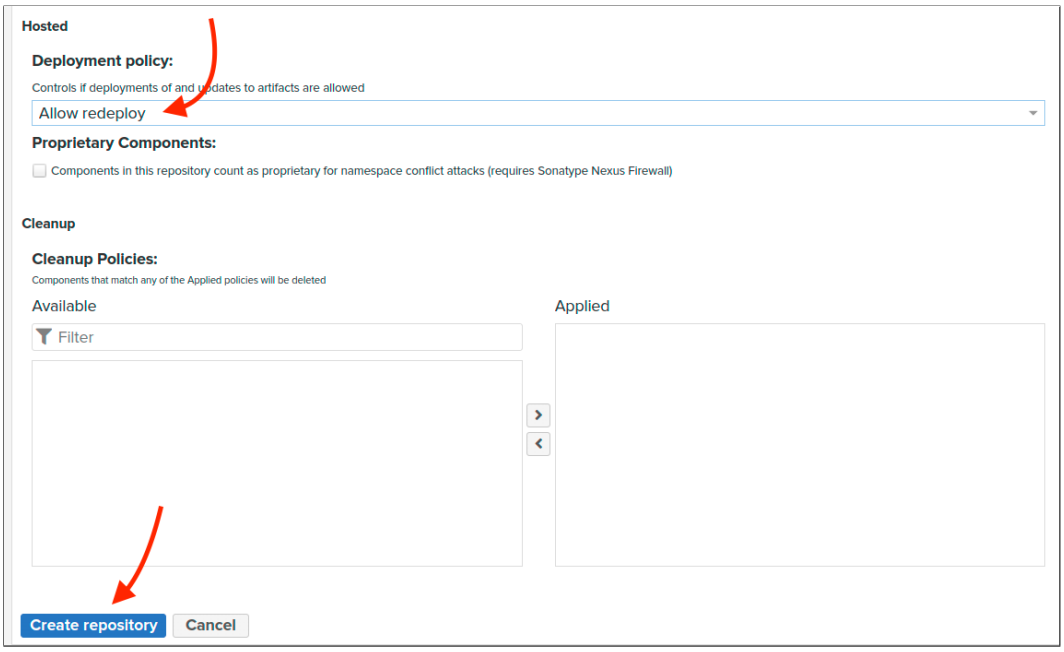

Figure 4.62: Confirming the Creation of the repository

4.5.4.2 Triggers

Using a Webhook to trigger the Release Pipeline/Continuous Deployment when we push on 'main' branch.The figures 4.63 and 4.64 shows the steps followed to create and configure the Deployment Webhook:

