

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:

The screenshot shows the AWS Console Home page. At the top, there's a navigation bar with the AWS logo, a 'Services' button, a search bar containing 'Search', and links for VPC, EC2, and S3. Below the navigation bar, the title 'Console Home' is displayed. A sidebar on the left lists 'Recently visited' services: EC2, S3, IAM, Billing and Cost Management, AWS Billing Conductor, CloudFormation, and AWS Cloud Map. The main content area is currently empty.

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

The screenshot shows the AWS service search results for 'ec2'. The search bar at the top contains 'ec2'. Below it, the results are categorized under 'Services (13)'. The 'Services' section shows a card for 'EC2' with the subtext 'Virtual Servers in the Cloud'. Other categories listed include 'Features (54)', 'Resources New', 'Documentation (34,128)', and 'Knowledge Articles (598)'.

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

The screenshot shows the EC2 Dashboard. On the left, a sidebar menu includes 'EC2 Dashboard', 'Instances', 'Images', and 'New'. The main content area has three sections: 'Resources' (listing 0 instances running, 0 auto scaling groups, 0 dedicated hosts, 0 elastic IPs, 3 instances, 3 key pairs, 0 load balancers, 0 placement groups, 0 security groups, 0 snapshots, and 3 volumes), 'Launch instance' (with a large orange 'Launch instance' button), and 'Service health' (listing the region as 'Asia Pacific (Mumbai)').

➔ 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
React-deployment

➔ 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

Search our full catalog including 1000s of application and OS images

Recents Quick Start

Amazon Linux macOS Ubuntu Windows Red Hat SUSE

aws Mac ubuntu Microsoft Red Hat SUSE

Amazon Machine Image (AMI)

Ubuntu Server 22.04 LTS (HVM), SSD Volume Type
ami-0287a05f0ef0e9d9a (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 proceed 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:

The screenshot shows the 'Configure storage' section of the AWS Lambda function creation wizard. It specifies a root volume of 8 GiB using gp2 storage. A note indicates that free-tier eligible customers can get up to 30 GB of EBS General Purpose (SSD) or Magnetic storage. Below this, a message states that the selected AMI contains more instance store volumes than the instance allows, and only the first 0 instance store volumes from the AMI will be accessible from the instance. A note about backup information and Data Lifecycle Manager policies is present. The 'File systems' section shows 0x File systems. On the right, configuration details include the Software Image (AMI) as Canonical, Ubuntu, 22.04 LTS, and the Virtual server type (instance type) as t2.micro. A note about the Free tier is shown, mentioning it includes 750 hours of t2.micro (or t3.micro) in the Regions in which the instance is running. Buttons for 'Cancel', 'Launch instance', and 'Review commands' are at the bottom.

→ The instance has been launched successfully:

The screenshot shows the 'Instances (1)' list in the AWS Lambda console. The instance is named 'React-deployment' and has an ID of i-0f3379216d9771671. Its state is listed as 'Pending'. A search bar at the top is empty.

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

The screenshot shows a terminal session on an AWS Lambda instance. The terminal window is titled '2. React'. The session displays a welcome message for Ubuntu 22.04.3 LTS (GNU/Linux 6.2.0-1012-aws x86_64). It provides documentation, management, and support links. System information is shown, including load average (0.18408203125), memory usage (5%), swap usage (0%), processes (111), users logged in (0), and an IPv4 address for eth0 (172.31.19.184). A note states that expanded security maintenance for applications is not enabled. It also mentions 0 updates can be applied immediately.

```

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]
```

```
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
```

