1. REACT APP DEPLOYMENT WITH DOCKER

PROBLEM STATEMENT:

ABC Tech is developing an e-commerce website for a client and requires an efficient deployment solution for the React application. The challenge is to deploy the React app using Docker in a way that streamlines the process, ensures easy management, and allows for cost-effective hosting.

USE CASE SCENARIO:

➔ **Business Requirement:** ABC Tech aims to deploy the client's e-commerce React application using Docker, leveraging a multi-stage Docker build approach. This deployment strategy should automate the build and deployment processes, promoting simplicity and efficiency.

➔ **Technical Challenge:** The objective is to containerize the React app, making it easily deployable across various environments. Additionally, automation tools like Jenkins and bash scripts will be employed to automate the Docker image build and deployment to Docker Hub, ensuring a straightforward and cost-effective solution for hosting static content.

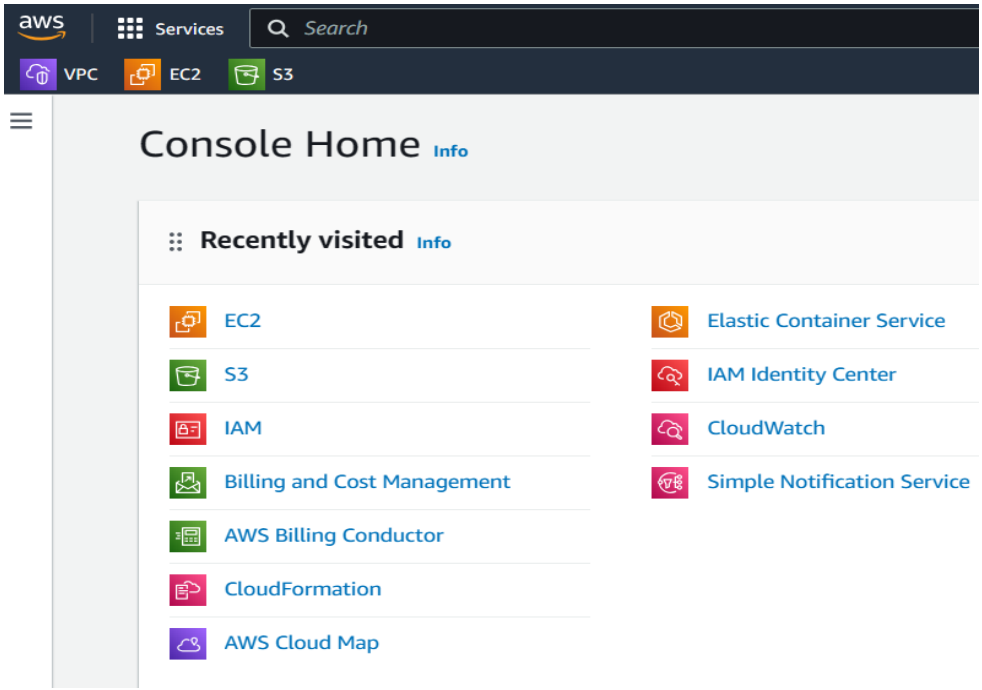
SOLUTION:

REQUIREMENTS:

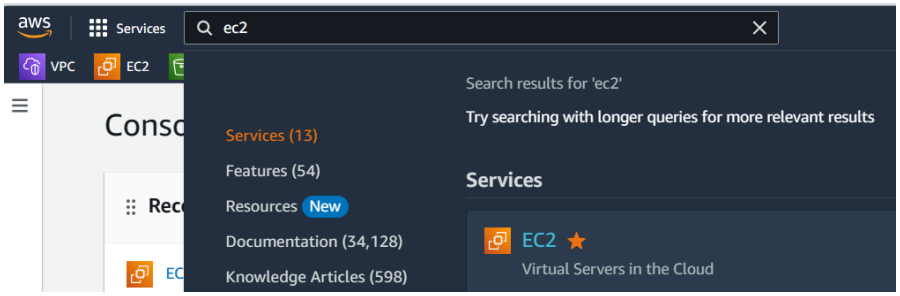
- ❖ AWS Cloud
- ❖ AWS EC2 instance
- ❖ Git & GitHub
- ❖ Docker
- ❖ Java
- ❖ Jenkins

Step:1 – Launching an EC2 instance:

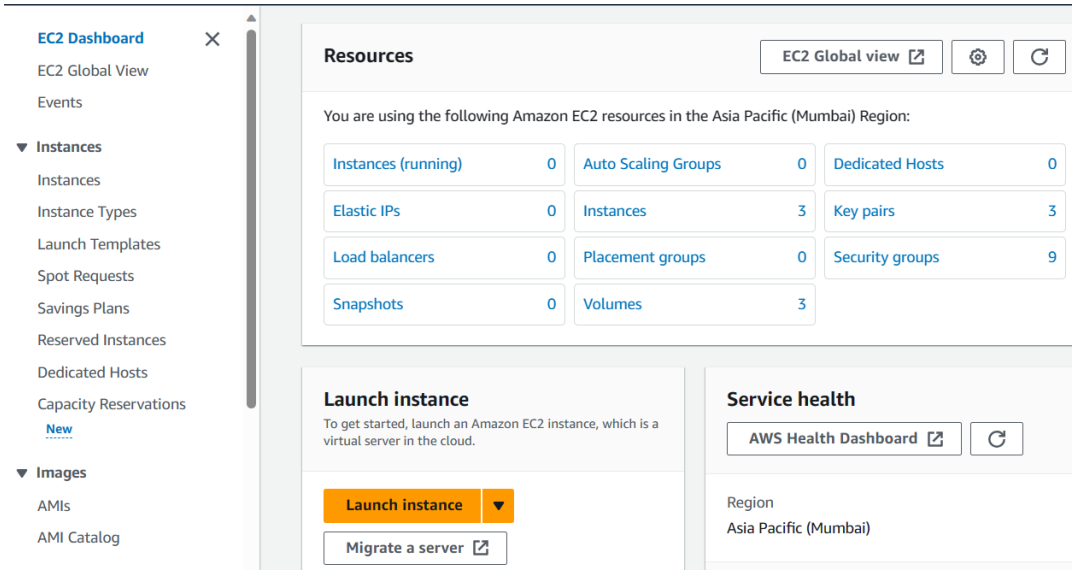
➔ First login into your AWS instance:



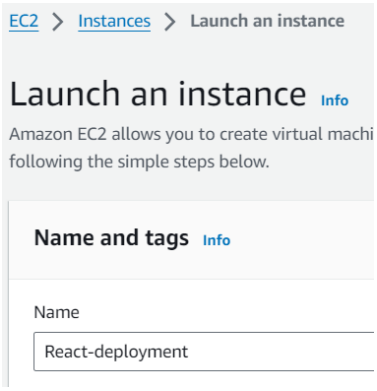
➔ Then on service search panel search EC2, click that one:



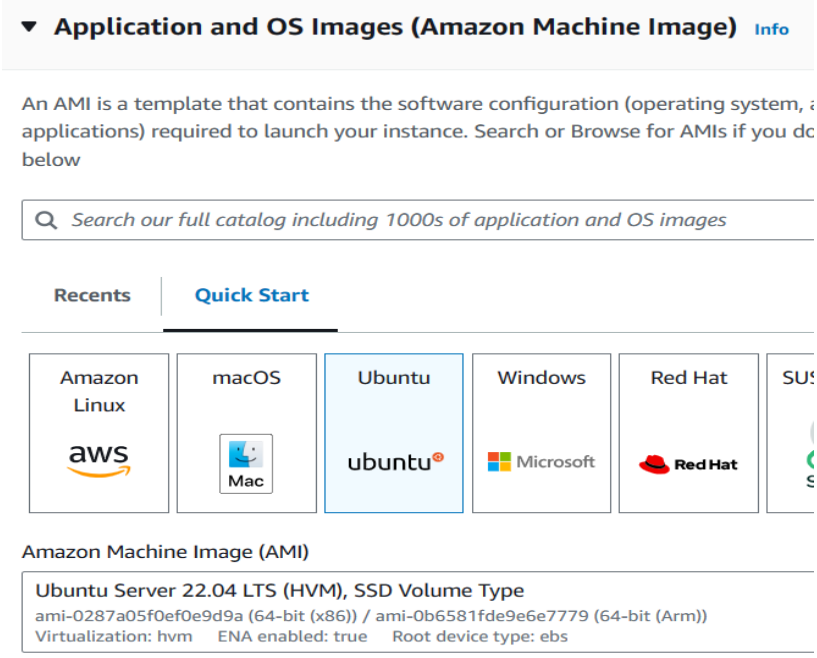
➔ Then click launch instances, for creating an EC2 instance:



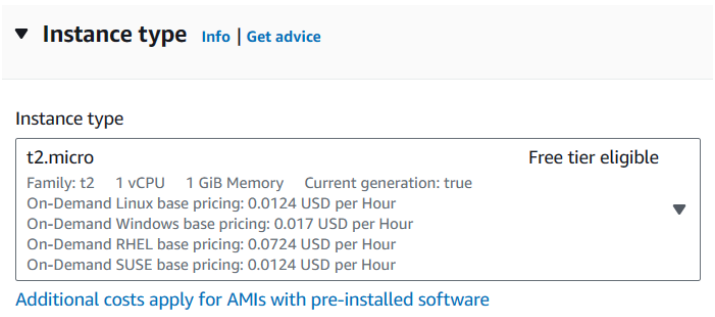
➔ Then name the instance according to your preferences:



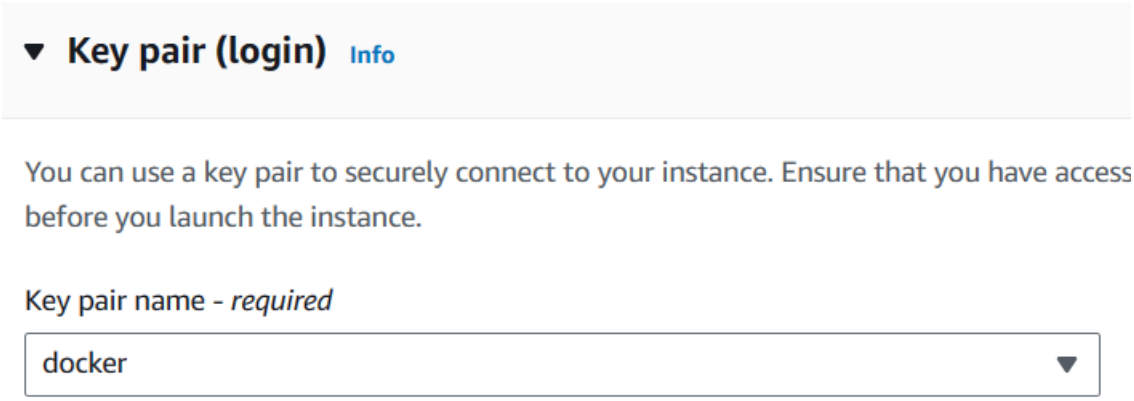
➔ Then select the operating system according to your preferences:



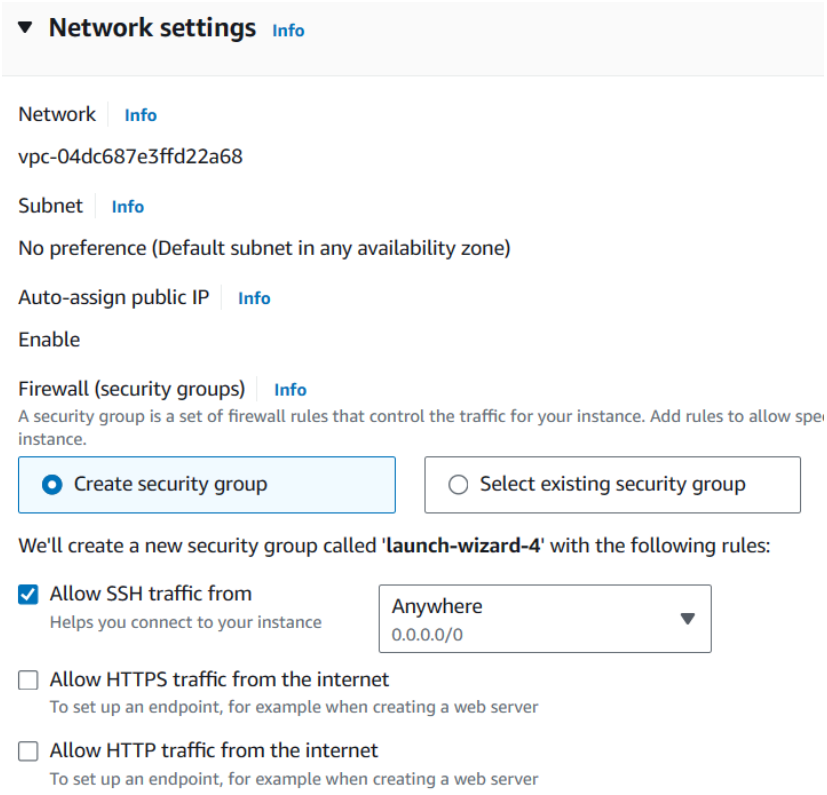
➔ Then select the instance type: according to your preferences, but here I am selecting **t2.micro**



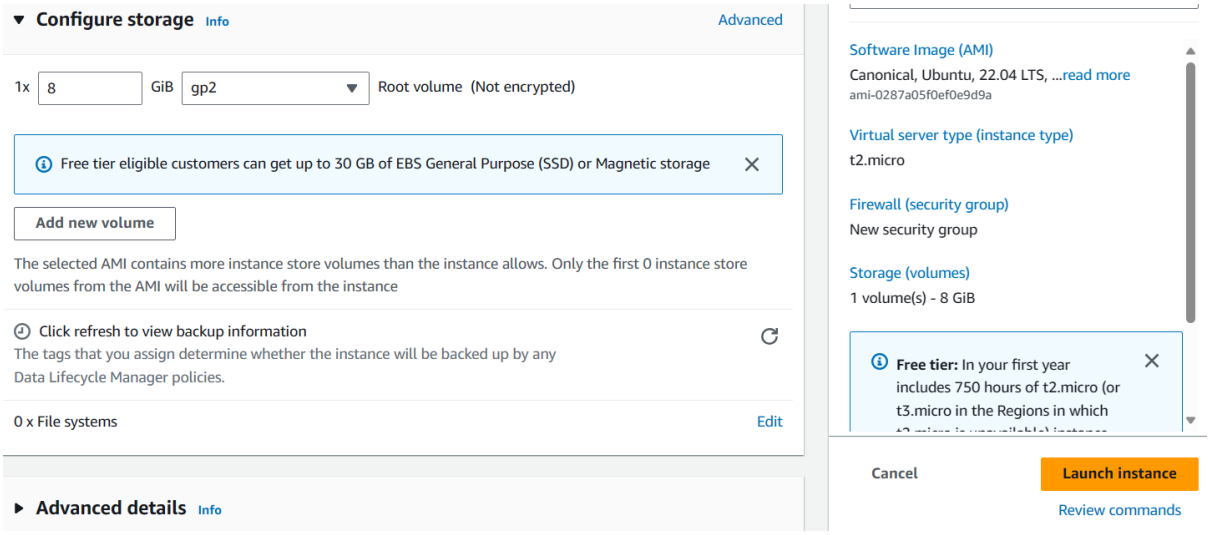
➔ Then select the key pair, according to your preferences, but here I am **proceeding with key pair option**, you can go with proceed with **without key pair option:**



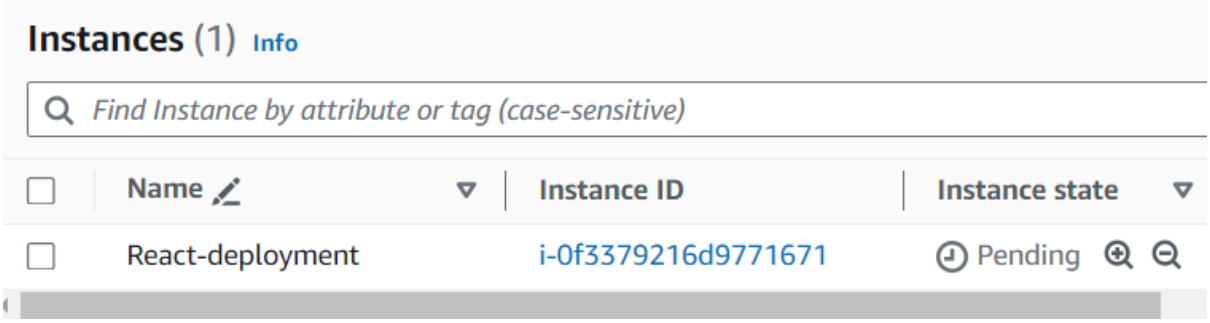
➔ Then keeping the default options under network settings:



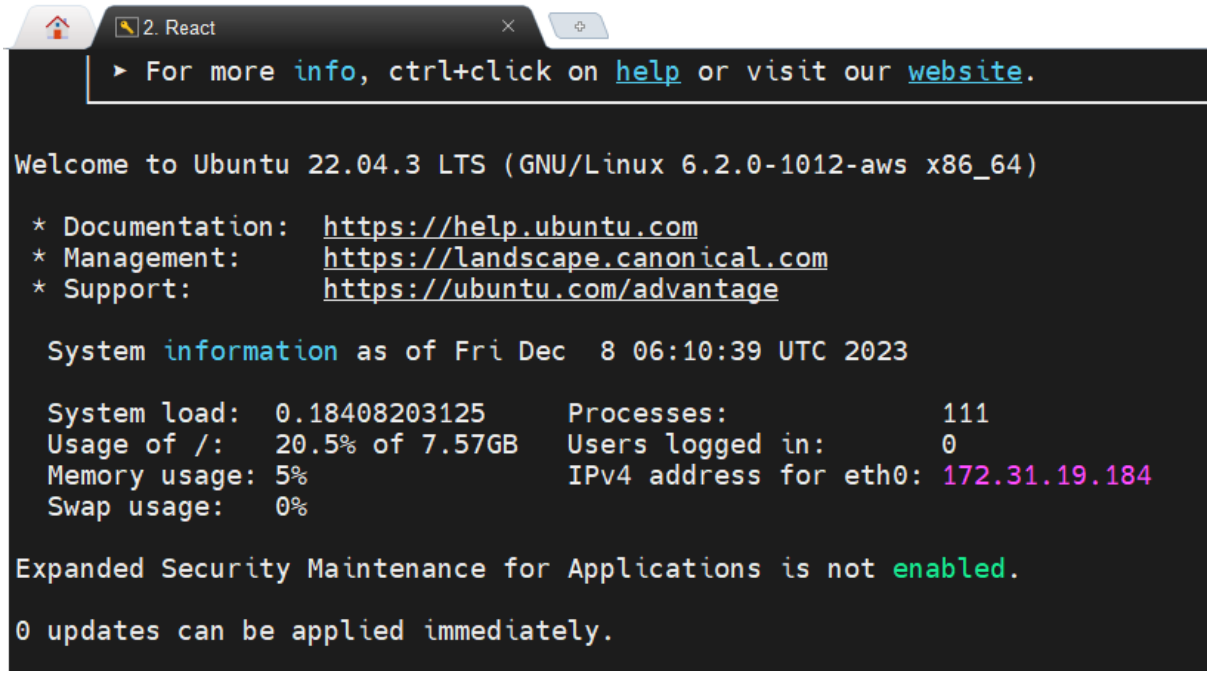
➔ Then keeping default options for the rest of the settings, click launch instance:



➔ The instance has been launched successfully:



➔ Connect the created instance with instance connect or with putty:



➔ Installing the necessary software's & services for this task:

- **Docker**
- **Java**
- **Jenkins**

By creating a shell file to install the necessary packages: the shell file 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 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 "This is the Java package - "
java --version

echo "This is Jenkins package - "
jenkins --version

echo "This is Docker package - "
docker --version
```

➔ Changing the file permission and executing it:

```
ubuntu@ip-172-31-19-184:~$ sudo su
root@ip-172-31-19-184:/home/ubuntu#
root@ip-172-31-19-184:/home/ubuntu# vi service.sh
root@ip-172-31-19-184:/home/ubuntu# chmod +x service.sh
root@ip-172-31-19-184:/home/ubuntu# ./service.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]
Get:5 http://security.ubuntu.com/ubuntu jammy-security InRelease [110 kB]
```

```
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
root@ip-172-31-19-184:/home/ubuntu#
```

All the packages have been installed successfully:

Step:3 – Dockerization of React application:

➔ I am going to use the React application from the GitHub repository: by using **git clone** command:

```
root@ip-172-31-19-184:/home/ubuntu# git clone https://github.com/Ravivarman16/react-App-Deployment-with-Docker.git
Cloning into 'react-App-Deployment-with-Docker'...
remote: Enumerating objects: 28, done.
remote: Counting objects: 100% (28/28), done.
remote: Compressing objects: 100% (21/21), done.
remote: Total 28 (delta 4), reused 25 (delta 4), pack-reused 0
Receiving objects: 100% (28/28), 242.28 KiB | 5.77 MiB/s, done.
Resolving deltas: 100% (4/4), done.
root@ip-172-31-19-184:/home/ubuntu# ls
react-App-Deployment-with-Docker  service.sh
root@ip-172-31-19-184:/home/ubuntu#
```

➔ Then going inside the cloned directory:

```
root@ip-172-31-19-184:/home/ubuntu/react-App-Deployment-with-Docker/react-code/route-app# ls -l
total 856
-rw-r--r-- 1 root root  80668 Dec  8 06:23 README.md
-rw-r--r-- 1 root root 782149 Dec  8 06:23 package-lock.json
-rw-r--r-- 1 root root    403 Dec  8 06:23 package.json
drwxr-xr-x 2 root root   4096 Dec  8 06:23 public
drwxr-xr-x 2 root root   4096 Dec  8 06:23 src
root@ip-172-31-19-184:/home/ubuntu/react-App-Deployment-with-Docker/react-code/route-app#
```

➔ Creating a dockerfile for above react 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;"]
```

➔ Building a docker image from the dockerfile:

s

```
root@ip-172-31-19-184:/home/ubuntu/react-App-Deployment-with-Docker/react-code/route-app# vi dockerfile
root@ip-172-31-19-184:/home/ubuntu/react-App-Deployment-with-Docker/react-code/route-app# docker build -t react-ci/cd .
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  913.4kB
Step 1/11 : FROM node:16-alpine as build
16-alpine: Pulling from library/node
7264a8db6415: Pull complete
eee371b9ce3f: Pull complete
93b3025fe103: Pull complete
d9059661ce70: Pull complete
Digest: sha256:a1f9d027912b58a7c75be7716c97cfbc6d3099f3a97ed84aa490be9dee20e787
Status: Downloaded newer image for node:16-alpine
 ---> 2573171e0124
Step 2/11 : WORKDIR /app
 ---> Running in 69020a1c1796
Removing intermediate container 69020a1c1796
 ---> 85e819a94585
Step 3/11 : COPY package.json .
 ---> 1534e4c0a41e
```