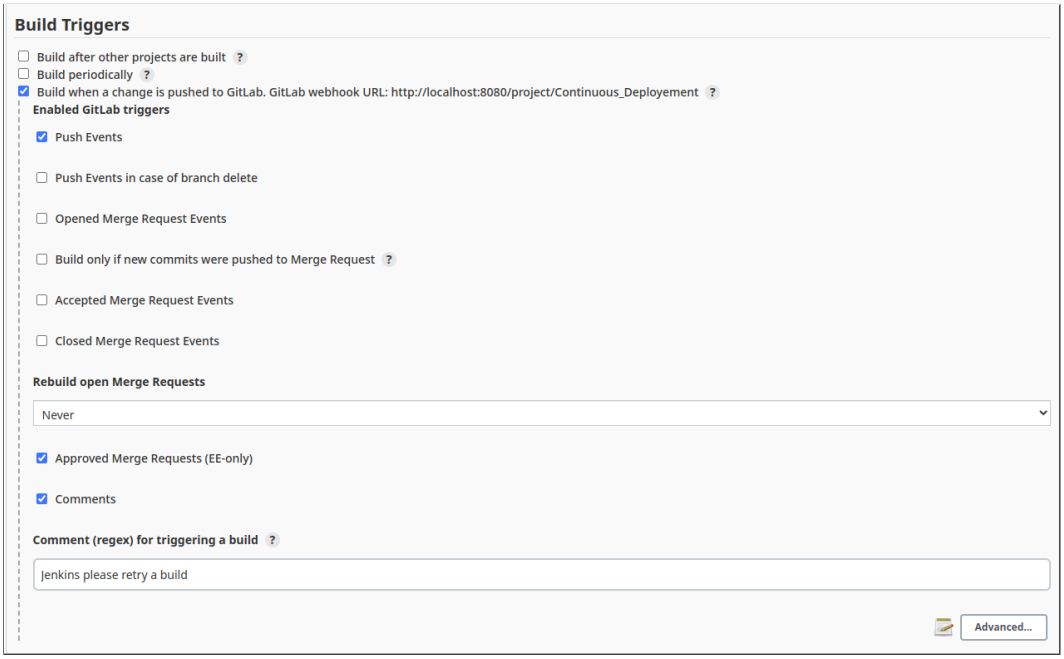

Figure 4.63: Continuous Deployment Trigger Configuration in Jenkins

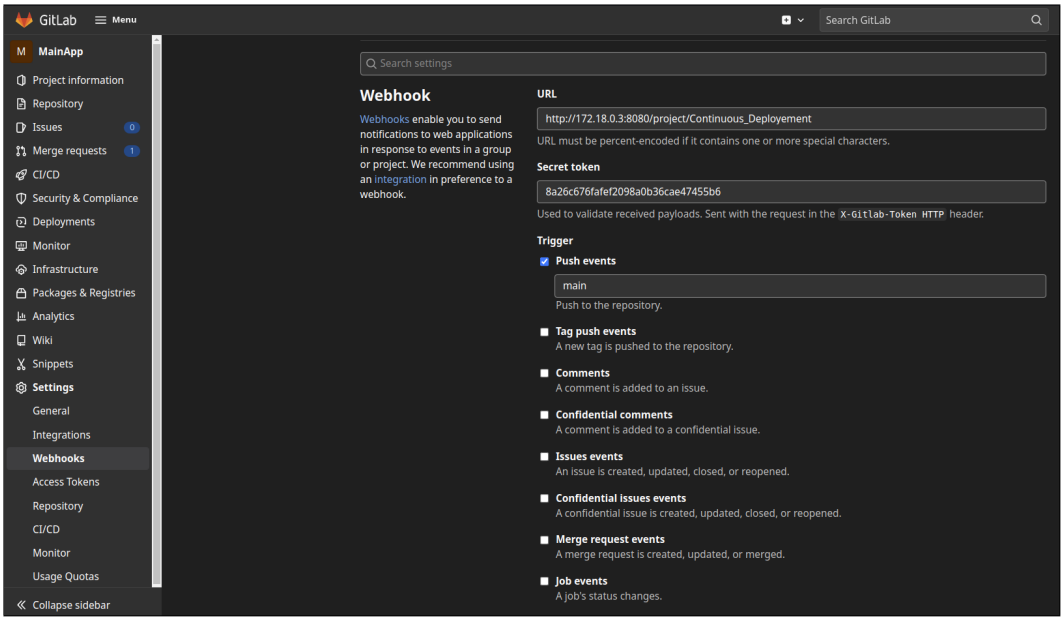

Figure 4.64: Continuous Deployment Webhook Configuration in Gitlab

4.5.4.3 Pipeline Stages

- Declaring The environment variable

  this section of the Jenkins file is similar to the previous section of "Daily Pipeline" and it is illustrated in 4.5.8 from line 3 to 14.

- Cloning the Code from the Gitlab Repository(From the Main Branch)

  In this stage we will clone the code from our Gitlab repository. The source code will be cloned from the branch that triggered the pipeline by running the jenkinsfile as illustrated in 4.5.7 from line 18 to 25

- Building the App using NPM for the front-end and MVN for the back-end

  This stage is similar to the previous pipeline and it is illustrated in 4.5.8 from line 28 to 37.

- Deploying the artifacts to the Nexus repository

  This stage is similar to Daily Pipeline and it is illustrated in annex4.5.8 from line 40 to 86 but the artifacts will be deployed as a version instead of a snapshot. The figure 4.65 shows that the artifacts were pushed as a release.