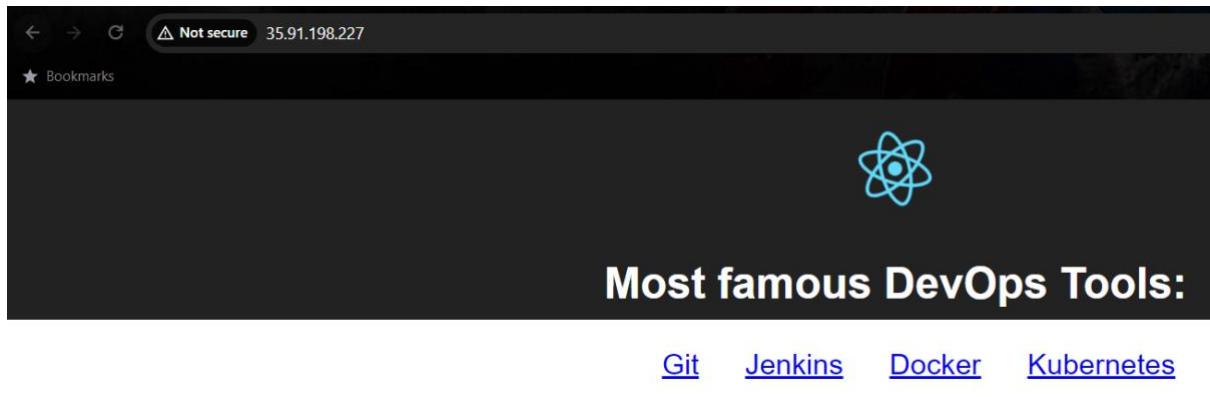
→ Checking the output of the docker image by running a container from the above image by using **docker run** command:

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
41dcee9e4269022691ff36310f560a4b71e9859f510544abbd89f73129b1c8d
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:

Getting Started

Unlock Jenkins

To ensure Jenkins is securely set up by the administrator, a password to the log ([not sure where to find it?](#)) and this file on the server:

`/var/lib/jenkins/secrets/initialAdminPassword`

Please copy the password from either location and paste it below.

Administrator password

- 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/initialAdminPas  
sword  
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	** Ionicons API Folders OWASP Markup Formatter ** Structs ** bouncycastle API ** Instance Identity ** JavaBeans Activation Framework (JAF) API ** JavaMail API ** Pipeline: Step API ** Token Macro
⌚ 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

Jenkins 2.426.1

[Skip and continue as admin](#)

[Save and Continue](#)

➔ 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.

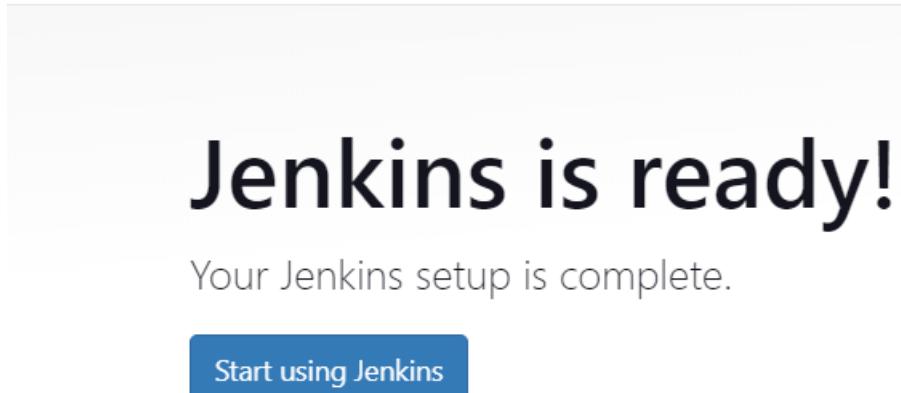
Jenkins 2.426.1

[Not now](#)

