NODEJS APPLICATION DEPLOYMENT TO EC2 WITH DOCKER

PROBLEM STATEMENT:

XYZ Corp is developing a Node.js-based web application and needs a robust deployment solution to efficiently build and deploy the application to EC2 instances. The challenge is to create a streamlined process using Jenkins, Docker, and AWS EC2, ensuring scalability and ease of management.

USE CASE SCENARIO:

- ➔ **Business Requirement:** XYZ Corp aims to deploy its Node.js application on AWS EC2 instances, leveraging Docker containers for consistency and scalability. The solution should integrate seamlessly with Jenkins, allowing for automated builds, tests, and deployments.

- ➔ **Technical Challenge:** The objective is to create a Dockerfile and Docker Compose file for the Node.js application, enabling easy containerization and deployment. Jenkins will be configured to trigger the build process, and the resulting Docker image will be deployed to EC2 instances, ensuring a reliable and efficient deployment pipeline for the Node.js application.

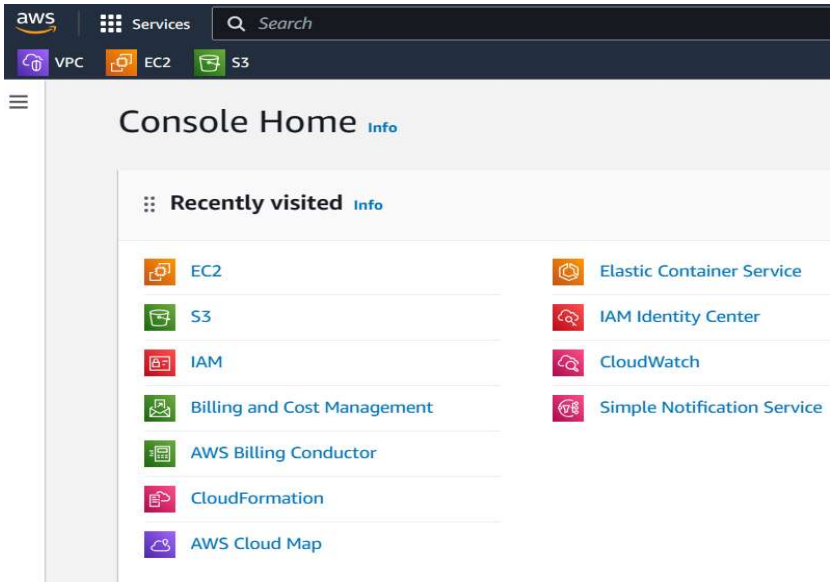
SOLUTION:

REQUIREMENTS:

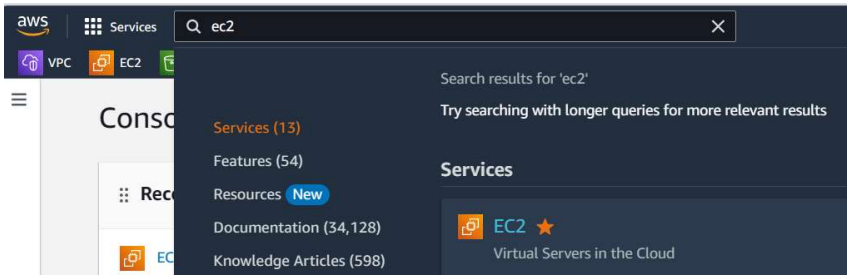
- ➔ AWS Cloud
- ➔ EC2 instance
- ➔ Docker
- ➔ Docker-compose
- ➔ GitHub
- ➔ Java
- ➔ Jenkins

Step:1 – Launching an EC2 Instance:

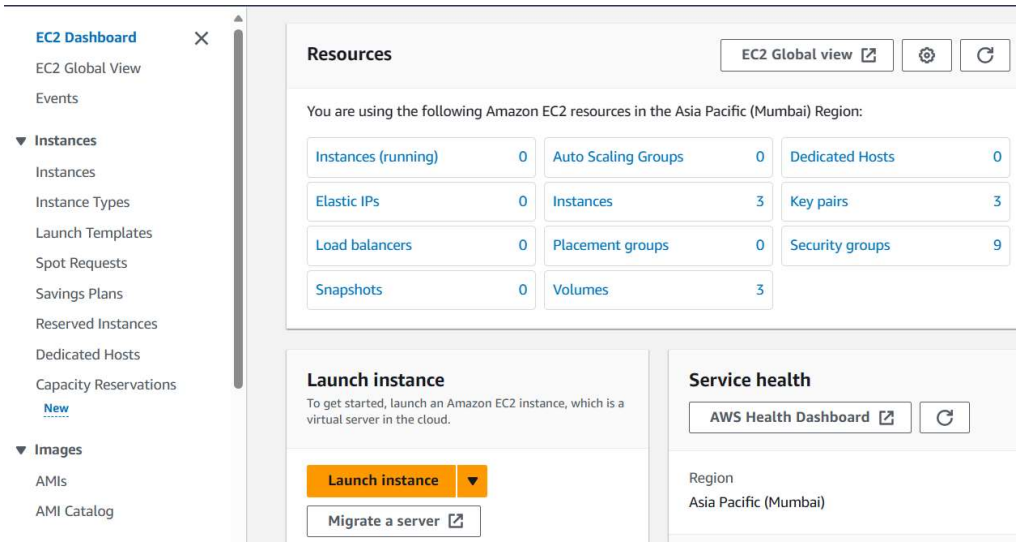
➔ First login into your AWS instance:



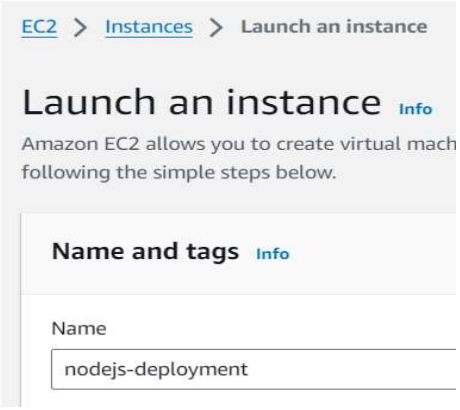
➔ Then on service search panel search EC2, click that one:



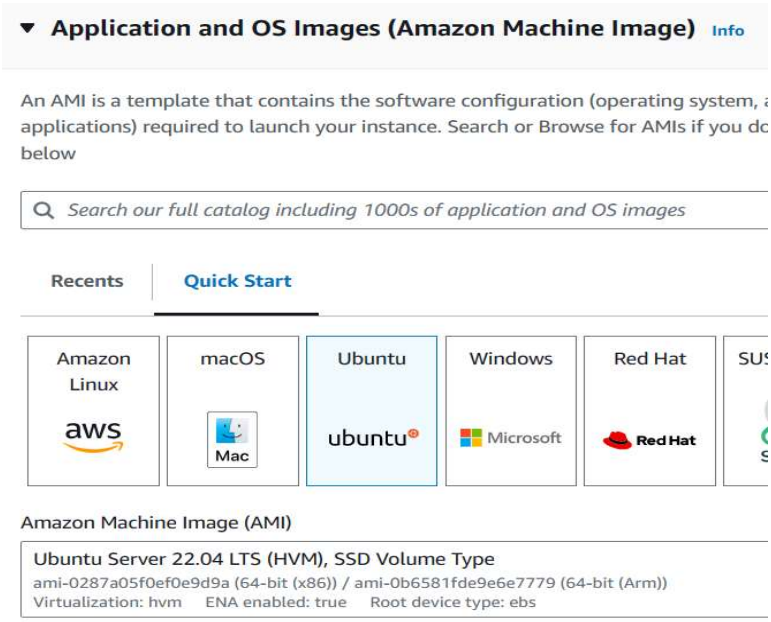
➔ Then click launch instances, for creating an EC2 instance:



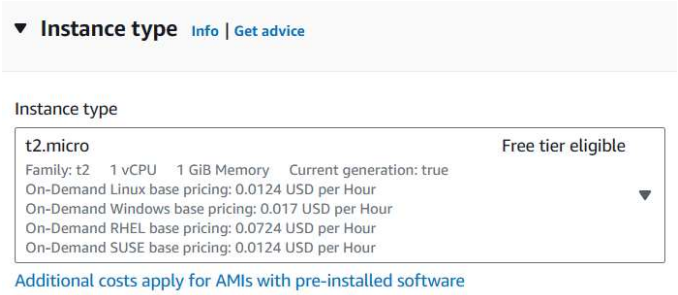
➔ Then name the instance according to your preferences:



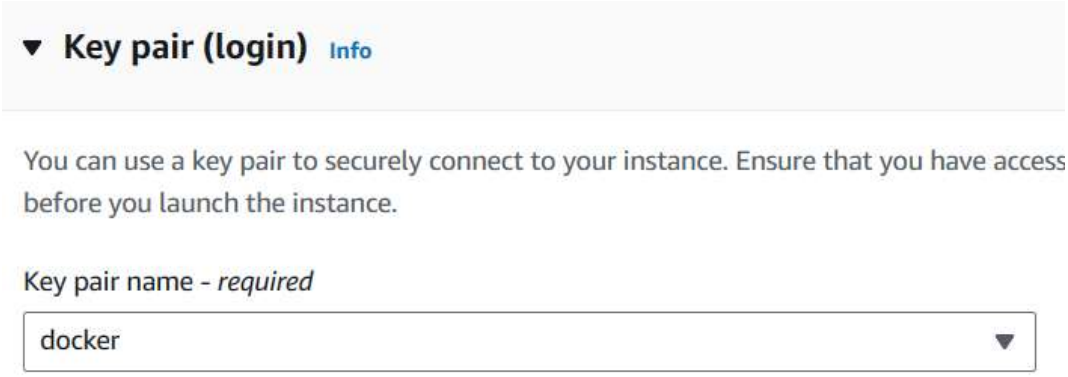
➔ Then select the operating system according to your preferences:



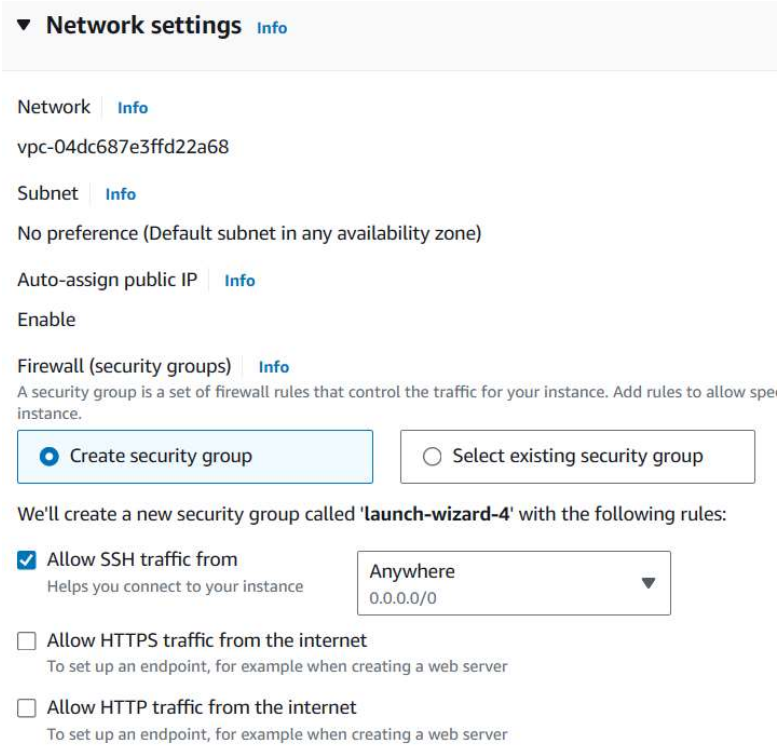
➔ Then select the instance type: according to your preferences, but here I am selecting **t2.micro**



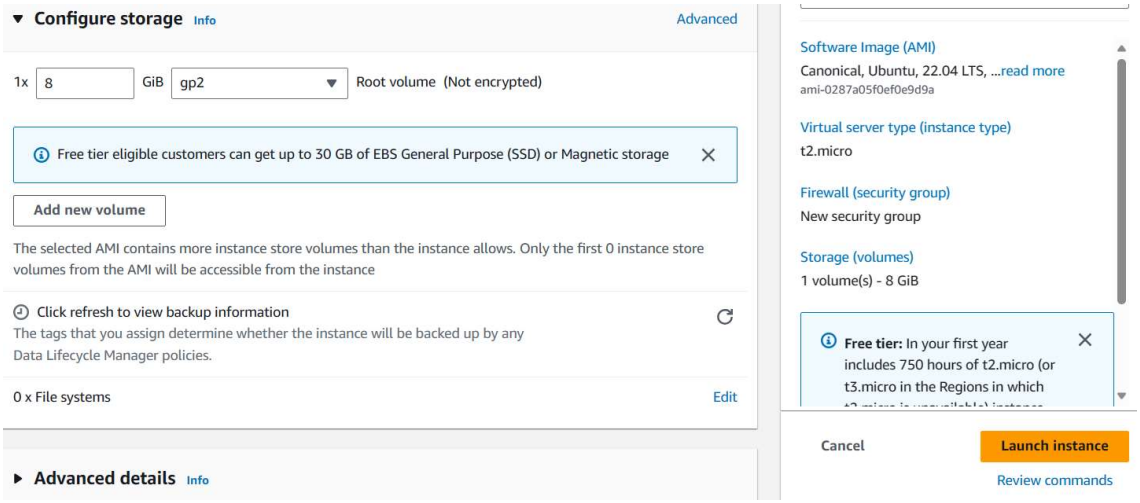
➔ Then select the key pair, according to your preferences, but here I am **proceeding with key pair option**, you can go with proceed with **without key pair option:**



➔ Then keeping the default options under network settings:

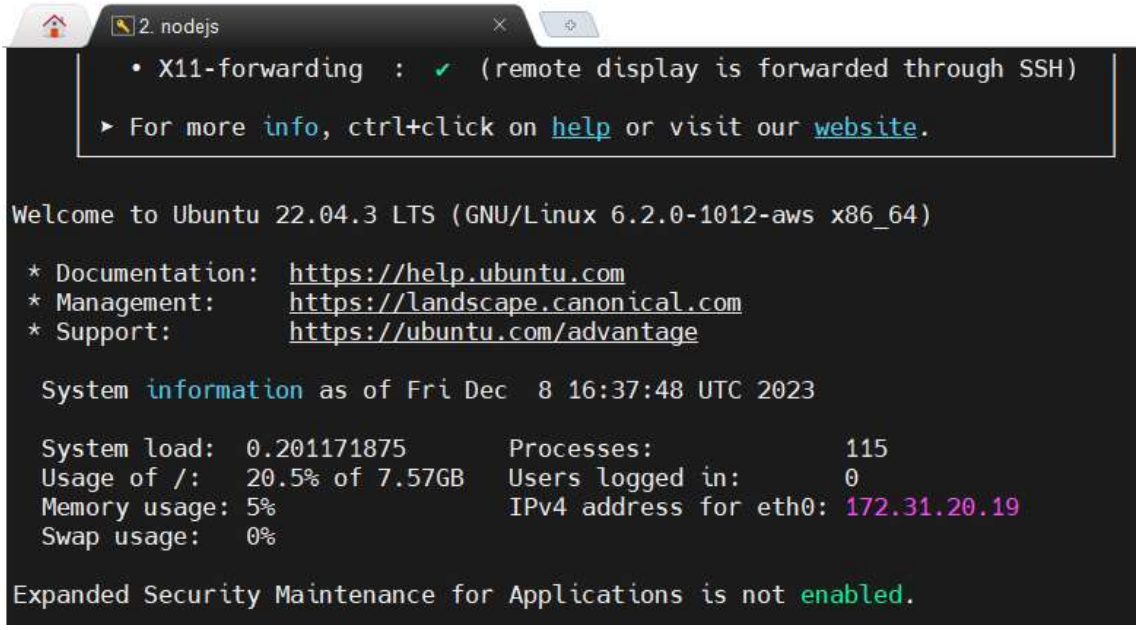


➔ Then keeping default options for the rest of the settings, click launch instance:



➔ The instance has been launched successfully:

➔ Connecting the instance:



Step:2 – Installing necessary packages required for this task:

➔ Creating a shell script, in that writing a necessary script to install these packages:

- ❖ **Java**
- ❖ **Docker**
- ❖ **Docker-compose**
- ❖ **Jenkins**

Script contains:

```
#!/bin/bash
```

```
#installing java:
apt-get update
apt-get install -y openjdk-11-jre

#installing docker:
apt-get update
apt-get install -y docker.io

#installing docker-compose:
apt-get update
apt-get install -y docker-compose

#installing jenkins:
sudo wget -O /usr/share/keyrings/jenkins-keyring.asc \
  https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key
echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc]
\
  https://pkg.jenkins.io/debian-stable binary/ | sudo tee \
  /etc/apt/sources.list.d/jenkins.list > /dev/null
sudo apt-get update
sudo apt-get install -y jenkins

#checking the installed services:
echo "-------------------------------------------"
echo "-------------------------------------------"
echo "-------------------------------------------"
echo "This is the Java package - "
java --version
echo "-------------------------------------------"
echo "This is Jenkins package - "
jenkins --version
echo "-------------------------------------------"
echo "This is Docker package - "
docker --version
echo "-------------------------------------------"
echo "This is Docker package - "
docker-compose --version
```

➔ Changing the file permission and executing it:

```
root@ip-172-31-20-19:/home/ubuntu# vi package.sh
root@ip-172-31-20-19:/home/ubuntu# chmod +x package.sh
root@ip-172-31-20-19:/home/ubuntu# ./package.sh
Hit:1 http://us-west-2.ec2.archive.ubuntu.com/ubuntu jammy InRelease
Get:2 http://us-west-2.ec2.archive.ubuntu.com/ubuntu jammy-updates InRelease [119 kB]
Get:3 http://us-west-2.ec2.archive.ubuntu.com/ubuntu jammy-backports InRelease [109 kB]
Get:4 http://us-west-2.ec2.archive.ubuntu.com/ubuntu jammy/universe amd64 Packages [14.1 MB]
```

➔ The packages have been installed successfully:

```
-----------------------------------------
-----------------------------------------
-----------------------------------------
This is the Java package -
openjdk 11.0.21 2023-10-17
OpenJDK Runtime Environment (build 11.0.21+9-post-Ubuntu-0ubuntu122.04)
OpenJDK 64-Bit Server VM (build 11.0.21+9-post-Ubuntu-0ubuntu122.04, mixed mode, sharing)
-----------------------------------------
This is Jenkins package -
2.426.1
-----------------------------------------
This is Docker package -
Docker version 24.0.5, build 24.0.5-0ubuntu1~22.04.1
-----------------------------------------
This is Docker package -
docker-compose version 1.29.2, build unknown
root@ip-172-31-20-19:/home/ubuntu#
```

Step:3 – Containerizing nodejs application by using docker and deploying it through Docker-compose:

➔ Assuming that you having Nodejs application like this:

```
root@ip-172-31-20-19:/home/ubuntu/food-delivery-react-redux-app# ls -l
total 1196
-rw-r--r-- 1 root root    1295 Dec  8 16:48 README.md
-rw-r--r-- 1 root root 1207543 Dec  8 16:48 package-lock.json
-rw-r--r-- 1 root root    1123 Dec  8 16:48 package.json
drwxr-xr-x 2 root root    4096 Dec  8 16:48 public
drwxr-xr-x 8 root root    4096 Dec  8 16:48 src
root@ip-172-31-20-19:/home/ubuntu/food-delivery-react-redux-app#
```