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

```
Successfully built 3c9a28e30e4e
Successfully tagged react-ci/cd:latest
root@ip-172-31-19-184:/home/ubuntu/react-App-Deployment-with-Docker/react-code/route-app# docker images
REPOSITORY    TAG          IMAGE ID       CREATED          SIZE
react-ci/cd   latest       3c9a28e30e4e   8 seconds ago    45MB
<none>        <none>       2fe9cbbb47b0   11 seconds ago   291MB
nginx         alpine       01e5c69afaf6   7 days ago       42.6MB
node          16-alpine    2573171e0124   3 months ago     118MB
```

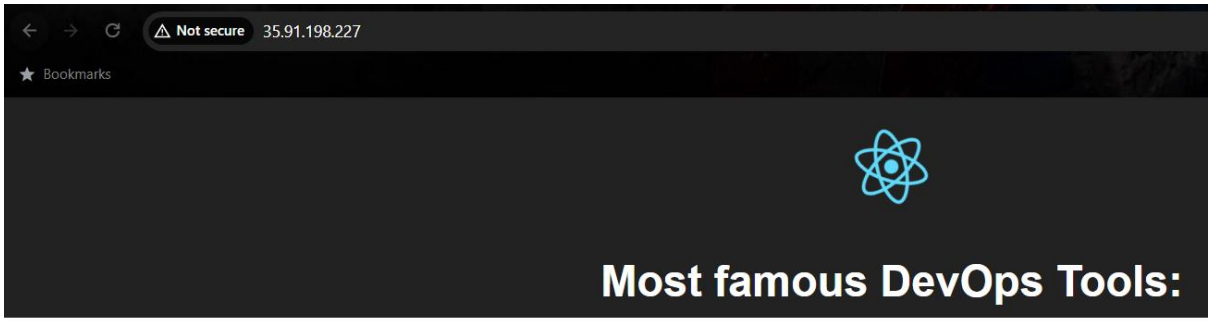
➔ Checking the output of the docker image by running a container from the above image by using **docker run** command:

docker run -d -it -p 80:80 <image-name>

```
root@ip-172-31-19-184:/home/ubuntu/react-App-Deployment-with-Docker/react-code/route-app# docker run -d -it -p 80:80 react-ci/cd
41dcee9e4269022691ff36310f560a4b71e9859f510544abbad89f73129b1c8d
root@ip-172-31-19-184:/home/ubuntu/react-App-Deployment-with-Docker/react-code/route-app# docker ps
CONTAINER ID   IMAGE         COMMAND                  CREATED             STATUS             PORTS
 NAMES
41dcee9e4269   react-ci/cd   "/docker-entrypoint.…"   5 seconds ago       Up 4 seconds       0.0.0.0:80->80/tcp, :::80->80/tcp
 zealous_feynman
```

➔ Checking the output by enabling port number 80 on the security group and pasting public ip address on the browser:

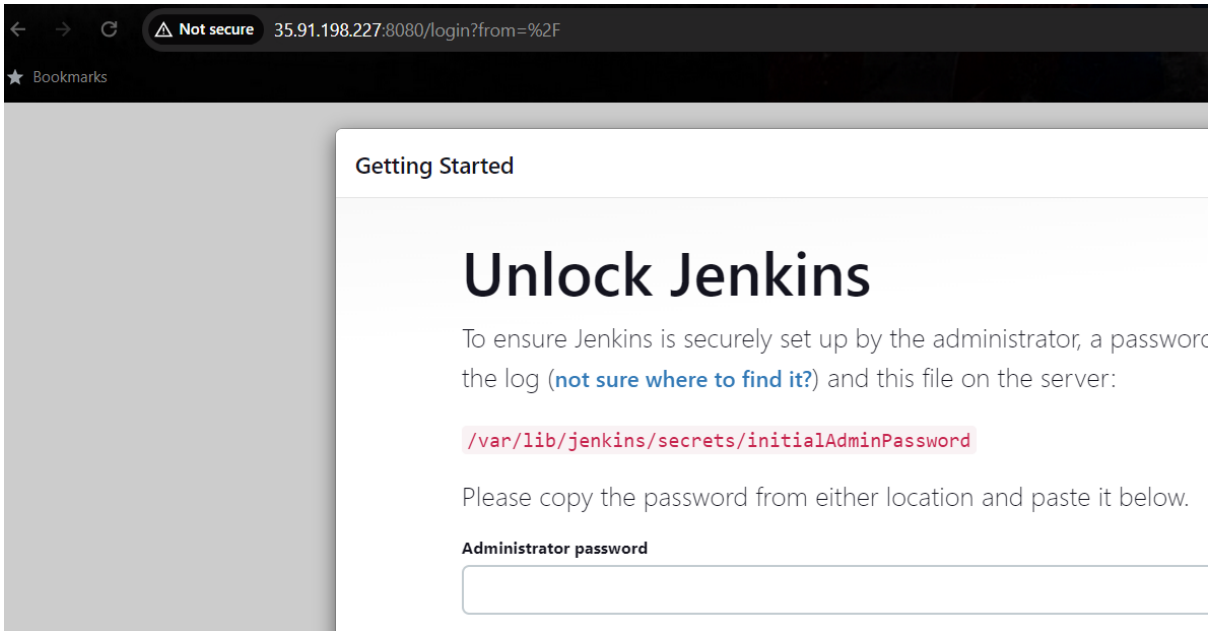
Browser output:



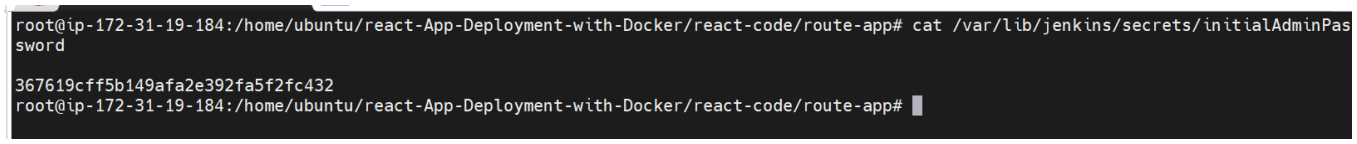
The docker image is working fine:

Step:4 - Setting up the Jenkins dashboard:

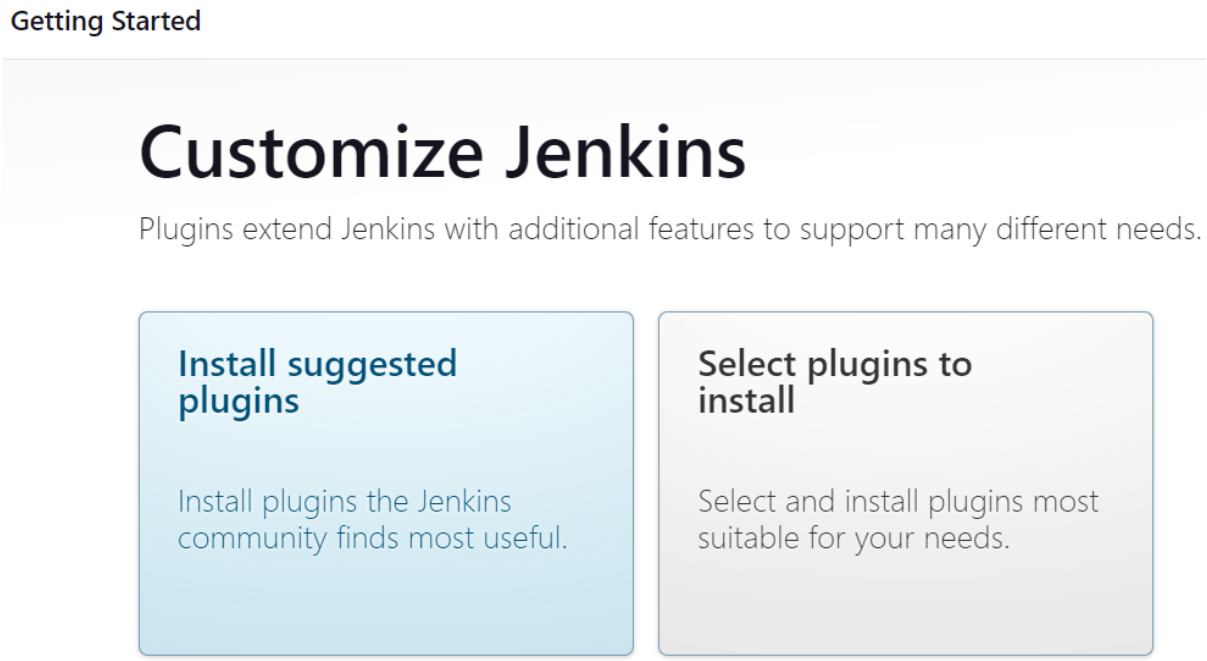
➔ Enabling the port number 8080 on the security group and pasting the public ip address along with the port number on the browser:



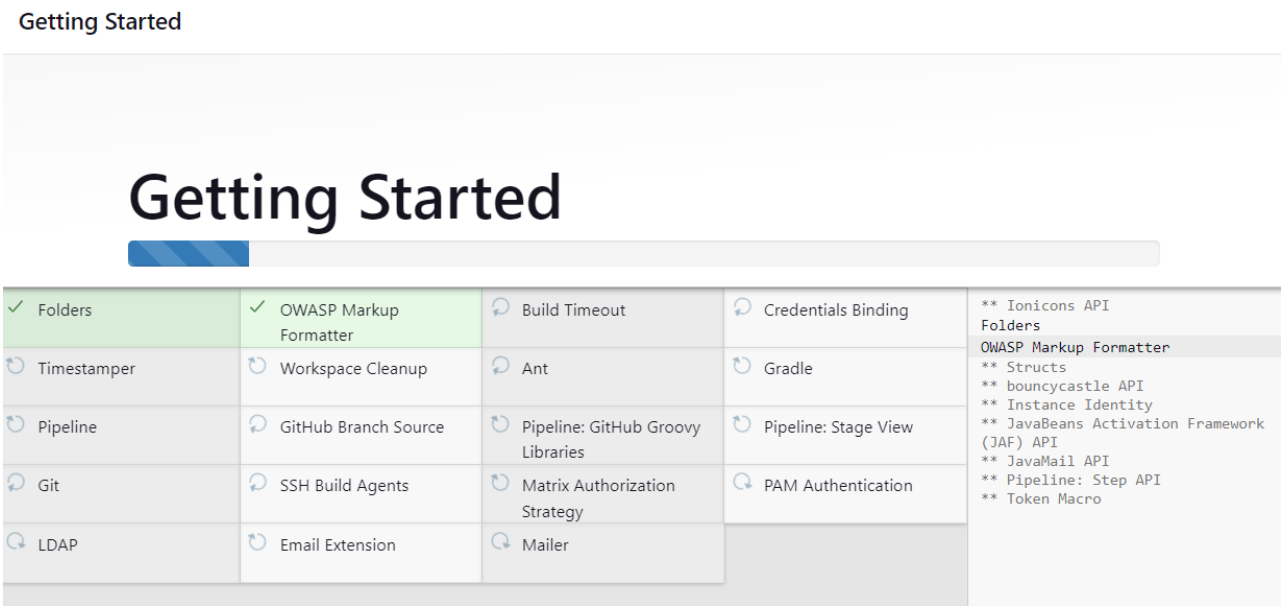
➔ We need to get the initial admin password by pasting the path on the command line: paste the password on the browser and proceed the next steps:



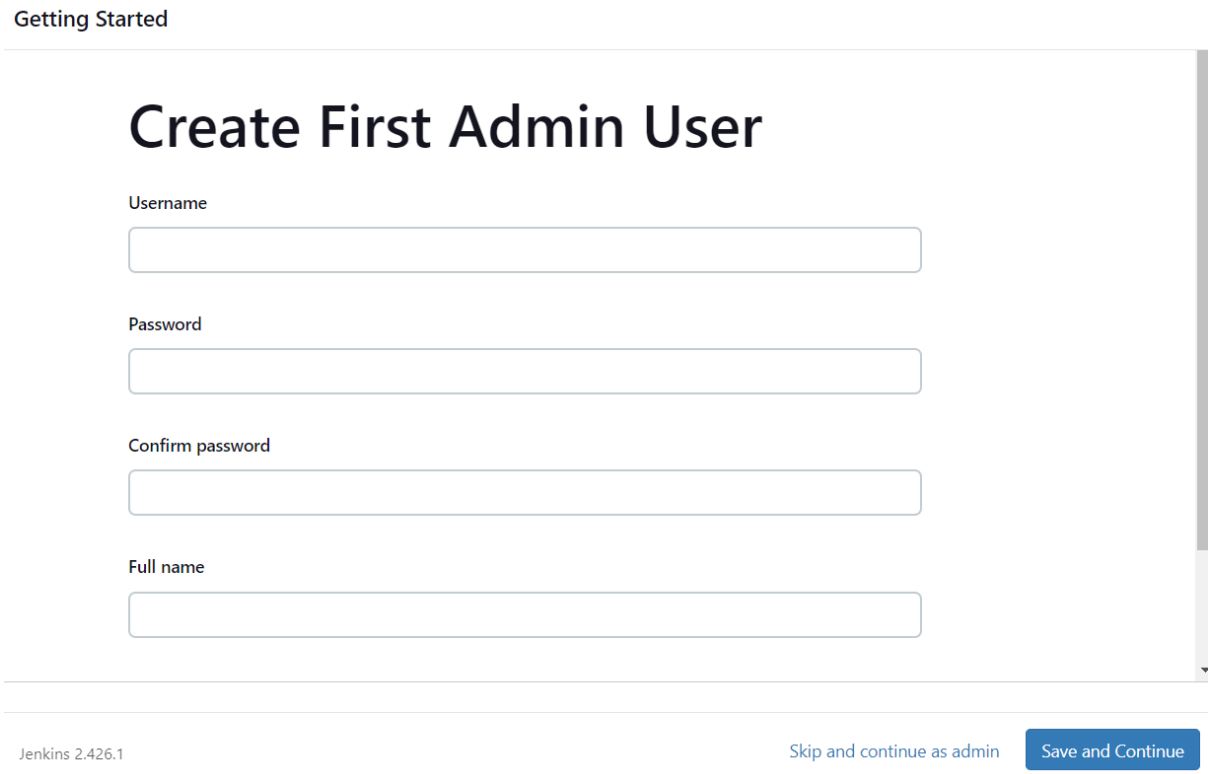
➔ Then click the install suggested plugins options:



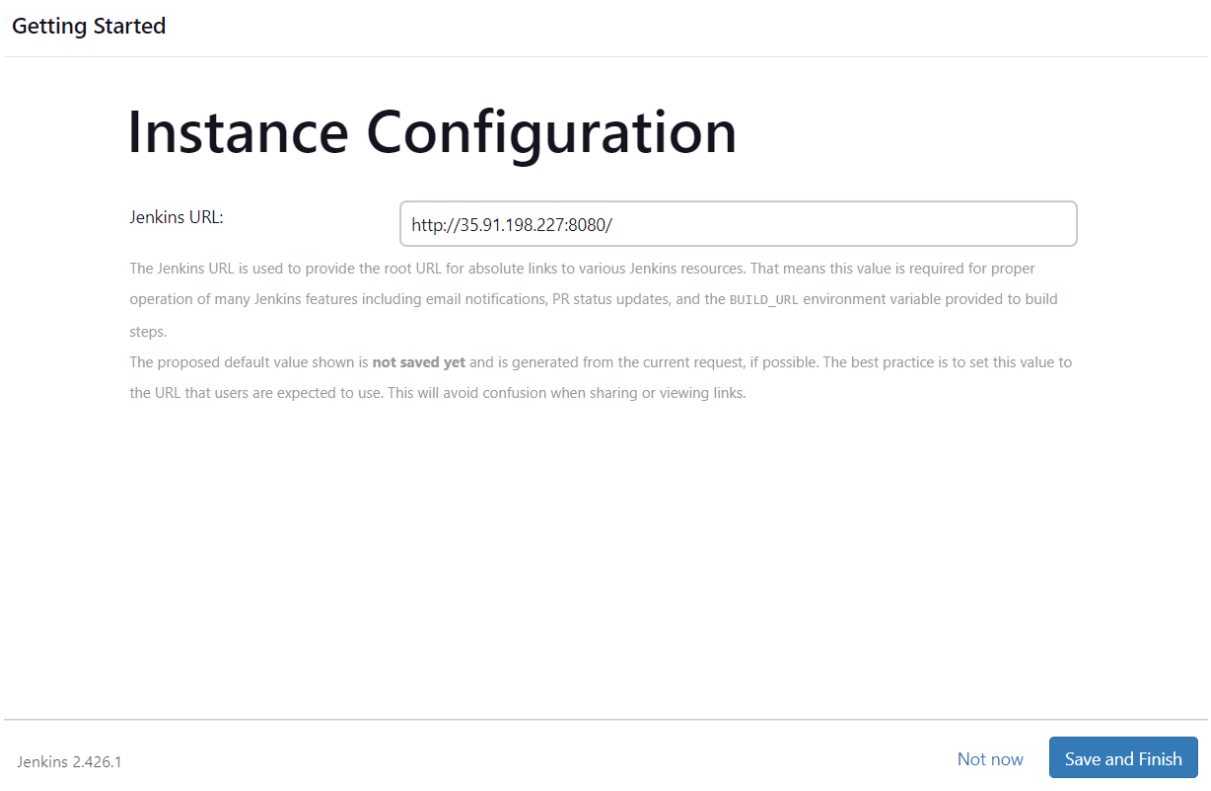
➔ Plugins will start to install:



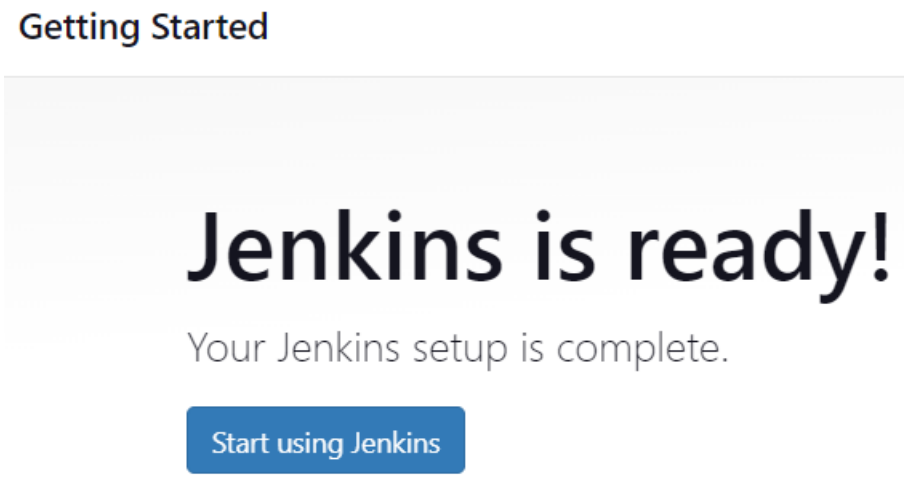
➔ Then we have to setup the credentials for Jenkins login purpose: click next:



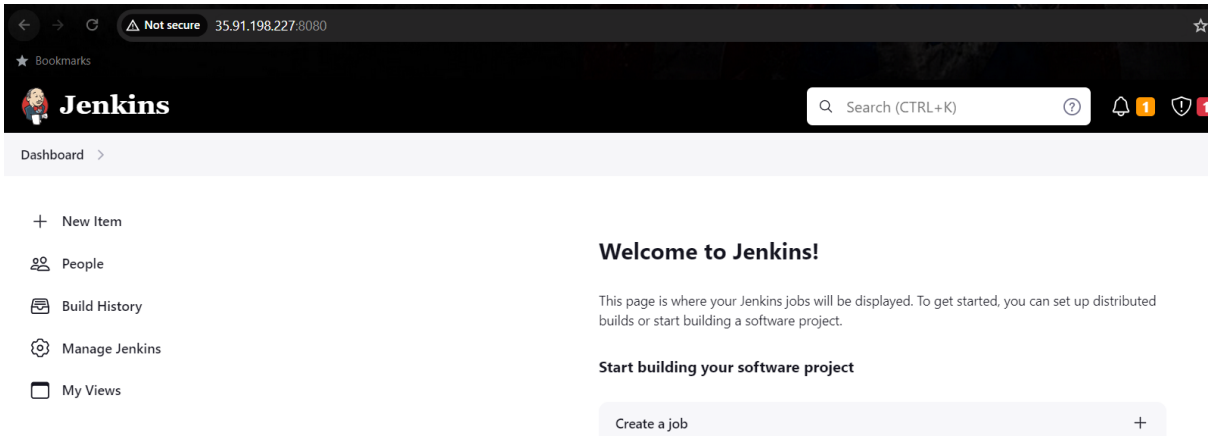
➔ Then click save and finish:



➔ Then we can able to see Jenkins is ready: click **start using Jenkins:**



➔ Jenkins dashboard:



Step:5 - Creating a script file for building & pushing the image to Docker Hub:

➔ Creating a script file for above purpose:

Script file contains:

```
#!/bin/bash

#login into DockerHub:
docker login -u $DOCKER_USERNAME -p $DOCKER_PASS
```

```
#stopping existing container:
docker stop react
docker rm react

#building a image:
docker build -t react-ci/cd .

#running a container from the created image:
docker run -d -it --name react -p 80:80 react-ci/cd

#pushing the image to dockerhub:
docker tag react-ci/cd ravivarman46/react-app:ci-cd
docker push ravivarman46/react-app:ci-cd
```

➔ Setting up Docker hub credentials environment variables:

```
root@ip-172-31-19-184:/home/ubuntu/react-app-deployment-with-docker/react-app-code# export DOCKER_USERNAME=ravivarman46
root@ip-172-31-19-184:/home/ubuntu/react-app-deployment-with-docker/react-app-code# echo $DOCKER_USERNAME
ravivarman46
root@ip-172-31-19-184:/home/ubuntu/react-app-deployment-with-docker/react-app-code# export DOCKER_PASS=dckr_pat_4RpB6x_mUNVKrFCLk2W5pq
BnywE
root@ip-172-31-19-184:/home/ubuntu/react-app-deployment-with-docker/react-app-code#
```

➔ Changing the file permission and executing it:

```
root@ip-172-31-19-184:/home/ubuntu/react-app-deployment-with-docker/react-app-code# vi build.sh
root@ip-172-31-19-184:/home/ubuntu/react-app-deployment-with-docker/react-app-code# chmod +x build.sh
root@ip-172-31-19-184:/home/ubuntu/react-app-deployment-with-docker/react-app-code# ./build.sh
WARNING! Using --password via the CLI is insecure. Use --password-stdin.
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
react
react
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  914.4kB
Step 1/11 : FROM node:16-alpine as build
 ---> 2573171e0124
Step 2/11 : WORKDIR /app
 ---> Using cache
 ---> 85e819a94585
Step 3/11 : COPY package.json .
 ---> Using cache
 ---> 1534e4c0a41e
Step 4/11 : RUN npm install
```

```
Step 9/11 : COPY --from=build /app/build .
 ---> Using cache
 ---> 2ade22b415cd
Step 10/11 : EXPOSE 80
 ---> Using cache
 ---> af2f1edd4d8a
Step 11/11 : CMD ["nginx", "-g", "daemon off;"]
 ---> Using cache
 ---> 3c9a28e30e4e
Successfully built 3c9a28e30e4e
Successfully tagged react-ci/cd:latest
832e3ade3bf4ddf686620befb9d01af96afa65f4fc7b17c1bf0470b55935d9fc
The push refers to repository [docker.io/ravivarman46/react-app]
bb2537bdb154: Layer already exists
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
ci-cd: digest: sha256:ded93dc858afb0de9455f0f22d918317f136f46a2eafb21c8dacbbd06582bade size: 2199
root@ip-172-31-19-184:/home/ubuntu/react-app-deployment-with-docker/react-app-code# docker ps
CONTAINER ID   IMAGE         COMMAND                  CREATED          STATUS          PORTS                                   NAMES
832e3ade3bf4   react-ci/cd   "/docker-entrypoint.…"   13 seconds ago   Up 12 seconds   0.0.0.0:80->80/tcp, :::80->80/tcp   react
root@ip-172-31-19-184:/home/ubuntu/react-app-deployment-with-docker/react-app-code#
```