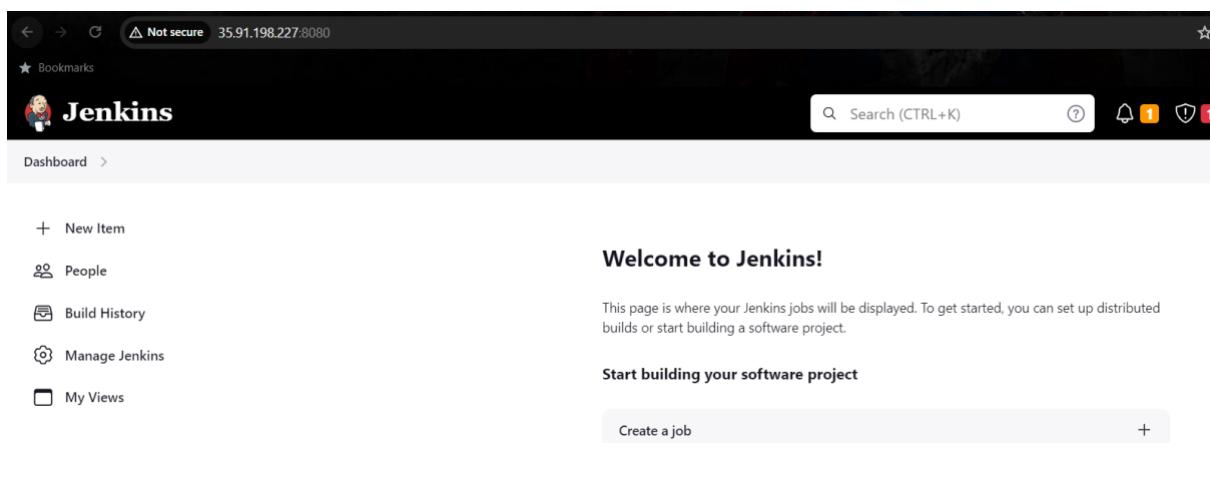
[Save and Finish](#)

→ 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=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:ded93dc858afb0de9455f0f22d918317f136f46a2eafb21c8dacbb06582bade 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
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

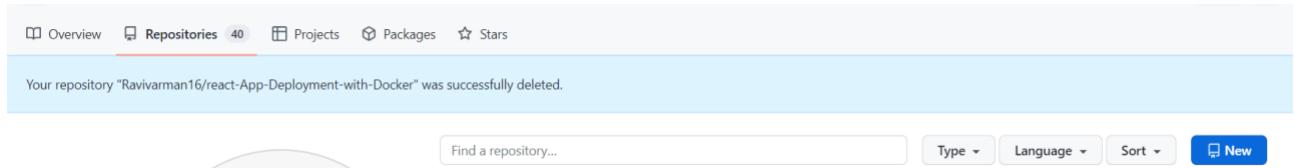
    stages {
        stage ('changing the file permission') {
            steps {
                sh ' chmod +x build.sh'
            }
        }