➔ Creating a dockerfile according to the application:

Dockerfile contains:

```
#choosing the base image as the build stage:
FROM node:16-alpine as build
```

```
#choosing working directory for the application:
WORKDIR /app

#copying the package.json file to app directory and
installing packages:
COPY package.json .
RUN npm install

#copying the rest of application code to the working
directory:
COPY . .

#building the application:
RUN npm run build

#second stage base image:
FROM nginx:alpine

#setting the working directory for this base image:
WORKDIR /usr/share/nginx/html/

#copying the first stage code to this stage
COPY --from=build /app/build .

#exposing the application:
EXPOSE 80

#Executing the application after creating image:
CMD ["nginx", "-g", "daemon off;"]
```

```
root@ip-172-31-20-19:/home/ubuntu/food-delivery-react-redux-app# ls -l
total 1196
-rw-r--r-- 1 root root    1295 Dec  8 16:48 README.md
-rw-r--r-- 1 root root 1207543 Dec  8 16:48 package-lock.json
-rw-r--r-- 1 root root    1123 Dec  8 16:48 package.json
drwxr-xr-x 2 root root    4096 Dec  8 16:48 public
drwxr-xr-x 8 root root    4096 Dec  8 16:48 src
root@ip-172-31-20-19:/home/ubuntu/food-delivery-react-redux-app# vi dockerfile
root@ip-172-31-20-19:/home/ubuntu/food-delivery-react-redux-app#
```

➔ Creating a docker image from the dockerfile by using **docker build** command:

```
root@ip-172-31-20-19:/home/ubuntu/food-delivery-react-redux-app#
root@ip-172-31-20-19:/home/ubuntu/food-delivery-react-redux-app# docker build -t nodejs:web .
DEPRECATED: The legacy builder is deprecated and will be removed in a future release.
            Install the buildx component to build images with BuildKit:
            https://docs.docker.com/go/buildx/

Sending build context to Docker daemon  30.12MB
Step 1/11 : FROM node:16-alpine as build
 ---> 2573171e0124
Step 2/11 : WORKDIR /app
 ---> Using cache
 ---> e7ec186a32a2
Step 3/11 : COPY package.json .
 ---> Using cache
 ---> 3843e06bea2e
Step 4/11 : RUN npm install
 ---> Using cache
 ---> 4b099804d6bc
Step 5/11 : COPY . .
 ---> Using cache
 ---> 7043a550bdd8
```

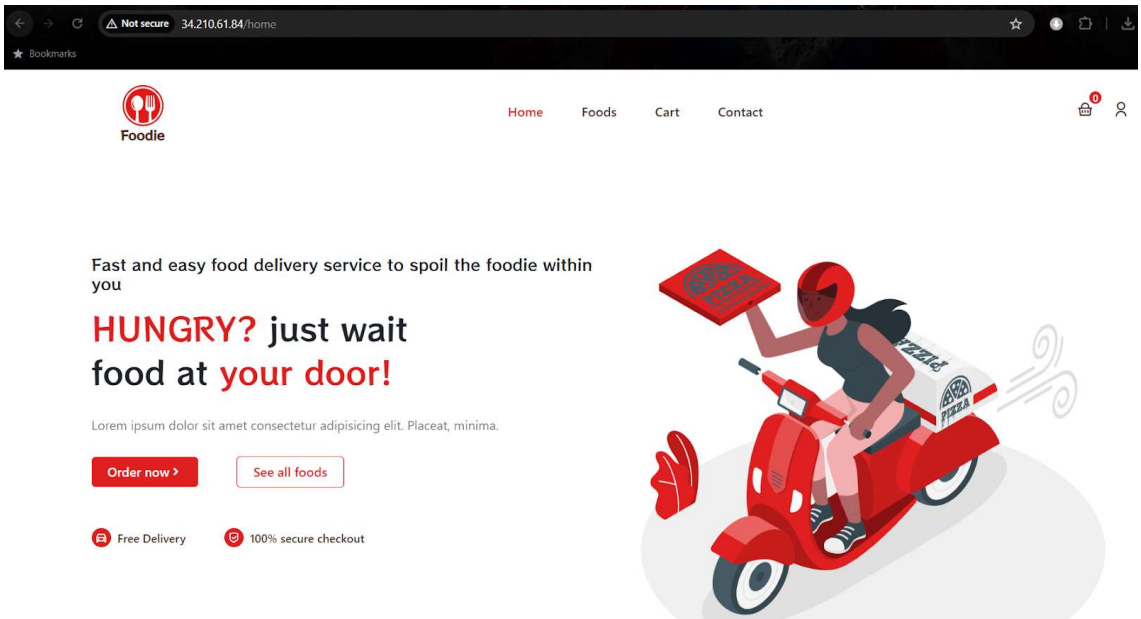
➔ Checking whether image is created or not by using **docker images** command:

```
Step 11/11 : CMD ["nginx", "-g", "daemon off;"]
 ---> Running in bd93527d571f
Removing intermediate container bd93527d571f
 ---> 5d27baeac2a8
Successfully built 5d27baeac2a8
Successfully tagged nodejs:web
root@ip-172-31-20-19:/home/ubuntu/food-delivery-react-redux-app# docker images
REPOSITORY   TAG         IMAGE ID       CREATED              SIZE
nodejs       web         5d27baeac2a8   About a minute ago   60MB
<none>       <none>      52ada153227e   2 minutes ago        624MB
nginx        alpine      01e5c69afaf6   7 days ago           42.6MB
node         16-alpine   2573171e0124   3 months ago         118MB
root@ip-172-31-20-19:/home/ubuntu/food-delivery-react-redux-app#
```

➔ checking the image by running container from the created image by using **docker run** command:

```
root@ip-172-31-20-19:/home/ubuntu/food-delivery-react-redux-app# docker run -d -it -p 80:80 nodejs:web
f4489dbb089ac10ade091523636bbccf0bb31f159a632beacdcb5f53016484cf
root@ip-172-31-20-19:/home/ubuntu/food-delivery-react-redux-app# docker ps
CONTAINER ID   IMAGE        COMMAND                  CREATED         STATUS         PORTS
f4489dbb089a   nodejs:web   "/docker-entrypoint.…"   4 seconds ago   Up 3 seconds   0.0.0.0:80->80/tcp, :::80->80/tcp
root@ip-172-31-20-19:/home/ubuntu/food-delivery-react-redux-app#
```

➔ checking the browser output:



The container is working fine:

➔ creating a docker-compose file for deployment purpose:

docker-compose.yml file contains:

```
version: '3'
services:
  nodejs-app:
    image: nodejs:web
    container_name: nodejs
    ports:
      - 80:80
    volumes:
      - nodejs-vol:/usr/share/nginx/html/

volumes:
  nodejs-vol:
    external: true
```

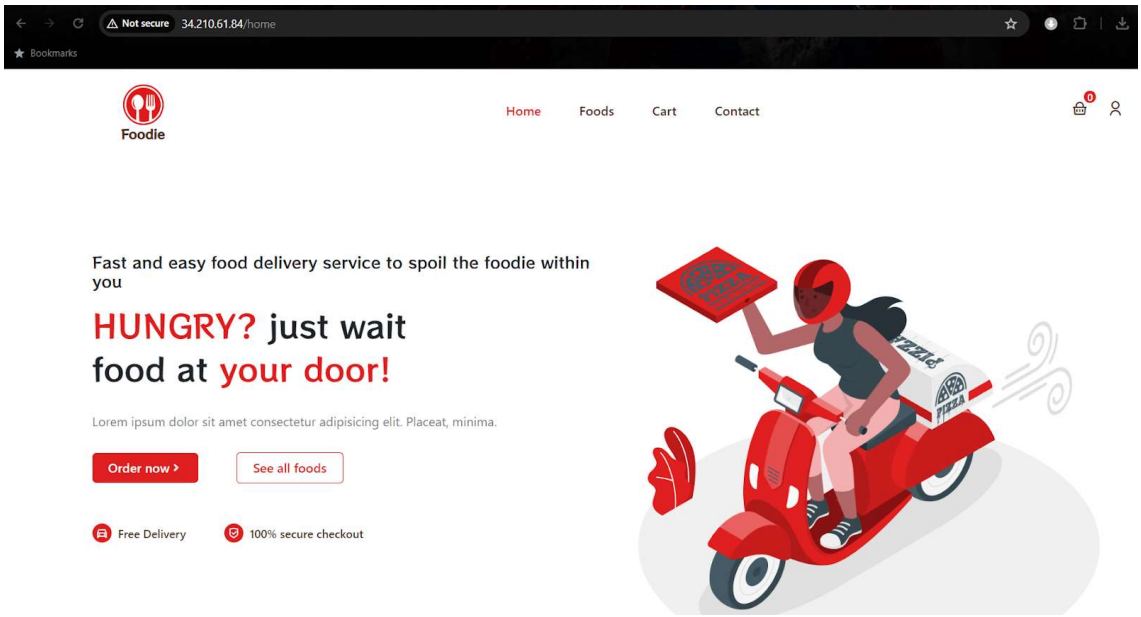
➔ creating a docker volume: by using **docker volume create** command:

```
root@ip-172-31-20-19:/home/ubuntu/food-delivery-react-redux-app# docker volume create nodejs-vol
nodejs-vol
root@ip-172-31-20-19:/home/ubuntu/food-delivery-react-redux-app# docker volume ls
DRIVER    VOLUME NAME
local     nodejs-vol
root@ip-172-31-20-19:/home/ubuntu/food-delivery-react-redux-app# vi docker-compose.yml
root@ip-172-31-20-19:/home/ubuntu/food-delivery-react-redux-app#
```

