**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.

------------------------------------------------------------------------------------------------

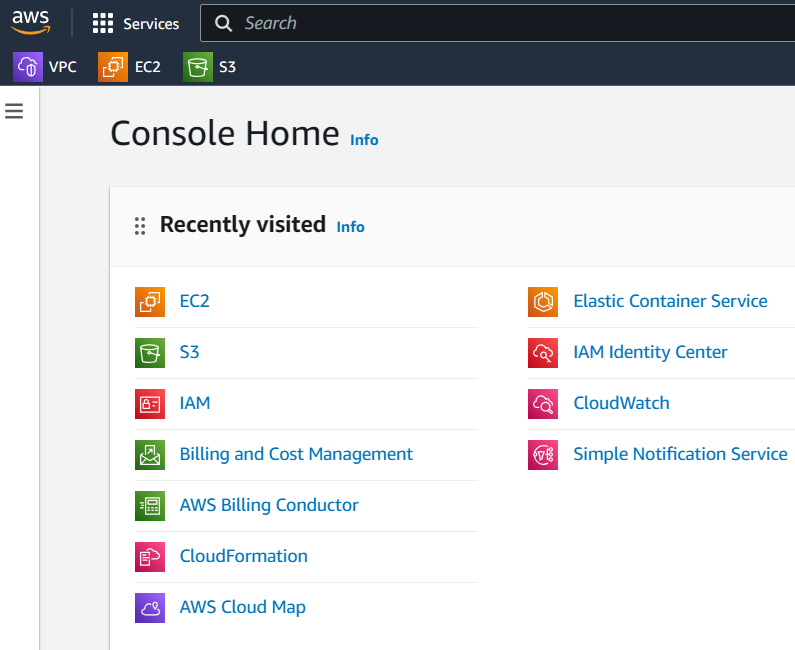
**SOLUTION:**

**REQUIREMENTS:**

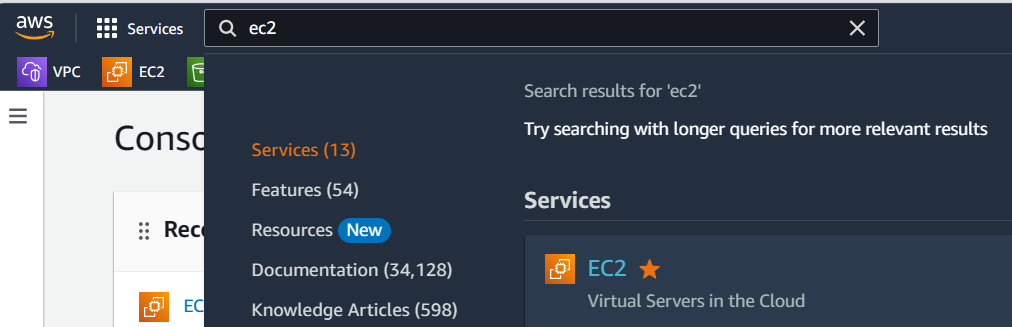
* AWS Cloud
* AWS EC2 instance
* Git & GitHub
* Docker
* Java
* Jenkins

**Step:1 – Launching an EC2 instance:**

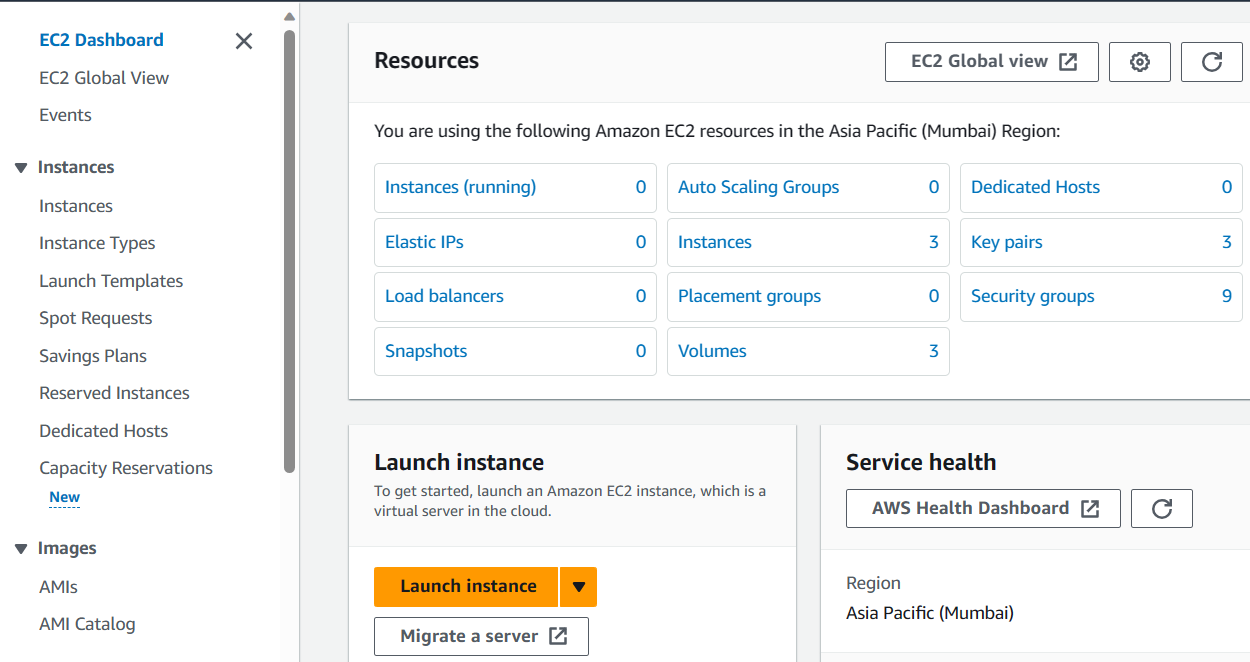
* First login into your AWS instance:



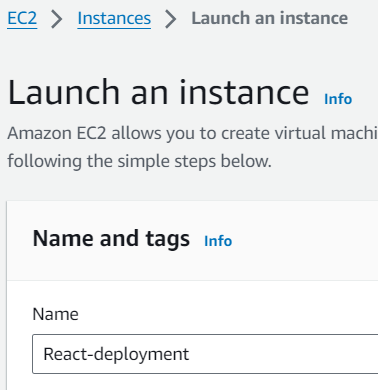
* Then on service search panel search EC2, click that one:



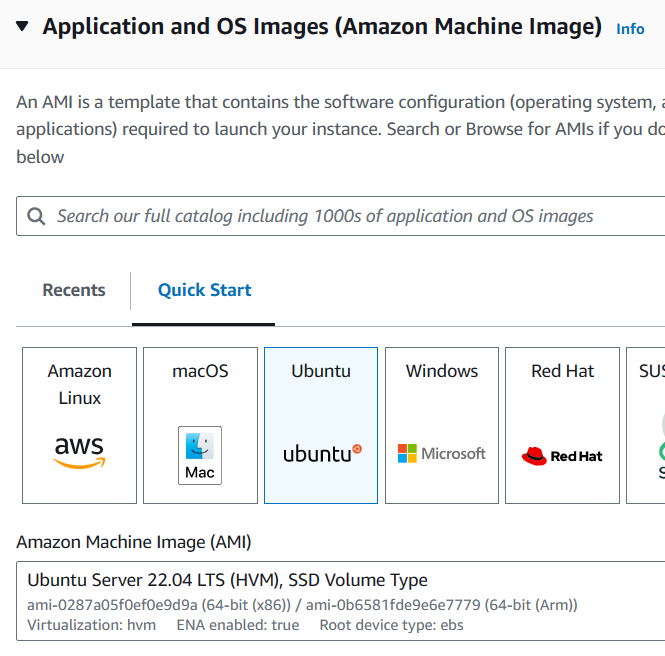
* Then click launch instances, for creating an EC2 instance:



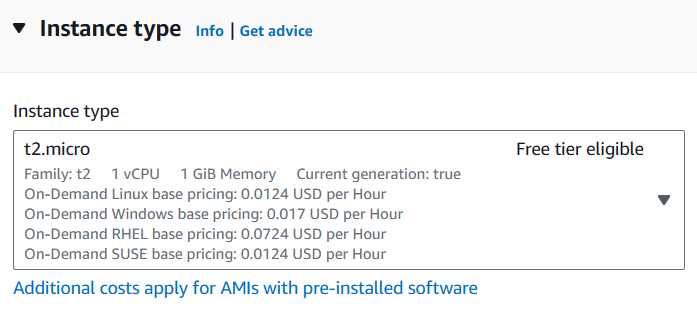
* Then name the instance according to your preferences:



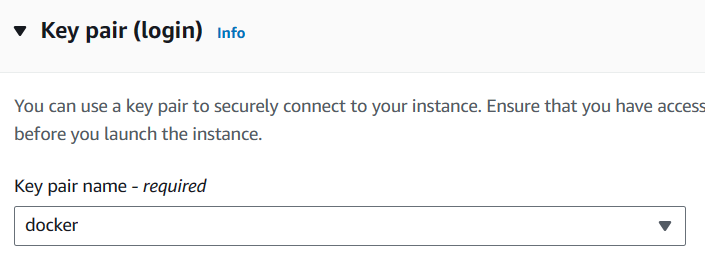
* Then select the operating system according to your preferences:



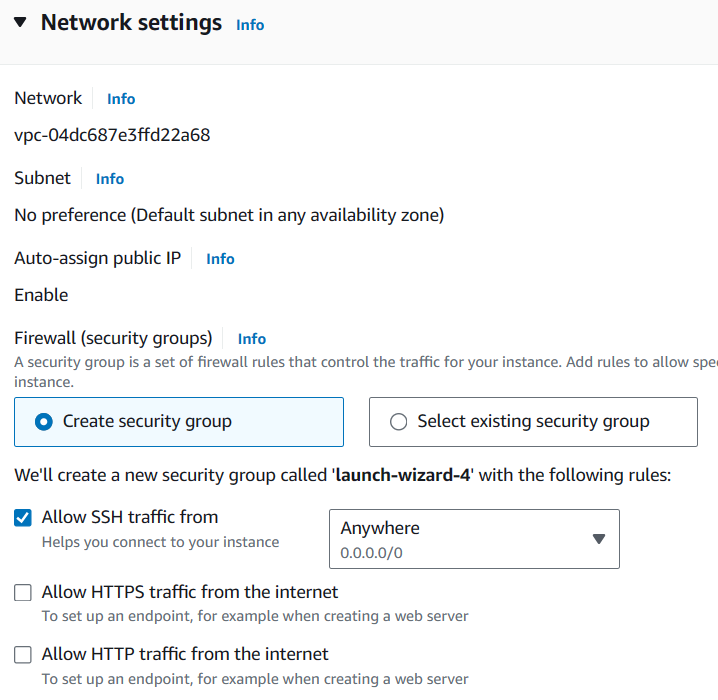
* Then select the instance type: according to your preferences, but here I am selecting **t2.micro**

