
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.

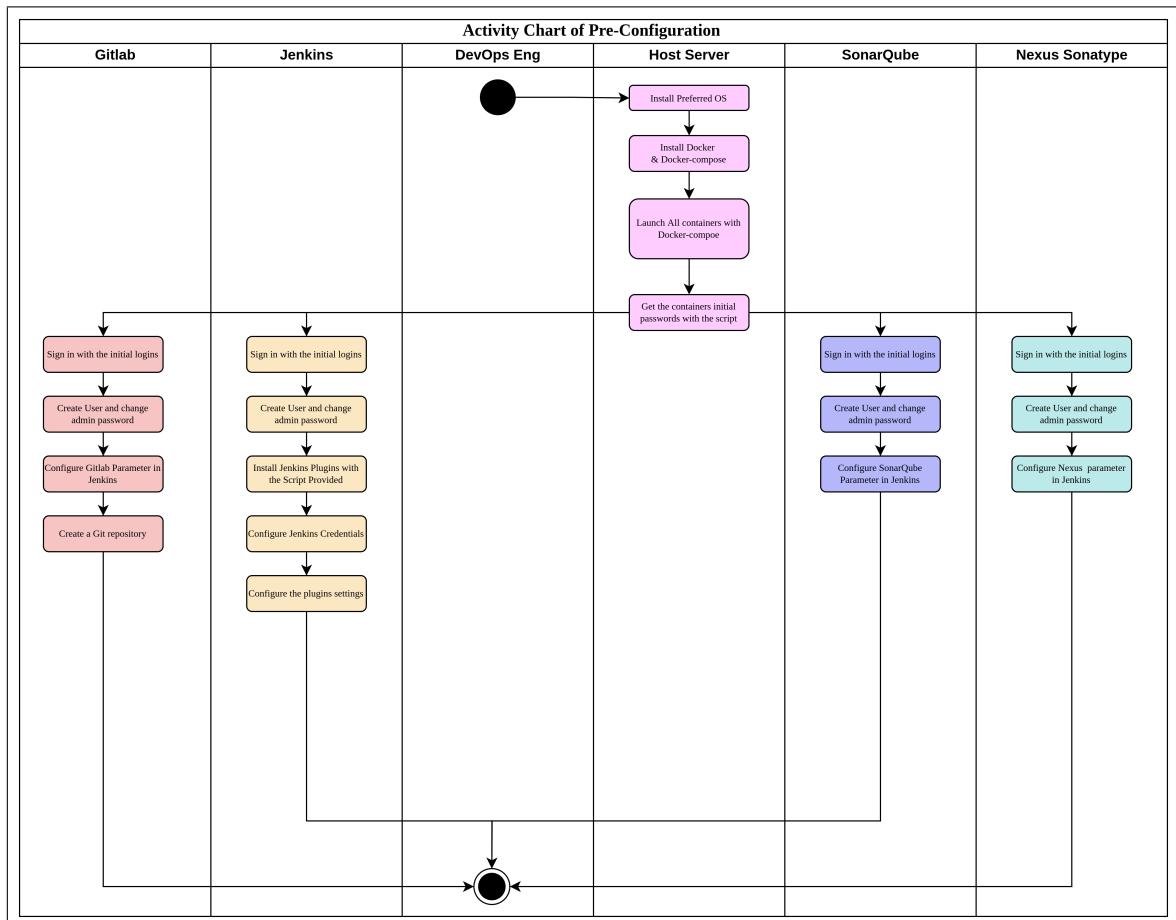


Figure 4.1: Activity Chart of Pre-configuration

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.

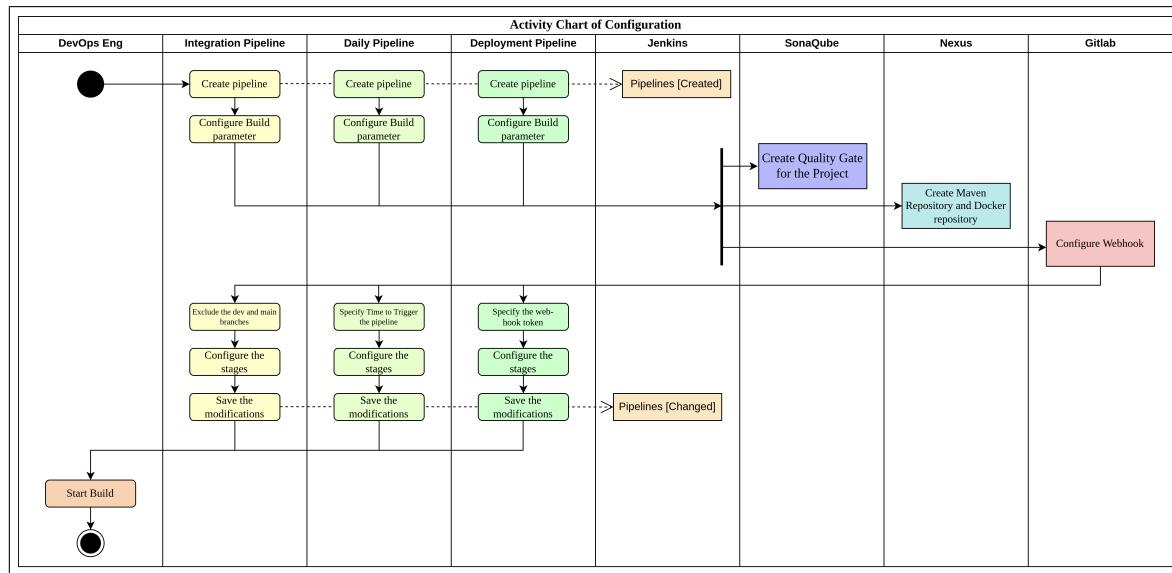


Figure 4.2: Activity Chart of configuration

4.1.3 Continuous Integration pipeline

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

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

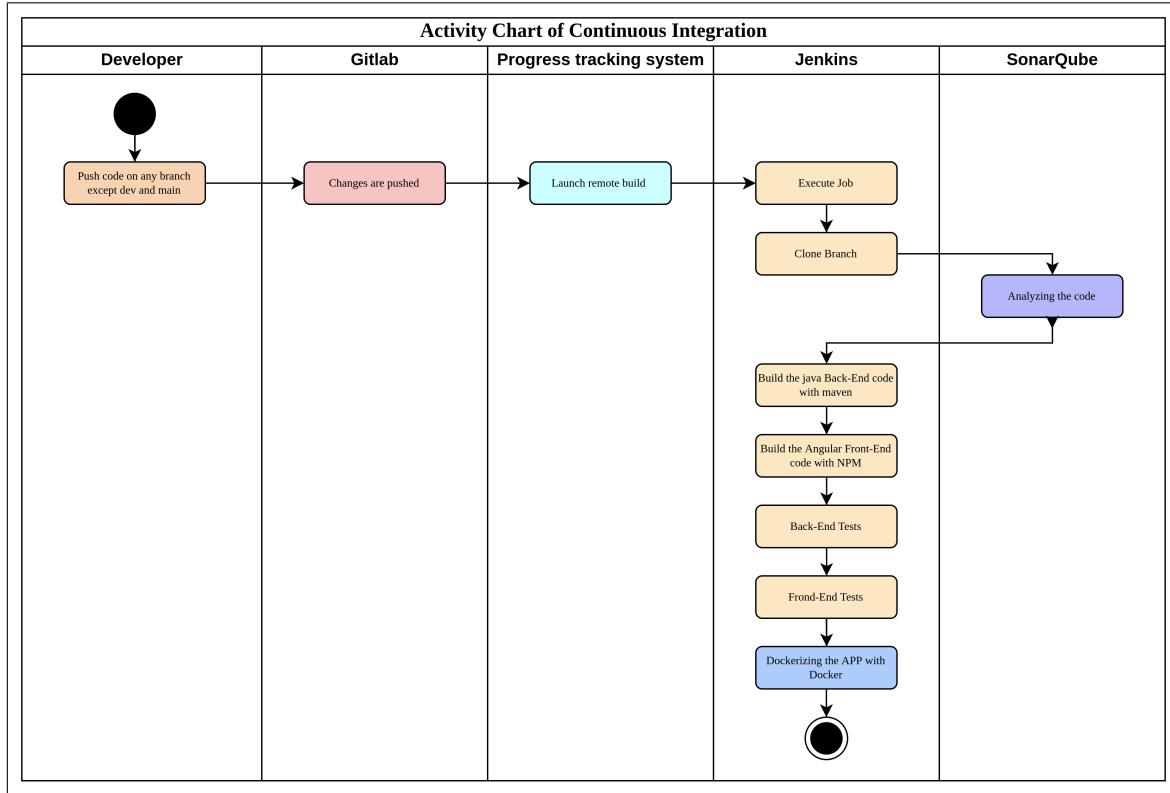


Figure 4.3: Continuous-Integration's Activity Chart

4.1.4 Daily pipeline

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

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

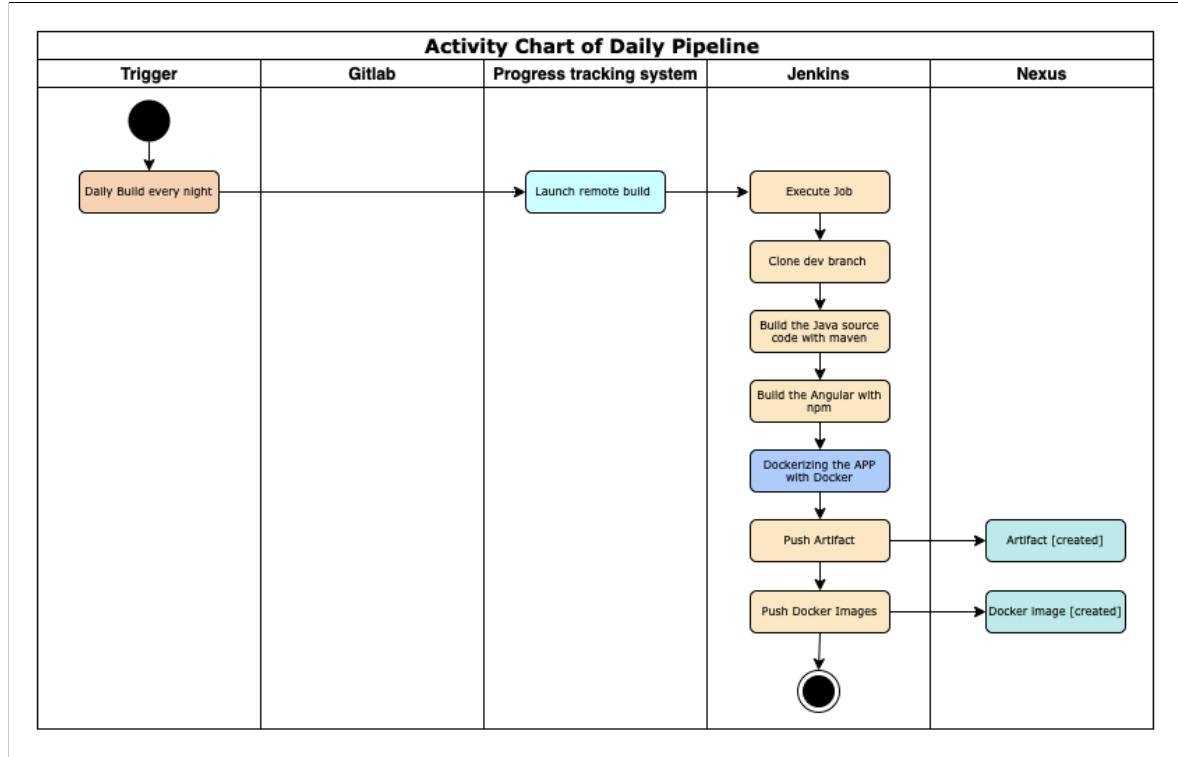


Figure 4.4: Daily pipeline's Activity Chart

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.