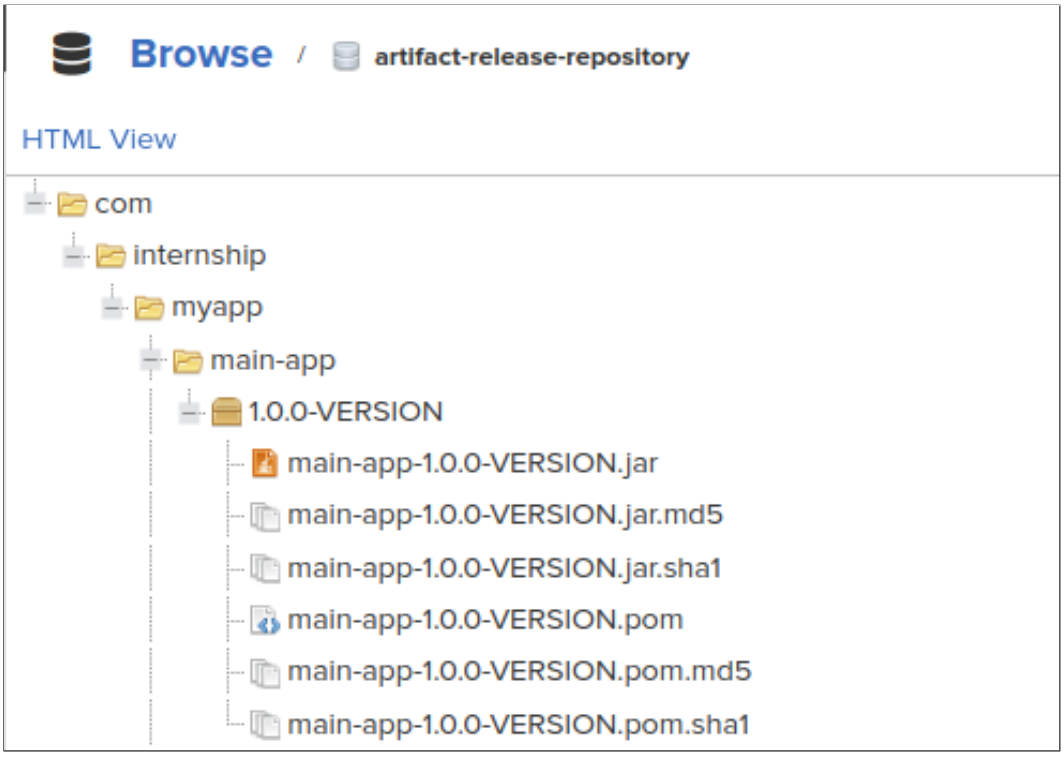

Figure 4.65: Deploying the artifacts to the nexus Release Repository

- Dockerizing the App

  In this stage using Docker will build a docker image for our app along with the environment it needs to run as illustrated in 4.5.8 from line 88 to 92.

- Deploying the docker image to the Nexus repository

  In this stage after creating the image we will deploy it to a docker repository in Nexus so we can roll-back directly to a stable older version in case of a problem and deploy it to production server as illustrated in 4.5.7 from line 94 to 100.

- Starting The Production Server

  In this Stage our Application is ready to move to production, so will start the Front-End and Back-End containers as illustrated in 4.5.8 from line 102 to 106. The figure 4.66 shows the application running after the continuous deployment pipeline is finished