➔ Deploying the application through docker-compose:

```
root@ip-172-31-20-19:/home/ubuntu/food-delivery-react-redux-app# docker-compose up -d
Creating network "food-delivery-react-redux-app_default" with the default driver
Creating nodejs ... done
root@ip-172-31-20-19:/home/ubuntu/food-delivery-react-redux-app# docker-compose ps
 Name                 Command               State                Ports
-------------------------------------------------------------------------------------
nodejs   /docker-entrypoint.sh ngin ...   Up      0.0.0.0:80->80/tcp,:::80->80/tcp
root@ip-172-31-20-19:/home/ubuntu/food-delivery-react-redux-app#
```

Browser output:



Step:4 - Creating Build and Deployment scripts:

➔ Setting up Docker hub credentials as environmental variables:
➔ Creating build.sh for building docker image:

Build.sh contains:

```
#!/bin/bash
#login into dockerhub:
docker login -u $DOCKER_USER -p $DOCKER_PASS

#building the image:
docker build -t nodejs:web .
docker images

#stopping the already running container:
docker stop nodejs
docker rm nodejs
```

➔ Changing the file permission and executing it:

```
root@ip-172-31-20-19:/home/ubuntu/food-delivery-react-redux-app# vi build.sh
root@ip-172-31-20-19:/home/ubuntu/food-delivery-react-redux-app# chmod +x build.sh
root@ip-172-31-20-19:/home/ubuntu/food-delivery-react-redux-app# ./build.sh
```

```
Step 10/11 : EXPOSE 80
 ---> Using cache
 ---> fc7e4dd06771
Step 11/11 : CMD ["nginx", "-g", "daemon off;"]
 ---> Using cache
 ---> 5d27baeac2a8
Successfully built 5d27baeac2a8
Successfully tagged nodejs:web
REPOSITORY   TAG         IMAGE ID       CREATED                  SIZE
<none>       <none>      98b8e5e447c0   Less than a second ago   624MB
nodejs       web         5d27baeac2a8   14 minutes ago           60MB
<none>       <none>      52ada153227e   16 minutes ago           624MB
nginx        alpine      01e5c69afaf6   7 days ago               42.6MB
node         16-alpine   2573171e0124   4 months ago             118MB
nodejs
nodejs
root@ip-172-31-20-19:/home/ubuntu/food-delivery-react-redux-app#
```

The script is working fine:

➔ Creating a Deploy.sh script for deployment of the container through docker-compose:

Deploy.sh file contains:

```
#!/bin/bash
```

```
#deploying the container:
docker-compose up -d

#tagging the image:
docker tag nodejs:web ravivarman46/nodejs:cicd

#pushing an image to dockerhub:
docker push ravivarman46/nodejs:cicd
echo "the image has been pushed to docker hub"
```

➔ Changing the file permission and executing it:

```
root@ip-172-31-20-19:/home/ubuntu/food-delivery-react-redux-app# vi deploy.sh
root@ip-172-31-20-19:/home/ubuntu/food-delivery-react-redux-app# chmod +x d
deploy.sh           docker-compose.yml  dockerfile
root@ip-172-31-20-19:/home/ubuntu/food-delivery-react-redux-app# chmod +x deploy.sh
root@ip-172-31-20-19:/home/ubuntu/food-delivery-react-redux-app# ./deploy.sh
Creating nodejs ... done
The push refers to repository [docker.io/ravivarman46/nodejs]
f7013e16d4b2: Pushed
a34b395c0ca3: Layer already exists
5e728486380e: Layer already exists
b968c967e155: Layer already exists
92ef9174e989: Layer already exists
28c7b3c0b176: Layer already exists
fbed1f6990ee: Layer already exists
dd731ddf52be: Layer already exists
9fe9a137fd00: Layer already exists
cicd: digest: sha256:bad153ed58e0563e78a323a437f22404a437cf06a48712c12feda33004d418d9 size: 2201
the image has been pushed to docker hub
root@ip-172-31-20-19:/home/ubuntu/food-delivery-react-redux-app#
```

Deploy.sh is working is fine:

Step:5 – Creating a Jenkinsfile for Jenkins job: pipeline (optional)

Jenkinsfile contains:

```
pipeline {
    agent any

    stages {
        stage('Login into DockerHub') {
            steps {

withCredentials([usernamePassword(credentialsId: 'docker',
passwordVariable: 'DOCKER_PASSWORD', usernameVariable:
'DOCKER_USERNAME')]) {
```

```
                    sh "echo \$DOCKER_PASSWORD | docker
login -u \$DOCKER_USERNAME --password-stdin docker.io"
                }
            }
        }

        stage('Changing the File Permission') {
            steps {
                sh 'chmod +x build.sh'
                sh 'chmod +x deploy.sh'
            }
        }

        stage('Executing the File') {
            steps {
                sh './build.sh'
                sh './deploy.sh'
            }
        }
    }
}
```

Step:6 – Creating a GitHub repository and pushing all the files:

➔ Creating a new repository on GitHub:

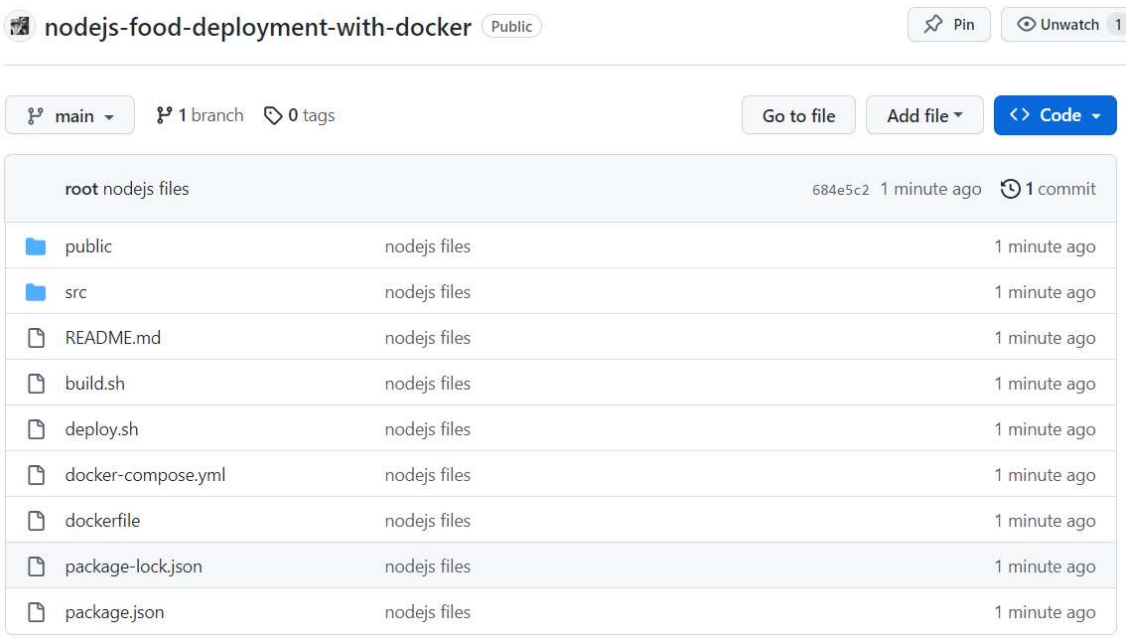
➔ Copying the https code and cloning it on command line:

➔ Copying all the files to the cloned directory and pushing it to the GitHub repository:

```
root@ip-172-31-20-19:/home/ubuntu/food-delivery-react-redux-app# mv * nodejs-food-deployment-with-docker/
mv: cannot move 'nodejs-food-deployment-with-docker' to a subdirectory of itself, 'nodejs-food-deployment-wit
oyment-with-docker'
root@ip-172-31-20-19:/home/ubuntu/food-delivery-react-redux-app# ls
nodejs-food-deployment-with-docker
root@ip-172-31-20-19:/home/ubuntu/food-delivery-react-redux-app# ls nodejs-food-deployment-with-docker/
README.md  build.sh  deploy.sh  docker-compose.yml  dockerfile  package-lock.json  package.json  public  src
root@ip-172-31-20-19:/home/ubuntu/food-delivery-react-redux-app#
```

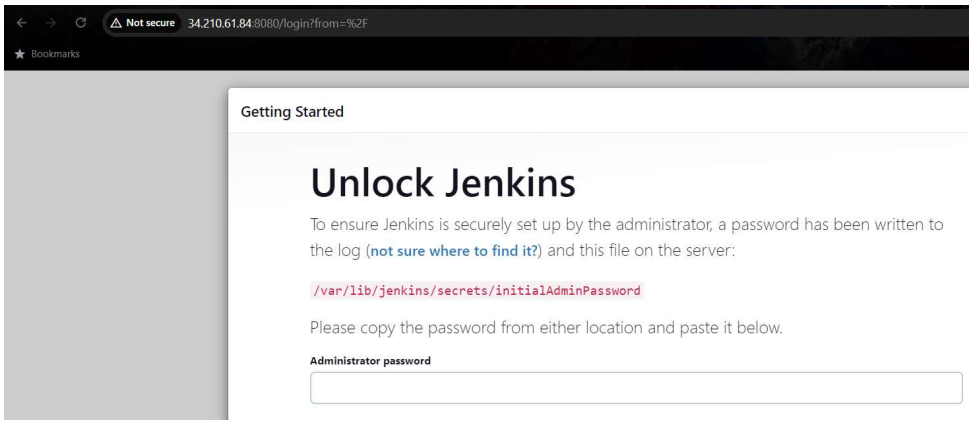
```
root@ip-172-31-20-19:/home/ubuntu/food-delivery-react-redux-app/nodejs-food-deployment-with-docker# git push origin main
Username for 'https://github.com': Ravivarman16
Password for 'https://Ravivarman16@github.com':
Enumerating objects: 118, done.
Counting objects: 100% (118/118), done.
Delta compression using up to 2 threads
Compressing objects: 100% (108/108), done.
Writing objects: 100% (118/118), 13.63 MiB | 15.13 MiB/s, done.
Total 118 (delta 3), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (3/3), done.
To https://github.com/Ravivarman16/nodejs-food-deployment-with-docker.git
 * [new branch]      main -> main
root@ip-172-31-20-19:/home/ubuntu/food-delivery-react-redux-app/nodejs-food-deployment-with-docker#
```