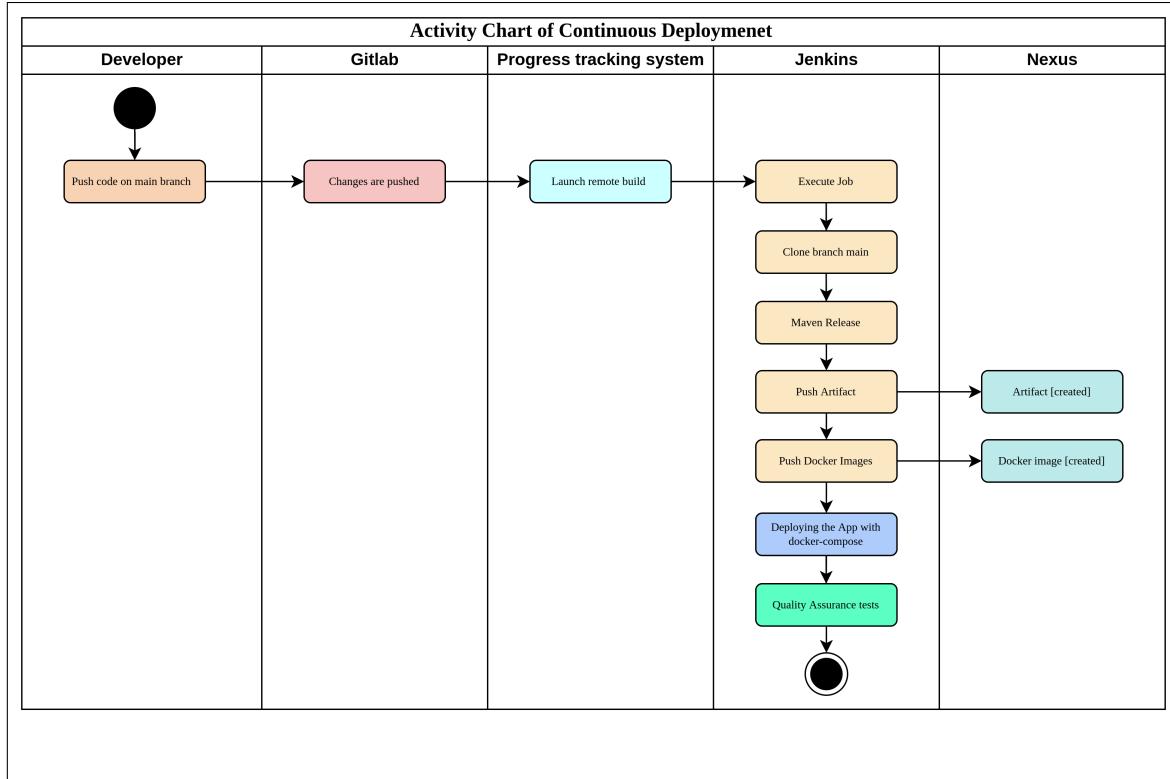


Figure 4.5: Continuous Deployment's Activity Chart

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 to 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 map 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 its 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:

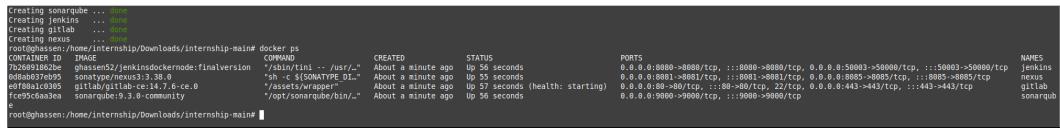
```
$ sudo docker-compose up -d
```

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
```



```

Creating sonarqube ... done
Creating jenkins ... done
Creating gitlab ... done
Creating nexus ... done
root@ghassen:/home/internship/downloads/internship-main# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS               NAMES
7f2a20037e09      sonatype/nexus3:3.38.0   "/bin/sh -c $SONATYPE_DL..."   About a minute ago   Up 55 seconds          0.0.0.0:8880->8880/tcp, 0.0.0.0:59883->59889/tcp, ...:59880->59880/tcp   nexus
edf88a1c03b5      gitlab/gitlab-ce:14.7.6-ce.0   "/assets/wrapper"    About a minute ago   Up 57 seconds (health: starting)   0.0.0.0:88->80/tcp, :88->88/tcp, 22/tcp, 0.0.0.0:443->443/tcp, ...:443->443/tcp   gitlab
fce95c0a3ea5      sonarqube:9.3.8-community   "/opt/sonarqube/bin..."   About a minute ago   Up 56 seconds          0.0.0.0:9000->9000/tcp, ...:9000->9000/tcp   sonarqube

```

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.

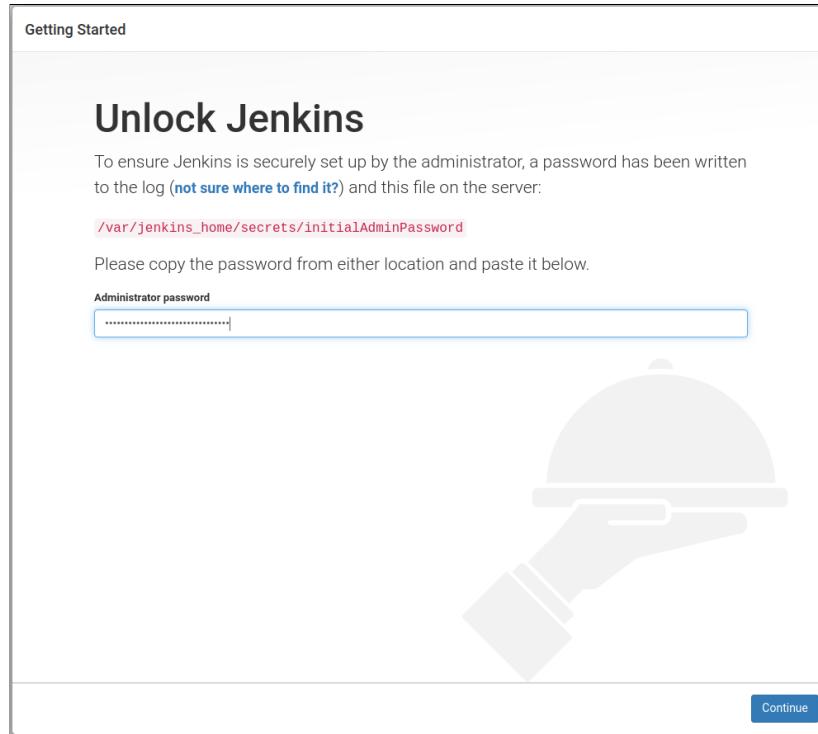


Figure 4.7: Entering Jenkins initial password

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:

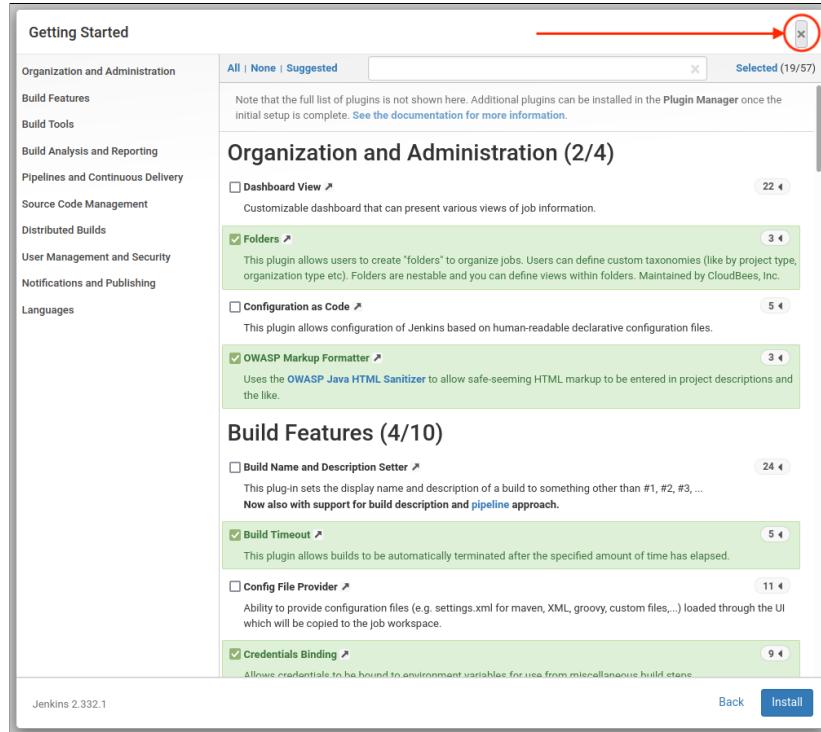


Figure 4.9: Jenkins User Interface

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

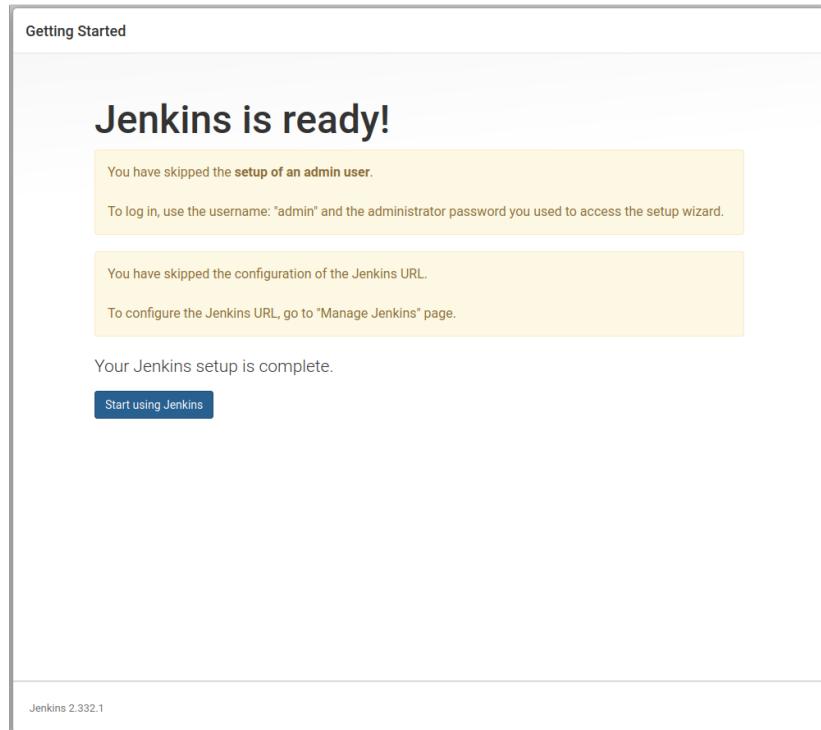


Figure 4.10: Jenkins User Interface

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

A screenshot of the Jenkins User Configuration page for the user 'admin'. It shows sections for 'Password' (with fields for 'Password' and 'Confirm Password'), 'SSH Public Keys' (empty), and 'Session Termination' (with 'Save' and 'Apply' buttons). A red box highlights the 'Password' section, and a red arrow points to the 'Save' button.

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

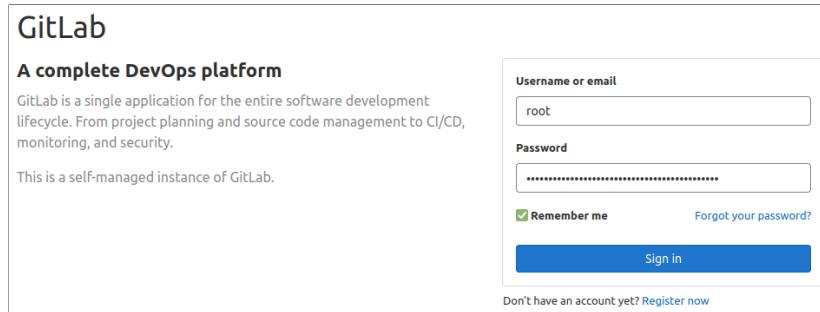


Figure 4.13: Gitlab's Sign in Menu

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

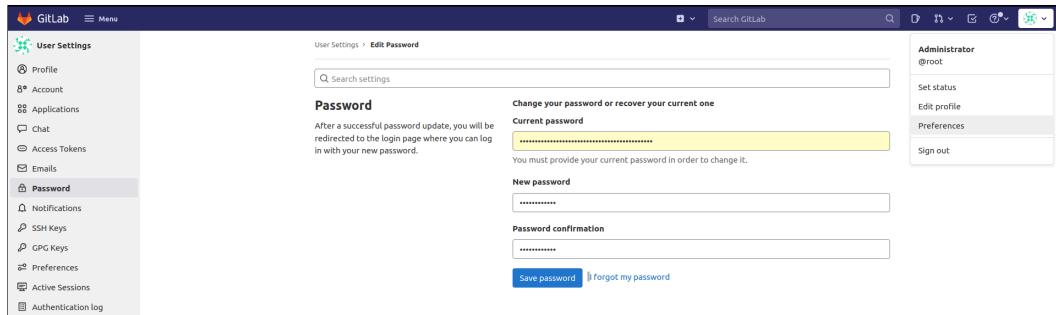


Figure 4.15: Changing Gitlab's password

4.3.5 SonarQube Container

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

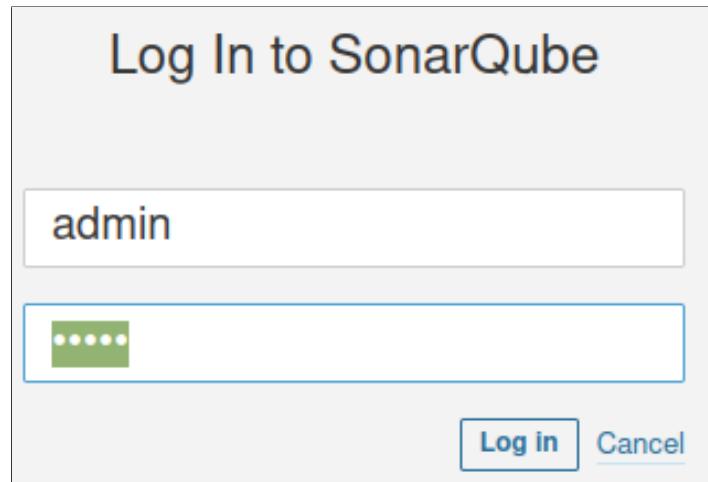


Figure 4.16: SonarQube Sign in

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

A screenshot of the SonarQube password update form. The title 'Update your password' is at the top. A note below says 'This account should not use the default password.' The form has three input fields: 'Old Password *' with four dots, 'New Password *' with six dots, and 'Confirm Password *' with six dots. Below the fields is a 'Update' button.

Figure 4.17: Change SonarQube password

The SonarQube Home Menu is shown in the figure 4.18

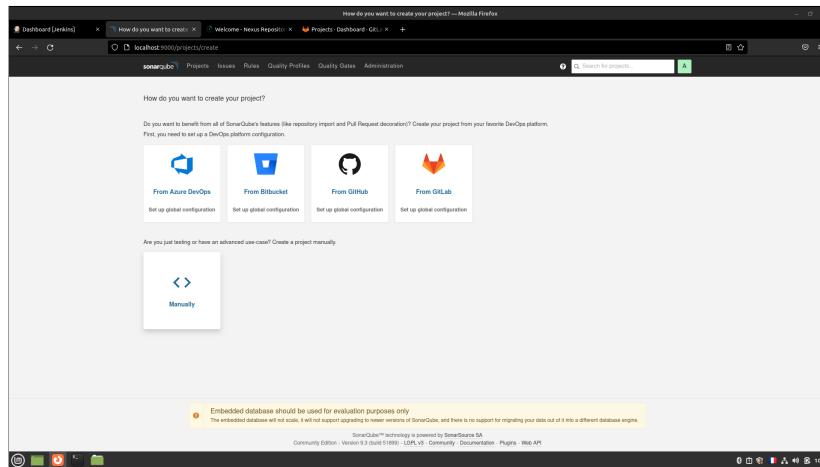


Figure 4.18: SonarQube Home Menu

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:

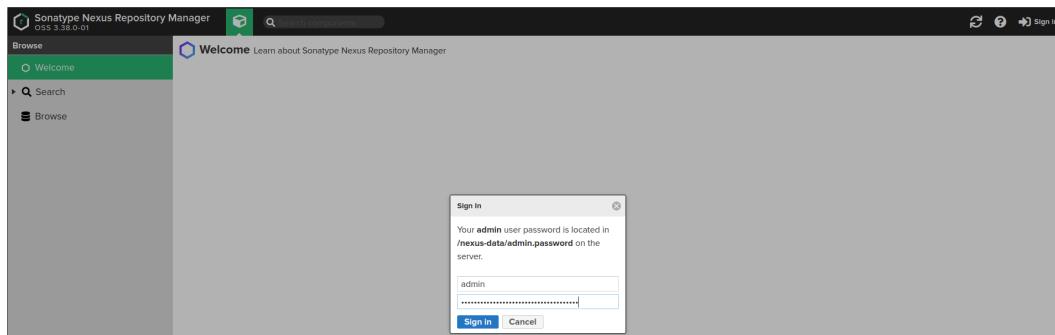


Figure 4.20: Sign In with initial password and admin as username

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

Please choose a password for the admin user 2 of 4

New password:

Confirm password:

[Back](#)

[Next](#)

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

Configure Anonymous Access 3 of 4

Enable anonymous access means that by default, users can search, browse and download components from repositories without credentials. Please **consider the security implications for your organization.**

Disable anonymous access should be chosen with care, as it **will require credentials for all** users and/or build tools.

[More information](#)

Enable anonymous access
 Disable anonymous access

[Back](#)

[Next](#)

Figure 4.22: Disable anonymous access

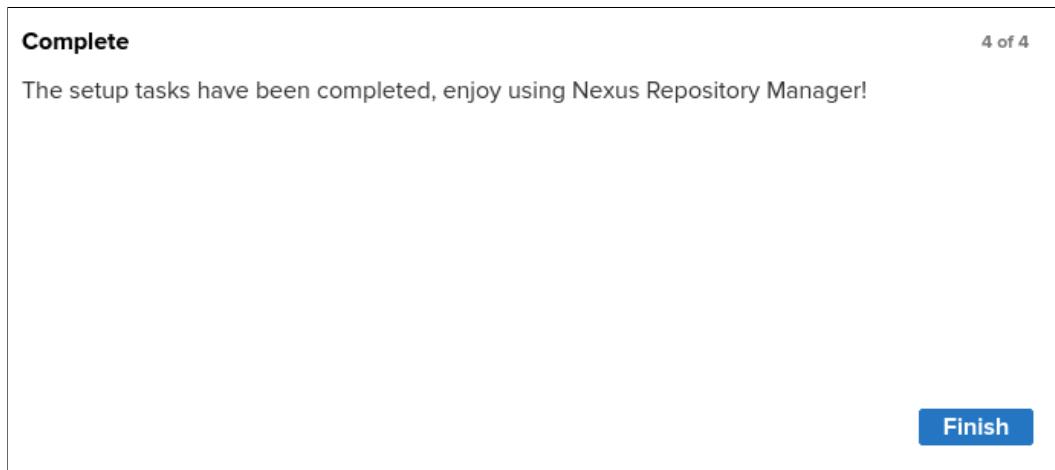


Figure 4.23: Nexus Setup Completed

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

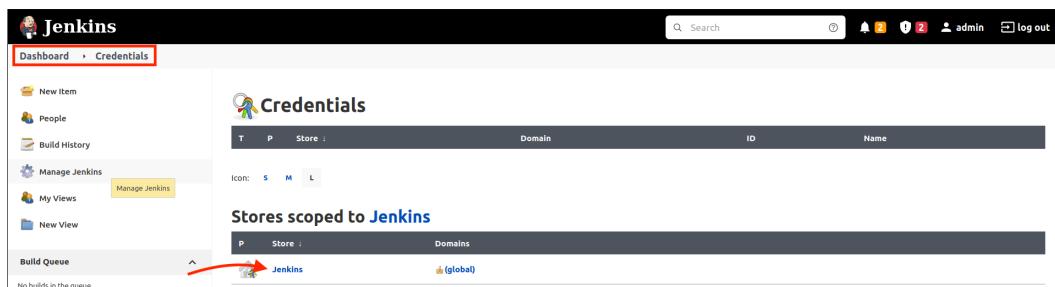


Figure 4.24: Jenkins Credentials Interface

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

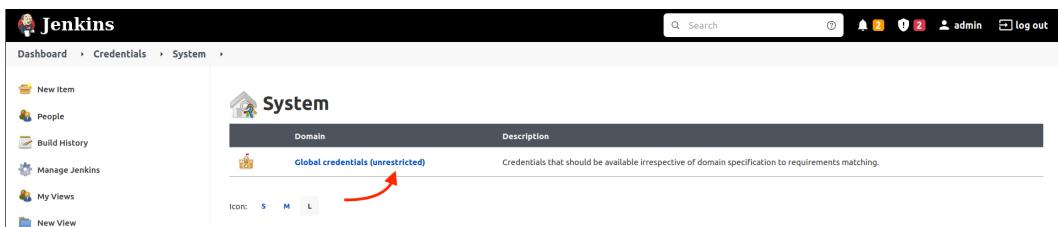


Figure 4.25: Credentials Interface

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

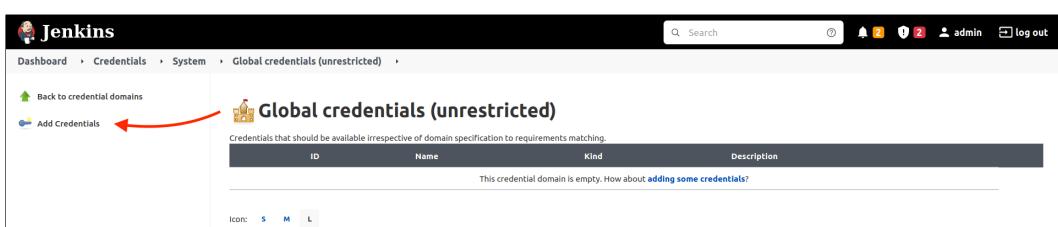


Figure 4.26: Adding Credentials

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

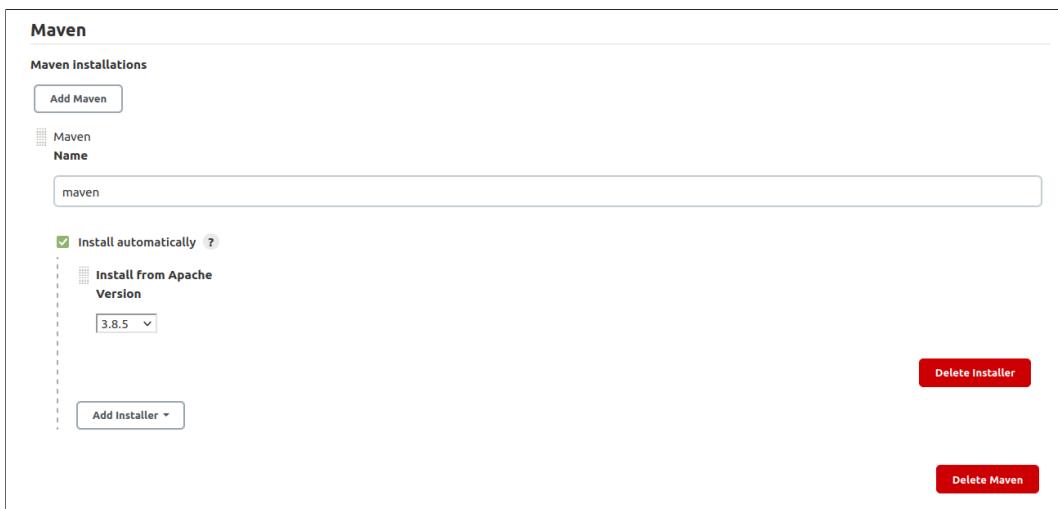


Figure 4.27: Integrating Maven with Jenkins

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

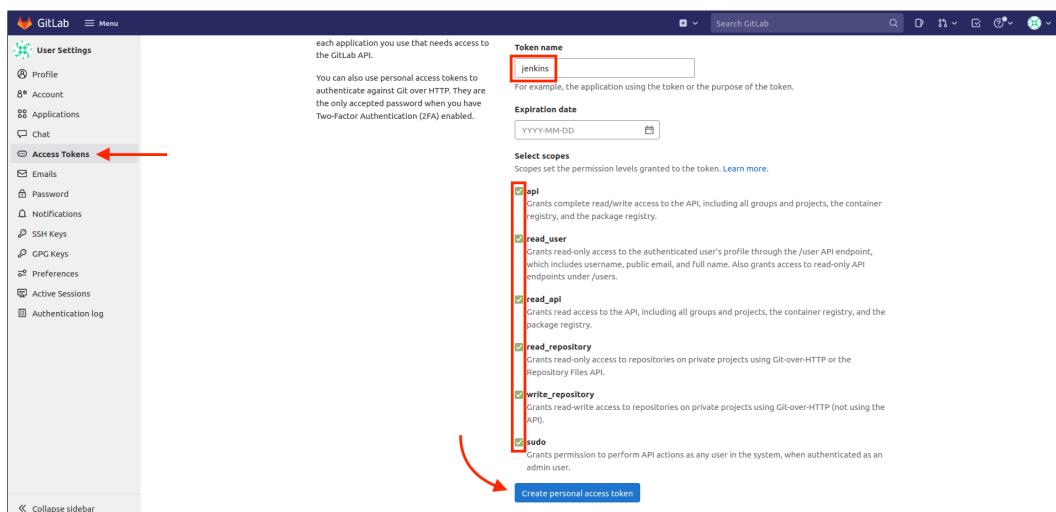


Figure 4.29: Access Token Parameter from Gitlab

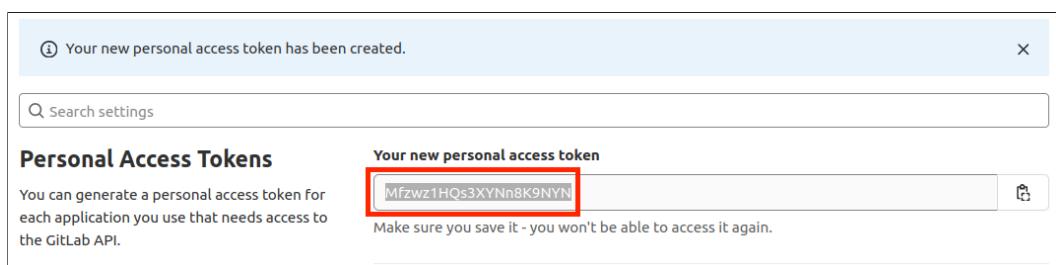


Figure 4.30: Get the Access Token from Gitlab

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

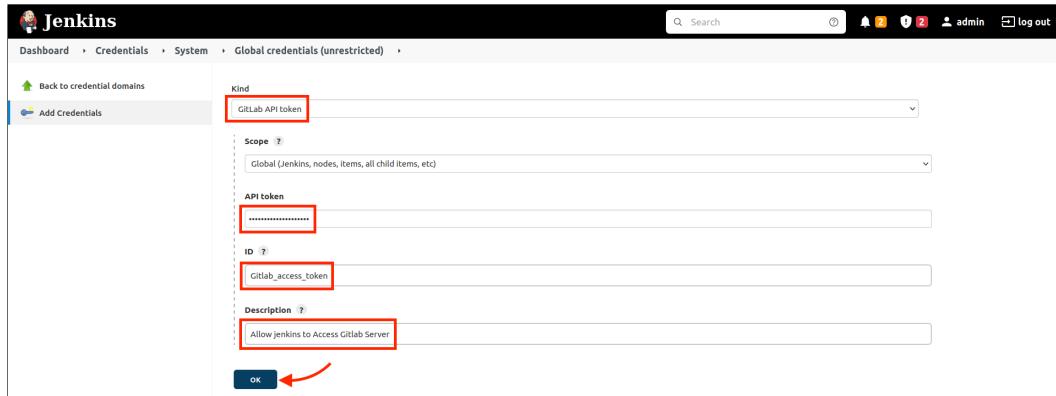


Figure 4.31: Configure Gitlab Credentials in Jenkins

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 .

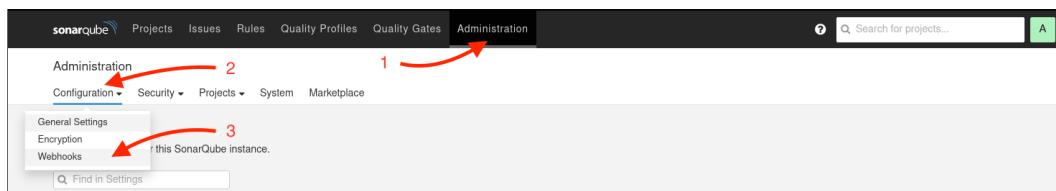


Figure 4.32: Accessing Webhooks settings

Chapter 4. Workflow Implementation

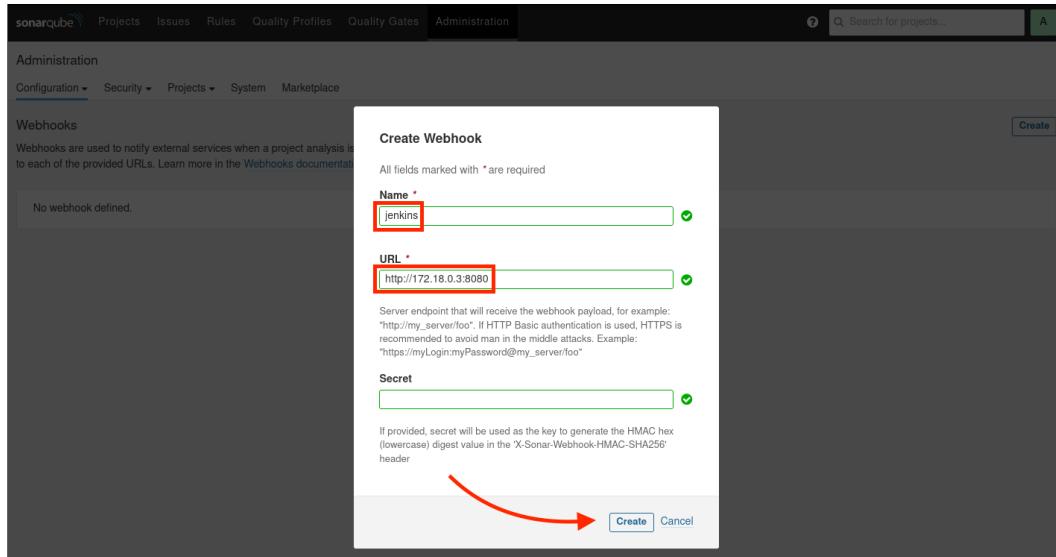


Figure 4.33: Creating the Webhook

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

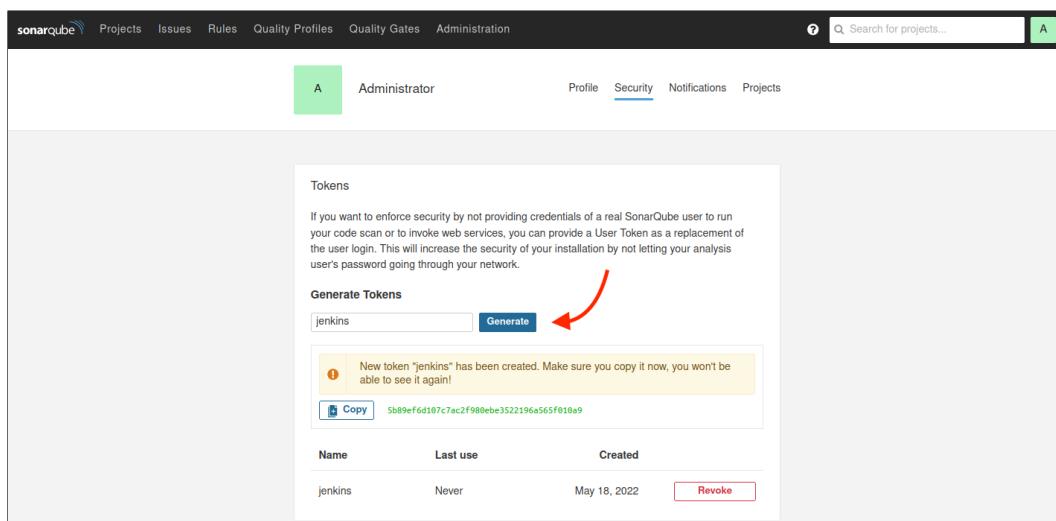


Figure 4.34: Get SonarQube Token

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

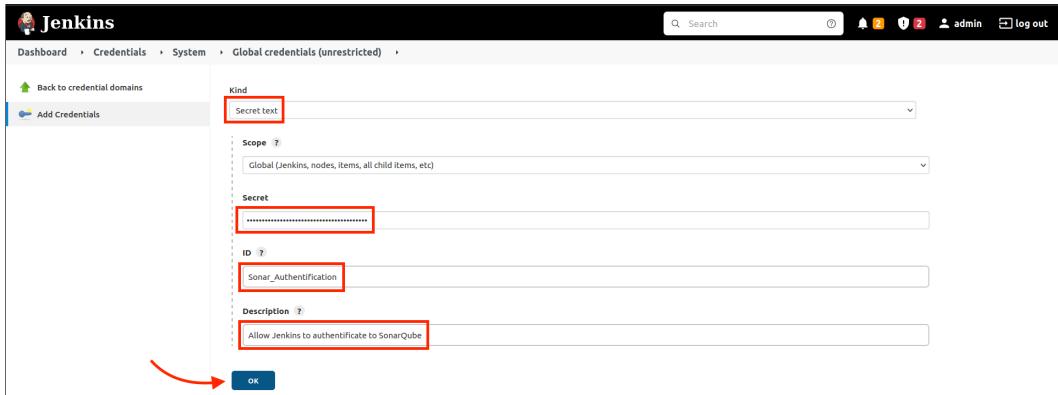


Figure 4.35: Configure SonarQube Credentials in Jenkins

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 .

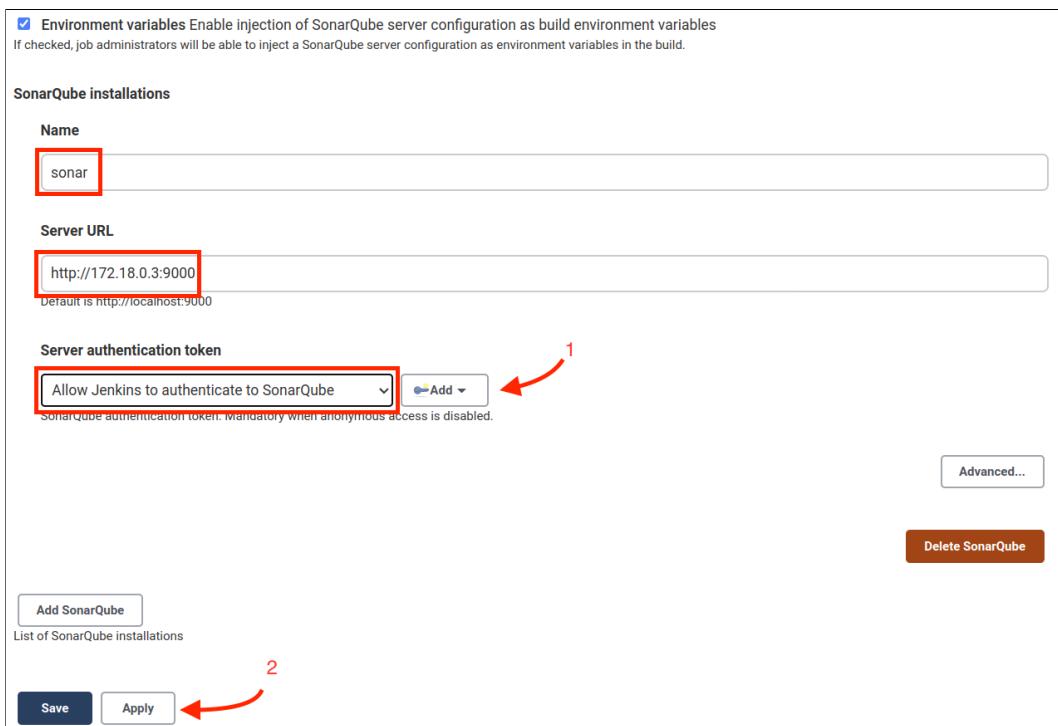


Figure 4.36: SonarQube Plugin in Jenkins

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

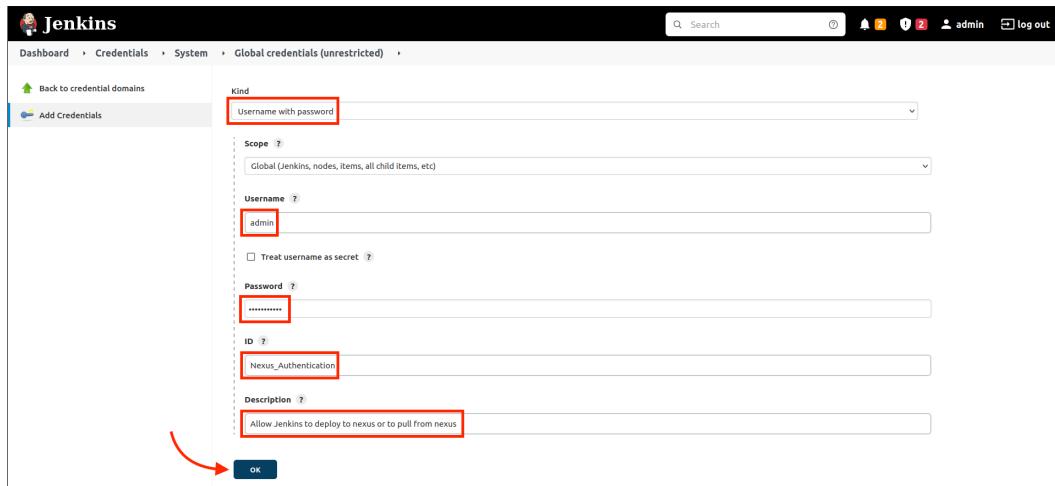


Figure 4.37: Configure Nexus Credentials in Jenkins

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

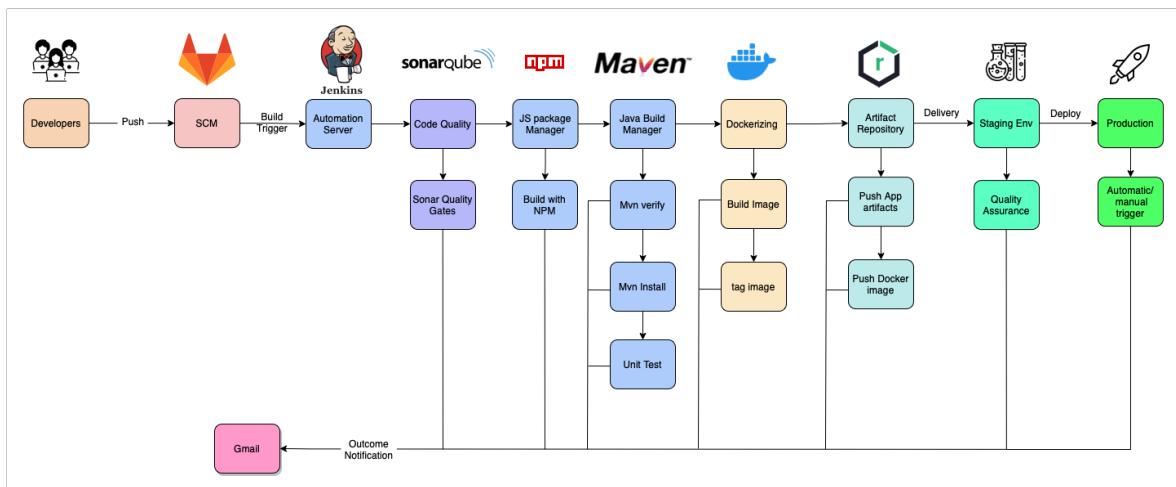


Figure 4.38: DevOps Pipeline

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.