****

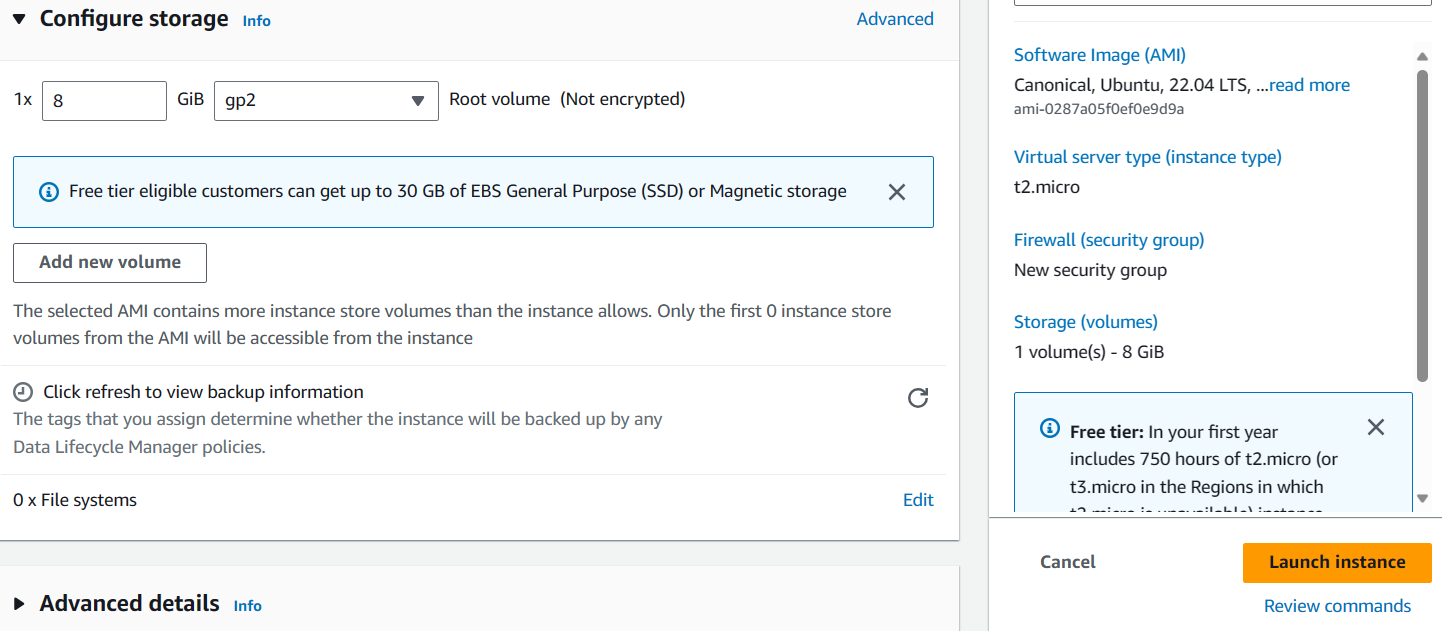
* Then select the key pair, according to your preferences, but here I am **proceeding with key pair option**, you can go with proceed with **without key pair option:**



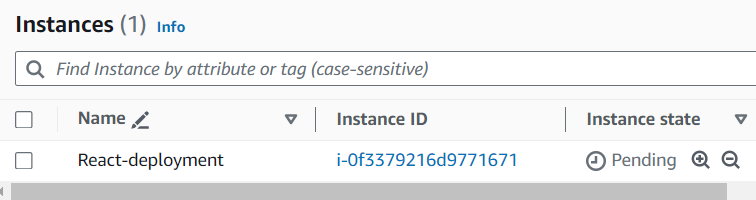
* Then keeping the default options under network settings:



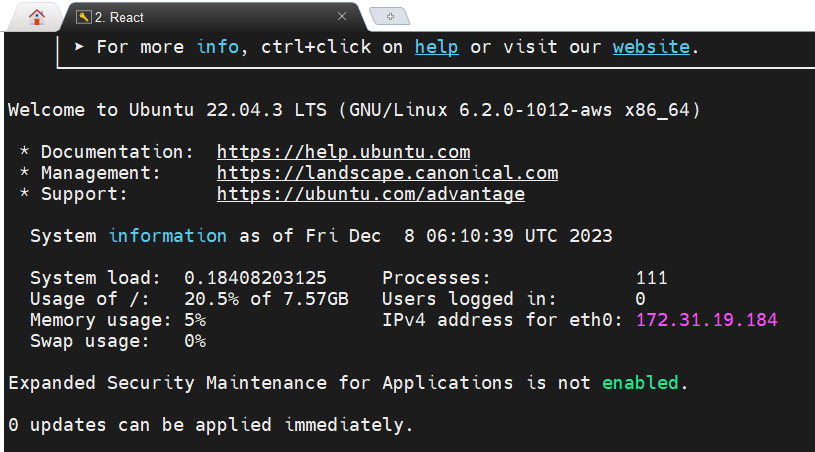
* Then keeping default options for the rest of the settings, click launch instance:



* 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

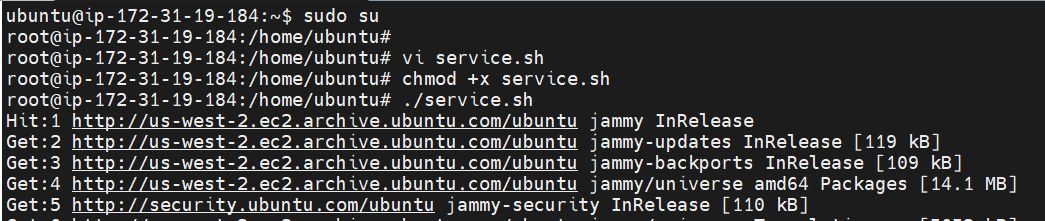
echo "This is Jenkins package - "

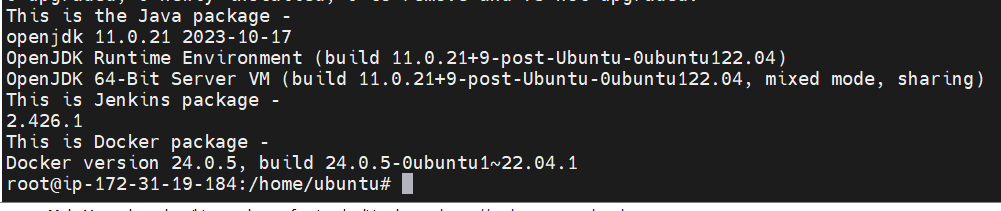
jenkins --version

echo "This is Docker package - "

docker --version

* Changing the file permission and executing it:



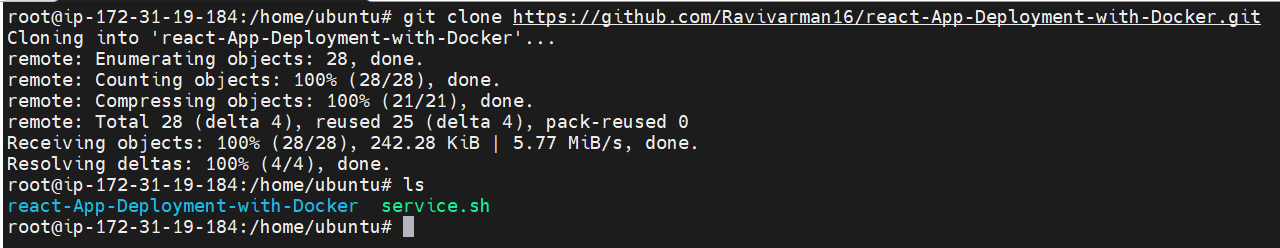


All the packages have been installed successfully:

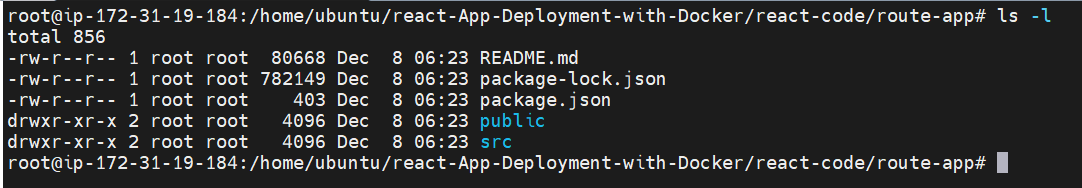
------------------------------------------------------------------------------------------------

**Step:3 – Dockerization of React application:**

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

****

* Then going inside the cloned directory:



* 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:

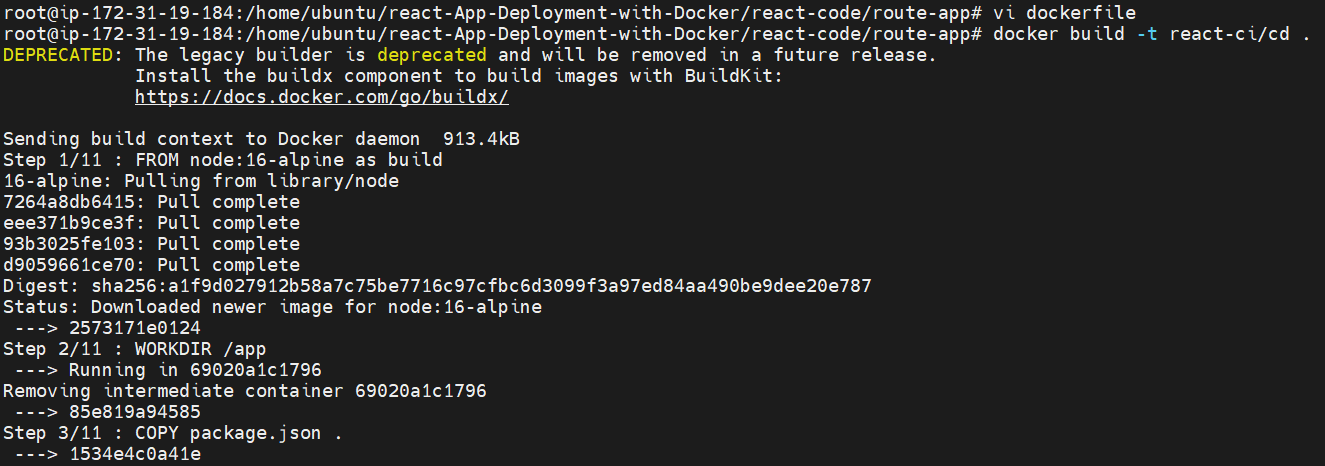
EXPOSE 80

#Executing the application after creating image:

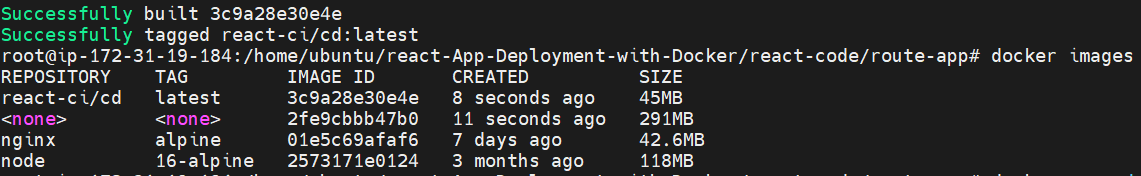
CMD ["nginx", "-g", "daemon off;"]

* Building a docker image from the dockerfile:

**s**

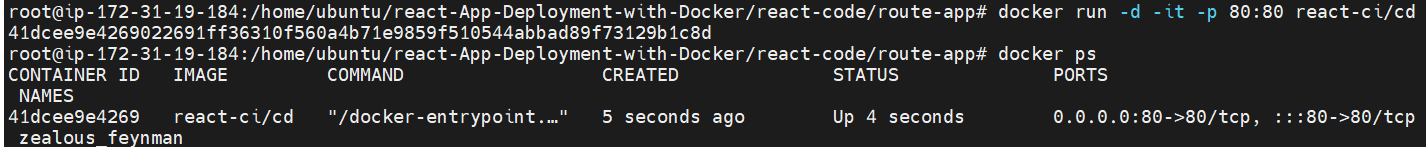


* Checking whether the image is created or not by using **docker images** command:



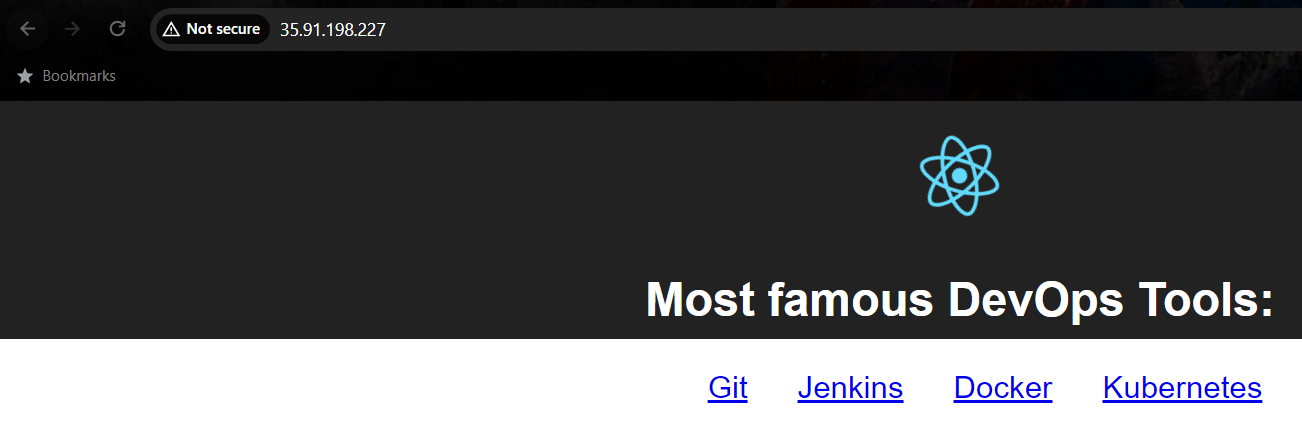
* Checking the output of the docker image by running a container from the above image by using **docker run** command:

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



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

**Browser output:**

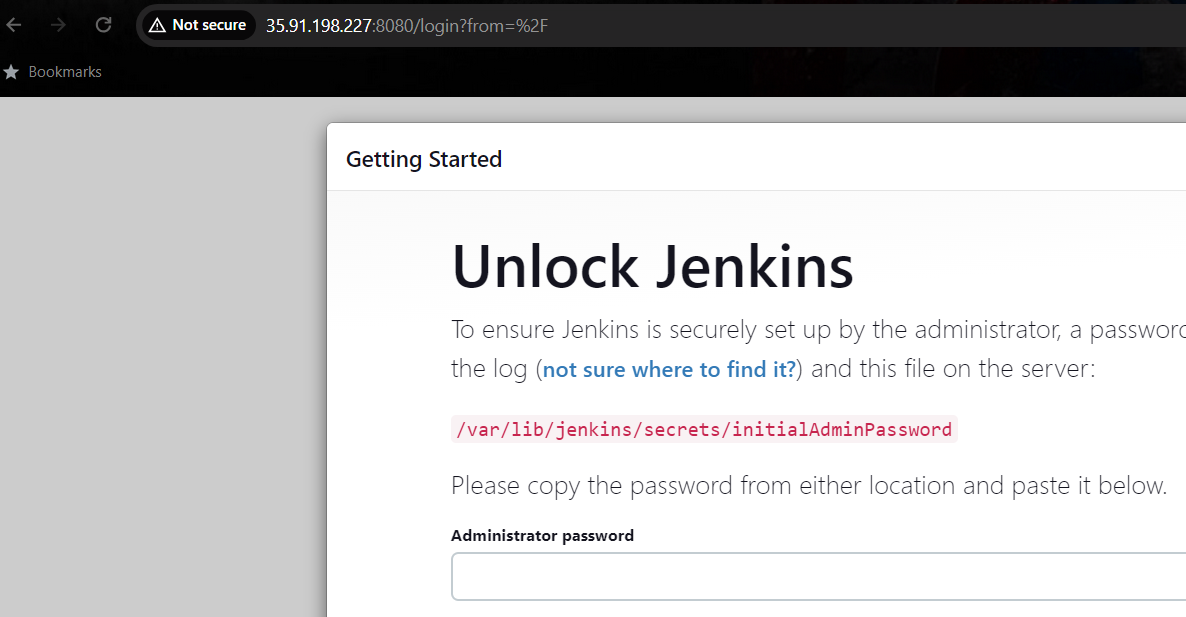


The docker image is working fine:

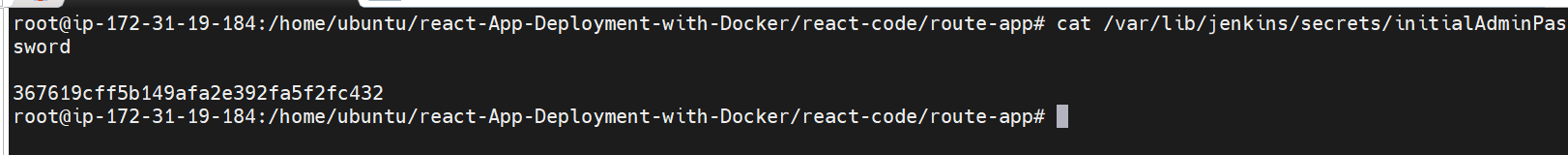
------------------------------------------------------------------------------------------------

**Step:4 - Setting up the Jenkins dashboard:**

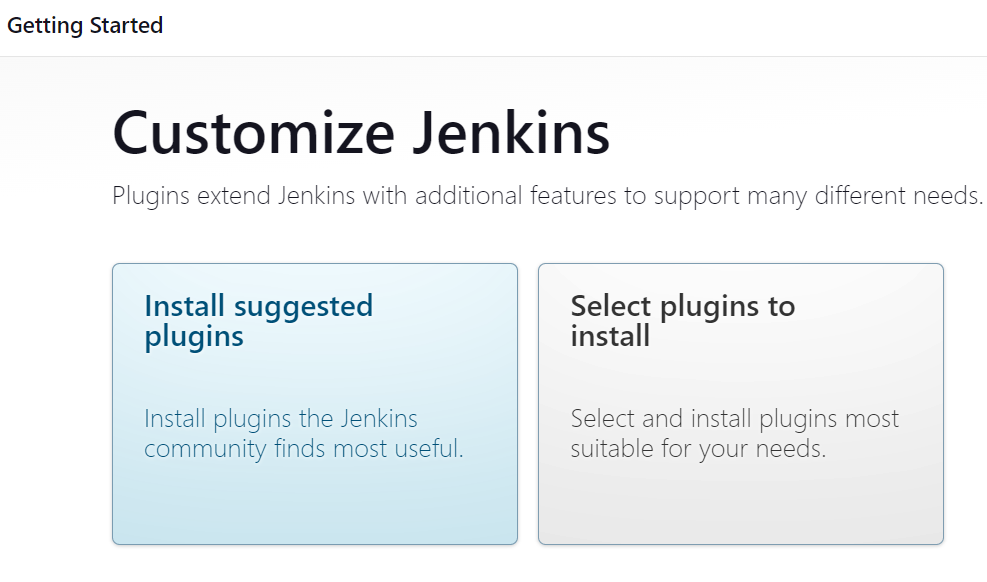
* Enabling the port number 8080 on the security group and pasting the public ip address along with the port number on the browser:

****

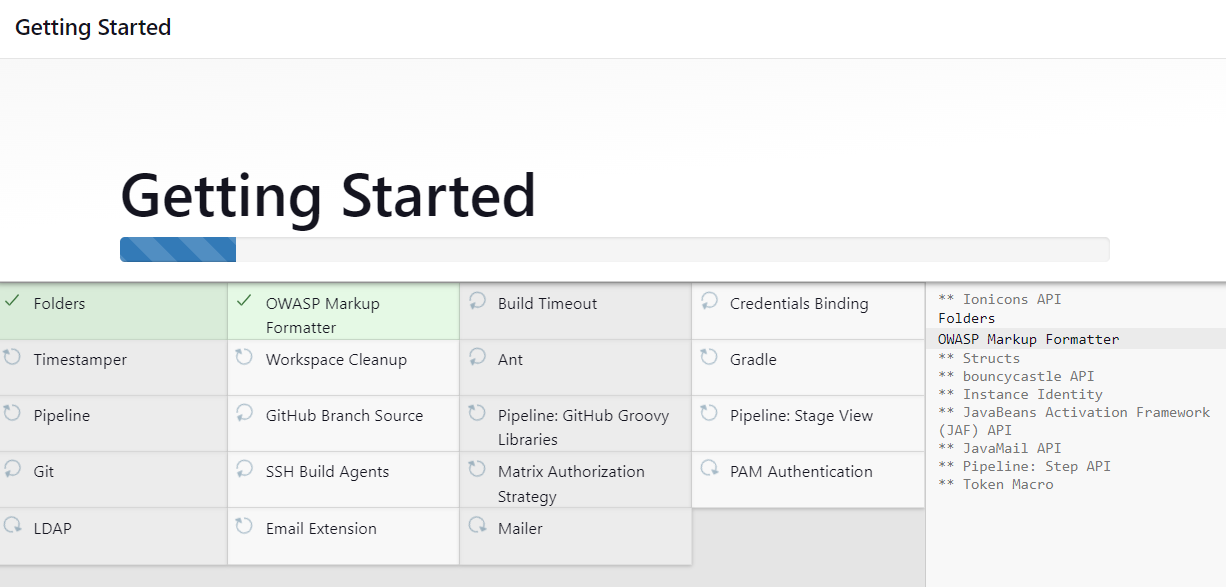
* We need to get the initial admin password by pasting the path on the command line: paste the password on the browser and proceed the next steps:

****

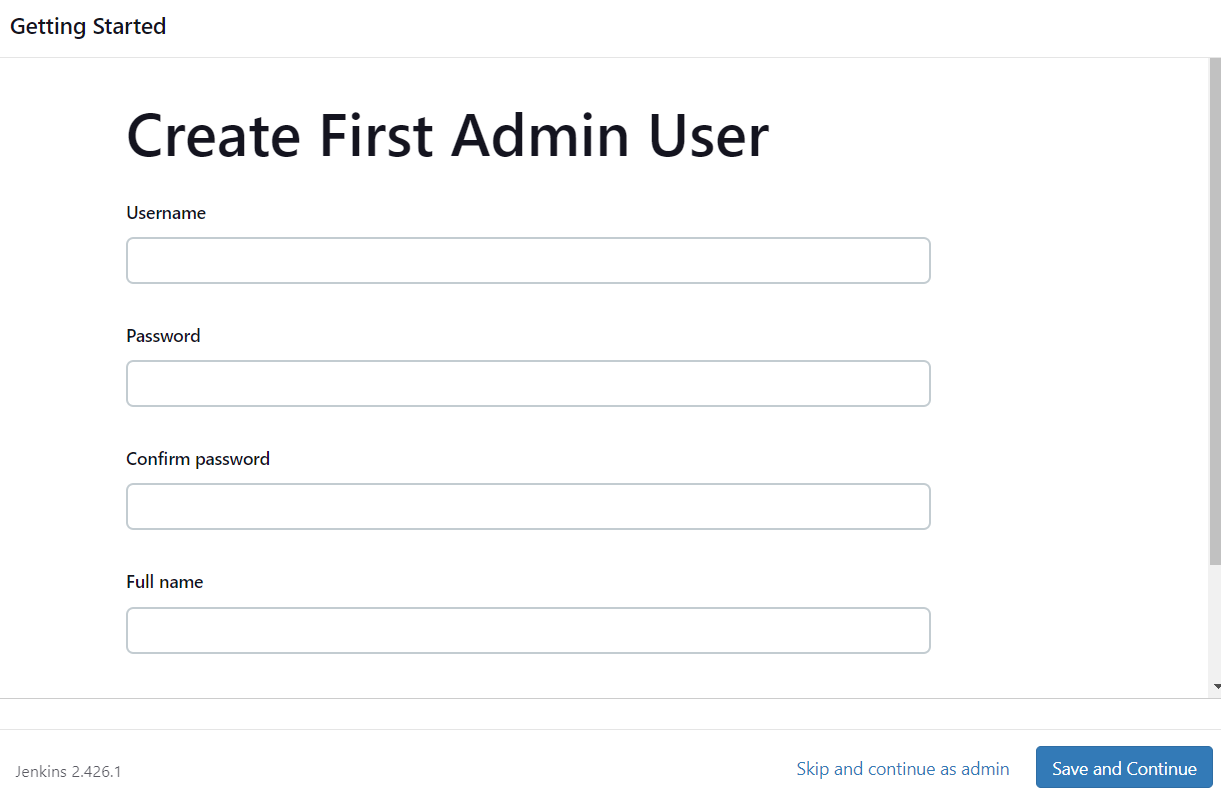
* Then click the install suggested plugins options:

****

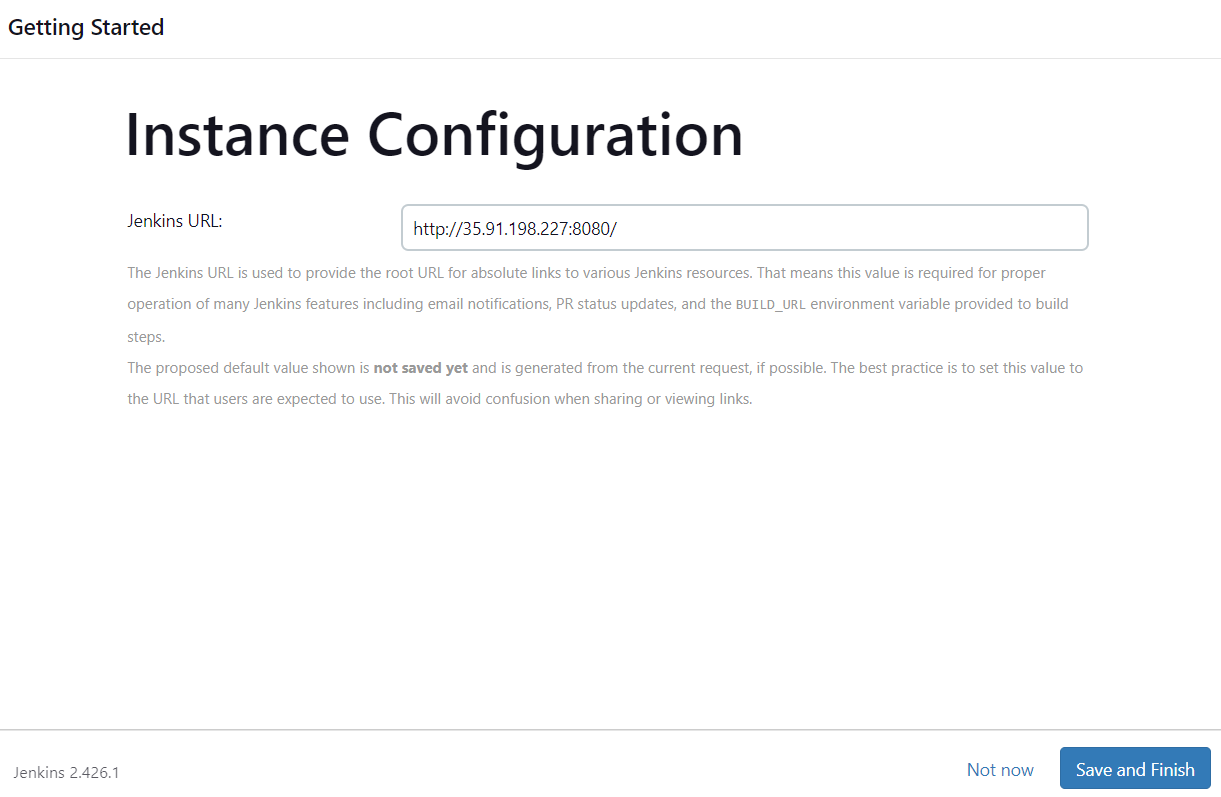
* Plugins will start to install:



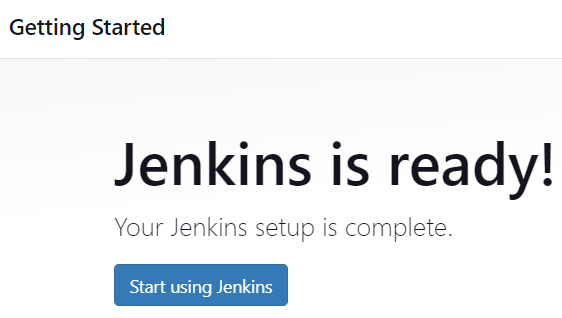
* Then we have to setup the credentials for Jenkins login purpose: click next:



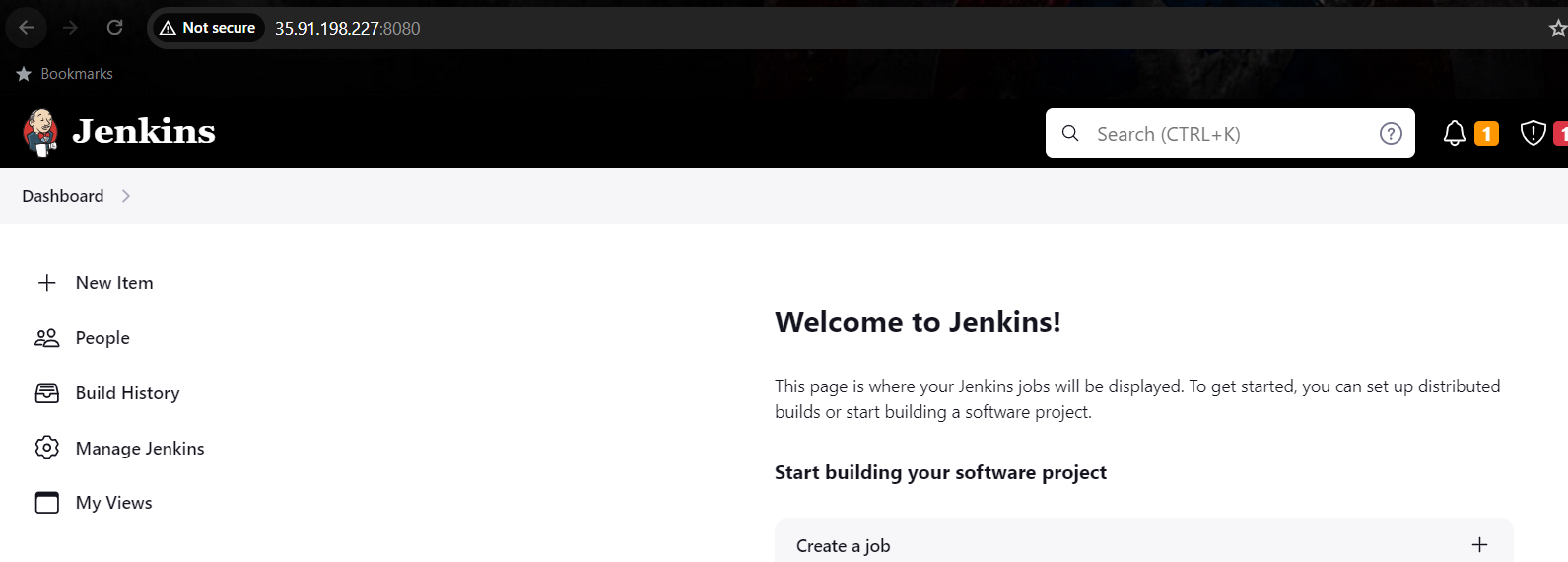
* Then click save and finish:



* 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:

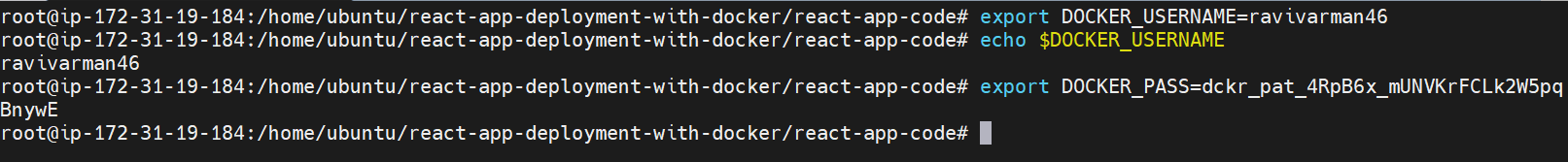
docker run -d -it --name react -p 80:80 react-ci/cd

#pushing the image to dockerhub:

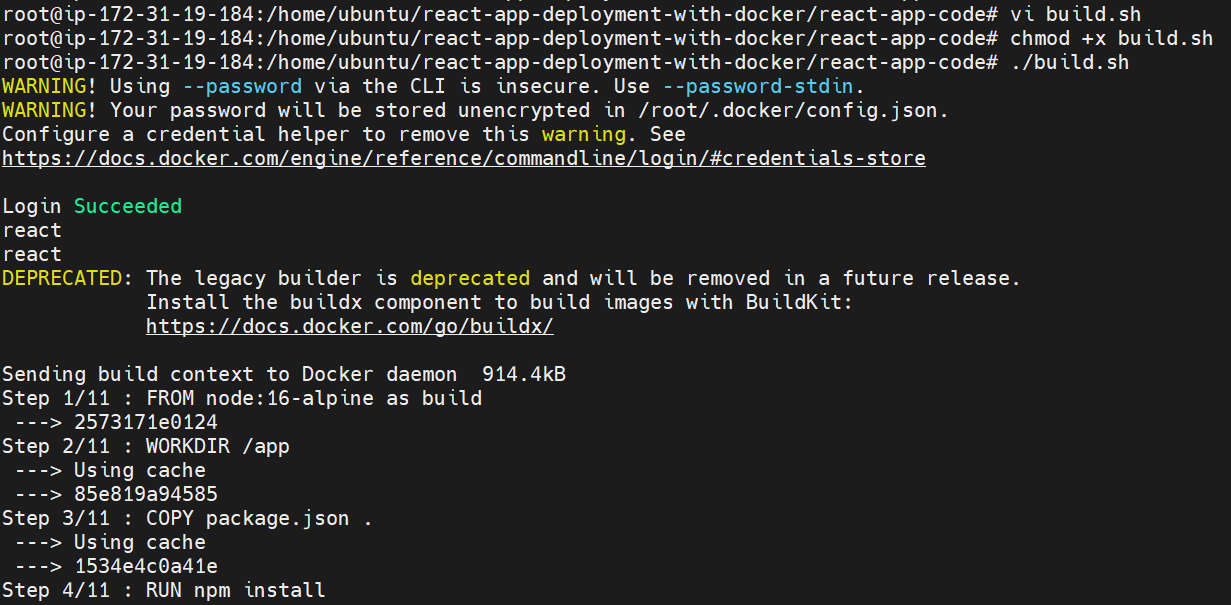
docker tag react-ci/cd ravivarman46/react-app:ci-cd

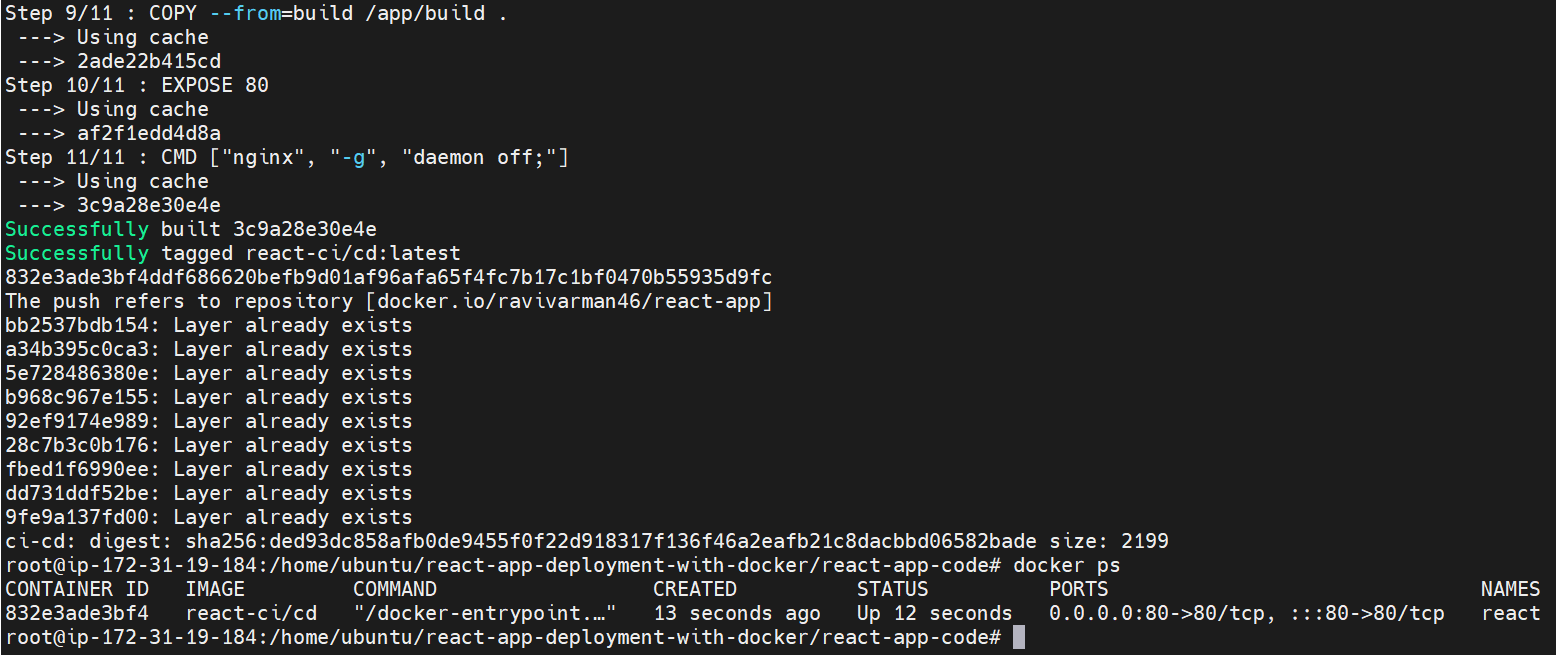
docker push ravivarman46/react-app:ci-cd