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

```

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

- Just login into GitHub account:



A screenshot of the GitHub repository list page. At the top, there are navigation links: Overview, Repositories 40, Projects, Packages, and Stars. Below the header, a message says "Your repository 'Ravivarman16/react-App-Deployment-with-Docker' was successfully deleted." A search bar labeled "Find a repository..." is followed by filters for Type, Language, Sort, and a "New" button.

- 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 *

 Ravivarman16 / react-app deployment wit

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:

The screenshot shows a GitHub repository page. At the top, there's a header with navigation links: Code, Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. A search bar is located at the top right. Below the header, the repository name 'react-app-deployment-with-docker' is shown, along with a 'Public' badge. To the right of the repository name are buttons for Pin, Unwatch (with a count of 1), Fork (with a count of 0), Star (with a count of 0), and a dropdown menu. On the left, there's a 'Set up GitHub Copilot' section with a button to 'Get started with GitHub Copilot'. On the right, there's a 'Add collaborators to this repository' section with a 'Search for people' input field and a 'Invite collaborators' button. Below these sections, a 'Quick setup — if you've done this kind of thing before' box contains links for 'Set up in Desktop' (with a download icon), 'HTTPS', 'SSH', and the URL 'https://github.com/Ravivarman16/react-app-deployment-with-docker.git'. It also includes a note about creating a README, LICENSE, and .gitignore files.

→ 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:

Ravivarman16 / react-app-deployment-with-docker

Type ⌘ to search

< Code Issues Pull requests Actions Projects Wiki Security Insights Settings

react-app-deployment-with-docker Public

master 1 branch 0 tags

Go to file Add file <> Code

Ravivarman16 Update build.sh 924da2d now 12 commits

public	react-files	19 minutes ago
