

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.

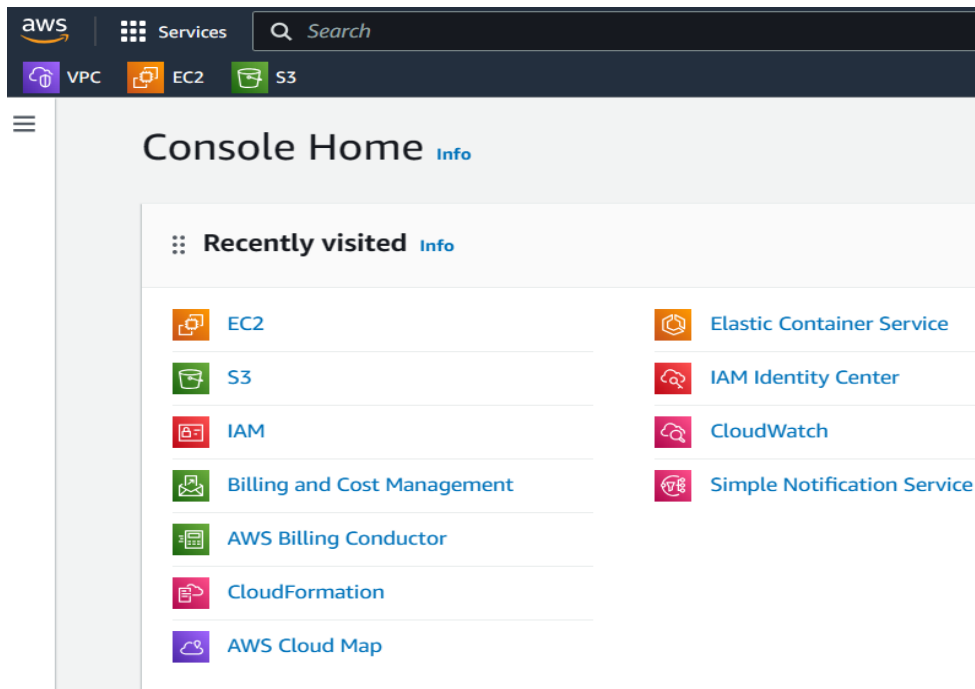
SOLUTION:

REQUIREMENTS:

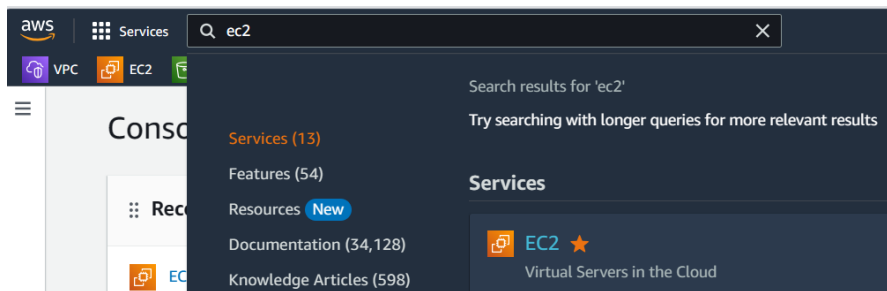
- ❖ AWS Cloud
- ❖ AWS EC2 instance
- ❖ Git & GitHub
- ❖ Docker
- ❖ Java
- ❖ Jenkins

Step:1 – Launching an EC2 instance:

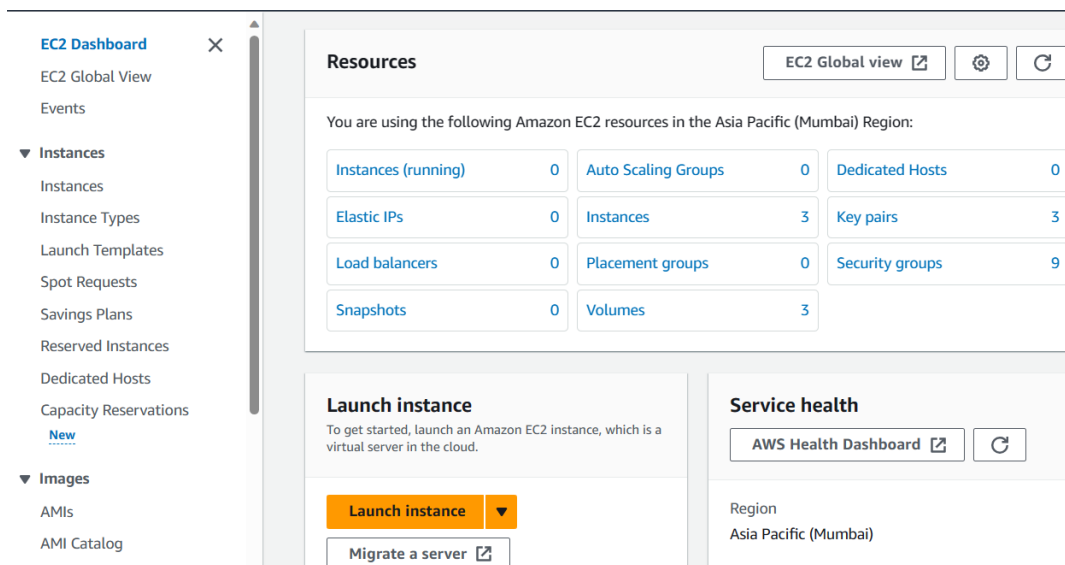
- ➔ First login into your AWS instance:



→ 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:

EC2 > Instances > Launch an instance

Launch an instance [Info](#)

Amazon EC2 allows you to create virtual machines following the simple steps below.

Name and tags [Info](#)

Name

➔ Then select the operating system according to your preferences:


▼ Application and OS Images (Amazon Machine Image) [Info](#)

An AMI is a template that contains the software configuration (operating system, applications) required to launch your instance. Search or Browse for AMIs if you do below


Recents

Quick Start


Amazon Linux




macOS




Ubuntu




Windows



Red Hat



SUSE



Amazon Machine Image (AMI)

Ubuntu Server 22.04 LTS (HVM), SSD Volume Type

ami-0287a05f0e0e9d9a (64-bit (x86)) / ami-0b6581fde9e6e7779 (64-bit (Arm))

Virtualization: hvm ENA enabled: true Root device type: ebs

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

▼ Instance type [Info](#) | [Get advice](#)

Instance type

t2.micro

Free tier eligible

Family: t2 1 vCPU 1 GiB Memory Current generation: true

On-Demand Linux base pricing: 0.0124 USD per Hour

On-Demand Windows base pricing: 0.017 USD per Hour

On-Demand RHEL base pricing: 0.0724 USD per Hour

On-Demand SUSE base pricing: 0.0124 USD per Hour

[Additional costs apply for AMIs with pre-installed software](#)

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

▼ Key pair (login) [Info](#)

You can use a key pair to securely connect to your instance. Ensure that you have access before you launch the instance.

Key pair name - *required*

docker ▼

➔ Then keeping the default options under network settings:

▼ Network settings [Info](#)

Network [Info](#)

vpc-04dc687e3ffd22a68

Subnet [Info](#)

No preference (Default subnet in any availability zone)

Auto-assign public IP [Info](#)

Enable

Firewall (security groups) [Info](#)

A security group is a set of firewall rules that control the traffic for your instance. Add rules to allow specific traffic to your instance.

☒ Create security group

☐ Select existing security group

We'll create a new security group called 'launch-wizard-4' with the following rules:

☒ Allow SSH traffic from

Helps you connect to your instance

Anywhere

0.0.0.0/0 ▼

☐ Allow HTTPS traffic from the internet

To set up an endpoint, for example when creating a web server

☐ Allow HTTP traffic from the internet

To set up an endpoint, for example when creating a web server

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

▼ Configure storage Info

Advanced

1x 8 GiB gp2 Root volume (Not encrypted)

Free tier eligible customers can get up to 30 GB of EBS General Purpose (SSD) or Magnetic storage

Add new volume

Click refresh to view backup information

The tags that you assign determine whether the instance will be backed up by any Data Lifecycle Manager policies.

0 x File systems

Edit

► Advanced details Info

Software Image (AMI)

Canonical, Ubuntu, 22.04 LTS, ...read more

ami-0287a05f0ef0e9d9a

Virtual server type (instance type)

t2.micro

Firewall (security group)

New security group

Storage (volumes)

1 volume(s) - 8 GiB

Free tier: In your first year includes 750 hours of t2.micro (or t3.micro in the Regions in which

Cancel

Launch instance

Review commands

➔ The instance has been launched successfully:

Instances (1) Info

Find Instance by attribute or tag (case-sensitive)

<input type="checkbox"/>	Name	Instance ID	Instance state
<input type="checkbox"/>	React-deployment	i-0f3379216d9771671	Pending

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

```

2. React
> For more info, ctrl+click on help or visit our website.

Welcome to Ubuntu 22.04.3 LTS (GNU/Linux 6.2.0-1012-aws x86_64)

* Documentation: https://help.ubuntu.com
* Management:   https://landscape.canonical.com
* Support:       https://ubuntu.com/advantage

System information as of Fri Dec 8 06:10:39 UTC 2023

System load: 0.18408203125 Processes: 111
Usage of /: 20.5% of 7.57GB Users logged in: 0
Memory usage: 5% IPv4 address for eth0: 172.31.19.184
Swap usage: 0%

Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