Build.sh is working fine:

Step:6 - Creating a Jenkinsfile:

➔ Creating a Jenkins file:

Jenkinsfile contains:

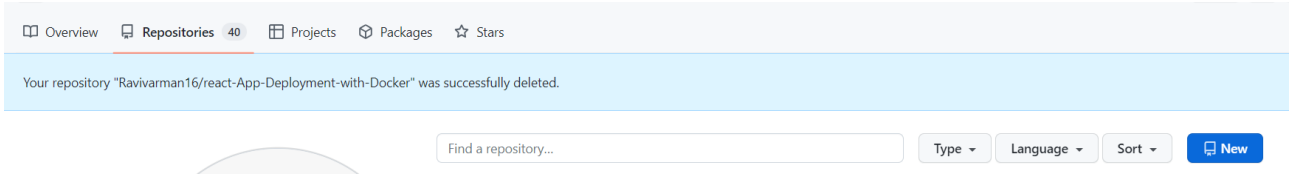
```
pipeline {
    agent any

    stages {
        stage ('changing the file permission') {
            steps {
                sh ' chmod +x build.sh'
            }
        }

        stage ('executing the file') {
            steps {
                sh './build.sh'
            }
        }
    }
}
```

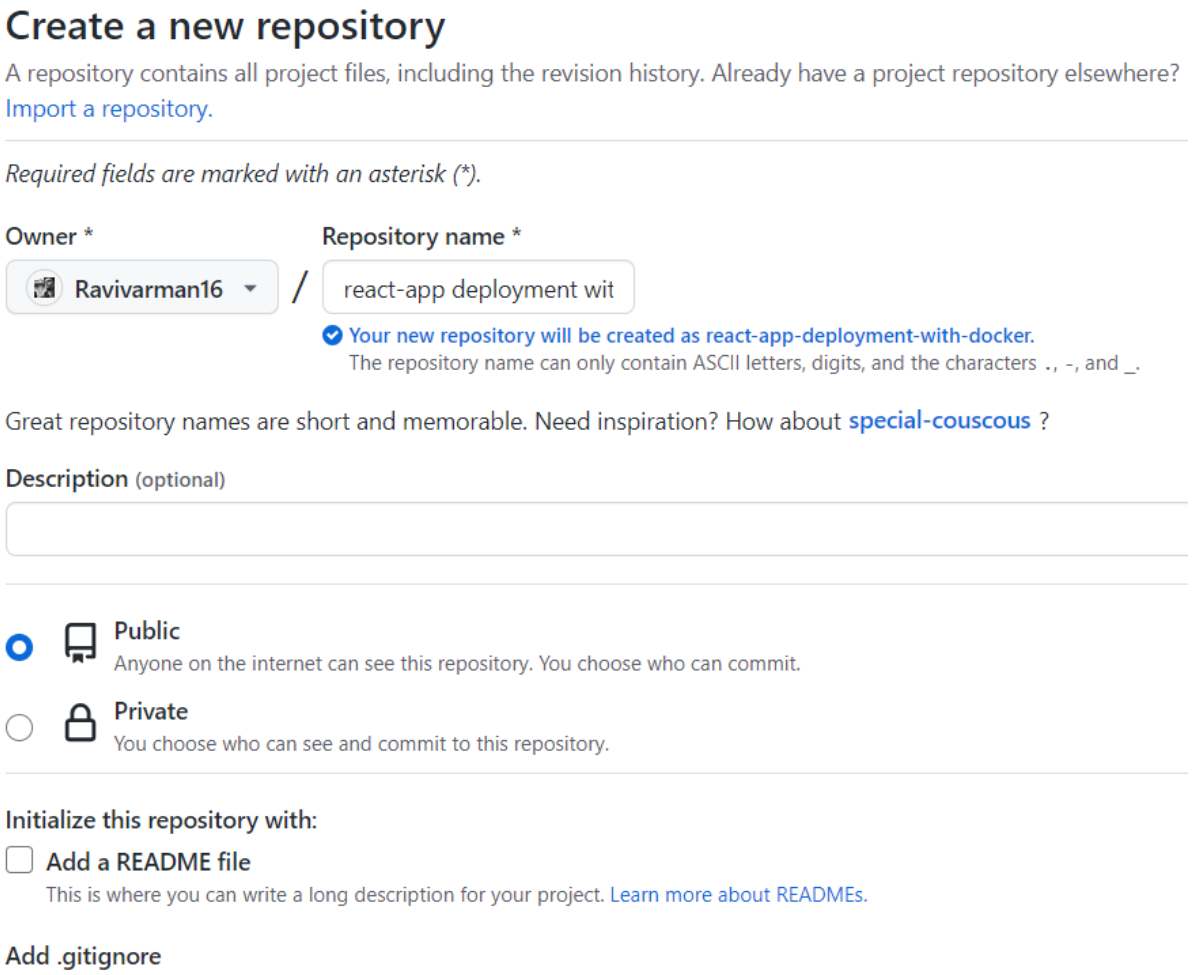
Step:7 - Creating a Github-repo & pushing the files on it:

➔ Just login into GitHub account:

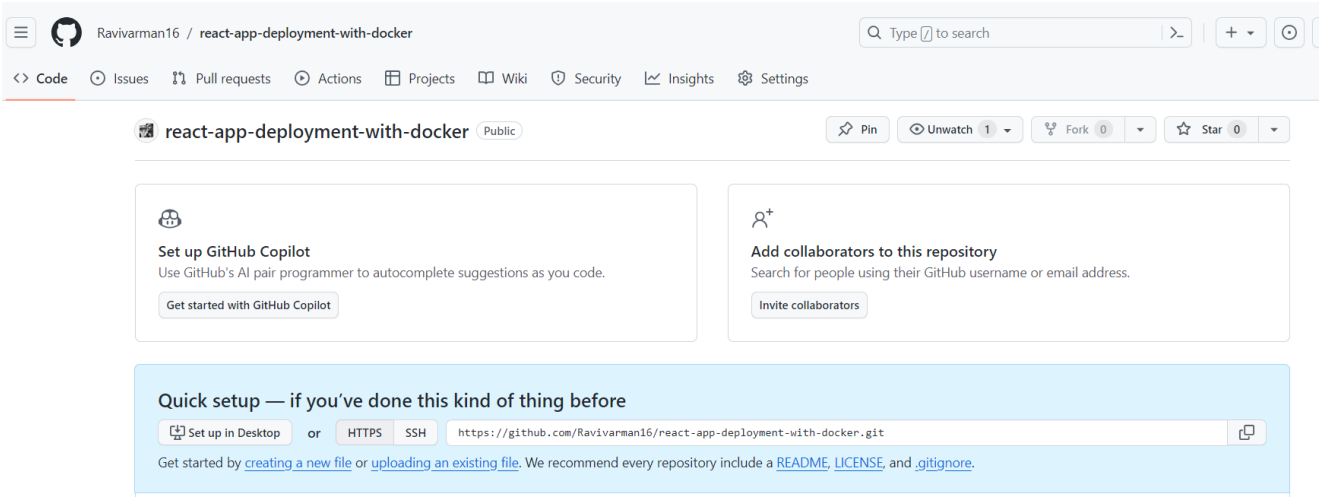


➔ Then just click new under repository section:

➔ Then create repository according to your preferences:



➔ The repository has been created successfully: now just copy the https URL and come back to command line:



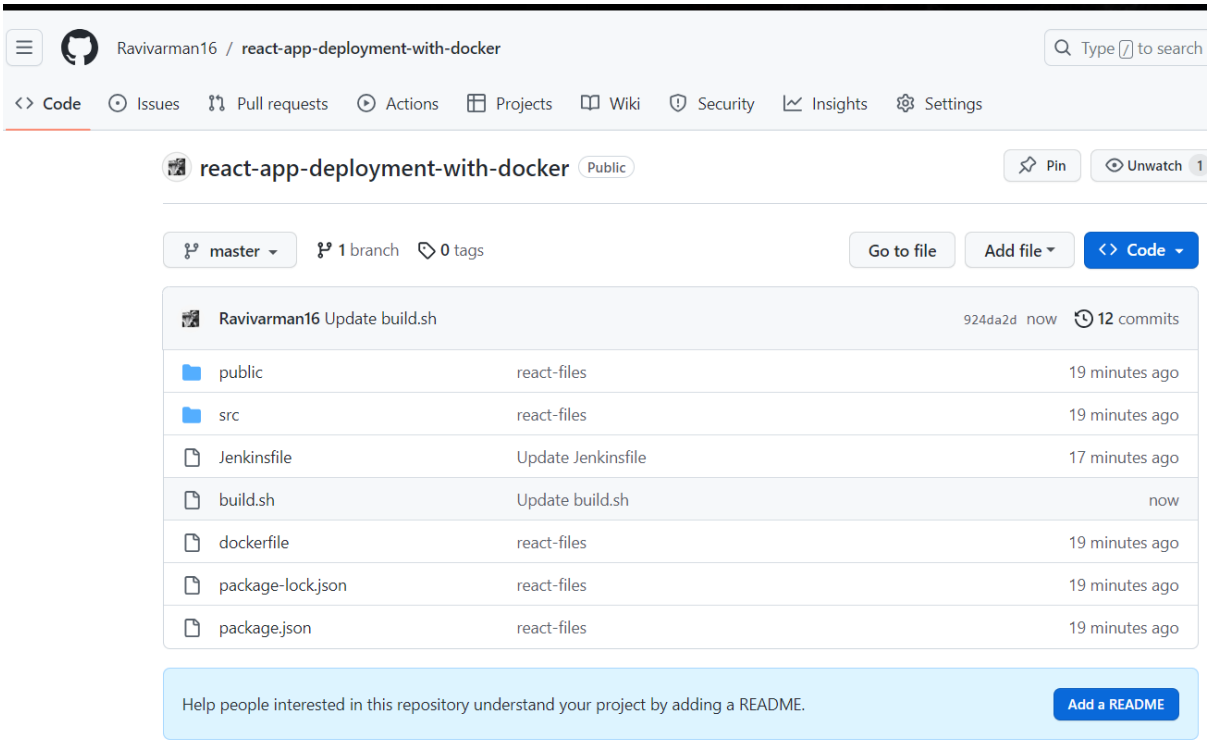
➔ Then clone the repository:

```
root@ip-172-31-19-184:/home/ubuntu/react-App-Deployment-with-Docker/react-code/route-app# cd ..
root@ip-172-31-19-184:/home/ubuntu/react-App-Deployment-with-Docker/react-code# git clone https://github.com/Ravivarman16/react-app-deployment-with-docker.git
Cloning into 'react-app-deployment-with-docker'...
warning: You appear to have cloned an empty repository.
root@ip-172-31-19-184:/home/ubuntu/react-App-Deployment-with-Docker/react-code# ls
react-app-deployment-with-docker  route-app
root@ip-172-31-19-184:/home/ubuntu/react-App-Deployment-with-Docker/react-code#
```