* Setting up Docker hub credentials environment variables:



* Changing the file permission and executing it:





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'

            }

        }

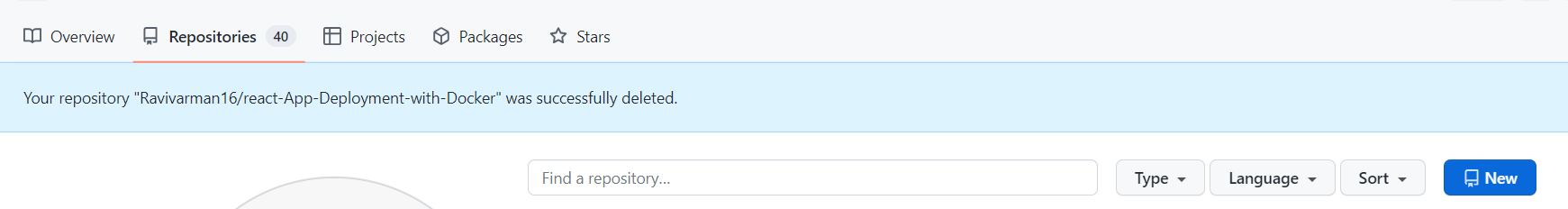
    }

}

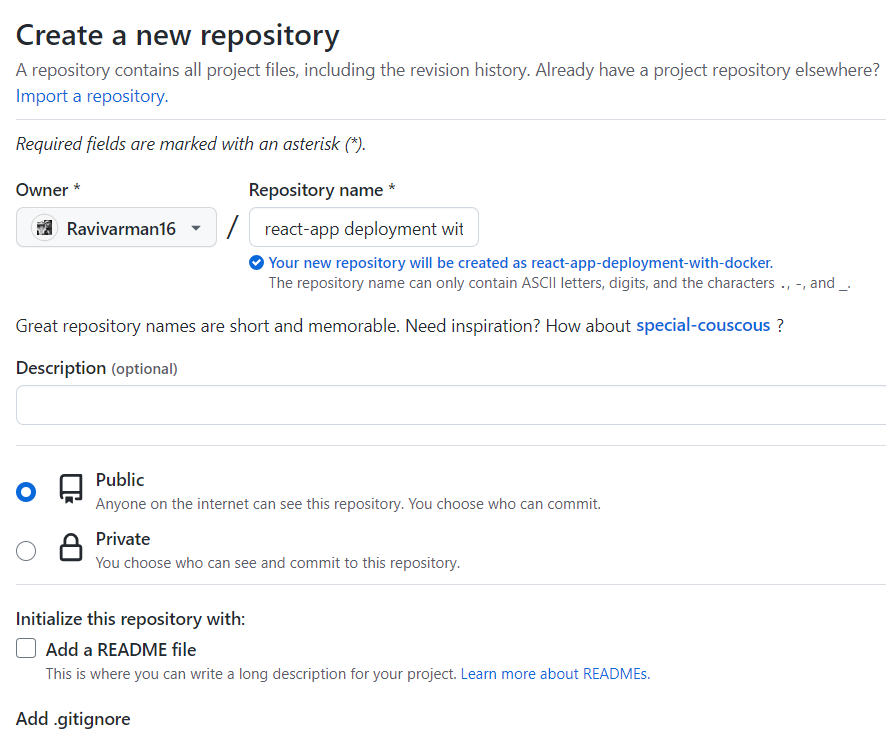
------------------------------------------------------------------------------------------------

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

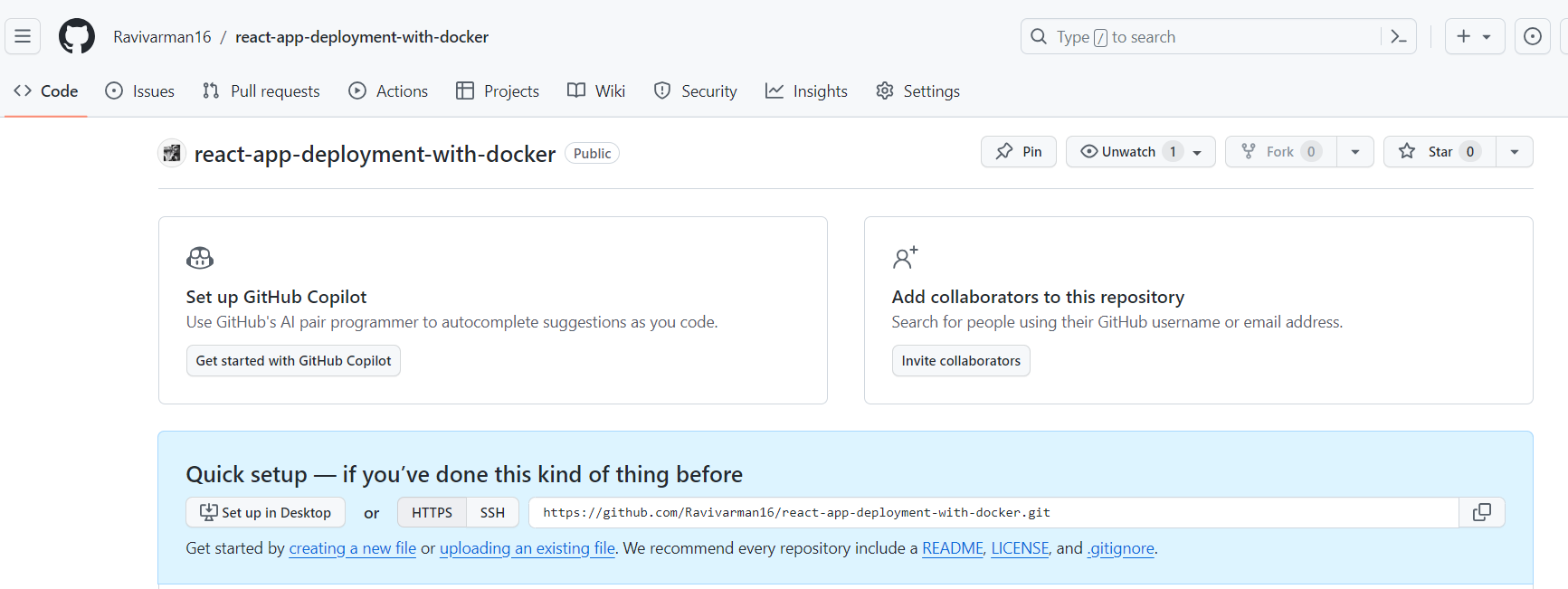
* Just login into GitHub account:

****

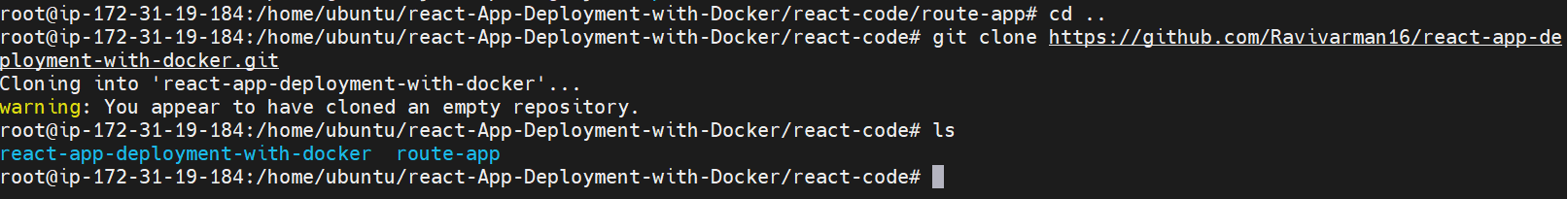
* Then just click new under repository section:
* Then create repository according to your preferences:

****

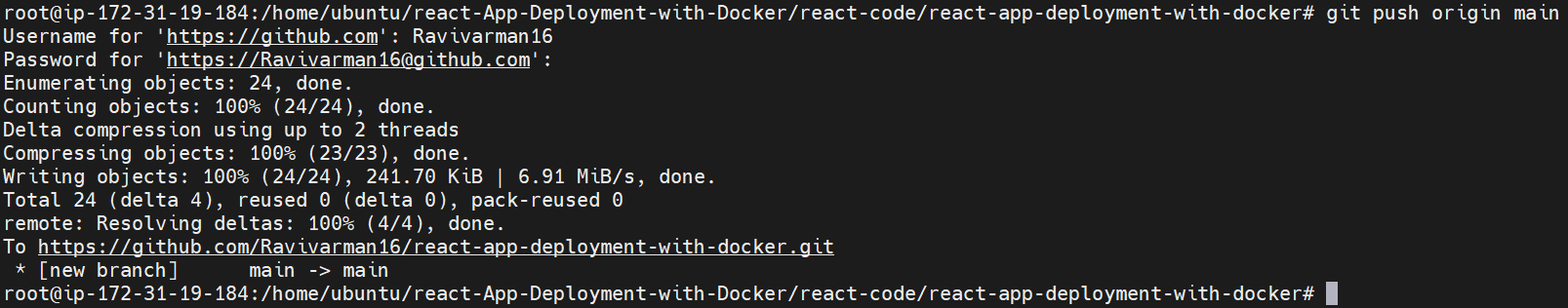
* Therepository has been created successfully: now just copy the https URL and come back to command line:

****

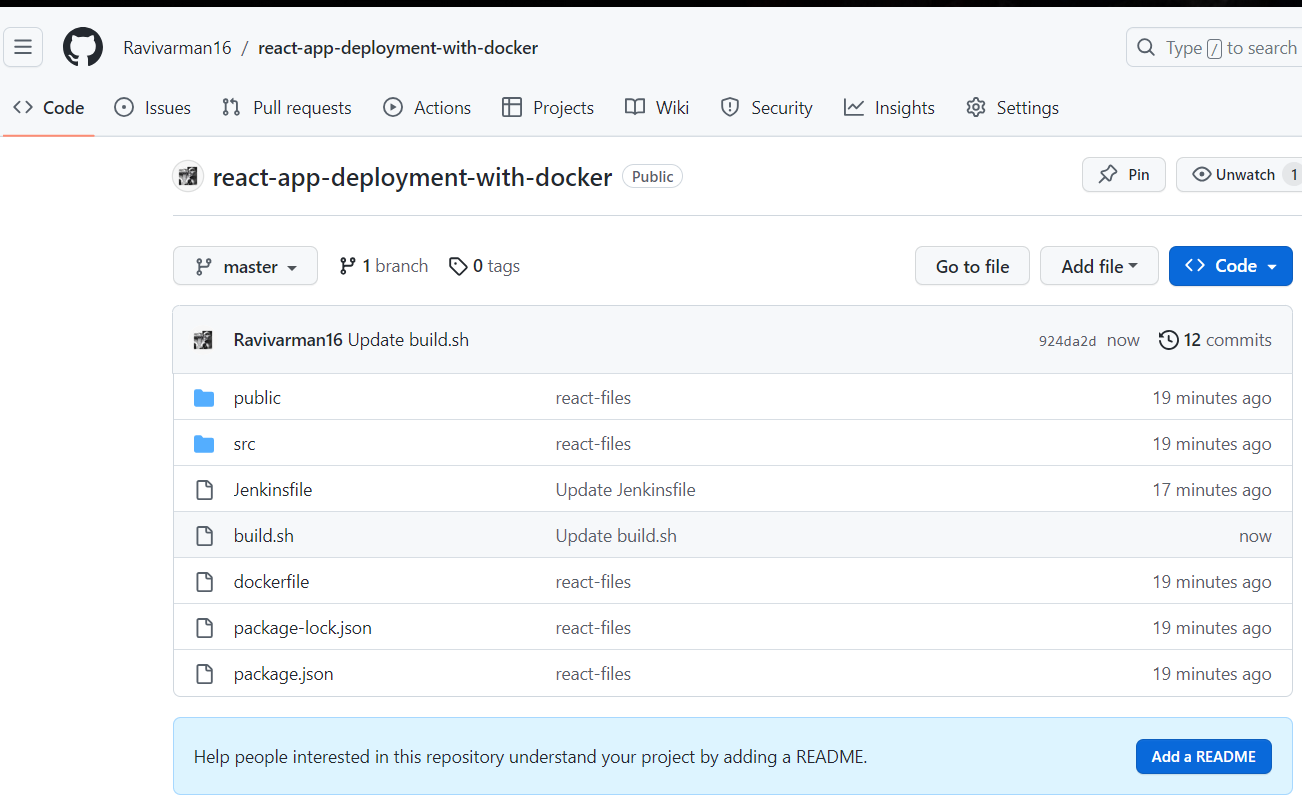
* Then clone the repository:



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



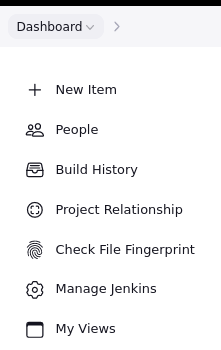
Checking the remote repository:



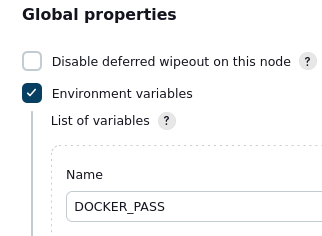
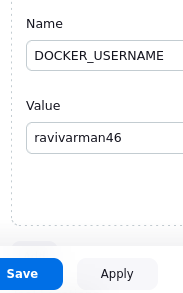
------------------------------------------------------------------------------------------------

**Step:8 – Setting up Environmental variables:**

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



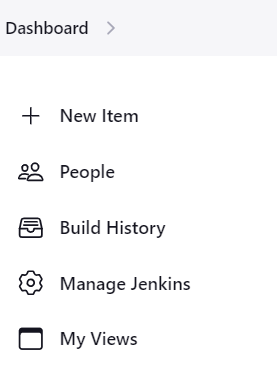
* Then click system: under global properties, select environment variables: set the variables: click save and apply

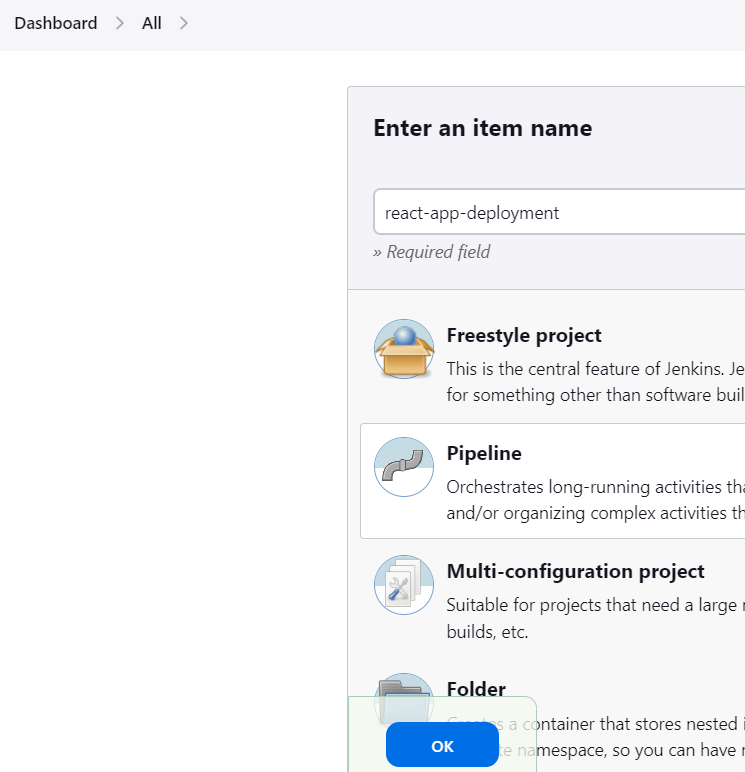
------------------------------------------------------------------------------------------------

**Step:9 - Setting up a CI/CD pipeline:**

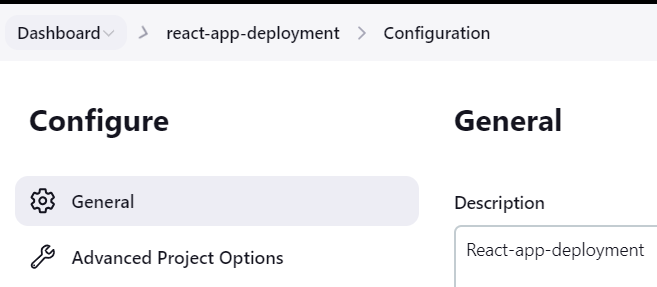
* On Jenkins dashboard, we can able to find out new item, click that one:



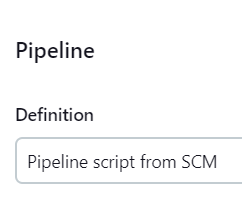
* Then name the job and select the job type as pipeline: click okay:



* Then under description give according to this task:



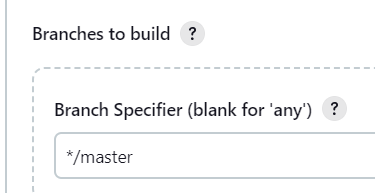
* Then under pipeline select **pipeline script from SCM:**



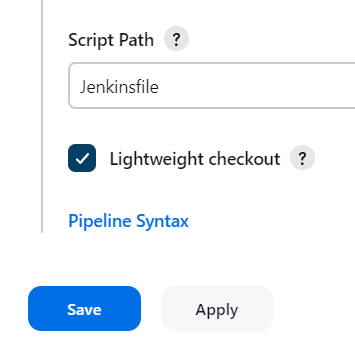
* Then under SCM select Git, enter the GitHub URL:



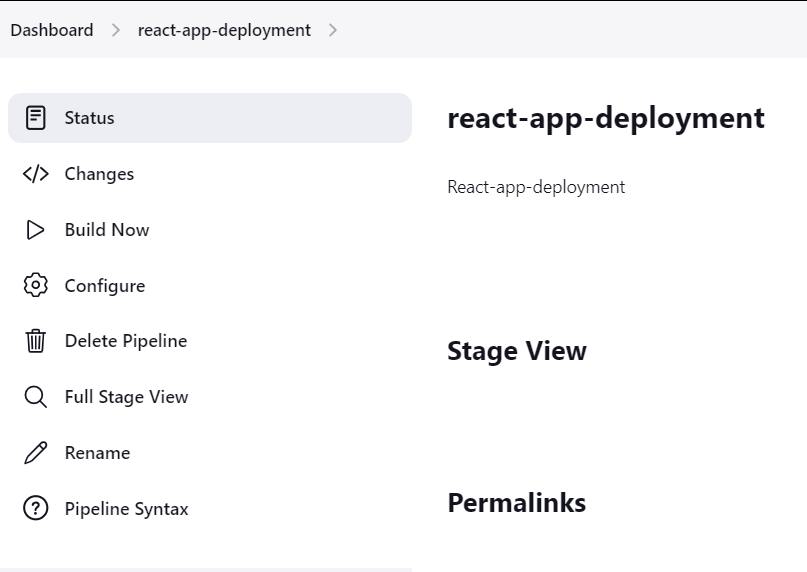
* Then select the branch:



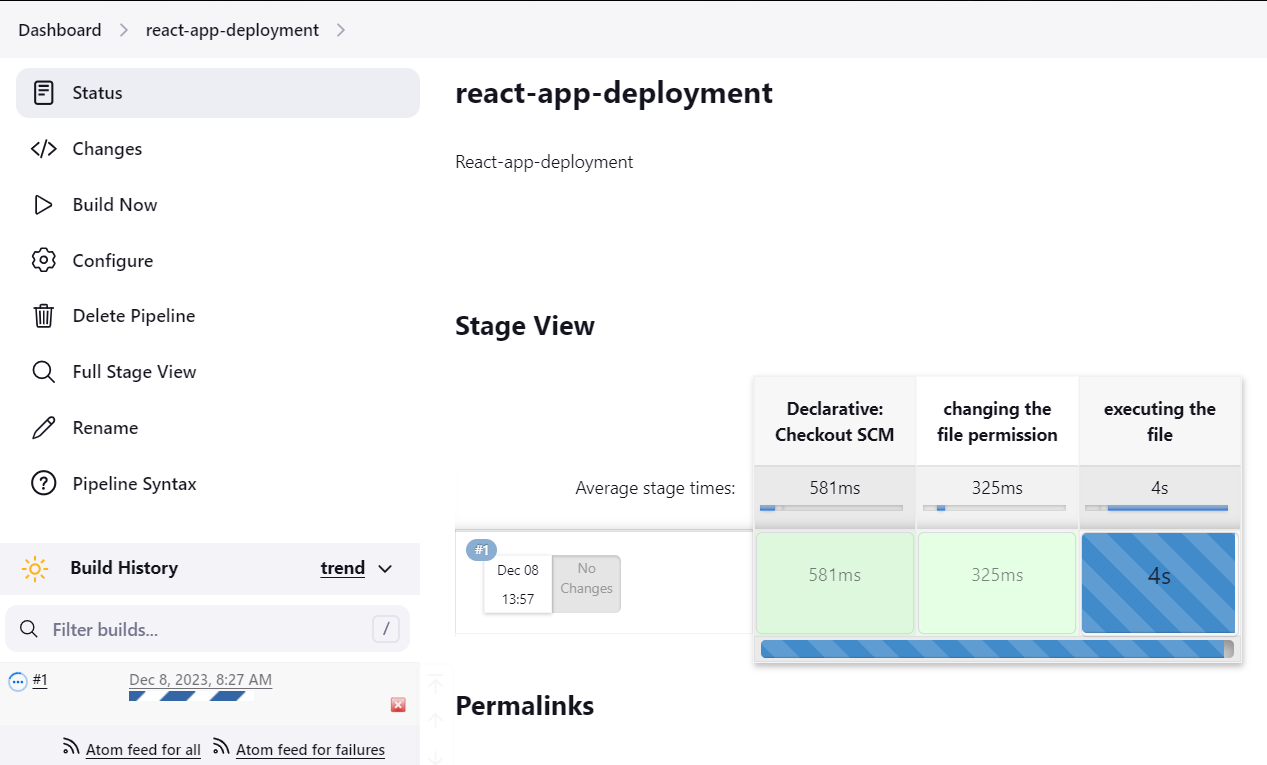
* Then enter the Jenkins file name: click apply and save:



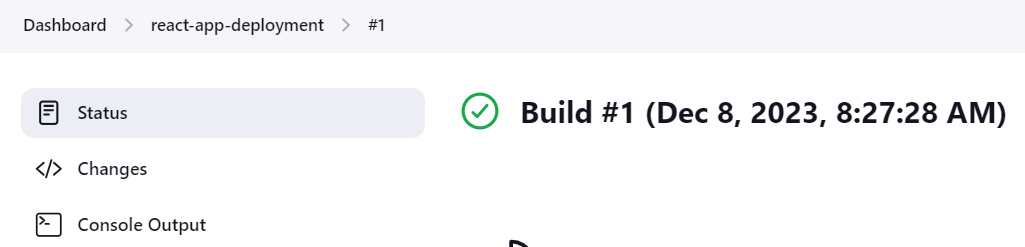
* Then we can able to see pipeline job has been created successfully:



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

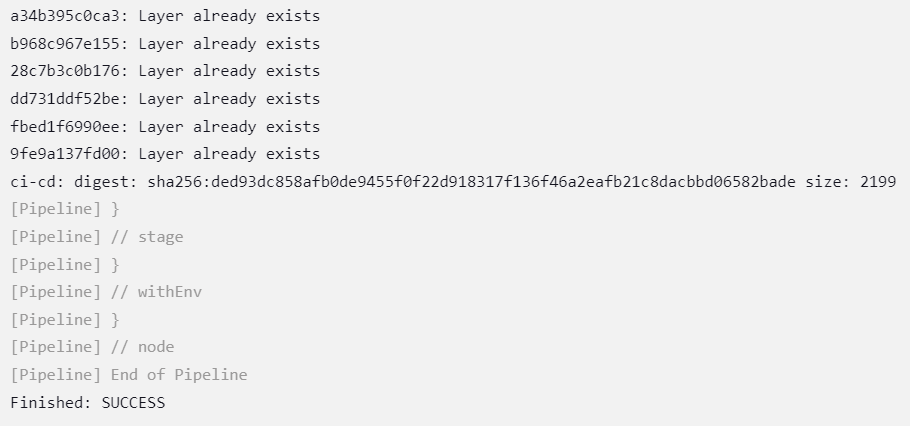


* We can able to see pipeline job executed successfully, click the job to know more details:

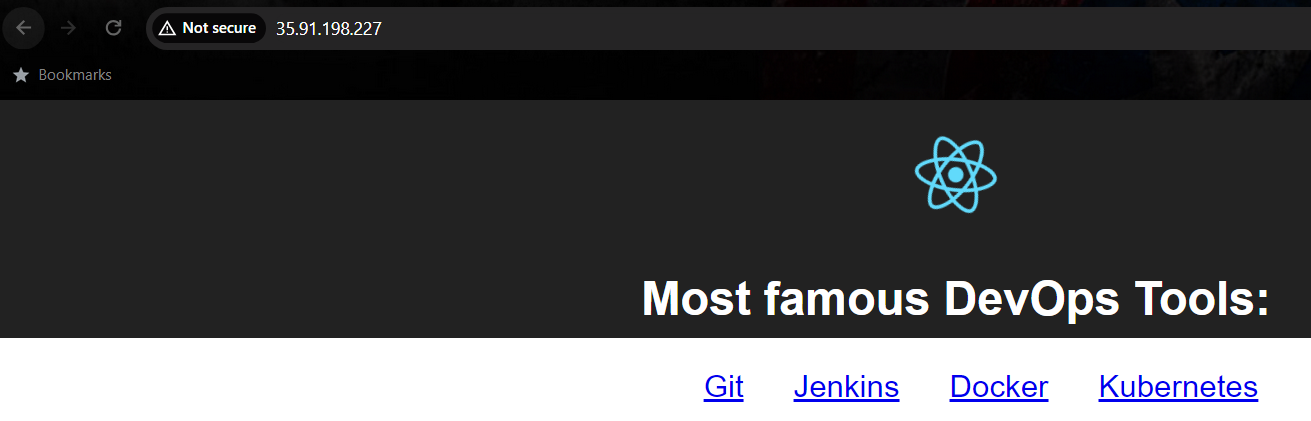


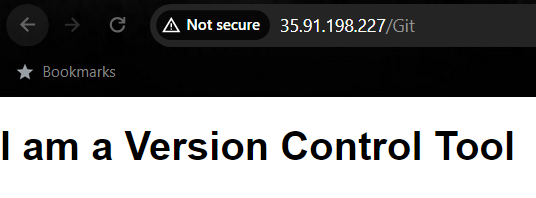
* Then click console output option: to know details working of this pipeline:

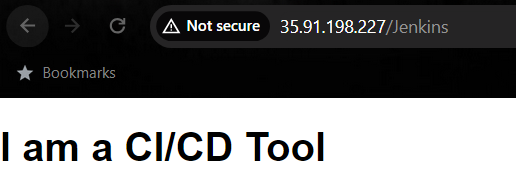


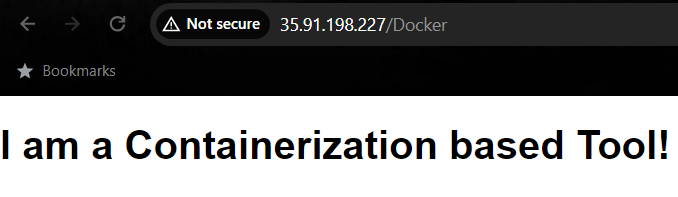


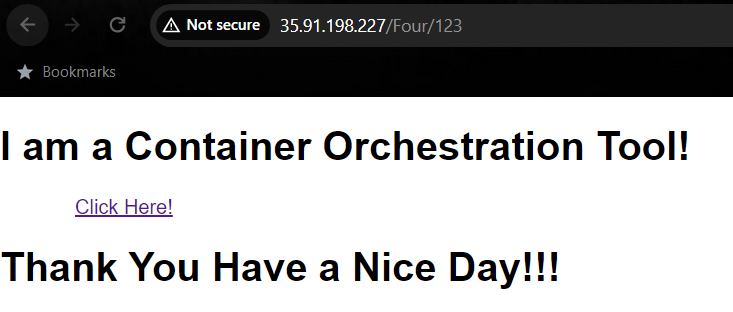
* Browser output of the container:



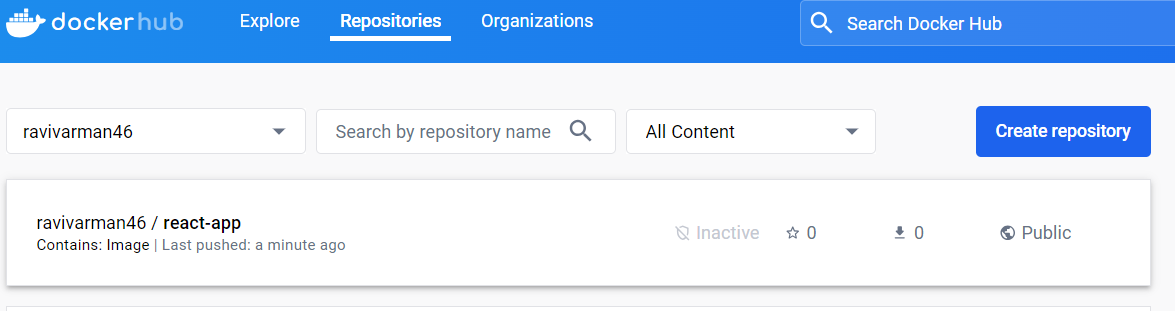


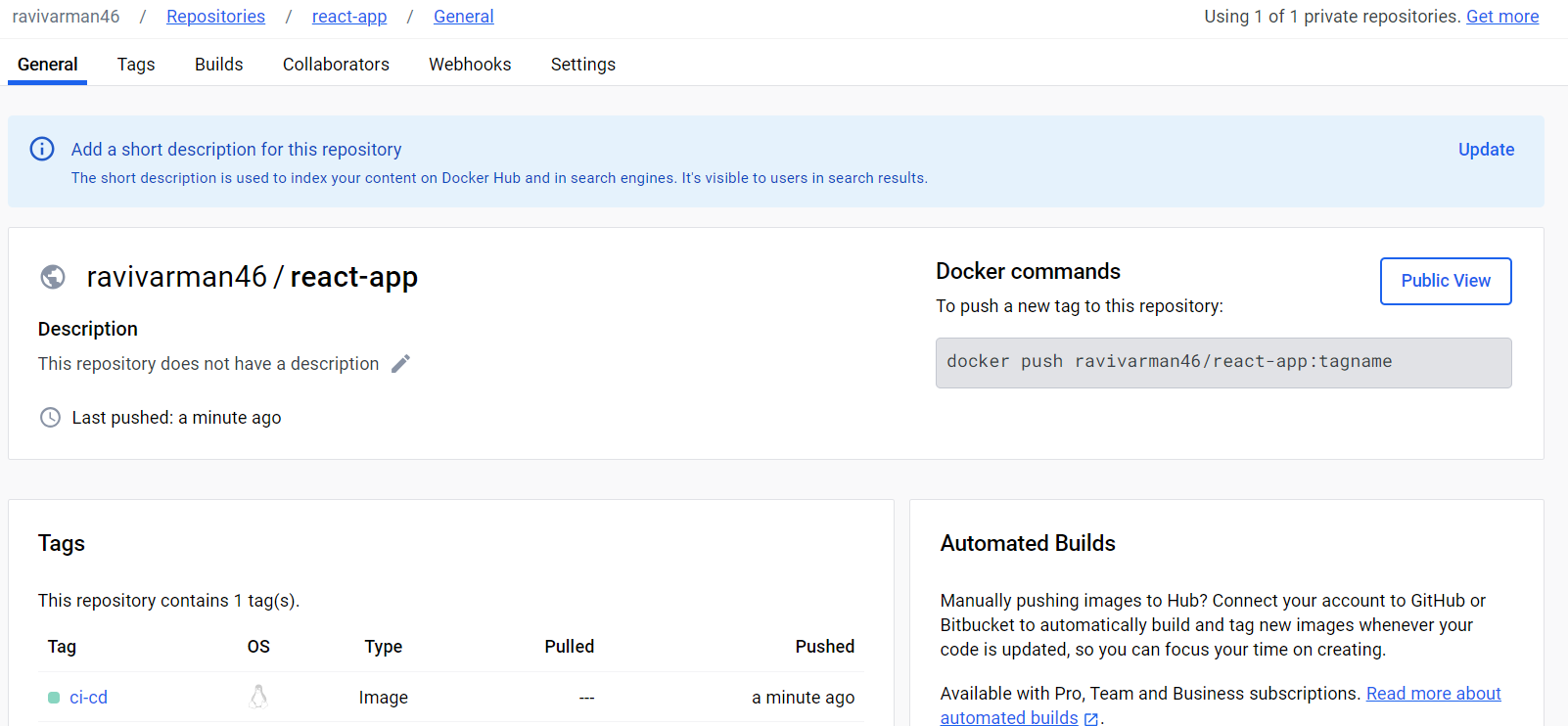






* Docker hub output:

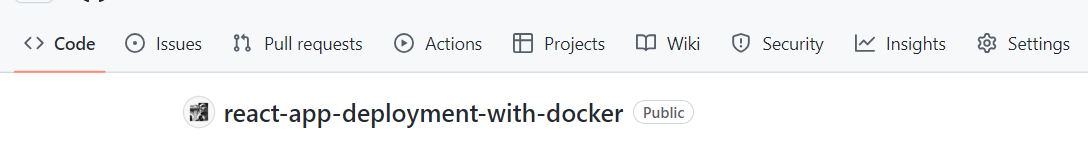




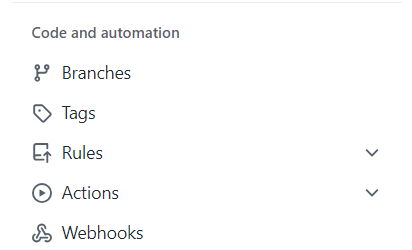
------------------------------------------------------------------------------------------------

**Step:10 - Making the pipeline automated:**

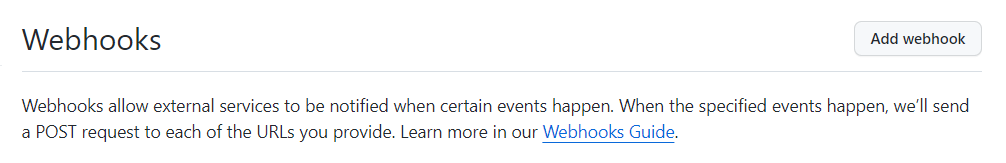
* On the GitHub repository, we can able to see settings, click that one:



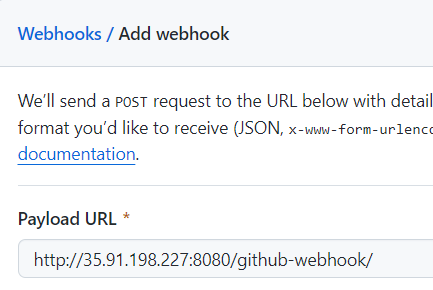
* Then on the left side, we can able to see webhooks, click that one:



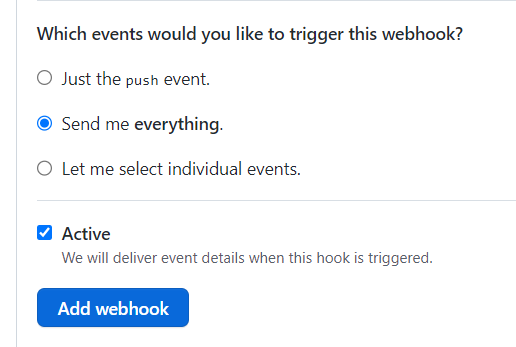
* Then we can able to see add webhook option, click that one:



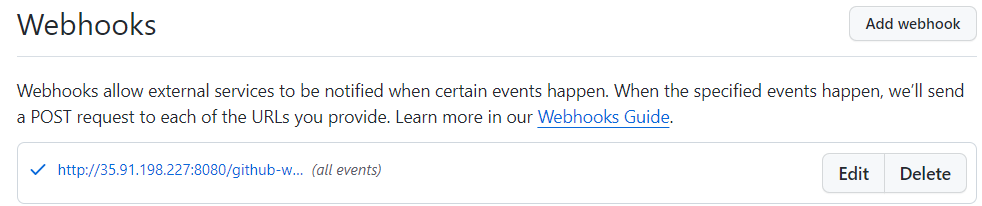
* Then under payload URL, enter the Jenkins URL along with github-webhook:



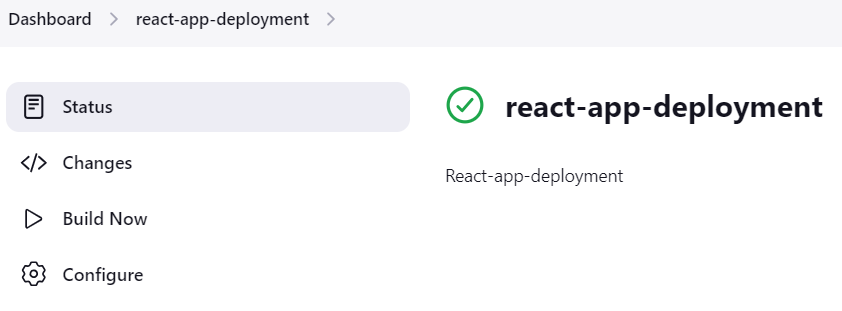
* Then select the events to trigger from GitHub: click add webhook:



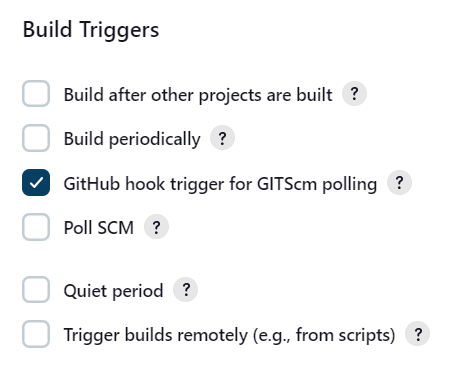
* Then we can see webhook has been created successfully:

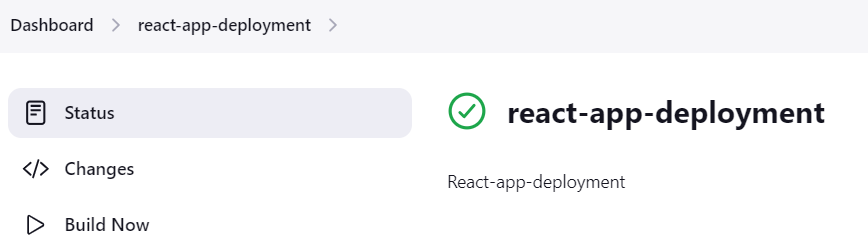


* Then on Jenkins pipeline job, we need make one change in settings, for that click configure:

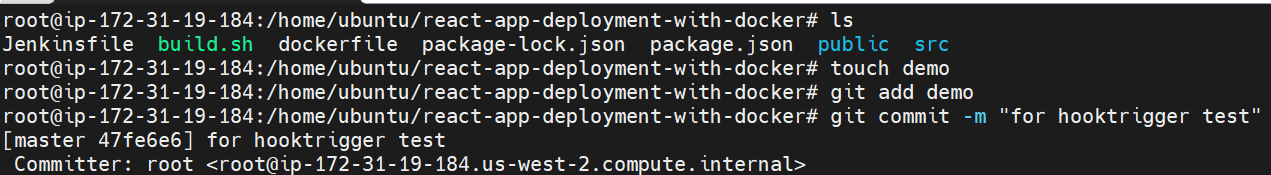


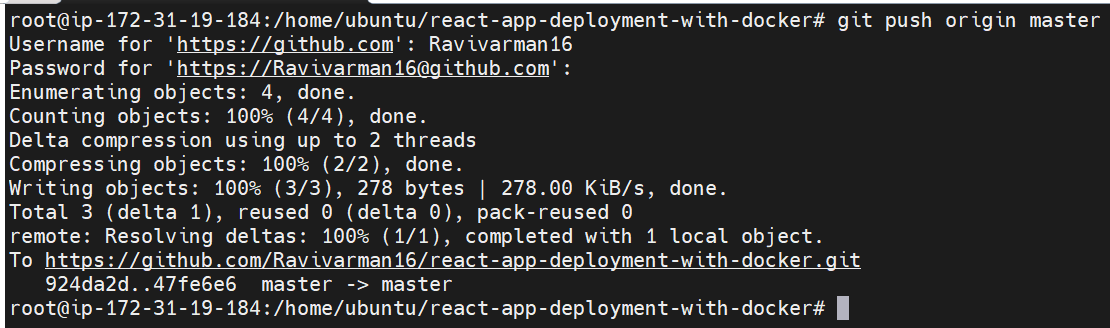
* Under build triggers enable: GitHub hook trigger for GITScm polling option. Save it



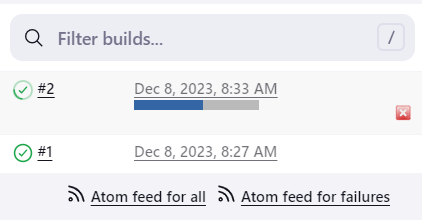


* Then on the command line, make small change, here I am creating a file, pushing it to remote repository:

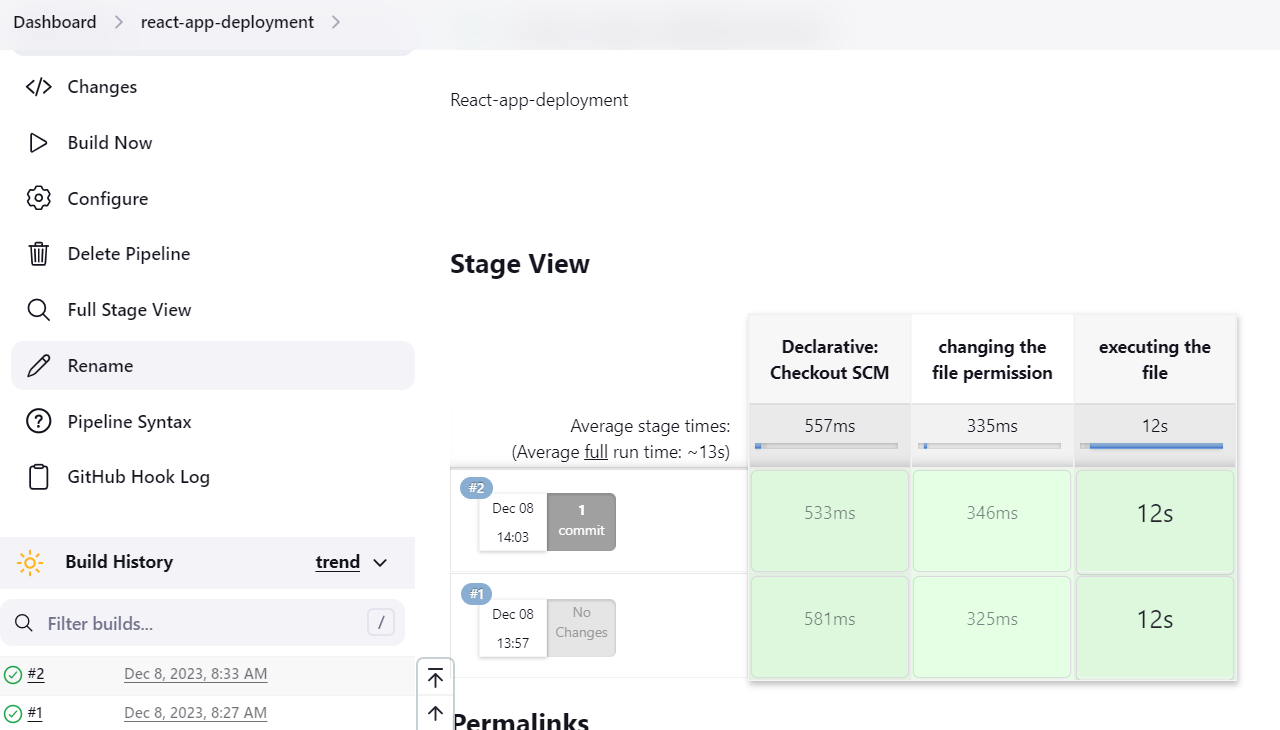




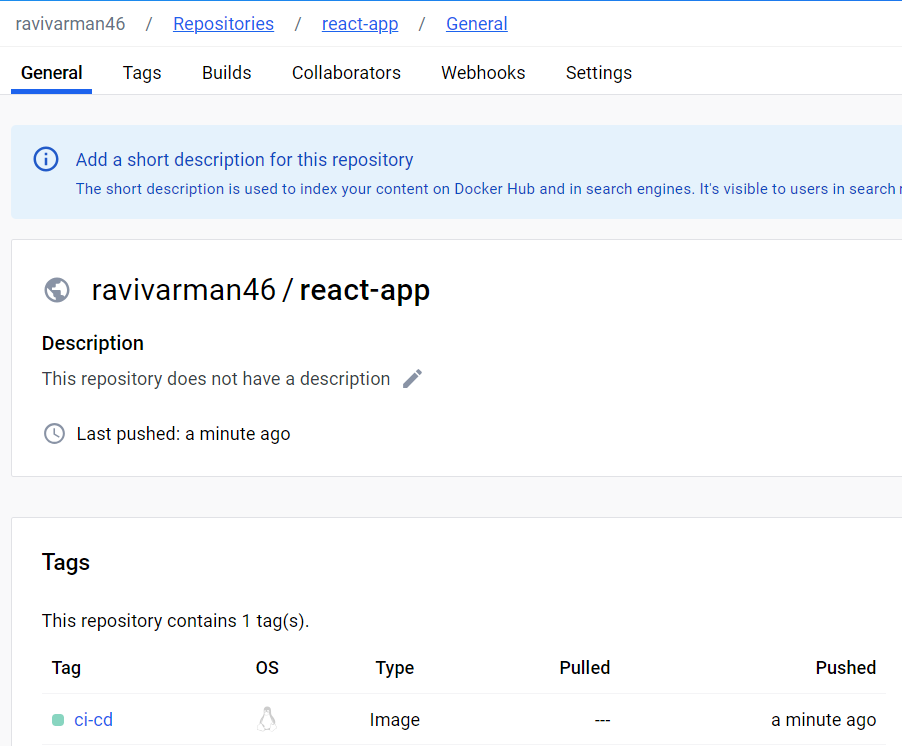
* After pushing it to the GitHub we can see in Jenkins job got triggered automatically:



* 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**](https://github.com/Ravivarman16/react-app-deployment-with-docker.git)