```

➔ 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]
```

```
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>

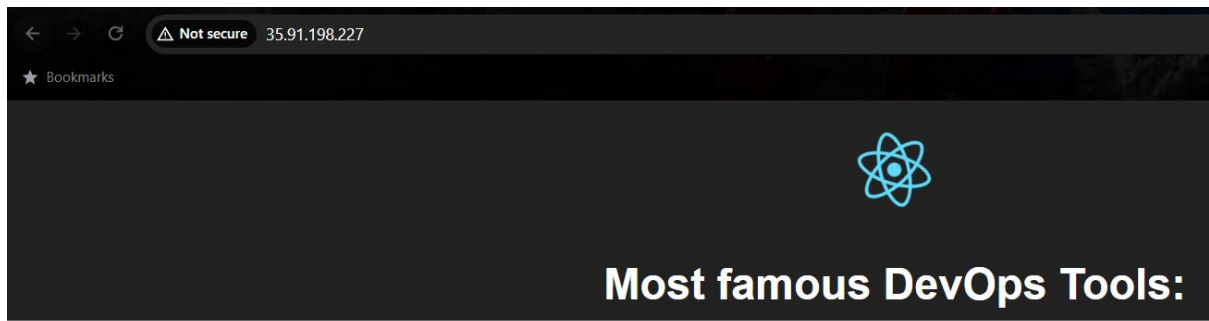
```
root@ip-172-31-19-184:/home/ubuntu/react-App-Deployment-with-Docker/react-code/route-app# docker run -d -it -p 80:80 react-ci/cd
41dcee9e4269022691ff36310f560a4b71e9859f510544abbad89f73129b1c8d
root@ip-172-31-19-184:/home/ubuntu/react-App-Deployment-with-Docker/react-code/route-app# docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
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:

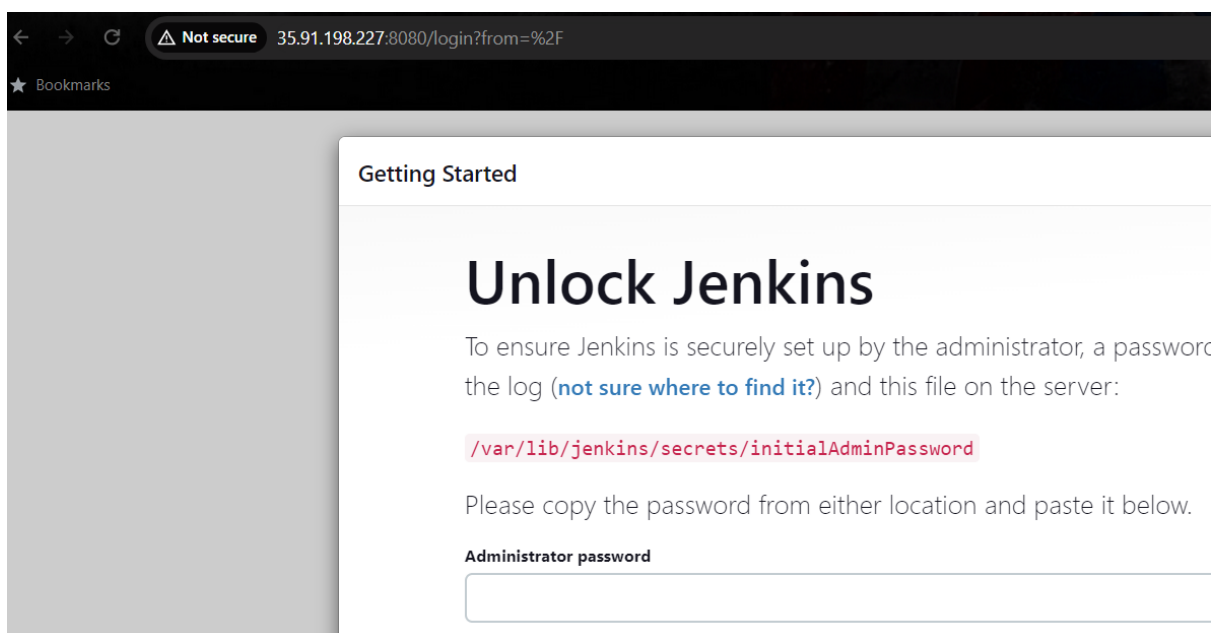


[Git](#) [Jenkins](#) [Docker](#) [Kubernetes](#)

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:

```
root@ip-172-31-19-184:/home/ubuntu/react-App-Deployment-with-Docker/react-code/route-app# cat /var/lib/jenkins/secrets/initialAdminPassword
367619cff5b149afa2e392fa5f2fc432
root@ip-172-31-19-184:/home/ubuntu/react-App-Deployment-with-Docker/react-code/route-app#
```

➔ Then click the install suggested plugins options:

Getting Started

Customize Jenkins

Plugins extend Jenkins with additional features to support many different needs.

Install suggested plugins

Install plugins the Jenkins community finds most useful.

Select plugins to install

Select and install plugins most suitable for your needs.

➔ Plugins will start to install:

Getting Started

Getting Started

✓ Folders	✓ OWASP Markup Formatter	🔄 Build Timeout	🔄 Credentials Binding	<div>** Ionicons API</div> <div>Folders</div> <div>OWASP Markup Formatter</div> <div>** Struts</div> <div>** bouncycastle API</div> <div>** Instance Identity</div> <div>** JavaBeans Activation Framework (JAF) API</div> <div>** JavaMail API</div> <div>** Pipeline: Step API</div> <div>** Token Macro</div>
🔄 Timestamper	🔄 Workspace Cleanup	🔄 Ant	🔄 Gradle	
🔄 Pipeline	🔄 GitHub Branch Source	🔄 Pipeline: GitHub Groovy Libraries	🔄 Pipeline: Stage View	
🔄 Git	🔄 SSH Build Agents	🔄 Matrix Authorization Strategy	🔄 PAM Authentication	
🔄 LDAP	🔄 Email Extension	🔄 Mailer		

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

Create First Admin User

Username

Password

Confirm password

Full name

➔ Then click save and finish:

Instance Configuration

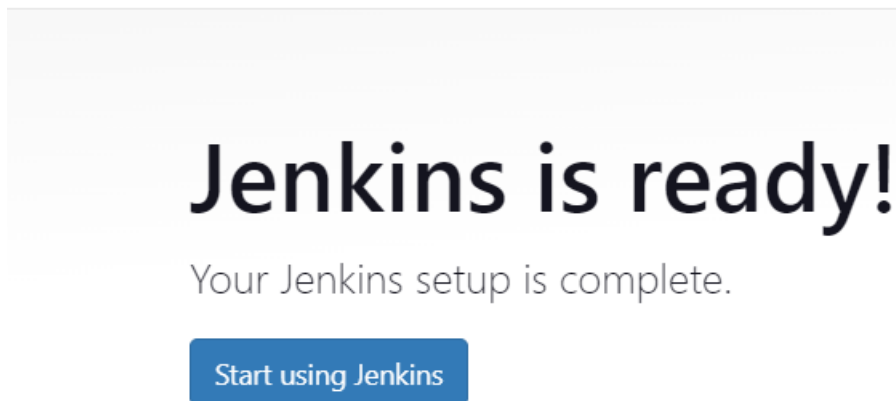
Jenkins URL:

The Jenkins URL is used to provide the root URL for absolute links to various Jenkins resources. That means this value is required for proper operation of many Jenkins features including email notifications, PR status updates, and the `BUILD_URL` environment variable provided to build steps.