src	react-files	19 minutes ago
Jenkinsfile	Update Jenkinsfile	17 minutes ago
build.sh	Update build.sh	now
dockerfile	react-files	19 minutes ago
package-lock.json	react-files	19 minutes ago
package.json	react-files	19 minutes ago

Help people interested in this repository understand your project by adding a README.

Add a README

Step:8 – Setting up Environmental variables:

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

Dashboard >

+ New Item

People

Build History

Project Relationship

Check File Fingerprint

Manage Jenkins

My Views

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

The screenshot shows the Jenkins Global properties configuration page. Under the 'Environment variables' section, a new variable 'DOCKER_USERNAME' is being added with the value 'ravivarman46'. Another variable 'DOCKER_PASS' is listed below it. At the bottom right are 'Save' and 'Apply' buttons.

Name	Value
DOCKER_USERNAME	ravivarman46

Save Apply

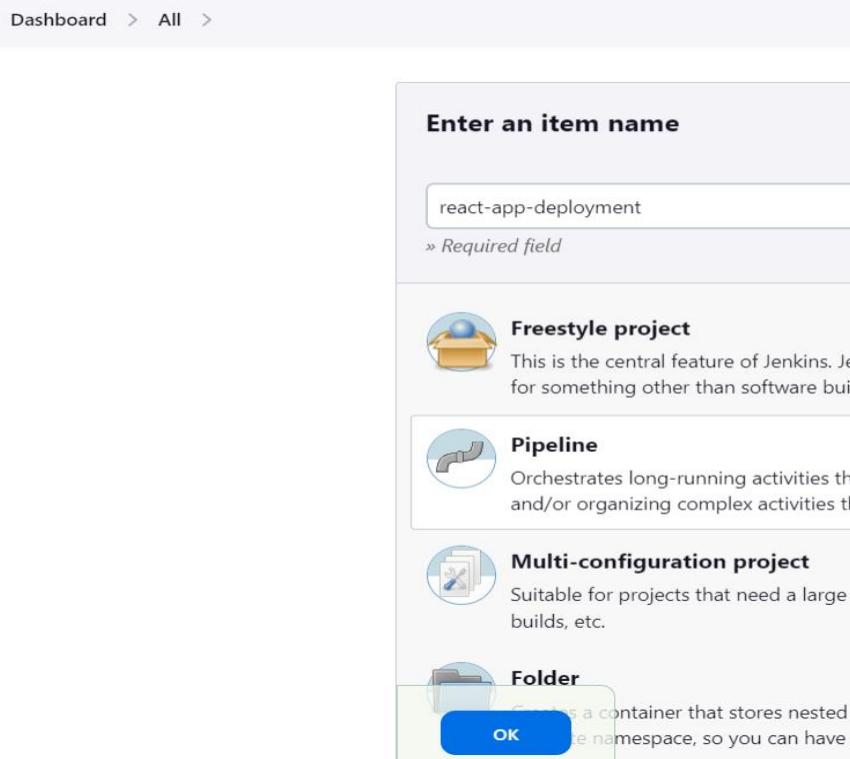
Step:9 - Setting up a CI/CD pipeline:

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

The screenshot shows the Jenkins dashboard with the 'New Item' option highlighted. Other options visible include 'People', 'Build History', 'Manage Jenkins', and 'My Views'.

- + New Item
- People
- Build History
- Manage Jenkins
- My Views

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



→ Then under description give according to this task:

The screenshot shows the Jenkins configuration page for the 'react-app-deployment' project. The top navigation bar shows 'Dashboard > react-app-deployment > Configuration'. In the 'Configure' section, the 'General' tab is selected. The 'Description' field contains the text 'React-app-deployment'.

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

The screenshot shows the Jenkins configuration page for the 'Pipeline' section. The 'Definition' section is expanded, showing the option '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](#)[Changes](#)[Build Now](#)[Configure](#)[Delete Pipeline](#)

react-app-deployment

React-app-deployment