Conditions			
Conditions on Overall Code			
Metric	Operator	Value	Edit
Bugs	is greater than	3	
Code Smells	is greater than	45	
Duplicated Blocks	is greater than	10	
Duplicated Lines	is greater than	200	
Major Issues	is greater than	20	
Minor Issues	is greater than	30	
Unit Tests	is less than	10	
Vulnerabilities	is greater than	3	

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

Chapter 4. Workflow Implementation

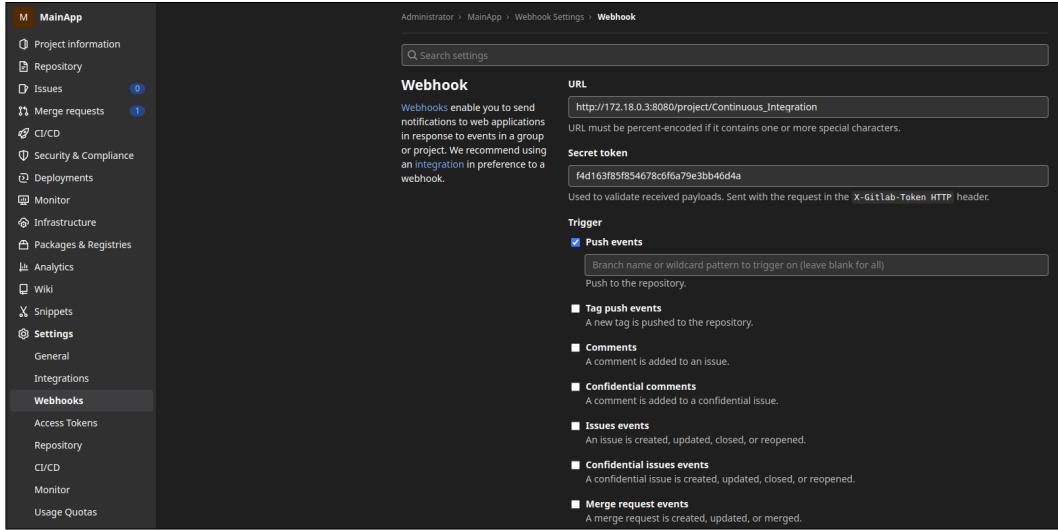


Figure 4.40: Integration Webhook Configuration

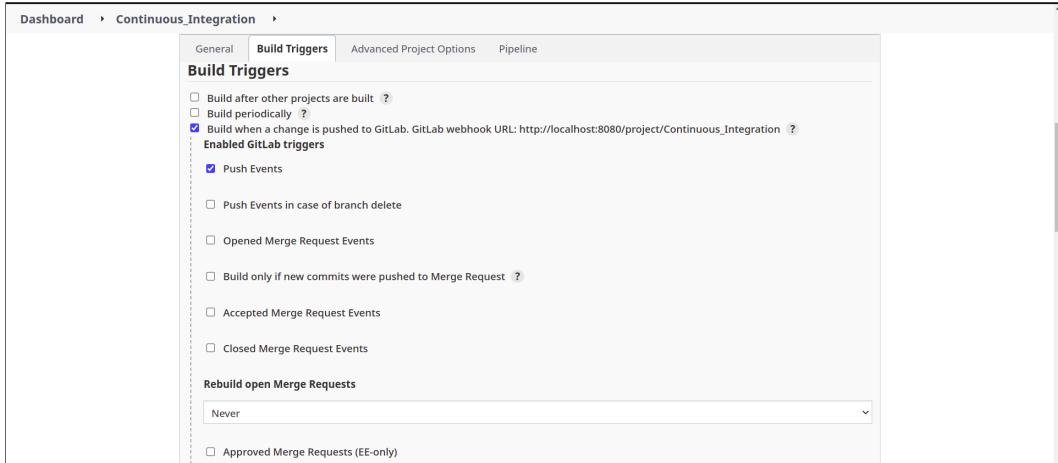


Figure 4.41: Continuous Integration Trigger Configuration



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.

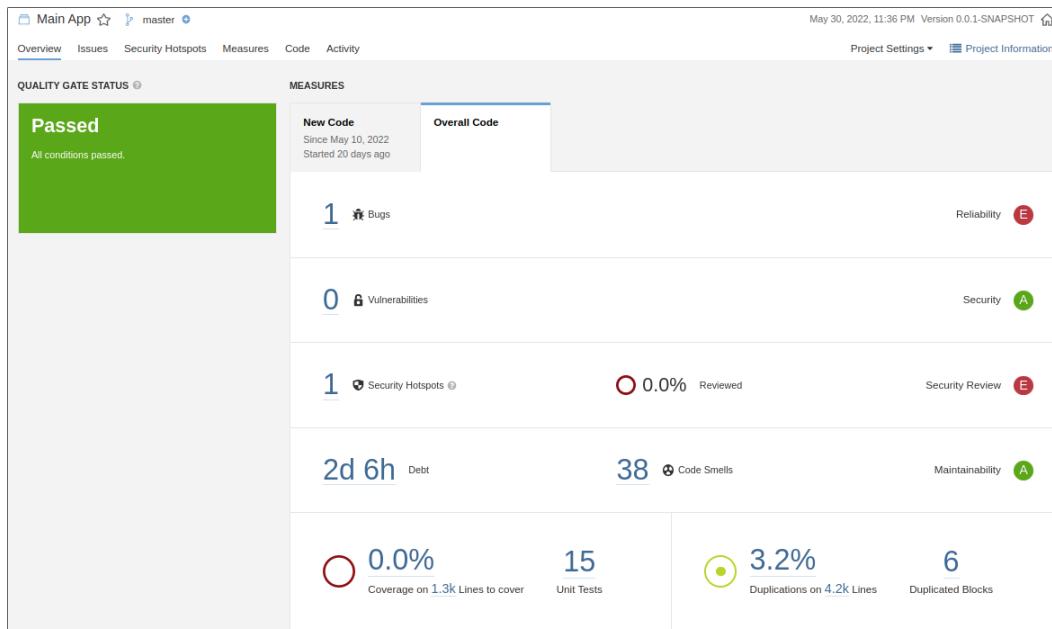


Figure 4.43: Successfully passing SonarQube Quality Gate

- 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

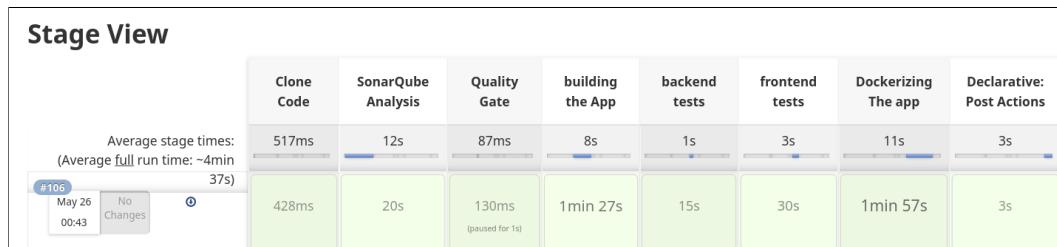


Figure 4.44: Continuous Integration Completed



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

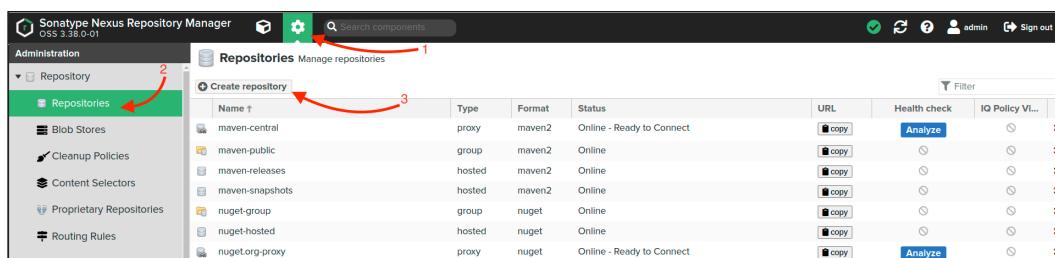


Figure 4.46: Creating the repository

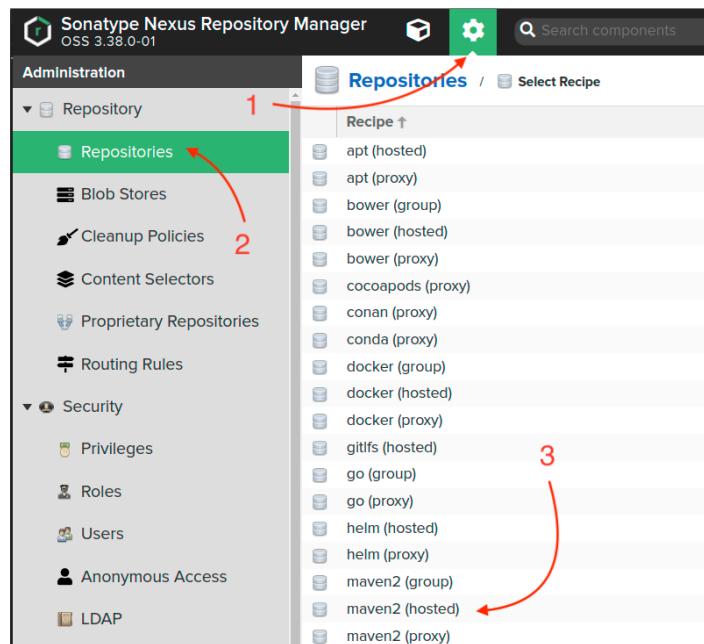


Figure 4.47: Choosing the Repository type

Repositories / **Select Recipe** / **Create Repository: maven2 (hosted)**

Name: artifact-repository

Online: If checked, the repository accepts incoming requests

Maven 2

Version policy: What type of artifacts does this repository store? **Snapshot**

Layout policy: Validate that all paths are maven artifact or metadata paths **Strict**

Content Disposition: Add Content-Disposition header as 'Attachment' to disable some content from being inline in a browser. **Inline**

Storage

Blob store: Blob store used to store repository contents **default**

Strict Content Type Validation: Validate that all content uploaded to this repository is of a MIME type appropriate for the repository format

Hosted

Deployment policy: Controls if deployments of and updates to artifacts are allowed **Allow redeploy**

Figure 4.48: Confirming the Creation of the repository

Hosted

Deployment policy: Controls if deployments of and updates to artifacts are allowed **Allow redeploy**

Proprietary Components: Components in this repository count as proprietary for namespace conflict attacks (requires Sonatype Nexus Firewall)

Cleanup

Cleanup Policies: Components that match any of the Applied policies will be deleted

Available	Applied
Filter	> <

Create repository **Cancel**

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

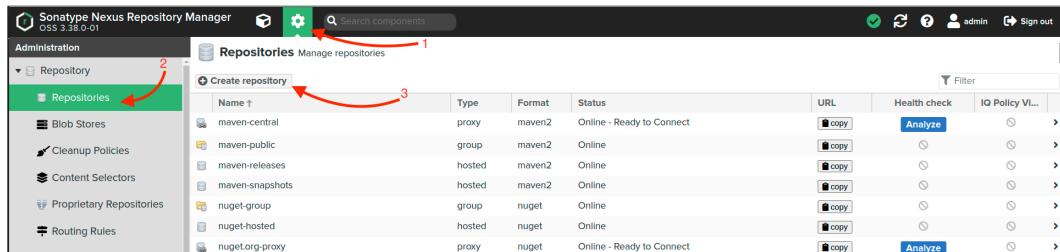


Figure 4.50: Creating the repository

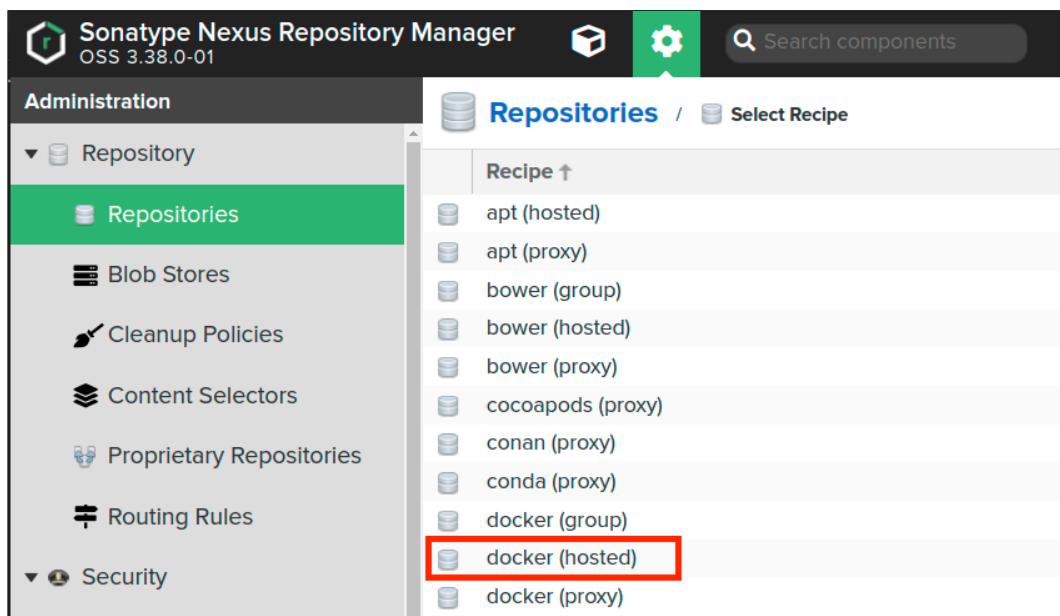


Figure 4.51: Choosing the repository type

Name: A unique identifier for this repository

Online: If checked, the repository accepts incoming requests

Repository Connectors