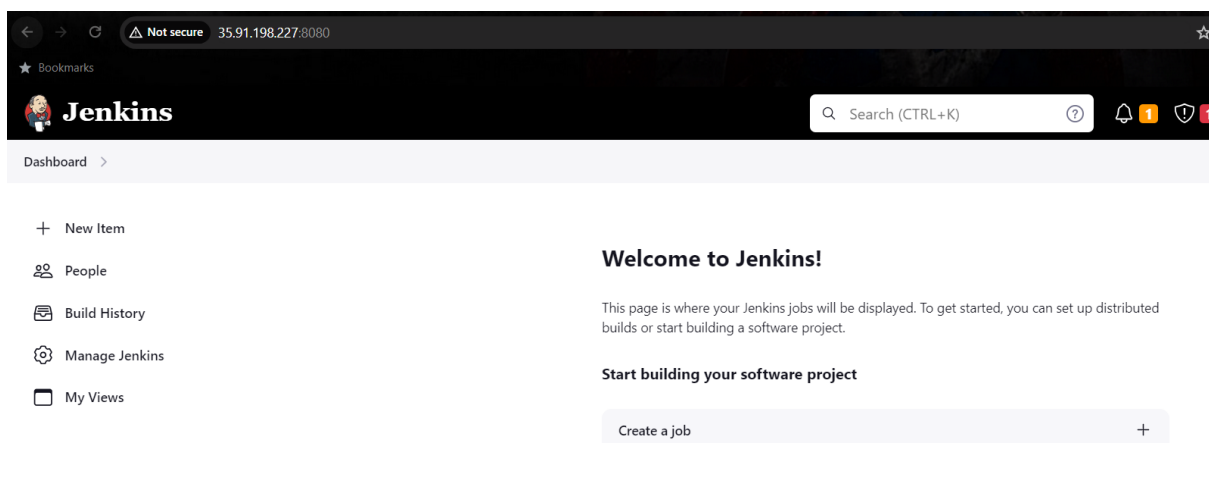
The proposed default value shown is **not saved yet** and is generated from the current request, if possible. The best practice is to set this value to the URL that users are expected to use. This will avoid confusion when sharing or viewing links.

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

Getting Started



➔ 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=dkr_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
832e3ade3bf4dddf686620befb9d01af96afa65f4fc7b17c1bf0470b55935d9fc
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

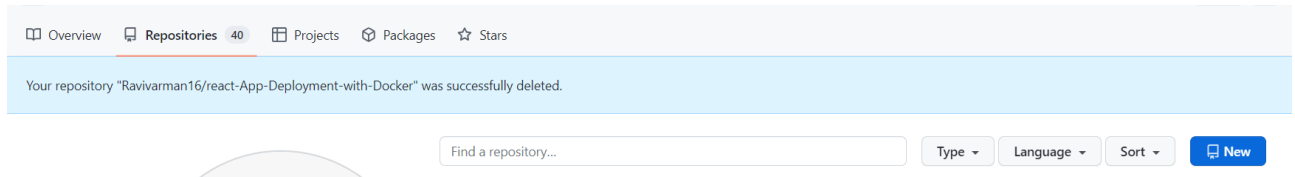
    stages {
        stage('changing the file permission') {
            steps {
                sh 'chmod +x build.sh'
            }
        }

        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:

Create a new repository

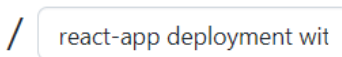
A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk ().*

Owner *



Repository name *



✓ Your new repository will be created as **react-app-deployment-with-docker**.

The repository name can only contain ASCII letters, digits, and the characters ., -, and _.

Great repository names are short and memorable. Need inspiration? How about [special-couscous](#) ?

Description (optional)



Public

Anyone on the internet can see this repository. You choose who can commit.



Private

You choose who can see and commit to this repository.

Initialize this repository with:

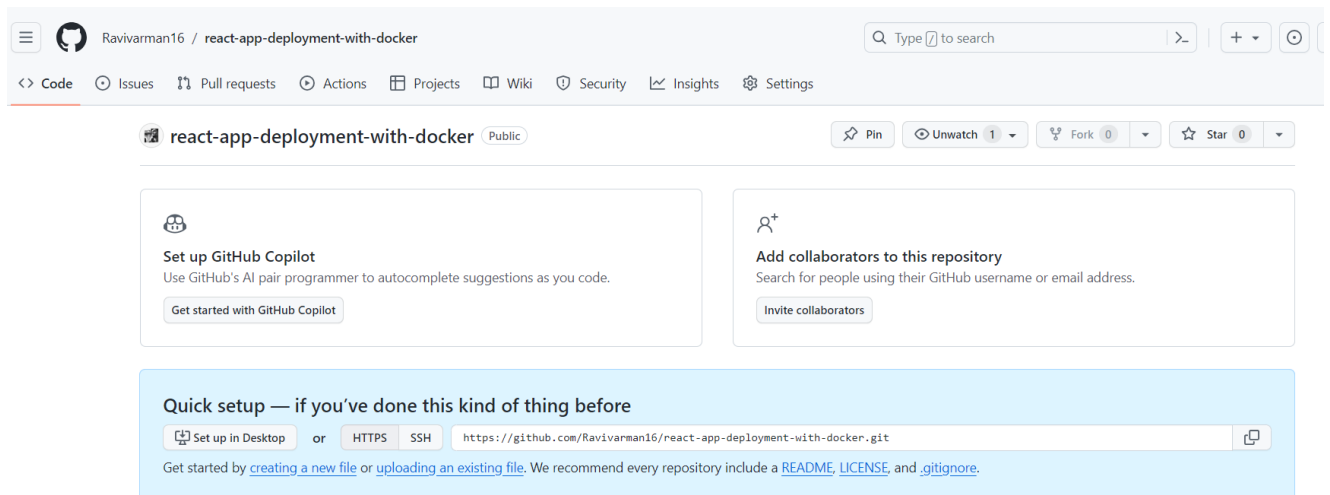


Add a README file

This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore

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



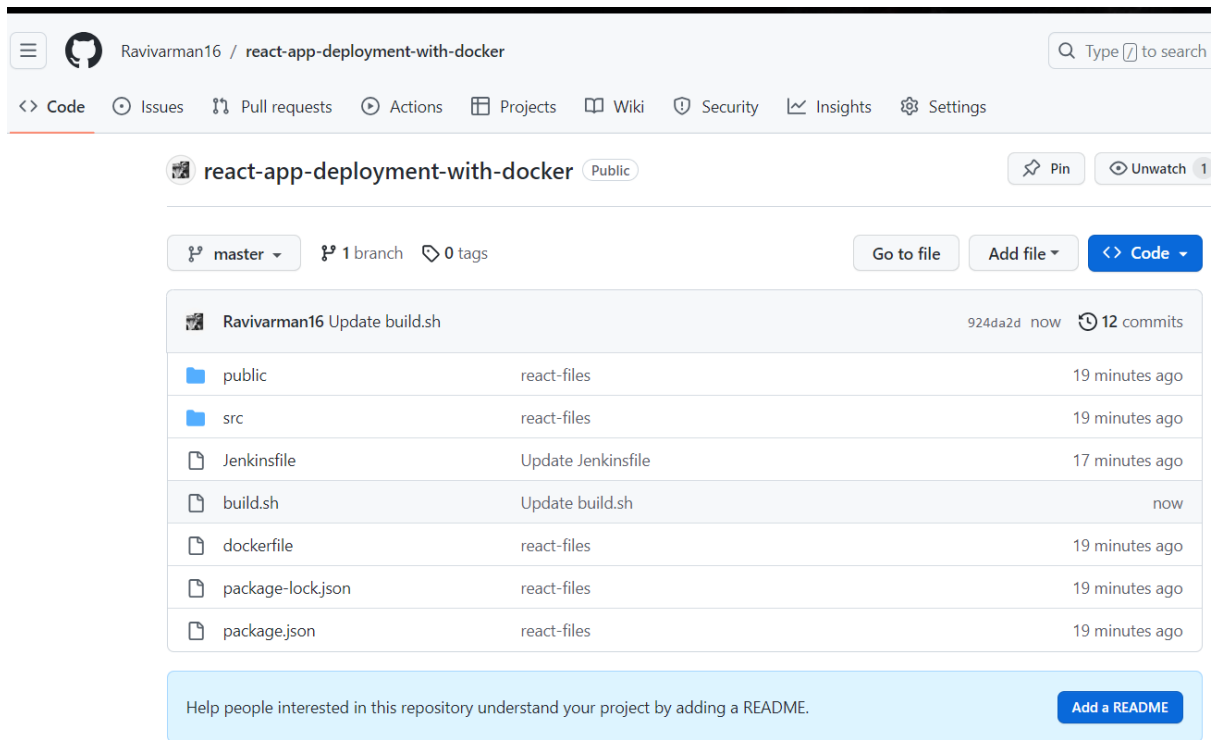
➔ Then clone the repository:

```
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:

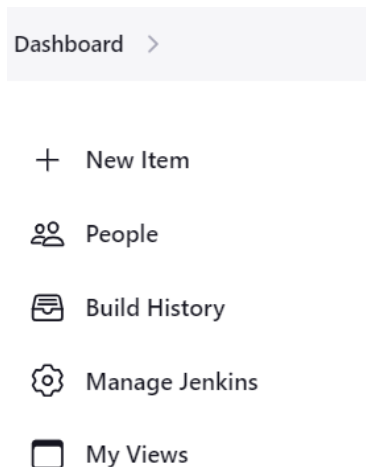
```
root@ip-172-31-19-184:/home/ubuntu/react-App-Deployment-with-Docker/react-code/react-app-deployment-with-docker# git push origin main
Username for 'https://github.com': Ravivarman16
Password for 'https://Ravivarman16@github.com':
Enumerating objects: 24, done.
Counting objects: 100% (24/24), done.
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:

Dashboard > All >

Enter an item name

react-app-deployment

» Required field



Freestyle project

This is the central feature of Jenkins. Je
for something other than software buil



Pipeline

Orchestrates long-running activities th
and/or organizing complex activities th



Multi-configuration project

Suitable for projects that need a large
builds, etc.



Folder

Creates a container that stores nested i
the namespace, so you can have r

OK

➔ Then under description give according to this task:

Dashboard > react-app-deployment > Configuration

Configure

General



General



Advanced Project Options

Description

React-app-deployment

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

Pipeline

Definition

Pipeline script from SCM

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

SCM ?

Git

Repositories ?

Repository URL ?

https://github.com/Ravivarman16/react-app-deployment-with-docker.git

➔ Then select the branch:

Branches to build ?

Branch Specifier (blank for 'any') ?

*/master

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

Script Path ?

Jenkinsfile

☒ Lightweight checkout ?

[Pipeline Syntax](#)

Save

Apply

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

 Status

react-app-deployment

 Changes

React-app-deployment

 Build Now

 Configure

 Delete Pipeline

Stage View

 Full Stage View

 Rename

 Pipeline Syntax

Permalinks

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

 Status

react-app-deployment

 Changes

React-app-deployment

 Build Now

 Configure


 Delete Pipeline


 Full Stage View

 Rename

 Pipeline Syntax

Stage View

 Build History **trend** v

 Filter builds... /

 #1 Dec 8, 2023, 8:27 AM

 Atom feed for all  Atom feed for failures


Permalinks

Average stage times:

Declarative: Checkout SCM	changing the file permission	executing the file
581ms	325ms	4s

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

 Status

 **Build #1 (Dec 8, 2023, 8:27:28 AM)**

 Changes

 Console Output

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

Dashboard > react-app-deployment > #1

Status

Changes

Console Output

View as plain text

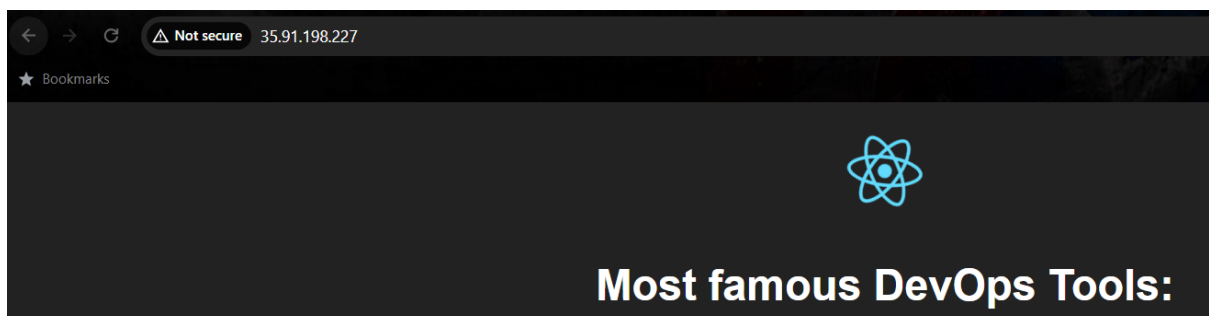
Edit Build Information

✓ Console Output

Started by user [jenkins](#)
Obtained Jenkinsfile from git <https://github.com/Ravivarman16/react-app-deployment-with-docker.git>
[Pipeline] Start of Pipeline
[Pipeline] node
Running on [Jenkins](#) in /var/lib/jenkins/workspace/react-app-deployment
[Pipeline] f