➔ Checking the GitHub repository:



Step:7 – Setting up Jenkins dashboard:

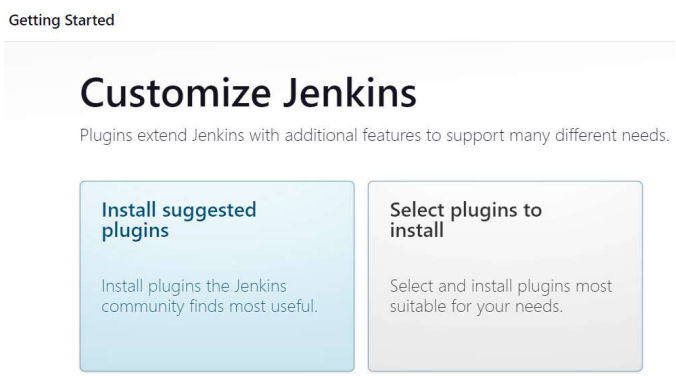
➔ Copy the public ip address of the instance along with the port number 8080 on the browser:

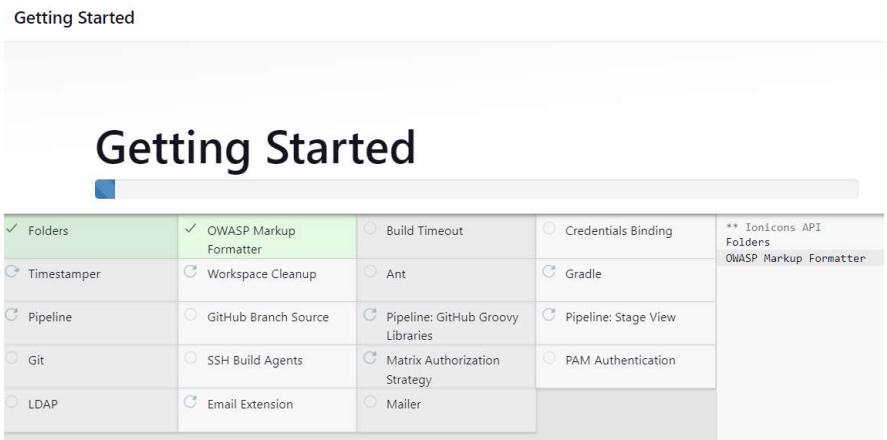


➔ Just copy the path paste it on the command line to get the password and paste it there and click continue:

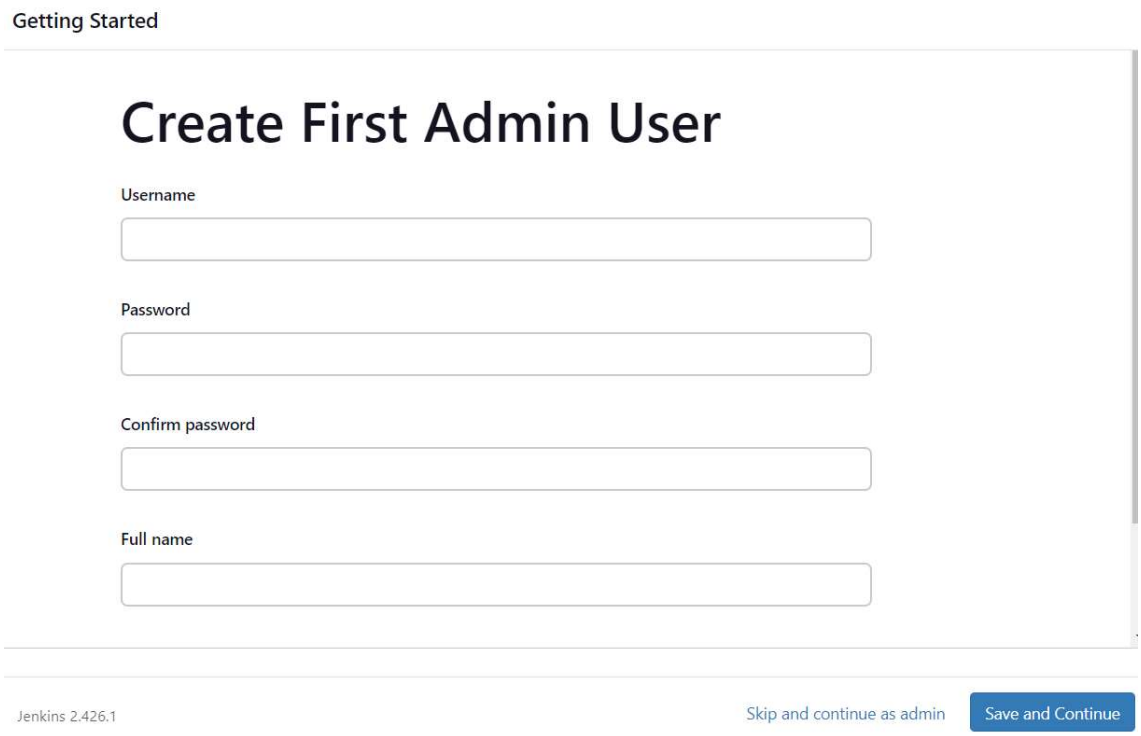
```
root@ip-172-31-20-19:/home/ubuntu/food-delivery-react-redux-app# cat /var/lib/jenkins/secrets/initialAdminPassword

ccf291decc85474eb472b396deb39180
root@ip-172-31-20-19:/home/ubuntu/food-delivery-react-redux-app#
```

➔ Then select install suggested plugins option, it will start installing plugins:

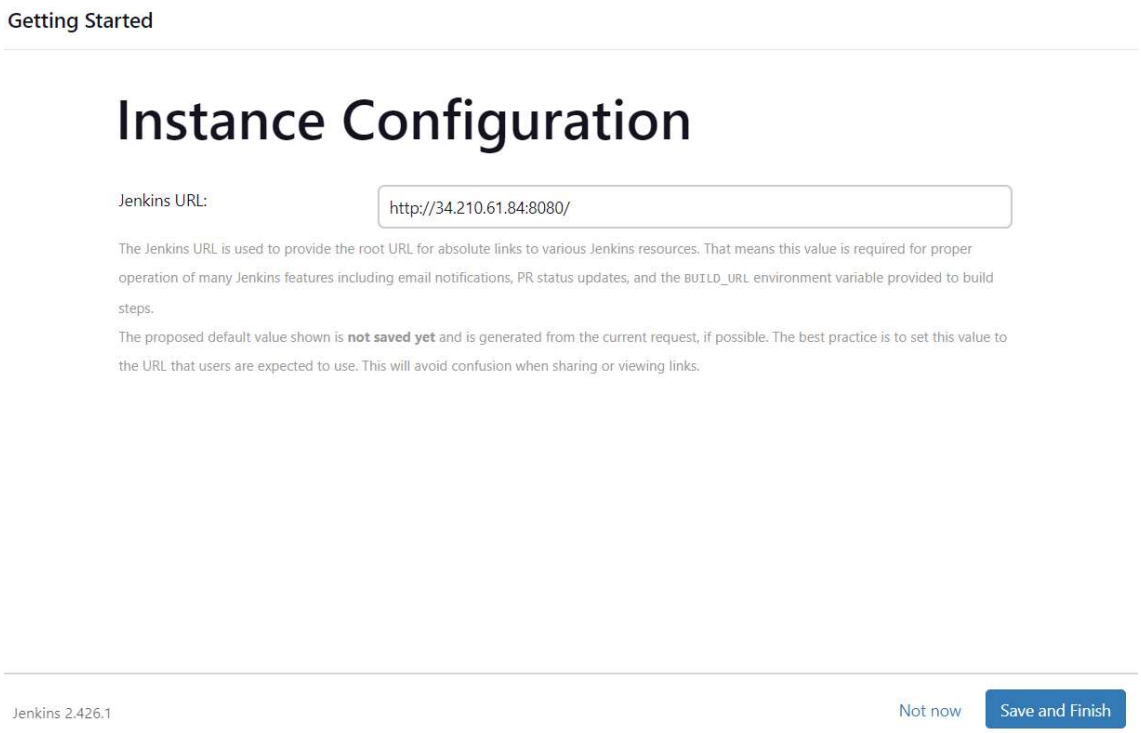




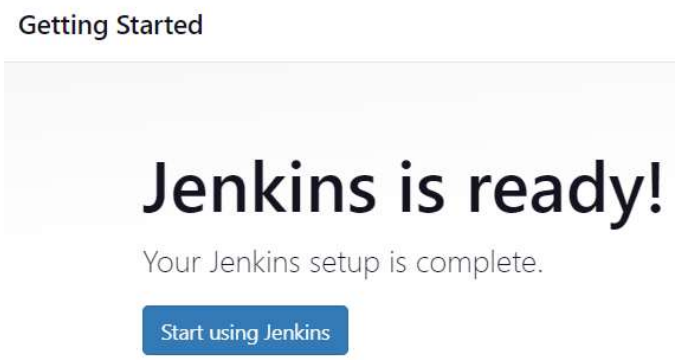
➔ Then we can able to see credentials dashboard just setup the credentials for Jenkins, click save and continue:



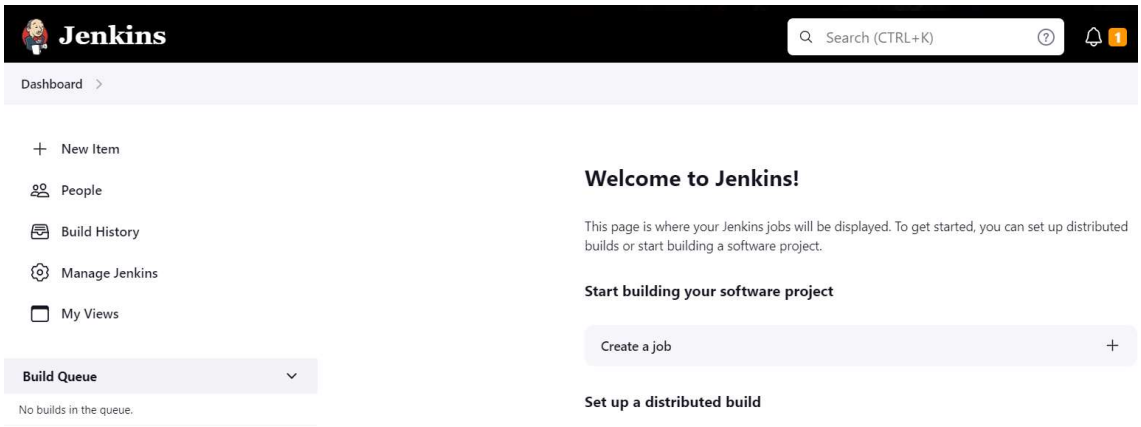
➔ Then click save and finish:



➔ Then click start using Jenkins option to see the Jenkins dashboard:

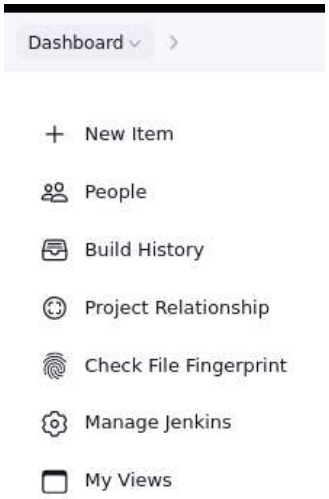


➔ We can able to the Jenkins dashboard:

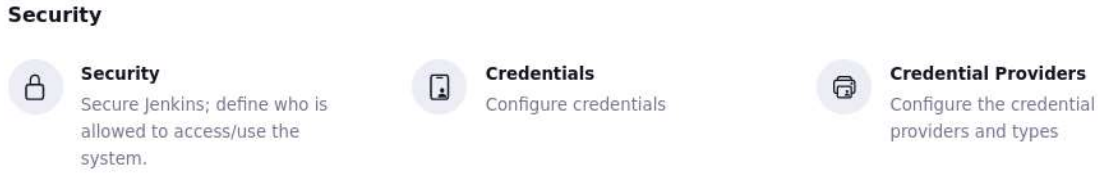


Step:8 – Setting Docker Hub credentials in Jenkins: (optional)

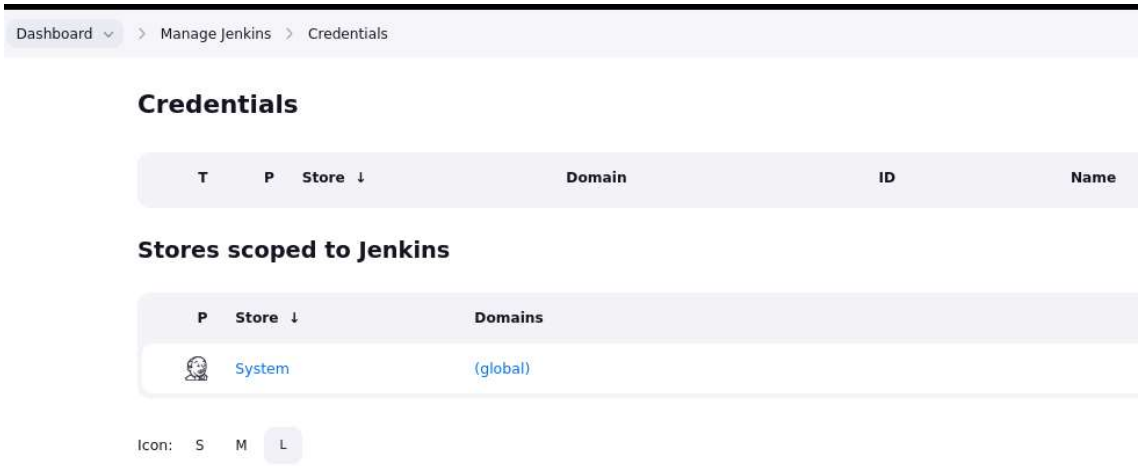
➔ On Jenkins dashboard, we can able to see manage Jenkins, click that one:



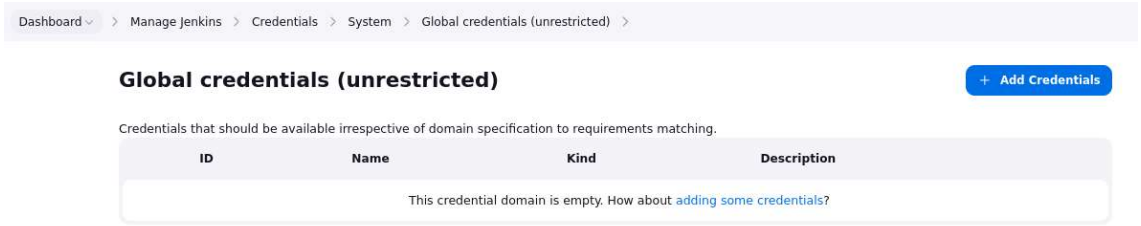
➔ We can able to see credentials, click that one:



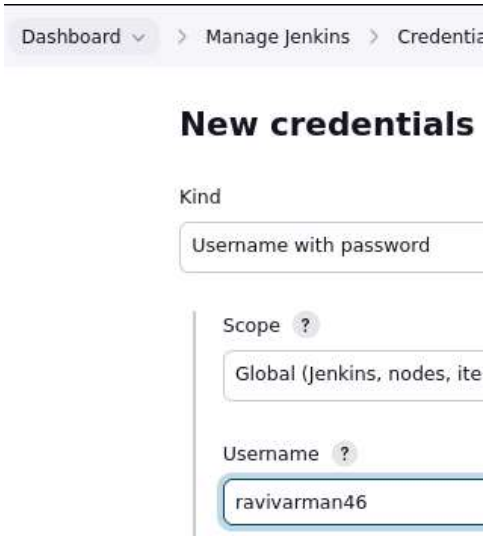
➔ Then click global one to add credentials:



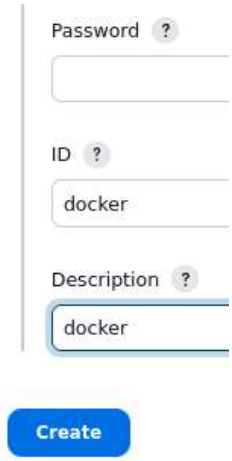
➔ Then click add credentials:



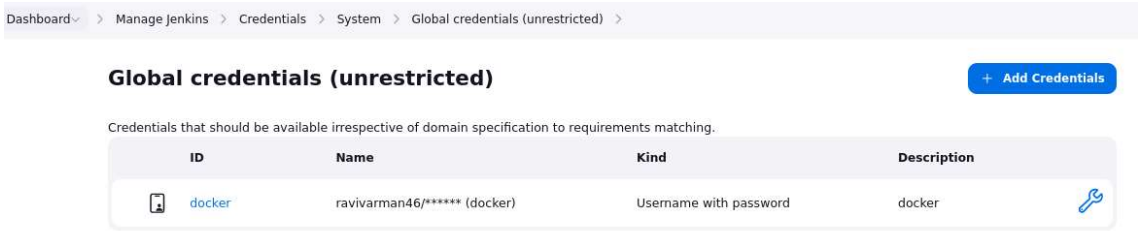
➔ Then give the docker hub username:



➔ Then give the Docker hub token, give the id as docker: click create:

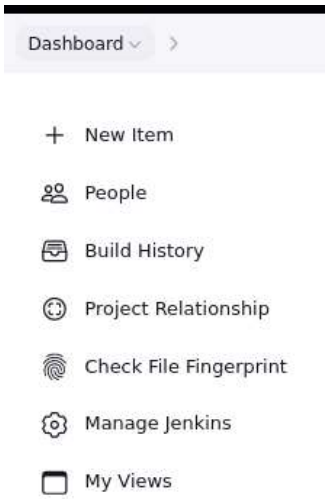


➔ The credentials have been created successfully:



Step:9 – Setting up Environmental variables:

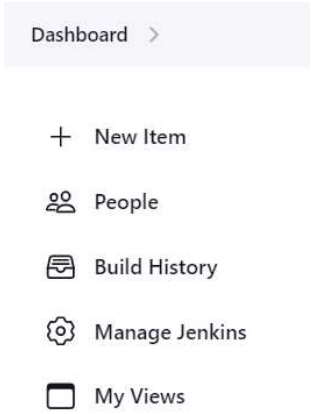
➔ On Jenkins dashboard we able to see manage Jenkins, click that one:



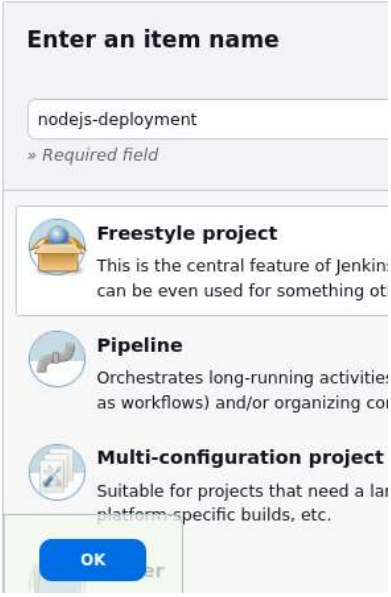
➔ Then click system: under global properties, select environment variables: set the variables: click save and apply