Connectors allow Docker clients to connect directly to hosted registries, but are not always required. Consult our documentation for which connector is appropriate for your use case. For information on scaling the repositories see our scaling documentation.

HTTP: Create an HTTP connector at specified port. Normally used if the server is behind a secure proxy.
 8085

HTTPS: Create an HTTPS connector at specified port. Normally used if the server is configured for https.

Allow anonymous docker pull:
 Allow anonymous docker pull (Docker Bearer Token Realm required)

Docker Registry API Support

Enable Docker V1 API:
 Allow clients to use the V1 API to interact with this repository

Figure 4.52: Configuring the Docker images repository

Hosted

Deployment policy:
 Controls if deployments of and updates to artifacts are allowed
 Allow redeploy

Proprietary Components:
 Components in this repository count as proprietary for namespace conflict attacks (requires Sonatype Nexus Firewall)

Cleanup

Cleanup Policies:
 Components that match any of the Applied policies will be deleted

Available	Applied
<input type="button" value="Filter"/> 	
<input type="button" value=">"/> <input type="button" value="<"/>	

Create repository **Cancel**

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

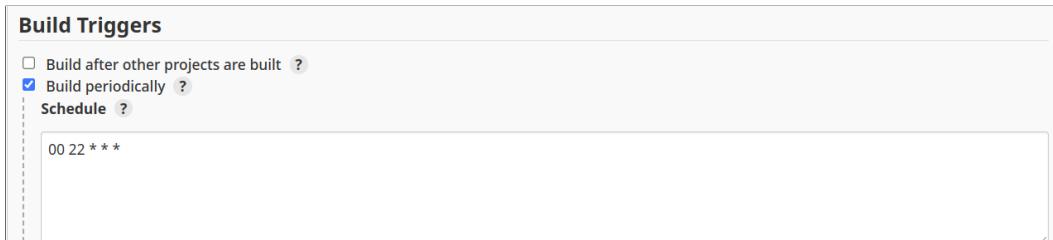


Figure 4.54: Daily pipeline trigger

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.

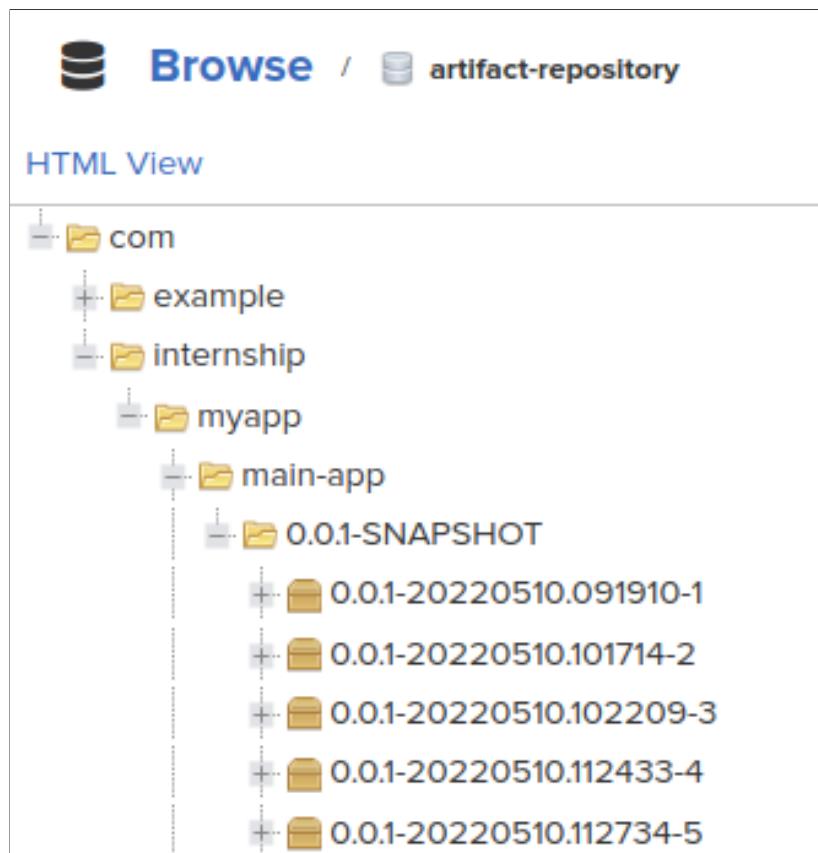


Figure 4.55: Artifacts pushed to nexus snapshot repository

- 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.

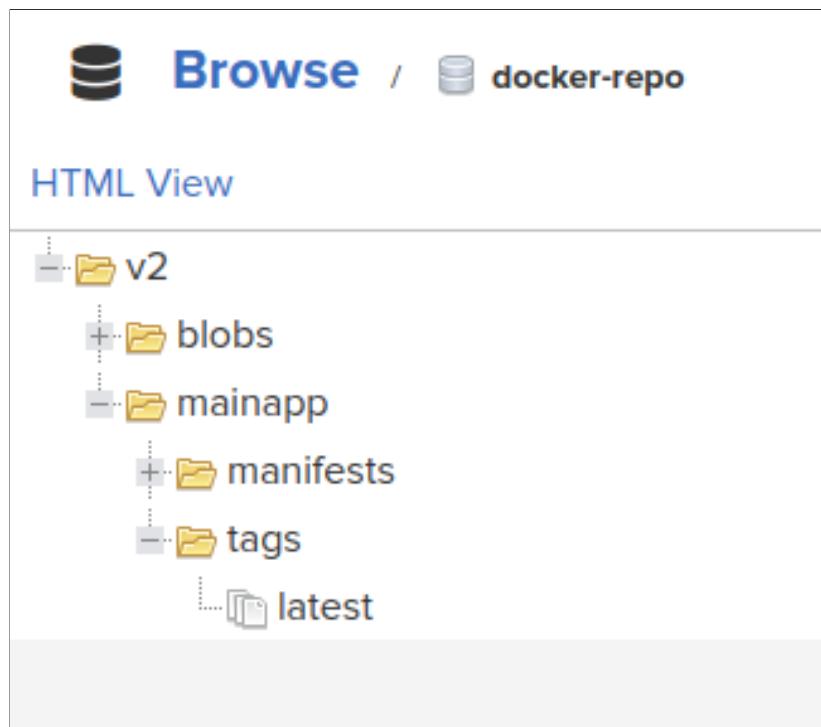


Figure 4.56: Docker image pushed to nexus repository

- 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

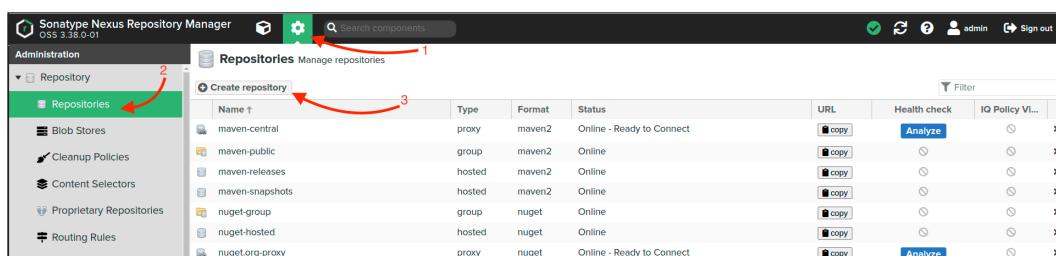


Figure 4.59: Creating the repository

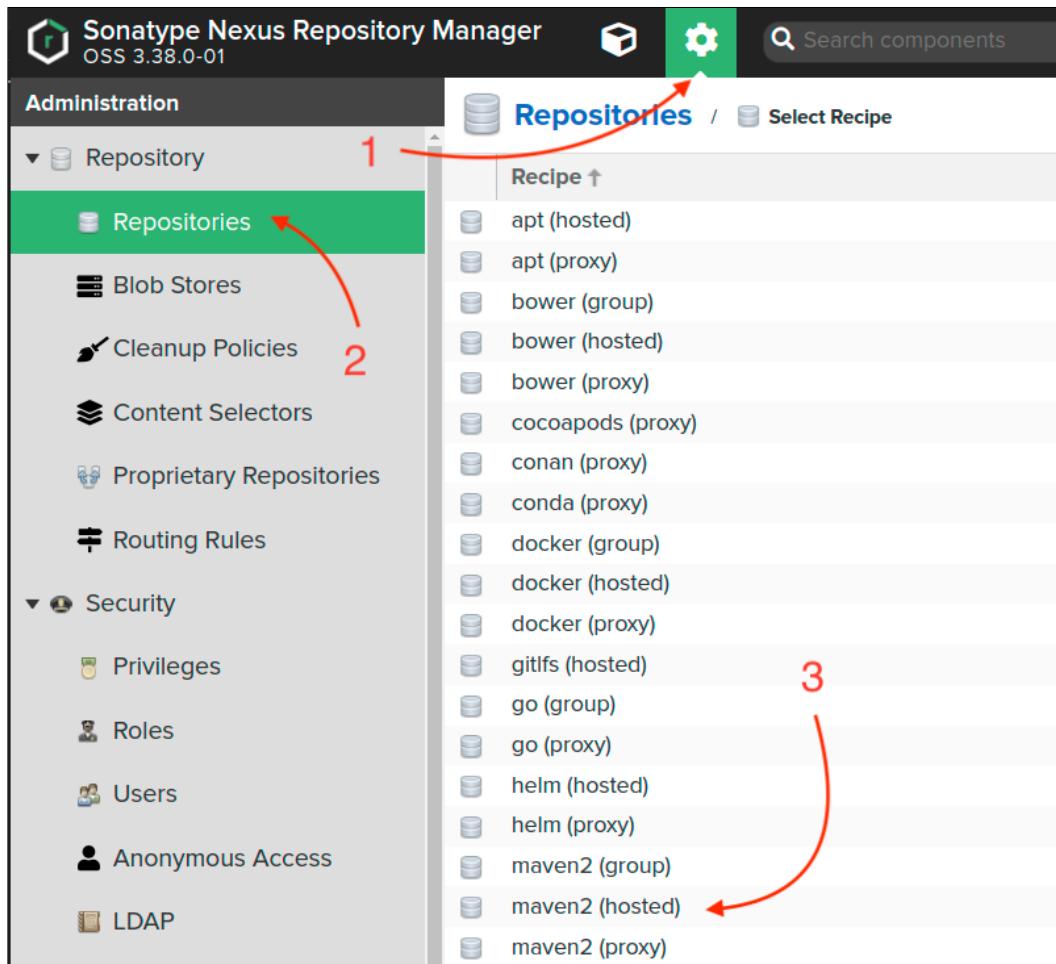


Figure 4.60: Choosing the Repository type

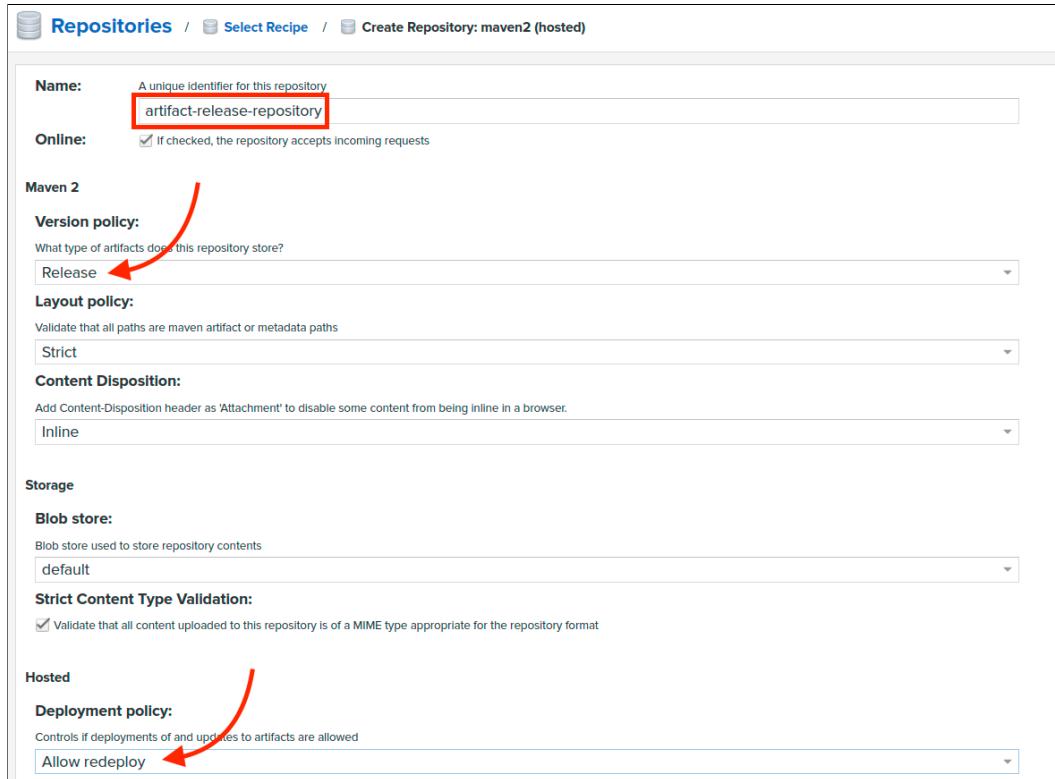


Figure 4.61: Configuring the Artifact Repository

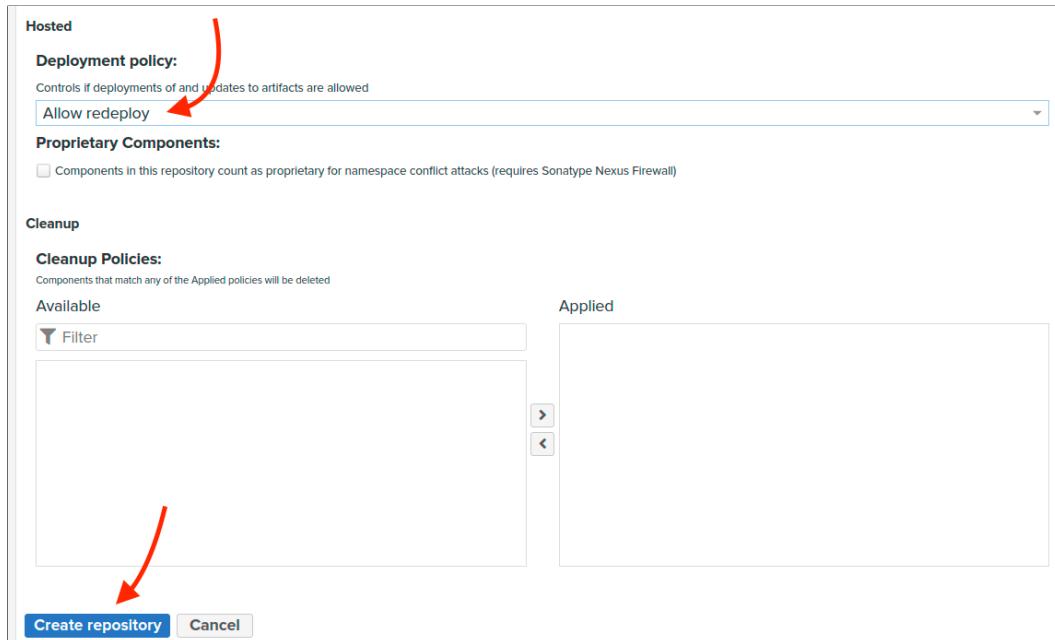


Figure 4.62: Confirming the Creation of the repository

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:

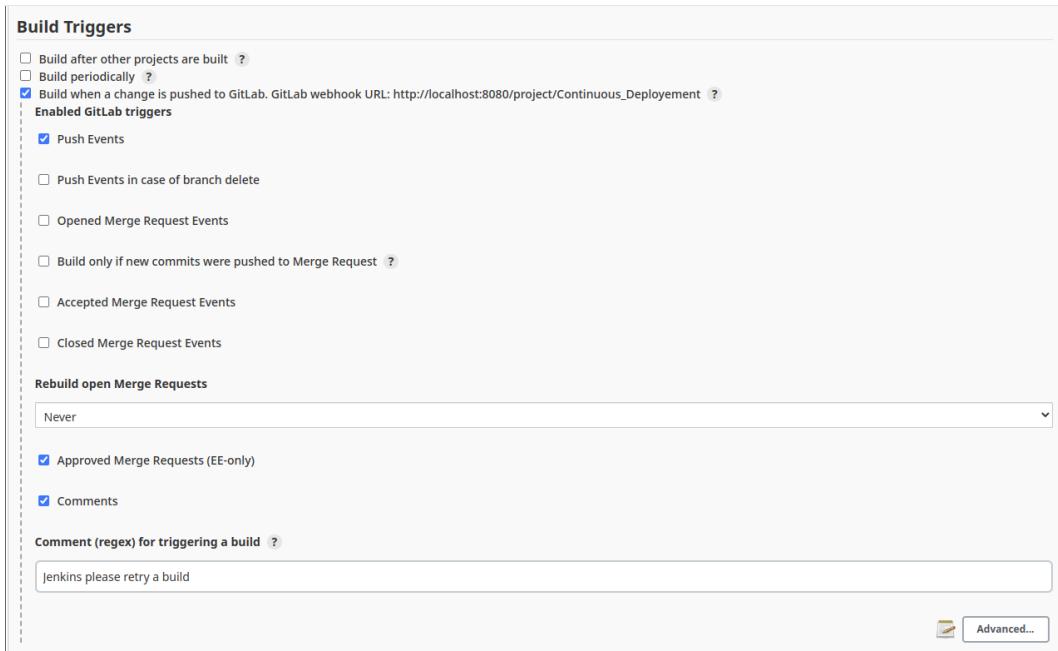


Figure 4.63: Continuous Deployment Trigger Configuration in Jenkins

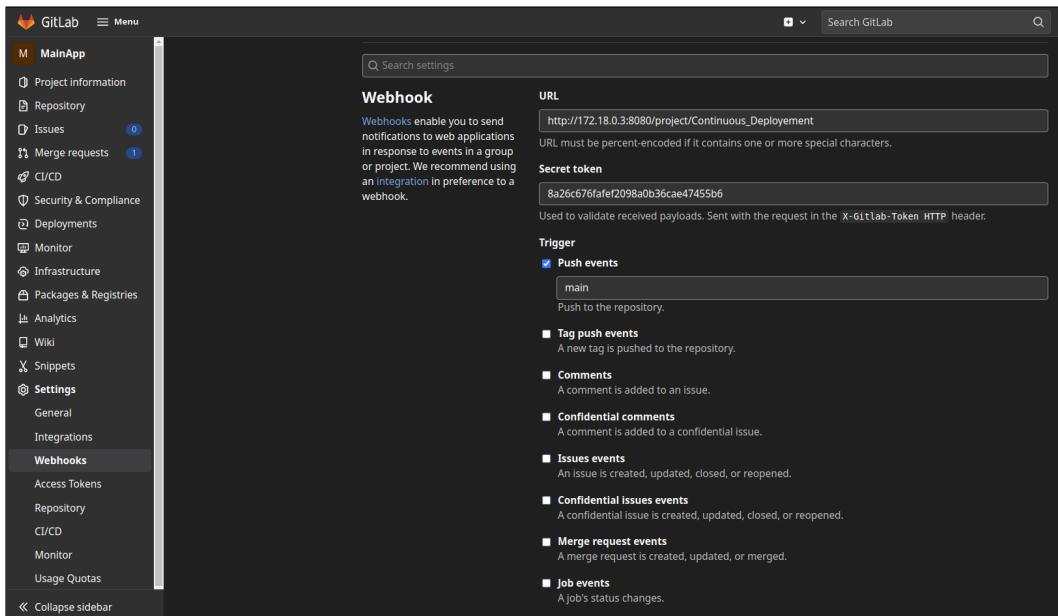


Figure 4.64: Continuous Deployment Webhook Configuration in Gitlab

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.

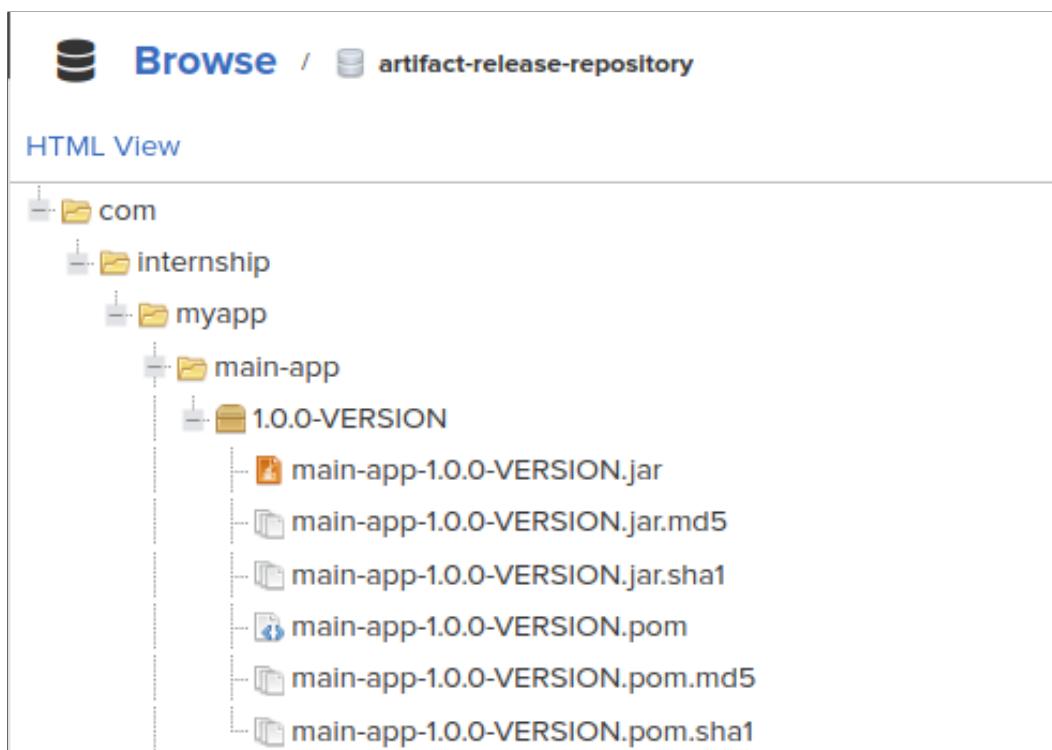


Figure 4.65: Deploying the artifacts to the nexus Release Repository

- 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

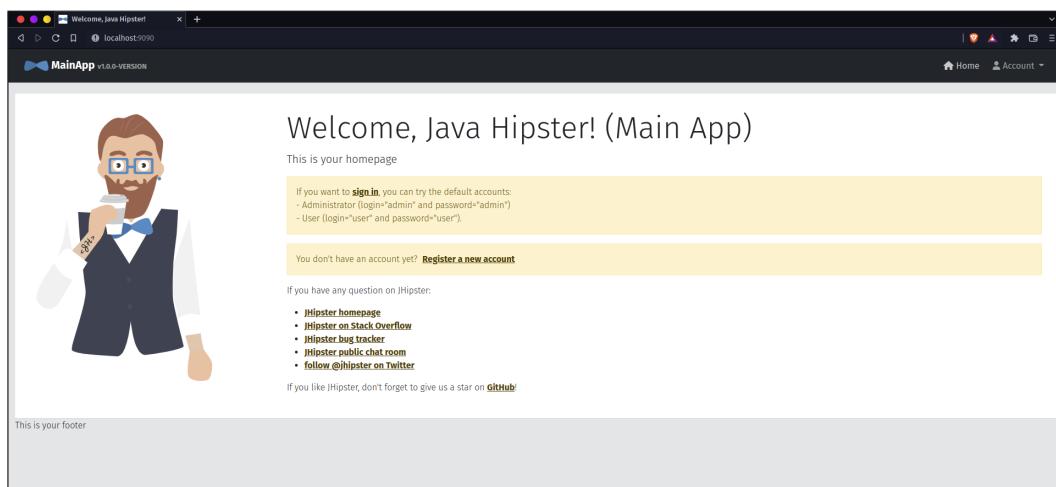


Figure 4.66: Application Running after the continuous deployment pipeline is finished

- 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

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

Listing 4.5.1: Run installdocker.bash Script to install Docker