```
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:



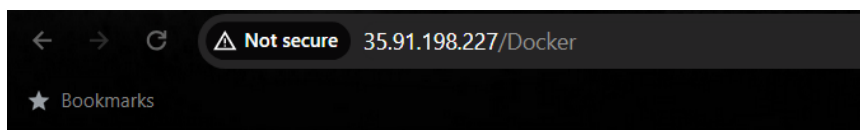
[Git](#) [Jenkins](#) [Docker](#) [Kubernetes](#)



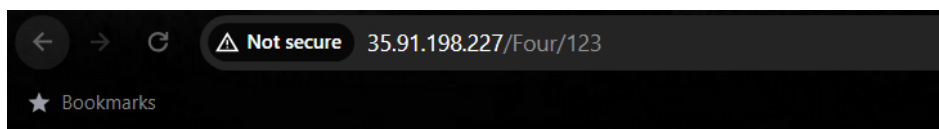
I am a Version Control Tool



I am a CI/CD Tool



I am a Containerization based Tool!

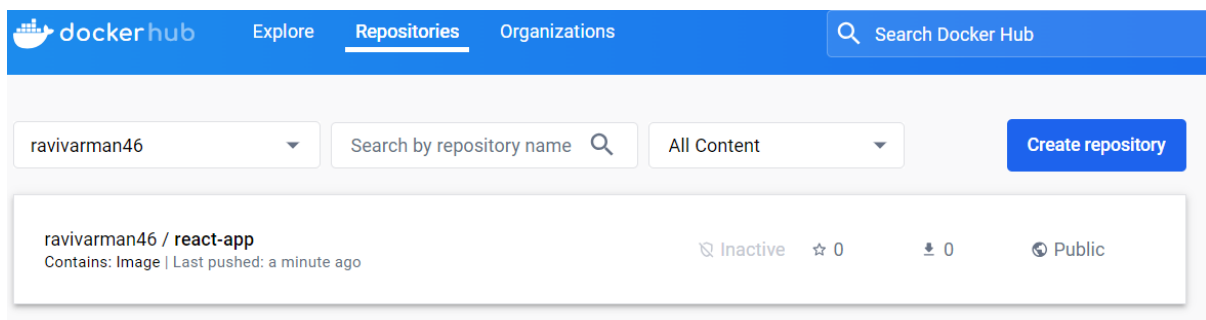


I am a Container Orchestration Tool!

[Click Here!](#)

Thank You Have a Nice Day!!!

➔ Docker hub output:



ravivarman46 / [Repositories](#) / [react-app](#) / [General](#) Using 1 of 1 private repositories. [Get more](#)

General Tags Builds Collaborators Webhooks Settings

Add a short description for this repository Update
The short description is used to index your content on Docker Hub and in search engines. It's visible to users in search results.

ravivarman46 / react-app

Description
This repository does not have a description

 Last pushed: a minute ago

Docker commands Public View
To push a new tag to this repository:

```
docker push ravivarman46/react-app:tagname
```

Tags
This repository contains 1 tag(s).