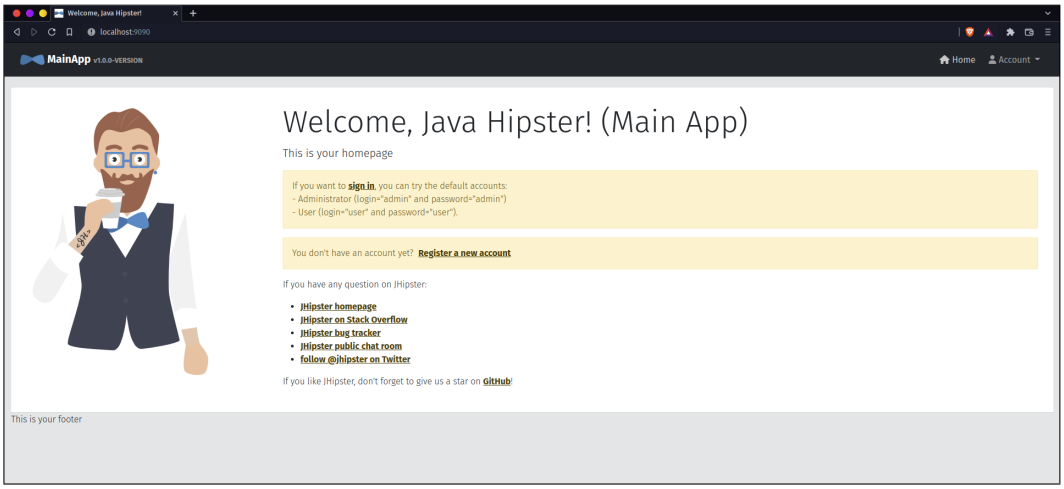

Figure 4.66: Application Running after the continuous deployment pipeline is finished

- Post Action

  This stage is similar to the previous pipeline and it is illustrated in the annex 4.5.8 from line 108 to 115. while the figure 4.67 shows that the mail is received when the Deployment Pipeline is successful.

Figure 4.67: Continuous Deployment Mail

The figure 4.68 shows the Jenkins dashboard output when the pipeline is successful.

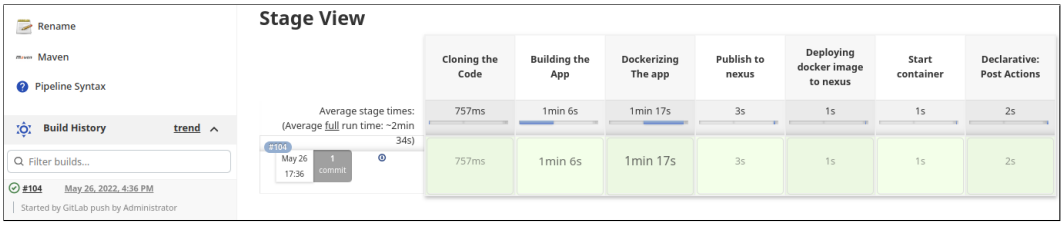

Figure 4.68: Continuous Deployment Stages

Conclusion

In this chapter, we explained using charts and more on how to create our DevOps platform from scratch and we have described the implementation of every part of our project. Moreover, we explained in details how to configure and run our project.

General Conclusion

As part of our end-of-studies internship, we provided VERMEG with a full CI/CD platform that helps automate the Integration and the Deployment of an application using docker.

In our report we have explained our technical choices as well as the main concepts used in our project.

Added to that we provided a full workflow explaining how the project works and how it can be integrated as well as providing all the scripts needed.

Throughout the internship, to successfully develop the platform, we adopted some powerful tools like Jenkins, Sonatype Nexus, SonarQube, GIT and Gitlab. Although we were confronted by time and technology constraints, it allowed us to sharpen our decision making and problem solving skills.

These four months of internship have been beneficial for us as we have learned a lot of new skills which helped us deepen our knowledge in the DevOps field.

As we mentioned previously, our system is working perfectly and all the problems encountered were solved successfully but there's still always room for improvement like making the CI/CD platform scalable. Therefore, one way to make the platform better is to add a containers Orchestrator like Kubernetes for example.

Annexes

```
#!/bin/bash
sudo apt install docker && apt install docker-compose
exec bash
```