Step:10 – Creating a Jenkins job:

➔ On the Jenkins dashboard, we can able to see new item click that one:

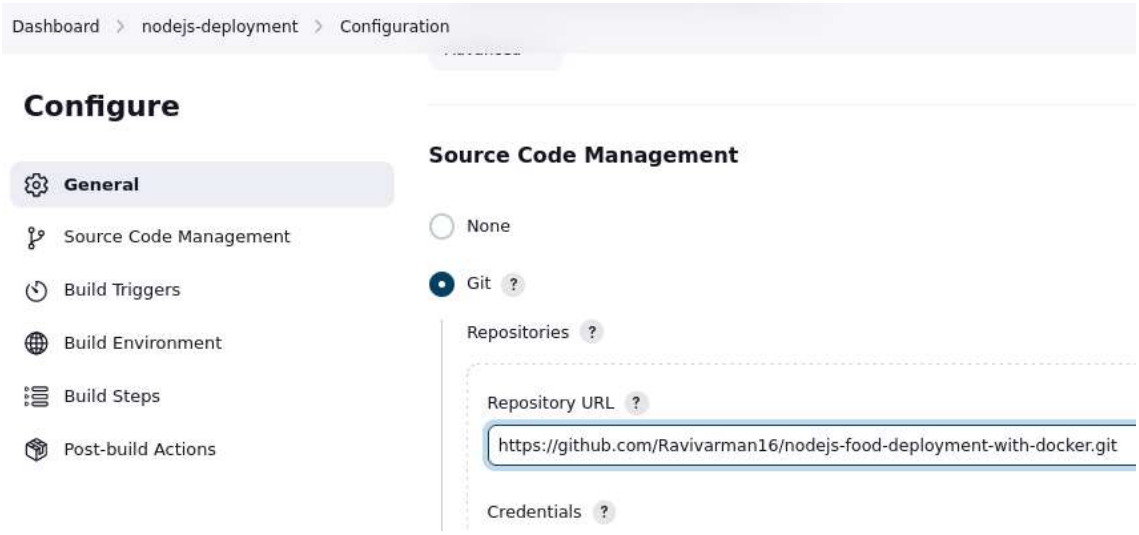


➔ Name the job and select **freestyle project** as job type:

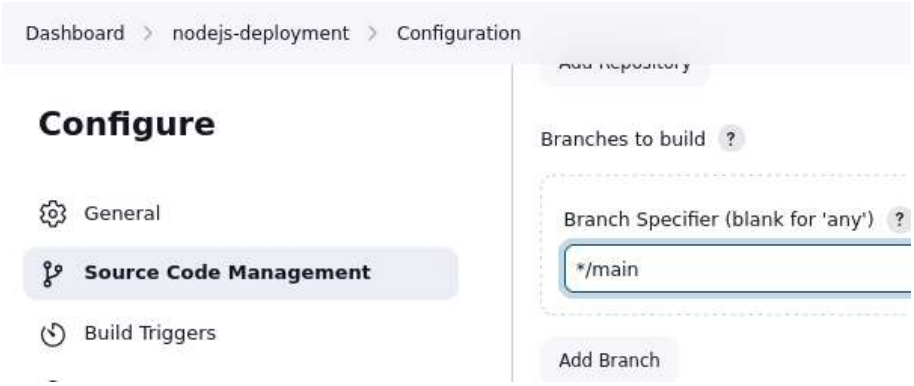


➔ Give the description according to your preferences:

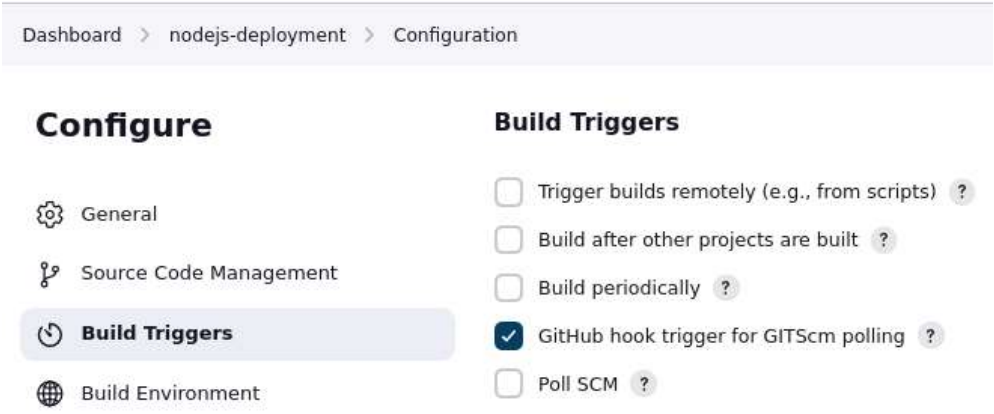
➔ Then under source-code select Git, enter your GitHub repository link:



➔ Then enter the branch name correctly:



➔ Then under build triggers, select GitHub hook trigger for GITScm polling option:



➔ Then under build steps, select execute shell option: enter the steps to be done:

chmod +x build.sh

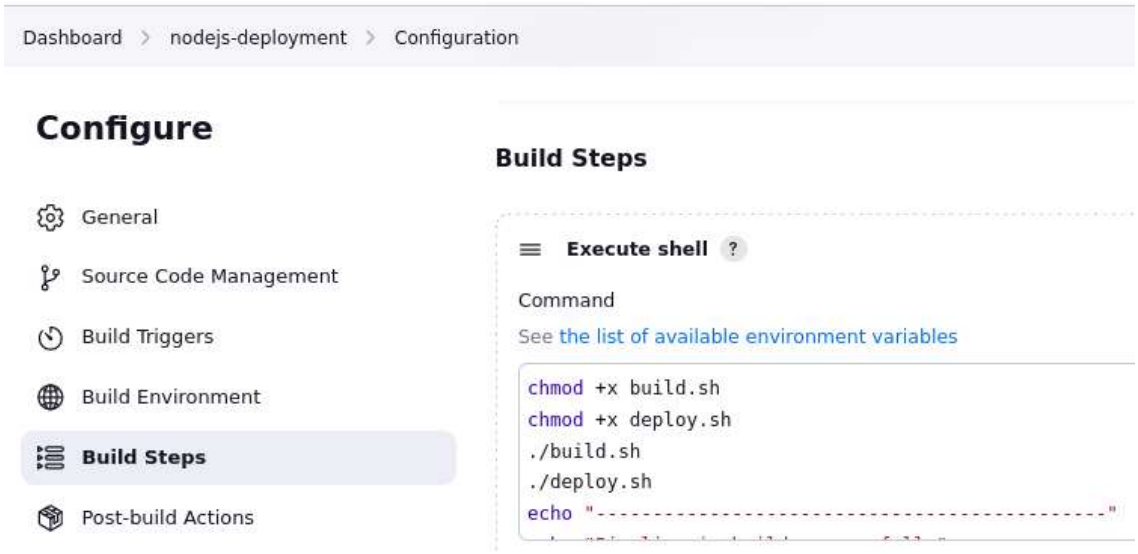
chmod +x deploy.sh

./build.sh

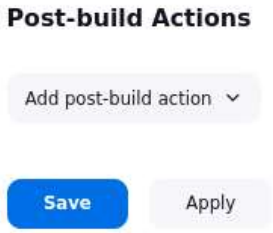
./deploy.sh

echo "----------"

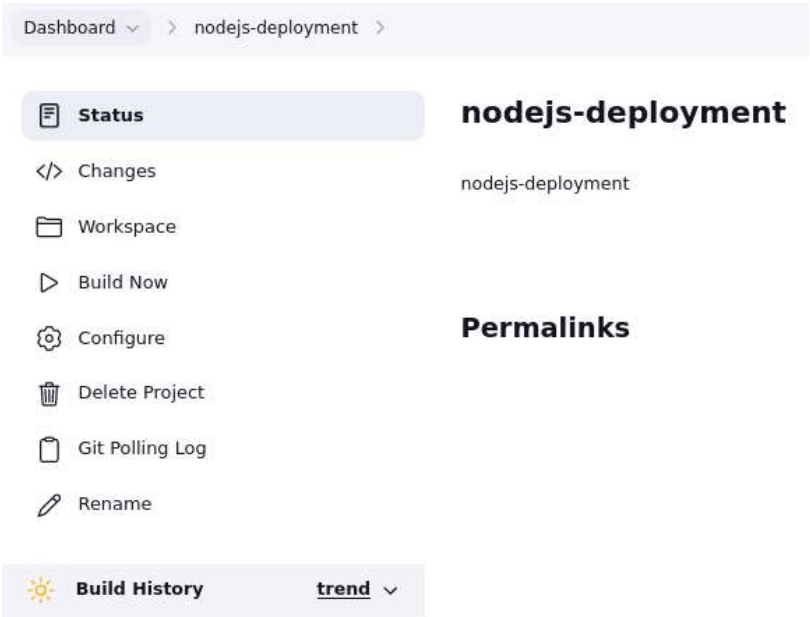
echo "Pipeline is build successfully"



➔ Then click apply & save:



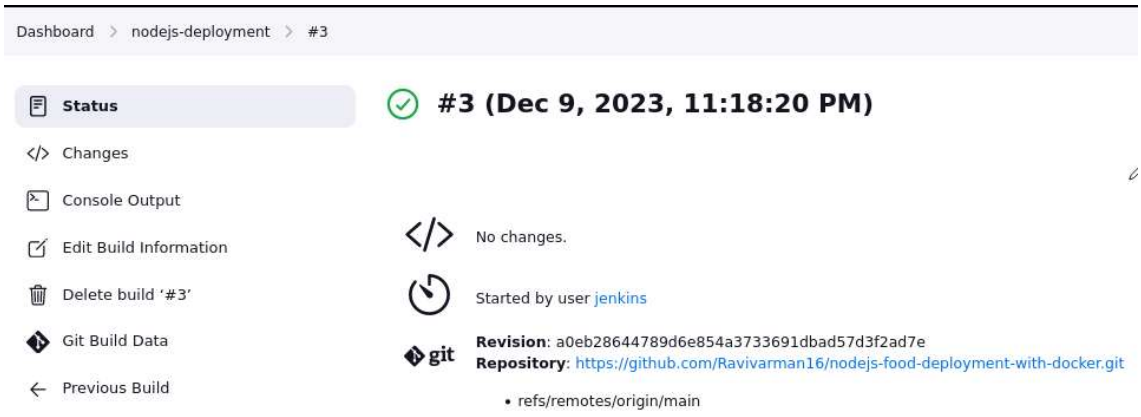
➔ Jenkins freestyle job is ready:



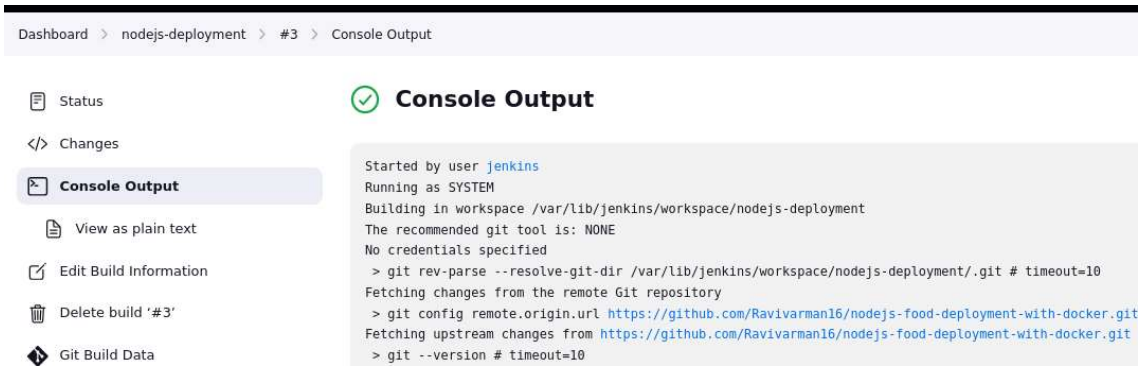
➔ Then click build now option to start the freestyle job: we can able to see job got initiated successfully:

We can able to see job started to run:

➔ Then click that one we can see output of the job:

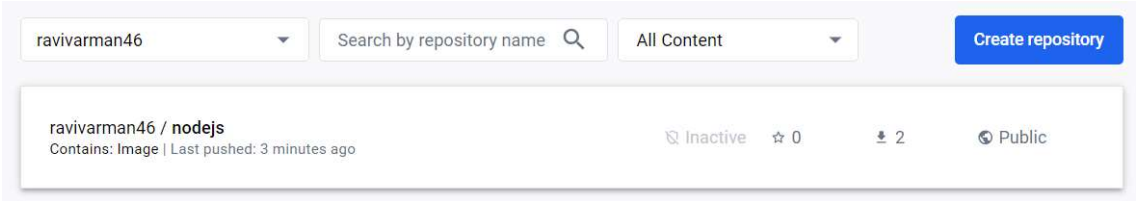


➔ Then click console output to know more details of this job:



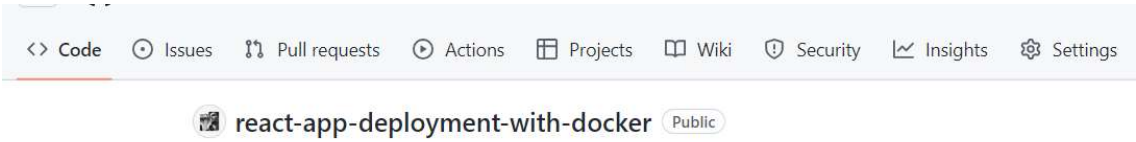

```
fbed1f6990ee: Layer already exists
28c7b3c0b176: Layer already exists
dd731ddf52be: Layer already exists
9fe9a137fd00: Layer already exists
cicd: digest: sha256:2c146fefbc3bb04b1554422fd86bd374a6ca7ecf5d202086ce25dfdb8bd06d5c size: 2201
the image has been pushed to docker hub
+ echo ---------------------------------------------
---------------------------------------------
+ echo Pipeline is build successfully
Pipeline is build successfully
Finished: SUCCESS
```

Docker Hub output:

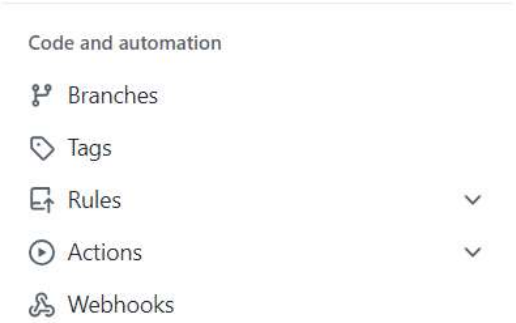


Step:11 – Making the pipeline automated by GitHub hook Trigger:

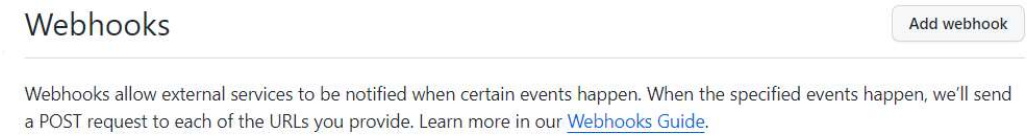
➔ On the GitHub repository, we can able to see settings, click that one:



➔ Then on the left side, we can able to see webhooks, click that one:



➔ Then we can able to see add webhook option, click that one:



➔ Then under payload URL, enter the Jenkins URL along with github-webhook:

Webhooks / Add webhook

We'll send a POST request to the URL below with detail
format you'd like to receive (JSON, x-www-form-urlenco
documentation.

Payload URL *

http://34.210.61.84:8080/github-webhook/

➔ Then select the event and click add webhooks:

Which events would you like to trigger this webhook?

- (●) Just the push event.
- ( ) Send me **everything**.
- ( ) Let me select individual events.

[x] **Active**
We will deliver event details when this hook is triggered.

Add webhook

➔ We can able to see webhook trigger is set perfectly:

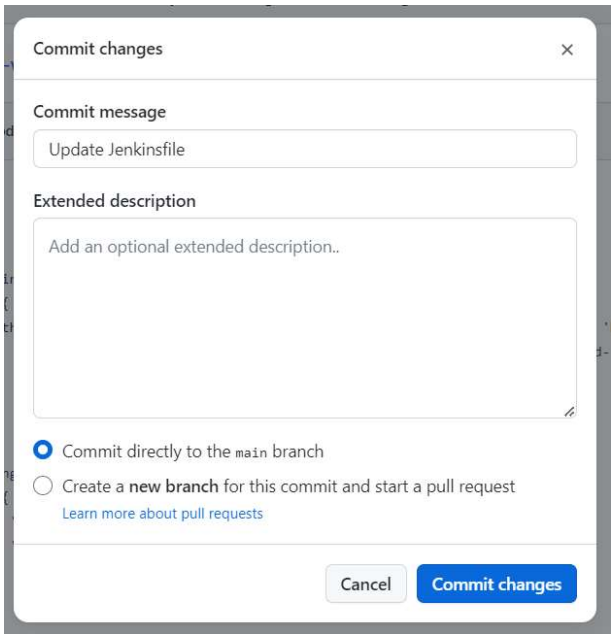
Webhooks

Add webhook

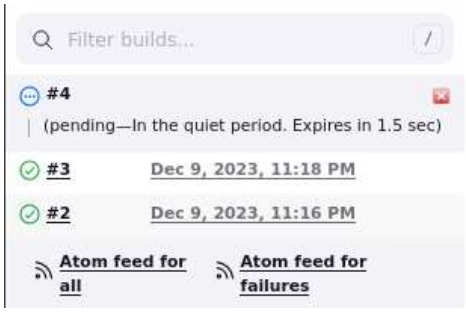
Webhooks allow external services to be notified when certain events happen. When the specified events happen, we'll send a POST request to each of the URLs you provide. Learn more in our Webhooks Guide.

✓ http://34.210.61.84:8080/github-we... *(push)* Edit Delete

➔ Checking it by making changes in GitHub:



➔ We can able to see the Jenkins job got triggered automatically:



Console output:

Benefits of above task:

- ➔ **Efficiency and Automation**: Automation through Jenkins enables the continuous integration and deployment of the Node.js application, reducing manual intervention and minimizing the risk of human errors.

  Docker containers provide a consistent and isolated environment for the application, ensuring that it runs reliably across different stages of development and deployment.

- ➔ **Scalability and Resource Optimization:** Leveraging Docker allows for efficient scaling of the application by deploying multiple instances of the Docker container. This ensures optimal resource utilization and the ability to handle varying workloads effectively.

  EC2 instances in AWS provide scalability by allowing the organization to adjust the compute capacity based on demand, ensuring the application can scale horizontally to meet changing performance requirements.

- ➔ **Cost-Effective Hosting and Resource Management**: Docker's lightweight containers reduce infrastructure costs by optimizing resource consumption, enabling more efficient utilization of server resources.

  AWS EC2 instances can be configured and scaled based on actual usage, helping to manage costs effectively. Additionally, the use of Jenkins for automation contributes to cost savings by streamlining the development and deployment processes.

All the files for the above task have been uploaded under this GitHub repository:

https://github.com/yasminjeelani/simple-fooddeliveryapp-deployment.git