```
1 version: '2'
2 services:
3   jenkins:
4     image: ghassenchams/jenkinsdockernode:finalversion
5     container_name: jenkins
6     privileged: true
7     user: root
8     volumes:
9       - ~/Documents/Internship/jenkins_data:/var/jenkins_home
10      - /var/run/docker.sock:/var/run/docker.sock
11   ports:
12     - 8080:8080
13     - 50003:50000
14   networks:
15     internship_default:
16       ipv4_address: 172.18.0.3
17
18   nexus:
19     image: sonatype/nexus3:3.38.0
20     container_name: nexus
21     privileged: true
22     user: root
23     volumes:
24       - ~/Documents/Internship/nexus-data:/nexus-data
25     ports:
26       - 8081:8081
27       - 8085:8085
28     networks:
29       internship_default:
30         ipv4_address: 172.18.0.4
31   sonarqube:
32     image: sonarqube:9.3.0-community
33     container_name: sonarqube
34     privileged: true
35     user: root
36     volumes:
```

```

37      - ~/Documents/Internship/sonarqube_data:/opt/sonarqube/data
38
39      - ~/Documents/Internship/sonarqube_extensions:/opt/sonarqube/extensions
40      - ~/Documents/Internship/sonarqube_logs:/opt/sonarqube/logs
41
42  ports:
43    - 9000:9000
44
45  networks:
46    internship_default:
47      ipv4_address: 172.18.0.5
48
49
50
51  gitlab:
52    image: 'gitlab/gitlab-ce:14.7.6-ce.0'
53    container_name: gitlab
54    privileged: true
55    user: root
56    volumes:
57      - ~/Documents/Internship/gitlab/config:/etc/gitlab
58      - ~/Documents/Internship/gitlab/logs:/var/log/gitlab
59      - ~/Documents/Internship/gitlab/data:/var/opt/gitlab
60
61  ports:
62    - 80:80
63    - 443:443
64
65  networks:
66    internship_default:
67      ipv4_address: 172.18.0.2
68
69
70
71  volumes:
72    nexus-data: {}
73    sonarqube_data:
74    sonarqube_extensions:
75    sonarqube_logs:
76
77  networks:
78    internship_default:
79      driver: bridge
80      ipam:
81        config:
82          - subnet: "172.18.0.0/16"

```

Listing 4.5.2: Docker-Compose File