Listing 4.5.1: Run installdocker.bash Script to install Docker

```
version: '2'
services:
  jenkins:
    image: ghassenchams/jenkinsdockernode:finalversion
    container_name: jenkins
    privileged: true
    user: root
    volumes:
      - ~/Documents/Internship/jenkins_data:/var/jenkins_home
      - /var/run/docker.sock:/var/run/docker.sock
    ports:
      - 8080:8080
      - 50003:50000
    networks:
      internship_default:
        ipv4_address: 172.18.0.3

  nexus:
    image: sonatype/nexus3:3.38.0
    container_name: nexus
    privileged: true
    user: root
    volumes:
      - ~/Documents/Internship/nexus-data:/nexus-data
    ports:
      - 8081:8081
      - 8085:8085
    networks:
      internship_default:
        ipv4_address: 172.18.0.4
  sonarqube:
    image: sonarqube:9.3.0-community
    container_name: sonarqube
    privileged: true
    user: root
    volumes:
```

```
      - ~/Documents/Internship/sonarqube_data:/opt/sonarqube/data
      - ~/Documents/Internship/sonarqube_extensions:/opt/sonarqube/extensions
      - ~/Documents/Internship/sonarqube_logs:/opt/sonarqube/logs
    ports:
      - 9000:9000
    networks:
      internship_default:
        ipv4_address: 172.18.0.5

  gitlab:
    image: 'gitlab/gitlab-ce:14.7.6-ce.0'
    container_name: gitlab
    privileged: true
    user: root
    volumes:
      - ~/Documents/Internship/gitlab/config:/etc/gitlab
      - ~/Documents/Internship/gitlab/logs:/var/log/gitlab
      - ~/Documents/Internship/gitlab/data:/var/opt/gitlab
    ports:
      - 80:80
      - 443:443
    networks:
      internship_default:
        ipv4_address: 172.18.0.2

volumes:
  nexus-data: {}
  sonarqube_data:
  sonarqube_extensions:
  sonarqube_logs:
networks:
    internship_default:
        driver: bridge
        ipam:
            config:
                - subnet: "172.18.0.0/16"
```

Listing 4.5.2: Docker-Compose File

```
#!/bin/bash
echo 'Jenkins admin password '
docker container exec jenkins sh -c "cat
↪  /var/jenkins_home/secrets/initialAdminPassowrd"
echo ' Nexus admin password '
```

```
docker container exec nexus bash -c "cat /nexus-data/admin.password "
echo ' Gitlab root password'
docker exec -it gitlab grep 'Password:' /etc/gitlab/initial_root_password
exec bash
```

Listing 4.5.3: initialpassword.bash Script

```
#!/bin/bash
docker cp plugins.bash jenkins:var/jenkins_home && \
docker container exec jenkins bash -c "bash var/jenkins_home/plugins.bash
exec bash
```

Listing 4.5.4: installplugins.bash

```
#!/bin/bash
jenkins-plugin-cli --plugins \
credentials:1087.v16065d268466 \
command-launcher:1.6 \
docker-java-api:3.1.5.2 \
docker-commons:1.19 \
sonar-quality-gates:1.3.1 \
sonar:2.14 \
docker-compose-build-step:1.0 \
docker-workflow:1.28 \
docker-plugin:1.2.7 \
git-server:1.10 \
git-client:3.11.0 \
git:4.11.0 \
gitlab-oauth:1.15 \
gitlab-logo:1.0.5 \
gitlab-plugin:1.5.29 \
gitlab-api:1.0.6 \
gitlab-merge-request-jenkins:2.0.0 \
junit:1.57 \
mailer:408.vd726a_1130320 \
maven-dependency-update-trigger:1.5 \
maven-deployment-linker:1.5.1 \
pipeline-maven:3.10.0 \
nexus-artifact-uploader:2.13 \
nexus-jenkins-plugin:3.14.405.v74e19a_0b_1a_1a_ \
nodejs:1.5.1 \
plain-credentials:1.8 \
docker-build-step:2.8 \
```

```
ssh-slaves:1.806.v2253cedd3295 \
sshd:3.1.0
exec bash
```