Tag	OS	Type	Pulled	Pushed
ci-cd		Image	---	a minute ago

Automated Builds

Manually pushing images to Hub? Connect your account to GitHub or Bitbucket to automatically build and tag new images whenever your code is updated, so you can focus your time on creating.

Available with Pro, Team and Business subscriptions. [Read more about automated builds](#).

Step:9 - Making the pipeline automated:

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

[Code](#) [Issues](#) [Pull requests](#) [Actions](#) [Projects](#) [Wiki](#) [Security](#) [Insights](#) [Settings](#)

react-app-deployment-with-docker Public

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

Code and automation

Branches

Tags

Rules

Actions

Webhooks

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

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](#).

➔ 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-urlencoded [documentation](#)).

Payload URL *

http://35.91.198.227:8080/github-webhook/

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

Which events would you like to trigger this webhook?

- ☐ Just the push event.
- ☒ Send me **everything**.
- ☐ Let me select individual events.

☒ Active

We will deliver event details when this hook is triggered.

Add webhook

➔ Then we can see webhook has been created successfully:

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://35.91.198.227:8080/github-w... (all events)

Edit

Delete

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

Dashboard > react-app-deployment >

 Status

 **react-app-deployment**

 Changes

React-app-deployment

 Build Now

 Configure

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

Build Triggers

- ☐ Build after other projects are built ?
- ☐ Build periodically ?
- ☒ GitHub hook trigger for GITScm polling ?
- ☐ Poll SCM ?
- ☐ Quiet period ?
- ☐ Trigger builds remotely (e.g., from scripts) ?

Dashboard > react-app-deployment >

 Status

 **react-app-deployment**

 Changes

React-app-deployment

 Build Now

➔ 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:

Filter builds...

Build	Status	Timestamp
#2	Success	Dec 8, 2023, 8:33 AM
#1	Success	Dec 8, 2023, 8:27 AM

Atom feed for all Atom feed for failures

➔ Job executed successfully:

Dashboard > react-app-deployment >

Changes React-app-deployment

Build Now

Configure

Delete Pipeline

Full Stage View

Rename

Pipeline Syntax

GitHub Hook Log

Stage View

Average stage times:
(Average full run time: ~13s)

Stage	Declarative: Checkout SCM	changing the file permission	executing the file
#2 (Dec 08 14:03, 1 commit)	533ms	346ms	12s
#1 (Dec 08 13:57, No Changes)	581ms	325ms	12s

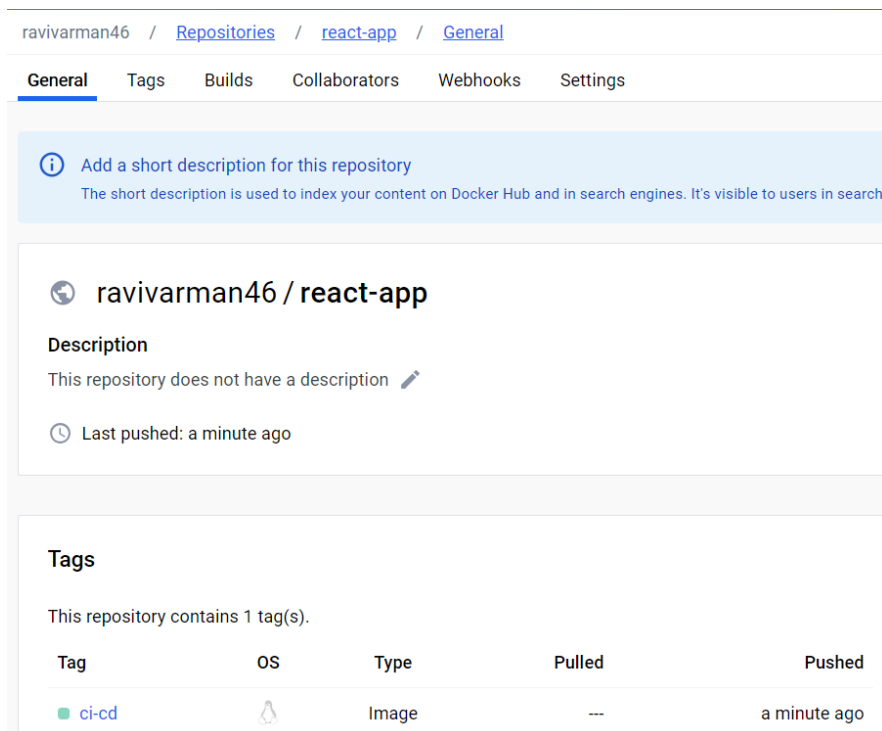
Build History trend

Filter builds...

Build	Status	Timestamp
#2	Success	Dec 8, 2023, 8:33 AM
#1	Success	Dec 8, 2023, 8:27 AM

Permalinks

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>