```

1  #!/bin/bash
2  echo 'Jenkins admin password '
3  docker container exec jenkins sh -c "cat
4    /var/jenkins_home/secrets/initialAdminPassword"
5  echo ' Nexus admin password '

```

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

Listing 4.5.3: initialpassword.bash Script

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

Listing 4.5.4: installplugins.bash

```
1 #!/bin/bash
2 jenkins-plugin-cli --plugins \
3 credentials:1087.v16065d268466 \
4 command-launcher:1.6 \
5 docker-java-api:3.1.5.2 \
6 docker-commons:1.19 \
7 sonar-quality-gates:1.3.1 \
8 sonar:2.14 \
9 docker-compose-build-step:1.0 \
10 docker-workflow:1.28 \
11 docker-plugin:1.2.7 \
12 git-server:1.10 \
13 git-client:3.11.0 \
14 git:4.11.0 \
15 gitlab-oauth:1.15 \
16 gitlab-logo:1.0.5 \
17 gitlab-plugin:1.5.29 \
18 gitlab-api:1.0.6 \
19 gitlab-merge-request-jenkins:2.0.0 \
20 junit:1.57 \
21 mailer:408.vd726a_1130320 \
22 maven-dependency-update-trigger:1.5 \
23 maven-deployment-linker:1.5.1 \
24 pipeline-maven:3.10.0 \
25 nexus-artifact-uploader:2.13 \
26 nexus-jenkins-plugin:3.14.405.v74e19a_0b_1a_1a_ \
27 nodejs:1.5.1 \
28 plain-credentials:1.8 \
29 docker-build-step:2.8 \
```

```

30 ssh-slaves:1.806.v2253cedd3295 \
31 sshd:3.1.0
32 exec bash

```

Listing 4.5.5: plugins.bash Script

```

1 pipeline {
2
3     agent any
4     stages {
5         // ----- We Cloned the repo from the parameter above -----
6         stage("Cloning the Code") {
7             steps {
8                 script {
9                     sh "echo cloning"
10                }
11            }
12        }
13    }
14    // ----- SonarQube Analysis -----
15    stage('SonarQube Analysis') {
16        steps{
17            script {
18
19                withSonarQubeEnv(installationName: 'sonar'){
20                    withMaven{
21                        sh "./mvnw sonar:sonar "
22                    }
23                }
24            }
25        }
26    }
27    // ----- Sonar Quality Gate -----
28    stage("Quality Gate") {
29        steps {
30            waitForQualityGate abortPipeline: true
31        }
32    }
33    // ----- Building The Project with mvn and npm -----
34    stage("Building the App") {
35        steps {
36            script {
37                sh "npm install"
38                withMaven {
39                    sh "./mvnw install"

```

```
40         }
41     }
42 }
43 }
44 // ----- Back-End testing -----
45 stage('Back-end tests') {
46     steps {
47         script{
48             try {
49                 sh "./mvnw test"
50             } catch(err) {
51                 throw err
52             } finally {
53                 step([\$/class: 'JUnitResultArchiver',
54                     ↳ allowEmptyResults: true, testResults:
55                     ↳ '**/target/surefire-reports/TEST-*.xml'])
56             }
57         }
58     }
59 }
60 // ----- Front-End testing -----
61 stage('Front-end tests') {
62     steps{
63         script{
64             try {
65                 sh 'npm install'
66                 sh "npm test"
67             } catch(err){
68                 throw err
69             } finally {
70                 step([\$/class: 'JUnitResultArchiver', allowEmptyResults:
71                     ↳ true, testResults:
72                     ↳ '**/target/test-results/karma/TESTS-*.xml'])
73             }
74         }
75     }
76 }
77 }
78 }
79 }
80 post {
81     success {
```

```

82         mail bcc: '', body: 'The CI pipeline was successful', cc:
83             ↵ 'chamssidine.abderrahim@etudiant-isi.utm.tn', from: '',
84             ↵ replyTo: '', subject: 'Successful Build', to:
85             ↵ 'oueslatighassen5201@gmail.com'
86     }
87 }
88 }
```

P

Listing 4.5.6: Continous Integration Jenkins File

```

1 pipeline {
2
3     agent any
4     environment {
5         // This can be nexus3 or nexus2
6         NEXUS_VERSION = "nexus3"
7         // This can be http or https
8         NEXUS_PROTOCOL = "http"
9         // Where your Nexus is running
10        NEXUS_URL = "172.18.0.4:8081"
11        // Repository where we will upload the artifact
12        NEXUS_REPOSITORY = "artifact-repository"
13        // Jenkins credential id to authenticate to Nexus OSS
14        NEXUS_CREDENTIAL_ID = "nexus-user-credentials"
15    }
16    stages {
17        // ----- Cloning the branch dev -----
18        stage("Clone Code") {
19            steps {
20                script {
21                    // Let's clone the source
22                    git branch: 'dev', credentialsId:
23                        ↵ 'be415b2c-6347-48a4-a24b-6cc743c73672', url:
24                        ↵ 'http://172.18.02/root/mainapp.git'
25                }
26            }
27        }
28    }
29    // ----- Code Analysis with SonarQube -----
30 }
```

```

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
70                     // step 'readMavenPom' is included in:
71                     // https://plugins.jenkins.io/pipeline-utility-steps
72                     def pom = readMavenPom file: 'pom.xml';
73                     writeMavenPom model: pom;

```

```

72         // Find built artifact under target folder
73         filesByGlob = findFiles(glob:
74             "target/*.${pom.packaging}");
75         // Print some info from the artifact found
76         echo "${filesByGlob[0].name} ${filesByGlob[0].path}"
77             ${filesByGlob[0].directory}
78             ${filesByGlob[0].length}
79             ${filesByGlob[0].lastModified}"
80         // Extract the path from the File found
81         artifactPath = filesByGlob[0].path;
82         // Assign to a boolean response verifying If the
83             artifact name exists
84         artifactExists = fileExists artifactPath;
85
86
87         if(artifactExists) {
88             echo "*** File: ${artifactPath}, group:
89                 ${pom.groupId}, packaging: ${pom.packaging},
90                 version ${pom.version}";
91
92             nexusArtifactUploader(
93                 nexusVersion: NEXUS_VERSION,
94                 protocol: NEXUS_PROTOCOL,
95                 nexusUrl: NEXUS_URL,
96                 groupId: pom.groupId,
97                 version: pom.version,
98                 repository: NEXUS_REPOSITORY,
99                 credentialsId: NEXUS_CREDENTIAL_ID,
100                artifacts: [
101                    // Artifact generated such as .jar, .ear
102                    // and .war files.
103                    [artifactId: pom.artifactId,
104                     classifier: '',
105                     file: artifactPath,
106                     type: pom.packaging],
107
108                    // Lets upload the pom.xml file for
109                    // additional information for Transitive
110                    // dependencies
111                    [artifactId: pom.artifactId,
112                     classifier: '',
113                     file: "pom.xml",
114                     type: "pom"]
115                ]
116            );
117
118        } else {
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
727
728
729
729
730
731
732
733
734
735
736
737
737
738
739
739
740
741
742
743
744
745
745
746
747
747
748
749
749
750
751
752
753
754
755
755
756
757
757
758
759
759
760
761
762
763
764
765
765
766
767
767
768
769
769
770
771
772
773
774
775
775
776
777
777
778
779
779
780
781
782
783
783
784
785
785
786
787
787
788
789
789
790
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1530
1531
1531
1532
1532
1533
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1540
1541
1541
1542
1542
1543
1543
1544
1544
1545
1545
1546
1546
1547
1547
1548
1548
1549
1549
1550
1550
1551
1551
1552
1552
1553
1553
1554
1554
1555
1555
1556
1556
1557
1557
1558
1558
1559
1559
1560
1560
1561
1561
1562
1562
1563
1563
1564
1564
1565
1565
1566
1566
1567
1567
1568
1568
1569
1569
1570
1570
1571
1571
1572
1572
1573
1573
1574
1574
1575
1575
1576
1576
1577
1577
1578
1578
1579
1579
1580
1580
1581
1581
1582
1582
1583
1583
1584
1584
1585
1585
1586
1586
1587
1587
1588
1588
1589
1589
1590
1590
1591
1591
1592
1592
1593
1593
1594
1594
1595
1595
1596
1596
1597
1597
1598
1598
1599
1599
1600
1600
1601
1601
1602
1602
1603
1603
1604
1604
1605
1605
1606
1606
1607
1607
1608
1608
1609
1609
1610
1610
1611
1611
1612
1612
1613
1613
1614
161
```

```

108                         error "*** File: \${artifactPath}, could not be
109                         ↵ found";
110                     }
111                 }
112             }
113             // ----- Dockerizing the app using Docker -----
114         stage("Dockerizing The App") {
115             steps {
116                 sh "./mvnw -Pprod verify jib:dockerBuild"
117             }
118         }
119         stage("deploying docker image to nexus") {
120             steps {
121                 sh "docker login -u admin -p nexusserver 172.18.0.4:8085 "
122                 sh "docker tag mainapp 172.18.0.4:8085/mainapp:latest"
123                 sh "docker push 172.18.0.4:8085/mainapp:latest"
124             }
125         }
126     }
127     post {
128         success {
129             mail bcc: '', body: 'The Daily pipeline was successful', cc:
130                         ↵ 'chamssidine.abderrahim@etudiant-isi.utm.tn', from: '',
131                         ↵ replyTo: '', subject: 'Successful Build', to:
132                         ↵ 'oueslatighassen5201@gmail.com'
133         }
134         failure {
135             mail bcc: '', body: 'The Daily pipeline was a failure ', cc:
136                         ↵ 'chamssidine.abderrahim@etudiant-isi.utm.tn', from: '',
137                         ↵ replyTo: '', subject: 'failed Build', to:
138                         ↵ 'oueslatighassen5201@gmail.com'
139         }
140     }
141 }
142 }
```

Listing 4.5.7: Daily Jenkins File

```

1 pipeline {
2     agent any
3     environment {
4         // This can be nexus3 or nexus2
5         NEXUS_VERSION = "nexus3"
6         // This can be http or https
7         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:
23             'be415b2c-6347-48a4-a24b-6cc743c73672', url:
24             'http://172.18.02/root/mainapp.git'
25         }
26       }
27
28     // ----- Building The Application -----
29     stage("Building the App") {
30       steps {
31         script {
32           sh "npm install"
33           withMaven {
34             sh "./mvnw install"
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
44             // step 'readMavenPom' is included in:
45             // https://plugins.jenkins.io/pipeline-utility-steps
46           def pom = readMavenPom file: 'pom.xml';
47           writeMavenPom model: pom;
48           // Find built artifact under target folder
49           filesByGlob = findFiles(glob:
50             "target/*.${pom.packaging}");
51           // Print some info from the artifact found

```

```

49      echo "${filesByGlob[0].name} ${filesByGlob[0].path}
50      ↪  ${filesByGlob[0].directory}
51      ↪  ${filesByGlob[0].length}
52      ↪  ${filesByGlob[0].lastModified}"
53      // Extract the path from the File found
54      artifactPath = filesByGlob[0].path;
55      // Assign to a boolean response verifying If the
56      ↪  artifact name exists
57      artifactExists = fileExists artifactPath;

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

62          nexusArtifactUploader(
63              nexusVersion: NEXUS_VERSION,
64              protocol: NEXUS_PROTOCOL,
65              nexusUrl: NEXUS_URL,
66              groupId: pom.groupId,
67              version: pom.version,
68              repository: NEXUS_REPOSITORY,
69              credentialsId: NEXUS_CREDENTIAL_ID,
70              artifacts: [
71                  // Artifact generated such as .jar, .ear
72                  ↪  and .war files.
73                  [artifactId: pom.artifactId,
74                  classifier: '',
75                  file: artifactPath,
76                  type: pom.packaging],

77                  // Lets upload the pom.xml file for
78                  ↪  additional information for Transitive
79                  ↪  dependencies
80                  [artifactId: pom.artifactId,
81                  classifier: '',
82                  file: "pom.xml",
83                  type: "pom"]
84              ]
85          );
86      } else {
87          error "*** File: \${artifactPath}, could not be
88          ↪  found";
89      }
90  }

```

```

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 → -----
103 stage('Start container') {
104     steps {
105         sh 'docker-compose -f src/main/docker/app.yml up -d'
106     }
107 }
108 post {
109     success {
110         mail bcc: '', body: 'The CD pipeline was successful ', cc:
111             ↳ 'chamssidine.abderrahim@etudiant-isi.utm.tn', from: '',
112             ↳ replyTo: '', subject: 'Successful Build', to:
113             ↳ 'oueslatighassen5201@gmail.com'
114     }
115     failure {
116         mail bcc: '', body: 'The CD pipeline was a failure ', cc:
117             ↳ 'chamssidine.abderrahim@etudiant-isi.utm.tn', from: '',
118             ↳ replyTo: '', subject: 'failed Build', to:
119             ↳ 'oueslatighassen5201@gmail.com'
120     }
121 }
122 }

```

Listing 4.5.8: Continuous Deployment Jenkins File