Listing 4.5.5: plugins.bash Script

```
pipeline {

    agent any
    stages {
       // ------ We Cloned the repo from the parameter above ------
        stage("Cloning the Code") {
            steps {
                script {
                    sh "echo cloning"
                }

            }
        }
           // ------ SonarQube Analysis -----
        stage('SonarQube Analysis') {
            steps{
                    script {

                                withSonarQubeEnv(installationName: 'sonar'){
                                   withMaven{
                                    sh "./mvnw sonar:sonar "
                                   }
                                }
                    }
            }
        }
         // ------ Sonar Quality Gate -----
        stage("Quality Gate") {
            steps {
                waitForQualityGate abortPipeline: true
            }
        }
        //       ------ Building The Project with mvn and npm -----
        stage("Building the App") {
            steps {
                script {
                    sh "npm install"
                    withMaven {
                    sh "./mvnw install"
```

```
                    }
                }
            }
        }
        //       ------ Back-End testing -----
        stage('Back-end tests') {
            steps {
                script{
                   try {
                        sh "./mvnw test"
                    } catch(err) {
                        throw err
                    } finally {
                      step([\$class: 'JUnitResultArchiver',
                      ↪  allowEmptyResults: true, testResults:
                      ↪  '**/target/surefire-reports/TEST-*.xml'])
                      }
                }
            }
        }
        //       ------ Front-End testing -----
    stage('Front-end tests') {
        steps{
            script{
                try {
                 sh 'npm install'
                 sh "npm test"
                } catch(err){
                     throw err
                  } finally {
             step([\$class: 'JUnitResultArchiver',  allowEmptyResults:
             ↪  true, testResults:
             ↪  '**/target/test-results/karma/TESTS-*.xml'])
            }
           }
        }
    }
        //       ------ Dockerizing The Project using docker ------
        stage("Dockerizing The app") {
             steps {
                sh "./mvnw -Pprod verify jib:dockerBuild"
            }
        }
    }
    post {
        success {
```

```
            mail bcc: '', body: 'The CI pipeline was successful', cc:
            ↪ 'chamssidine.abderrahim@etudiant-isi.utm.tn', from: '',
            ↪ replyTo: '', subject: 'Successful Build', to:
            ↪ 'oueslatighassen5201@gmail.com'
        }
        failure {
            mail bcc: '', body: 'The CI pipeline was a failure ', cc:
            ↪ 'chamssidine.abderrahim@etudiant-isi.utm.tn', from: '',
            ↪ replyTo: '', subject: 'failed Build', to:
            ↪ 'oueslatighassen5201@gmail.com'
        }
    }
}
```