➔ stage it, commit it and push it to the remote repository:

```
root@ip-172-31-19-184:/home/ubuntu/react-App-Deployment-with-Docker/react-code/react-app-deployment-with-docker# git push origin main
Username for 'https://github.com': Ravivarman16
Password for 'https://Ravivarman16@github.com':
Enumerating objects: 24, done.
Counting objects: 100% (24/24), done.
Delta compression using up to 2 threads
Compressing objects: 100% (23/23), done.
Writing objects: 100% (24/24), 241.70 KiB | 6.91 MiB/s, done.
Total 24 (delta 4), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (4/4), done.
To https://github.com/Ravivarman16/react-app-deployment-with-docker.git
 * [new branch]      main -> main
root@ip-172-31-19-184:/home/ubuntu/react-App-Deployment-with-Docker/react-code/react-app-deployment-with-docker#
```

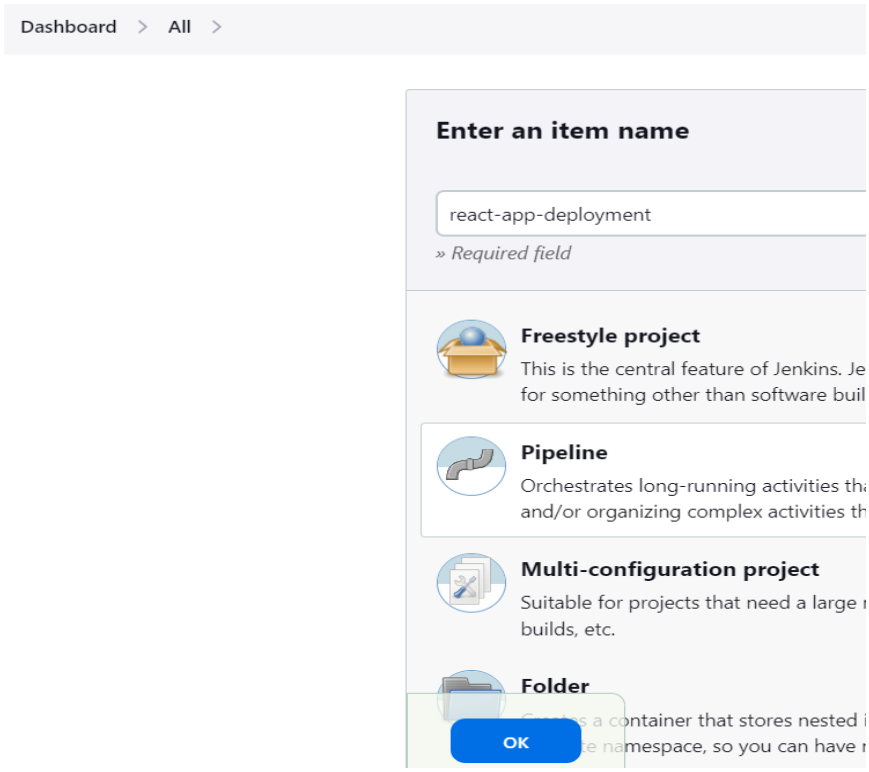
Checking the remote repository:



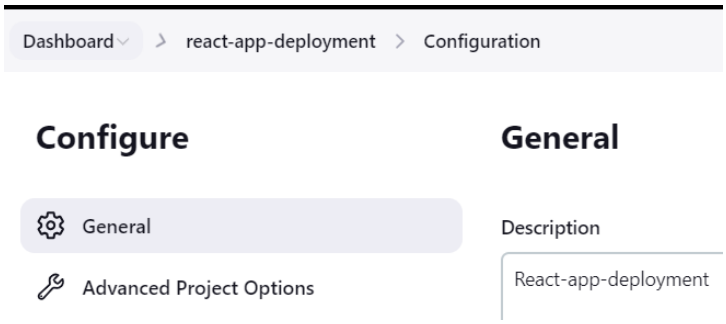
Step:8 - Setting up a CI/CD pipeline:

➔ On Jenkins dashboard, we can able to find out new item, click that one:

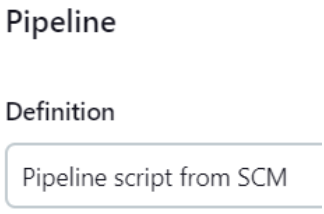
➔ Then name the job and select the job type as pipeline: click okay:



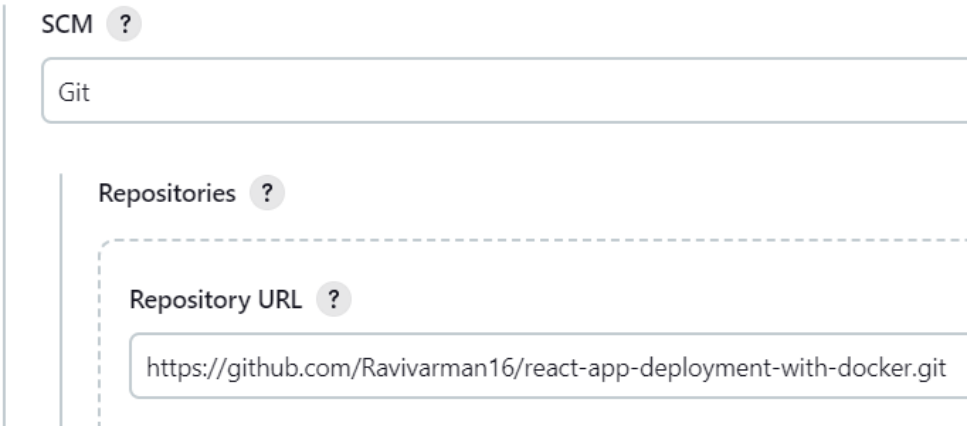
➔ Then under description give according to this task:



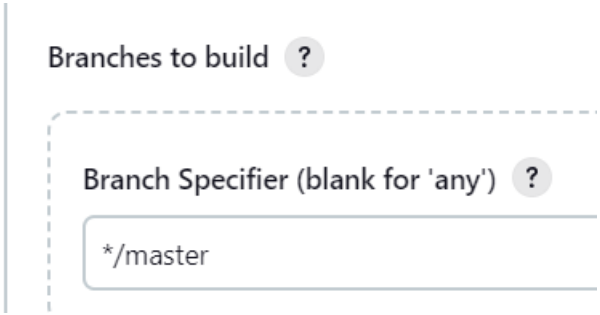
➔ Then under pipeline select **pipeline script from SCM:**



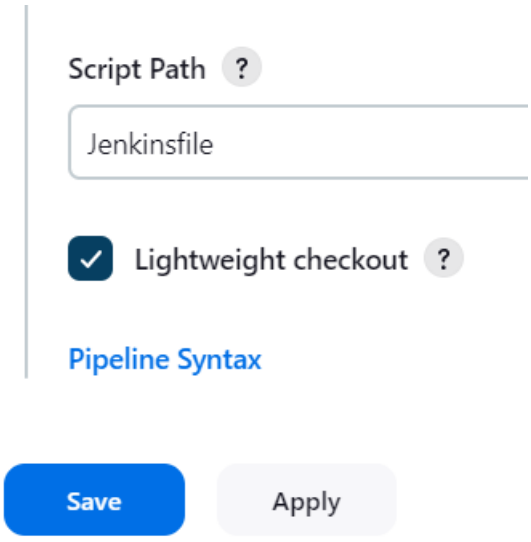
➔ Then under SCM select Git, enter the GitHub URL:



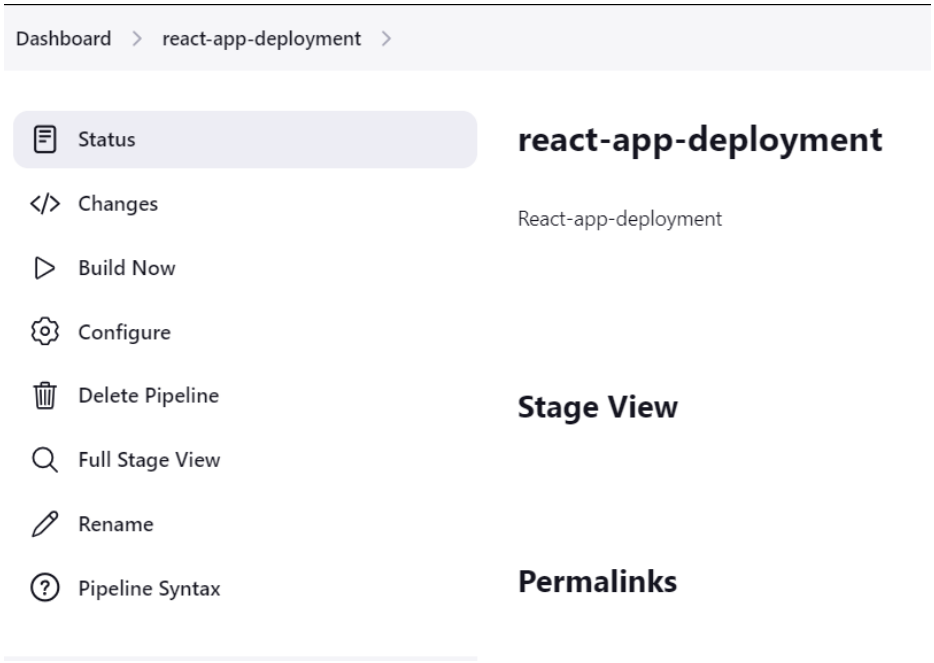
➔ Then select the branch:



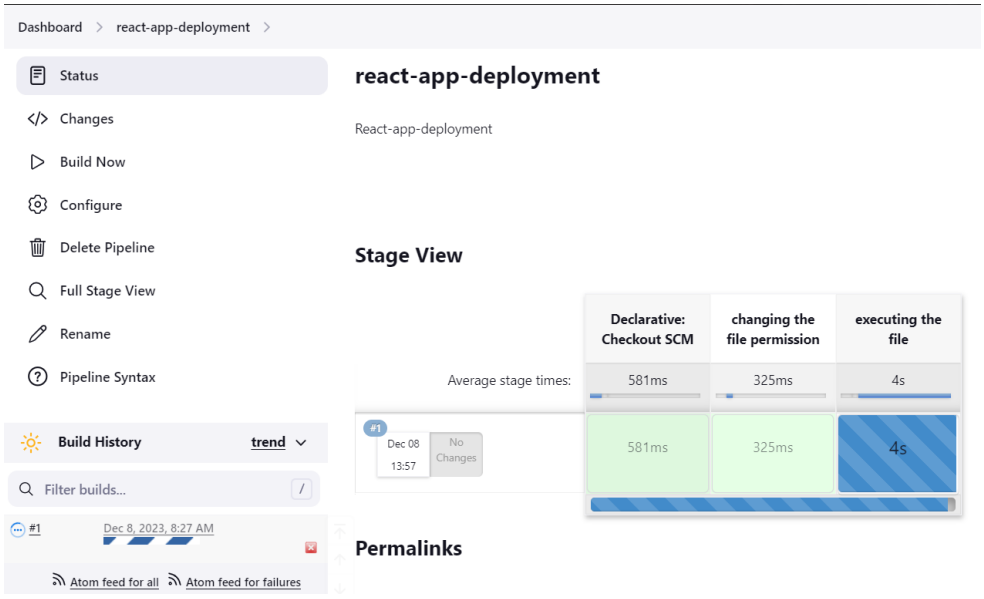
➔ Then enter the Jenkins file name: click apply and save:



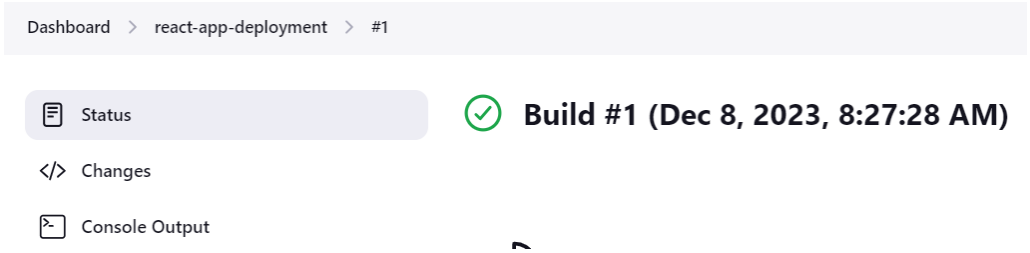
➔ Then we can able to see pipeline job has been created successfully:



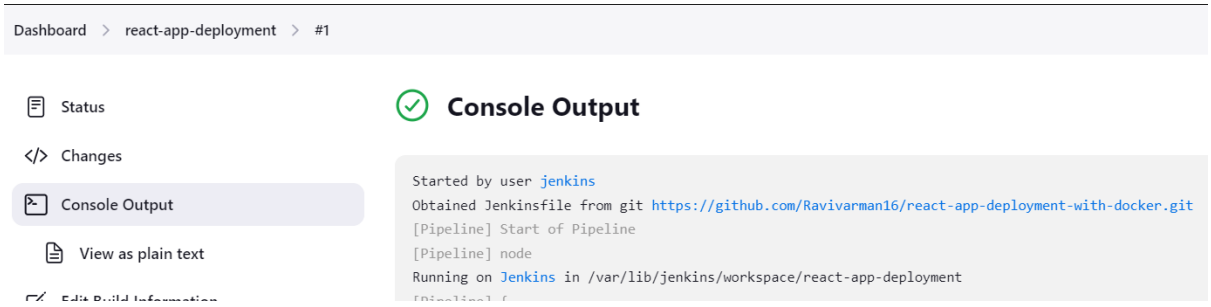
➔ Then click the build now option: pipeline job is executing fine:



➔ We can able to see pipeline job executed successfully, click the job to know more details:

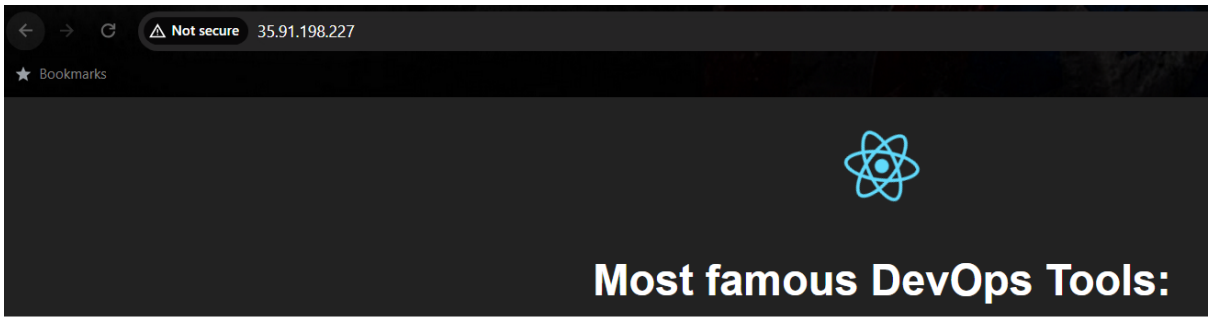


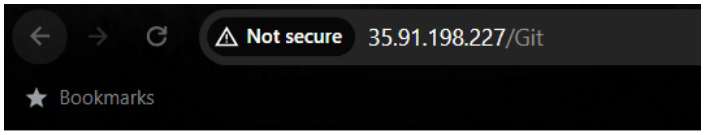
➔ Then click console output option: to know details working of this pipeline:



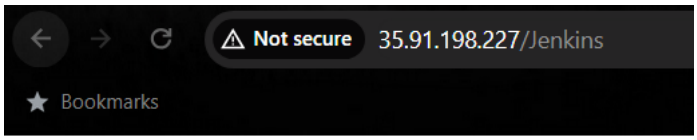

```
a34b395c0ca3: Layer already exists
b968c967e155: Layer already exists
28c7b3c0b176: Layer already exists
dd731ddf52be: Layer already exists
fbed1f6990ee: Layer already exists
9fe9a137fd00: Layer already exists
ci-cd: digest: sha256:ded93dc858afb0de9455f0f22d918317f136f46a2eafb21c8dacbbd06582bade size: 2199
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

➔ Browser output of the container:

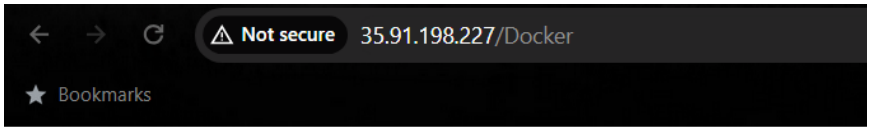




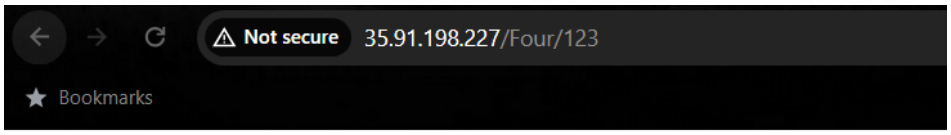
I am a Version Control Tool



I am a CI/CD Tool



I am a Containerization based Tool!

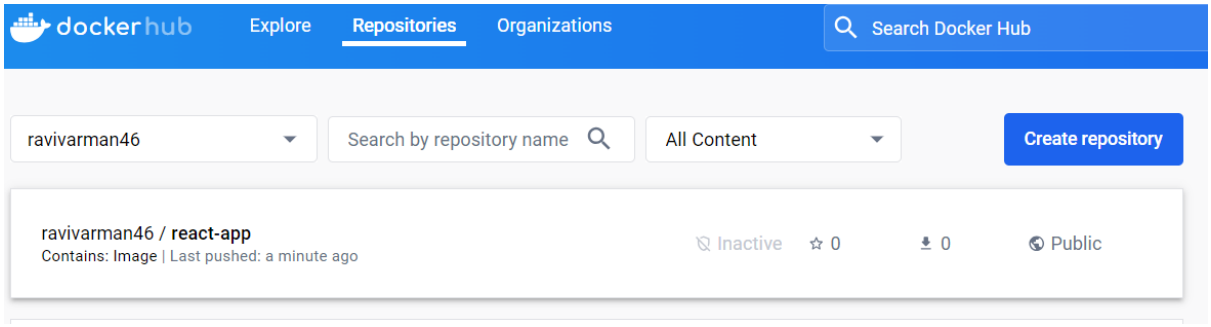


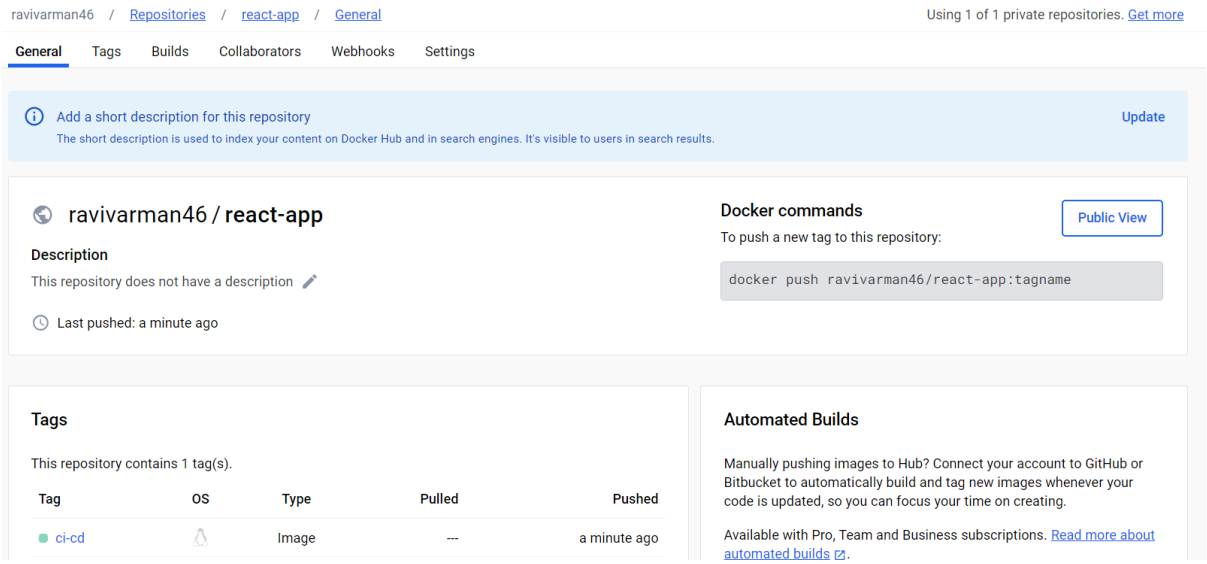
I am a Container Orchestration Tool!

Click Here!

Thank You Have a Nice Day!!!

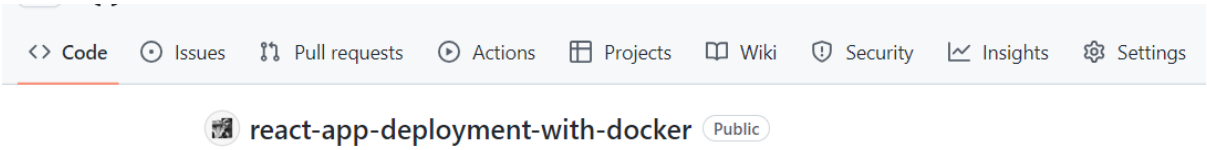
➔ Docker hub output:



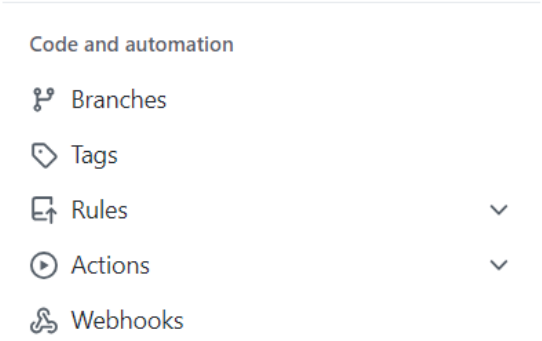


Step:9 - Making the pipeline automated:

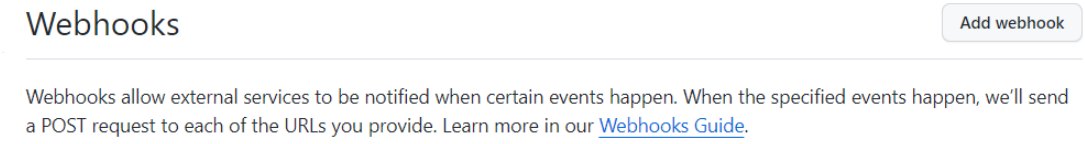
➔ On the GitHub repository, we can able to see settings, click that one:



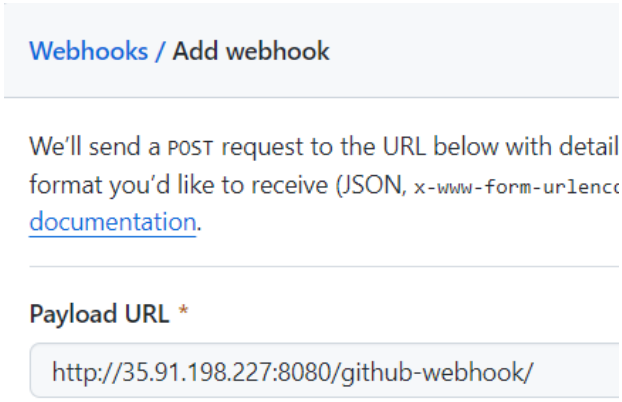
➔ Then on the left side, we can able to see webhooks, click that one:



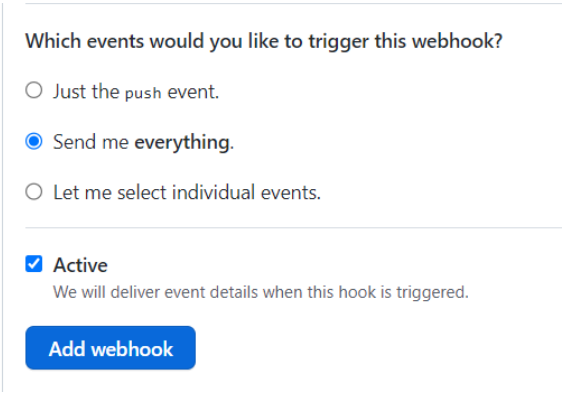
➔ Then we can able to see add webhook option, click that one:



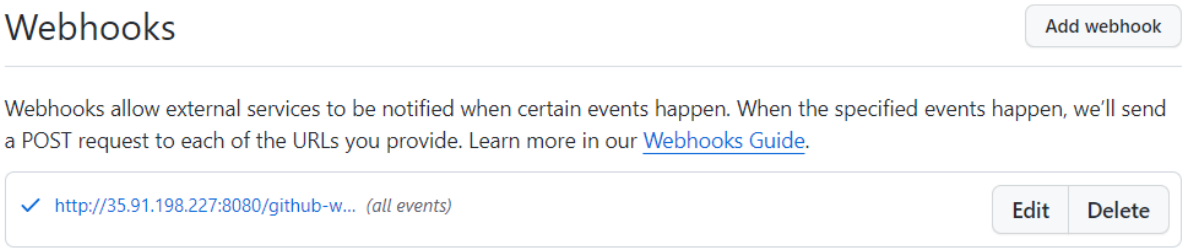
➔ Then under payload URL, enter the Jenkins URL along with github-webhook:



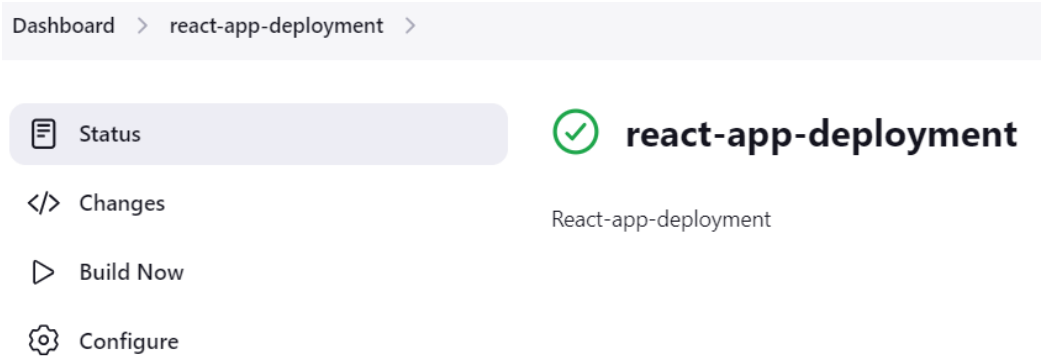
➔ Then select the events to trigger from GitHub: click add webhook:



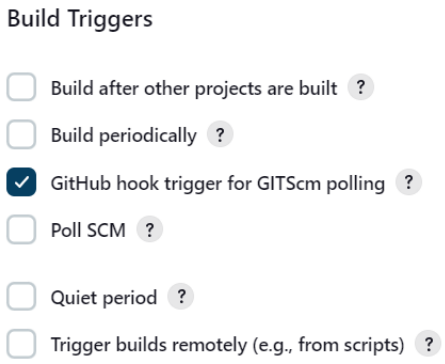
➔ Then we can see webhook has been created successfully:

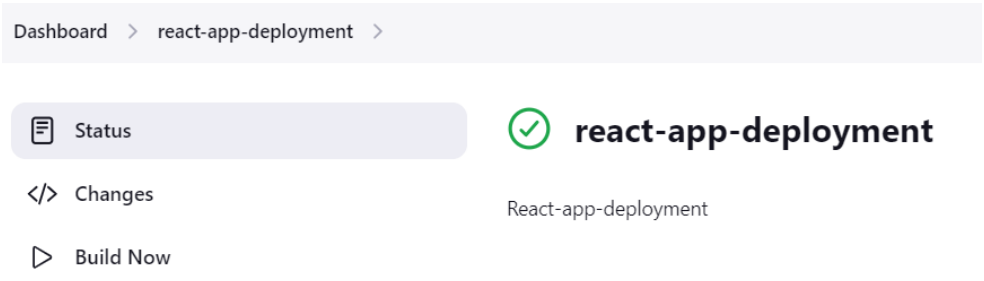


➔ Then on Jenkins pipeline job, we need make one change in settings, for that click configure:



➔ Under build triggers enable: GitHub hook trigger for GITScm polling option. Save it



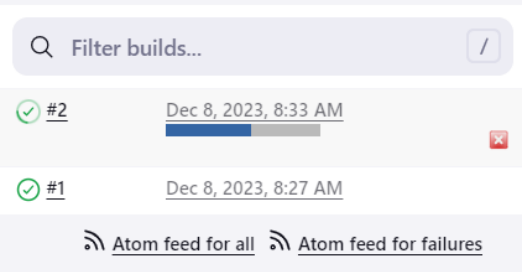


➔ Then on the command line, make small change, here I am creating a file, pushing it to remote repository:

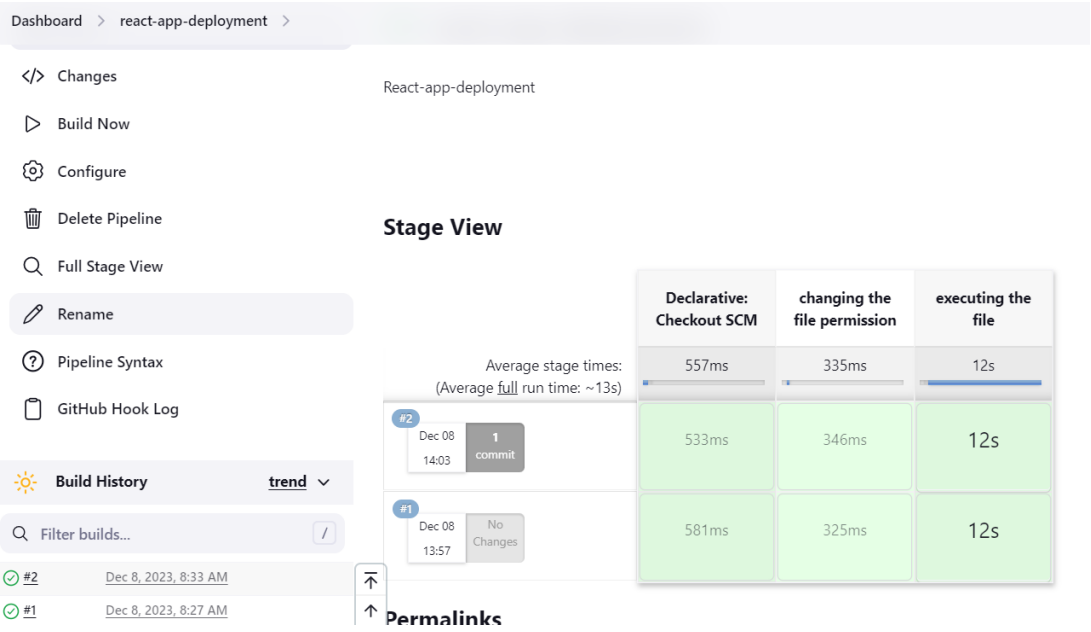
```
root@ip-172-31-19-184:/home/ubuntu/react-app-deployment-with-docker# ls
Jenkinsfile  build.sh  dockerfile  package-lock.json  package.json  public  src
root@ip-172-31-19-184:/home/ubuntu/react-app-deployment-with-docker# touch demo
root@ip-172-31-19-184:/home/ubuntu/react-app-deployment-with-docker# git add demo
root@ip-172-31-19-184:/home/ubuntu/react-app-deployment-with-docker# git commit -m "for hooktrigger test"
[master 47fe6e6] for hooktrigger test
 Committer: root <root@ip-172-31-19-184.us-west-2.compute.internal>
```

```
root@ip-172-31-19-184:/home/ubuntu/react-app-deployment-with-docker# git push origin master
Username for 'https://github.com': Ravivarman16
Password for 'https://Ravivarman16@github.com':
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 2 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 278 bytes | 278.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/Ravivarman16/react-app-deployment-with-docker.git
   924da2d..47fe6e6  master -> master
root@ip-172-31-19-184:/home/ubuntu/react-app-deployment-with-docker#
```

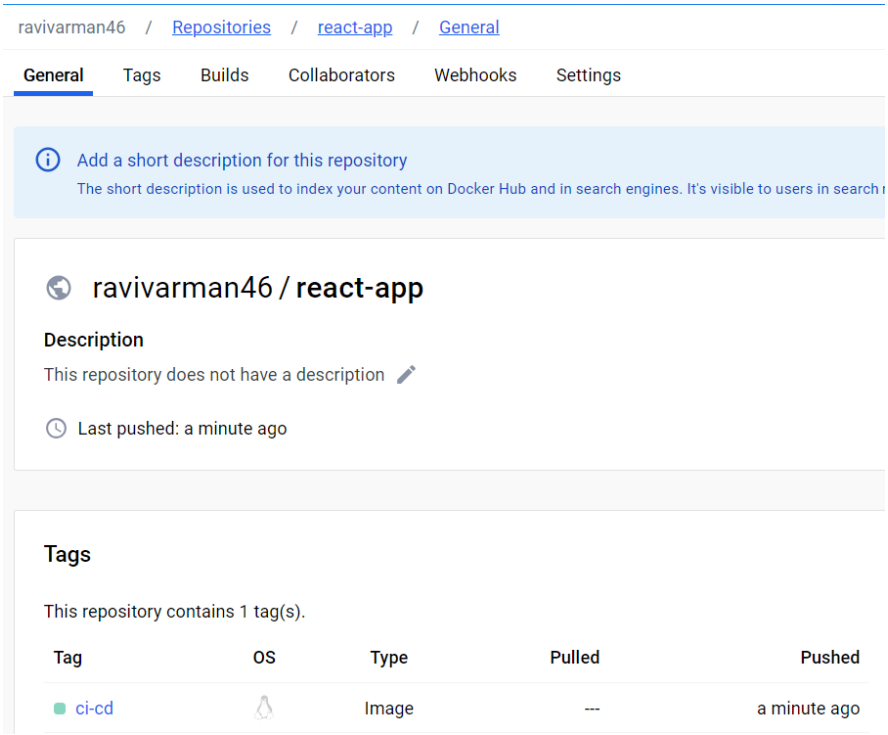
➔ After pushing it to the GitHub we can see in Jenkins job got triggered automatically:



➔ Job executed successfully:



Docker Hub output:



We can able to see the image got pushed into the Docker hub just now.

Benefits of above task:

➔ **Consistent Environments:** Docker ensures consistent development and deployment environments, minimizing "it works on my machine" issues and providing a reliable setup for developers across different stages.

➔ **Efficient Resource Utilization:** The multi-stage Docker build allows for a smaller final image, reducing the container's size and optimizing resource utilization. This results in faster deployment times and more efficient use of system resources.

➔ **Automated Deployment Pipeline:** The Jenkins pipeline automates the build, testing, and deployment processes, improving efficiency and reducing manual errors. This streamlined automation enhances the reliability and speed of delivering updates to production.

All the files for the above task have been uploaded under this GitHub repository: https://github.com/Ravivarman16/react-app-deployment-with-docker.git