P

Listing 4.5.6: Continous Integration Jenkins File

```
pipeline {

    agent any
    environment {
        // This can be nexus3 or nexus2
        NEXUS_VERSION = "nexus3"
        // This can be http or https
        NEXUS_PROTOCOL = "http"
        // Where your Nexus is running
        NEXUS_URL = "172.18.0.4:8081"
        // Repository where we will upload the artifact
        NEXUS_REPOSITORY = "artifact-repository"
        // Jenkins credential id to authenticate to Nexus OSS
        NEXUS_CREDENTIAL_ID = "nexus-user-credentials"
    }
    stages {
       // ------ Cloning the branch dev ------
        stage("Clone Code") {
            steps {
                script {
                    // Let's clone the source
                    git branch: 'dev', credentialsId:
                    ↪ 'be415b2c-6347-48a4-a24b-6cc743c73672', url:
                    ↪ 'http://172.18.02/root/mainapp.git'
                    }
            }
        }
           // ------ Code Analysis with SonaQube -----
```

```
27          stage('Analysing the Code') {
28              steps{
29                  script {
30                      withSonarQubeEnv(installationName: 'sonar'){
31                          withMaven{
32                              sh "./mvnw sonar:sonar "
33                          }
34                      }
35                  }
36              }
37          }
38                      // ------ Sonar Quality Gate -----
39          stage("Quality Gate ") {
40              steps {
41                  waitForQualityGate abortPipeline: true
42              }
43          }
44
45          //       ------ Building Angular Project with Npm -----
46          stage("Npm Install") {
47              steps {
48                  script {
49                      sh "npm install"
50                      sh "npm run build"
51                  }
52              }
53          }
54
55          //       ------ Building Jhipster Project with mvn -----
56          stage("MVN Build") {
57              steps {
58                  script {
59                      withMaven {
60                         sh "./mvnw install"
61                      }
62                  }
63              }
64          }
65                      // ------ Deploying the Artifact to Nexus -----
66          stage("Publish to Nexus") {
67              steps {
68                  script {
69                      // Read POM xml file using 'readMavenPom' step , this
                        ↪  step 'readMavenPom' is included in:
                        ↪  https://plugins.jenkins.io/pipeline-utility-steps
70                      def pom = readMavenPom file: 'pom.xml';
71                      writeMavenPom model: pom;
```

```
                    // Find built artifact under target folder
                    filesByGlob = findFiles(glob:
                    ↪  "target/*.${pom.packaging}");
                    // Print some info from the artifact found
                    echo "${filesByGlob[0].name} ${filesByGlob[0].path}
                    ↪  ${filesByGlob[0].directory}
                    ↪  ${filesByGlob[0].length}
                    ↪  ${filesByGlob[0].lastModified}"
                    // Extract the path from the File found
                    artifactPath = filesByGlob[0].path;
                    // Assign to a boolean response verifying If the
                    ↪  artifact name exists
                    artifactExists = fileExists artifactPath;

                    if(artifactExists) {
                        echo "*** File: ${artifactPath}, group:
                        ↪  ${pom.groupId}, packaging: ${pom.packaging},
                        ↪  version ${pom.version}";

                        nexusArtifactUploader(
                            nexusVersion: NEXUS_VERSION,
                            protocol: NEXUS_PROTOCOL,
                            nexusUrl: NEXUS_URL,
                            groupId: pom.groupId,
                            version: pom.version,
                            repository: NEXUS_REPOSITORY,
                            credentialsId: NEXUS_CREDENTIAL_ID,
                            artifacts: [
                                // Artifact generated such as .jar, .ear
                                ↪  and .war files.
                                [artifactId: pom.artifactId,
                                classifier: '',
                                file: artifactPath,
                                type: pom.packaging],

                                // Lets upload the pom.xml file for
                                ↪  additional information for Transitive
                                ↪  dependencies
                                [artifactId: pom.artifactId,
                                classifier: '',
                                file: "pom.xml",
                                type: "pom"]
                            ]
                        );

                    } else {
```

```
                        error "*** File: \${artifactPath}, could not be
                        ↪  found";
                    }
                }
            }
        }
                    // ------  Dockerizing the app using Docker -----
        stage("Dockerizing The App") {
            steps {
                sh "./mvnw -Pprod verify jib:dockerBuild"
            }
        }
        stage("deploying docker image to nexus") {
            steps {
                sh "docker login -u admin -p nexusserver 172.18.0.4:8085 "
                sh "docker tag mainapp 172.18.0.4:8085/mainapp:latest"
                sh "docker push 172.18.0.4:8085/mainapp:latest"
            }
        }
}
    post {
        success {
            mail bcc: '', body: 'The Daily pipeline was successful', cc:
            ↪  'chamssidine.abderrahim@etudiant-isi.utm.tn', from: '',
            ↪  replyTo: '', subject: 'Successful Build', to:
            ↪  'oueslatighassen5201@gmail.com'
        }
        failure {
            mail bcc: '', body: 'The Daily pipeline was a failure ', cc:
            ↪  'chamssidine.abderrahim@etudiant-isi.utm.tn', from: '',
            ↪  replyTo: '', subject: 'failed Build', to:
            ↪  'oueslatighassen5201@gmail.com'
        }
    }
}
```

Listing 4.5.7: Daily Jenkins File

```
pipeline {
    agent any
    environment {
        // This can be nexus3 or nexus2
        NEXUS_VERSION = "nexus3"
        // This can be http or https
        NEXUS_PROTOCOL = "http"
```

```
8           // Where your Nexus is running
9           NEXUS_URL = "172.18.0.4:8081"
10          // Repository where we will upload the artifact
11          NEXUS_REPOSITORY = "artifact-release-repository"
12          // Jenkins credential id to authenticate to Nexus OSS
13          NEXUS_CREDENTIAL_ID = "nexus-user-credentials"
14      }
15
16      stages {
17         //     -------- Cloning the branch main-------
18          stage("Cloning the Code") {
19              steps {
20                  script {
21                      // Let's clone the source
22                      git branch: 'main', credentialsId:
                        ↪  'be415b2c-6347-48a4-a24b-6cc743c73672', url:
                        ↪  'http://172.18.02/root/mainapp.git'
23                       }
24                  }
25          }
26
27          //       ------ Building The Application -----
28          stage("Building the App") {
29              steps {
30                  script {
31                      sh "npm install"
32                      withMaven {
33                      sh "./mvnw install"
34                      }
35                  }
36              }
37          }
38
39         //       ------ Deploying Jar Artifact to Nexus Repository -----
40          stage("Publish to nexus") {
41              steps {
42                  script {
43                      // Read POM xml file using 'readMavenPom' step , this
                        ↪  step 'readMavenPom' is included in:
                        ↪  https://plugins.jenkins.io/pipeline-utility-steps
44                      def pom = readMavenPom file: 'pom.xml';
45                      writeMavenPom model: pom;
46                      // Find built artifact under target folder
47                      filesByGlob = findFiles(glob:
                        ↪  "target/*.${pom.packaging}");
48                      // Print some info from the artifact found
```

```
49                      echo "${filesByGlob[0].name} ${filesByGlob[0].path}
                        ↪  ${filesByGlob[0].directory}
                        ↪  ${filesByGlob[0].length}
                        ↪  ${filesByGlob[0].lastModified}"
50                      // Extract the path from the File found
51                      artifactPath = filesByGlob[0].path;
52                      // Assign to a boolean response verifying If the
                        ↪  artifact name exists
53                      artifactExists = fileExists artifactPath;
54
55                      if(artifactExists) {
56                          echo "*** File: ${artifactPath}, group:
                            ↪  ${pom.groupId}, packaging: ${pom.packaging},
                            ↪  version ${pom.version}";
57
58                          nexusArtifactUploader(
59                              nexusVersion: NEXUS_VERSION,
60                              protocol: NEXUS_PROTOCOL,
61                              nexusUrl: NEXUS_URL,
62                              groupId: pom.groupId,
63                              version: pom.version,
64                              repository: NEXUS_REPOSITORY,
65                              credentialsId: NEXUS_CREDENTIAL_ID,
66                              artifacts: [
67                                  // Artifact generated such as .jar, .ear
                                    ↪  and .war files.
68                                  [artifactId: pom.artifactId,
69                                  classifier: '',
70                                  file: artifactPath,
71                                  type: pom.packaging],
72
73                                  // Lets upload the pom.xml file for
                                    ↪  additional information for Transitive
                                    ↪  dependencies
74                                  [artifactId: pom.artifactId,
75                                  classifier: '',
76                                  file: "pom.xml",
77                                  type: "pom"]
78                              ]
79                          );
80
81                      } else {
82                          error "*** File: \${artifactPath}, could not be
                            ↪  found";
83                      }
84                  }
85              }
```

```
86          }
87           //       ------ Dockerizing The Project using docker ------
88          stage("Dockerizing The app") {
89               steps {
90                  sh "./mvnw -Pprod verify jib:dockerBuild"
91              }
92          }
93          //------------- Deploying docker image to nexus ------------
94          stage("Deploying docker image to nexus") {
95              steps {
96                  sh "docker login -u admin -p nexusserver 172.18.0.4:8085 "
97                  sh "docker tag mainapp 172.18.0.4:8085/mainapp:latest"
98                  sh "docker push 172.18.0.4:8085/mainapp:latest"
99              }
100         }
101         //----------- Start both back-end and Front-end containers
            ↪  ----------
102         stage('Start container') {
103             steps {
104                 sh 'docker-compose -f src/main/docker/app.yml up -d'
105             }
106         }
107     }
108     post {
109         success {
110             mail bcc: '', body: 'The CD pipeline was successful ', cc:
                ↪ 'chamssidine.abderrahim@etudiant-isi.utm.tn', from: '',
                ↪ replyTo: '', subject: 'Successful Build', to:
                ↪ 'oueslatighassen5201@gmail.com'
111         }
112         failure {
113             mail bcc: '', body: 'The CD pipeline was a failure ', cc:
                ↪ 'chamssidine.abderrahim@etudiant-isi.utm.tn', from: '',
                ↪ replyTo: '', subject: 'failed Build', to:
                ↪ 'oueslatighassen5201@gmail.com'
114         }
115     }
116 }
```

Listing 4.5.8: Continuous Deployment Jenkins File