[Full Stage View](#)[Rename](#)[Pipeline Syntax](#)

Stage View

Permalinks

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

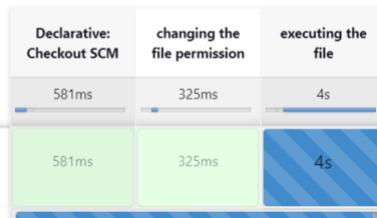
[Status](#)[Changes](#)[Build Now](#)[Configure](#)[Delete Pipeline](#)[Full Stage View](#)[Rename](#)[Pipeline Syntax](#)

react-app-deployment

React-app-deployment

Stage View

Average stage times:



Build History trend ▾

#1 Dec 08 13:57 No Changes

Filter builds... /

Permalinks

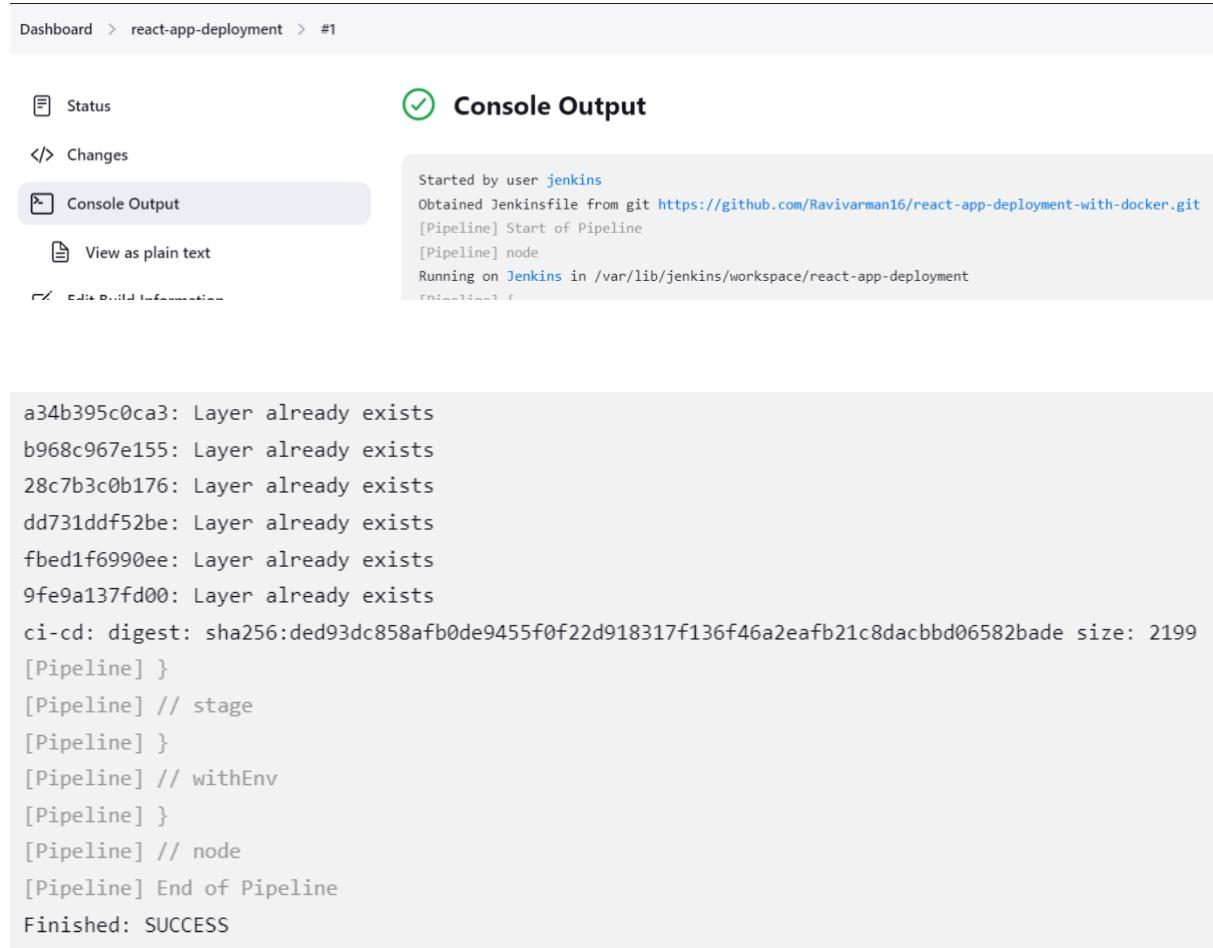
→ We can able to see pipeline job executed successfully, click the job to know more details:

[Status](#)

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

[Changes](#)[Console Output](#)

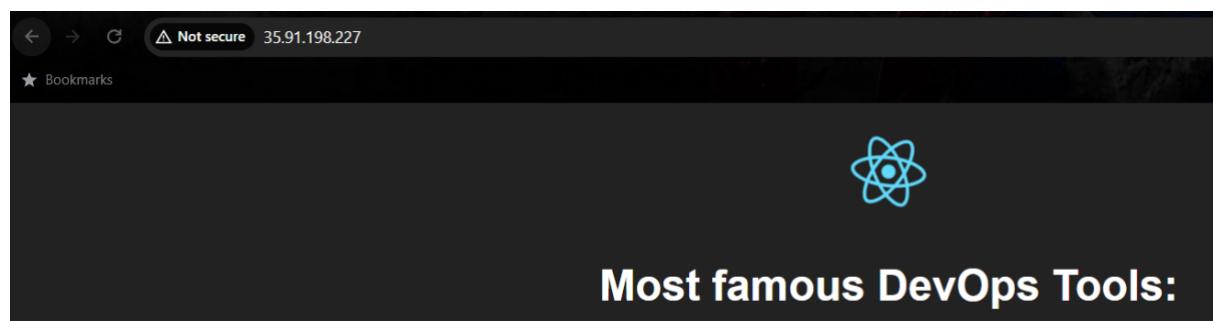
→ Then click console output option: to know details working of this pipeline:

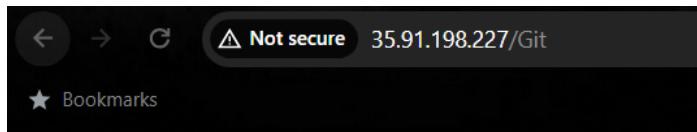


The screenshot shows the Jenkins Pipeline Console Output page for a job named 'react-app-deployment'. The top navigation bar shows 'Dashboard > react-app-deployment > #1'. On the left, there are links for 'Status', 'Changes', 'Console Output' (which is highlighted in blue), and 'View as plain text'. The main content area displays the Jenkins pipeline logs. It starts with a summary of the build environment: 'Started by user jenkins', 'Obtained Jenkinsfile from git https://github.com/Ravivarman16/react-app-deployment-with-docker.git', '[Pipeline] Start of Pipeline', '[Pipeline] node', and 'Running on Jenkins in /var/lib/jenkins/workspace/react-app-deployment'. The log then continues with the execution of the Jenkinsfile stages: 'a34b395c0ca3: Layer already exists', 'b968c967e155: Layer already exists', '28c7b3c0b176: Layer already exists', 'dd731ddf52be: Layer already exists', 'fbcd1f6990ee: Layer already exists', '9fe9a137fd00: Layer already exists', 'ci-cd: digest: sha256:ded93dc858afb0de9455f0f22d918317f136f46a2eafb21c8dacbb06582bade size: 2199', '[Pipeline] }', '[Pipeline] // stage', '[Pipeline] }', '[Pipeline] // withEnv', '[Pipeline] }', '[Pipeline] // node', '[Pipeline] End of Pipeline'. The build concludes with 'Finished: SUCCESS'.

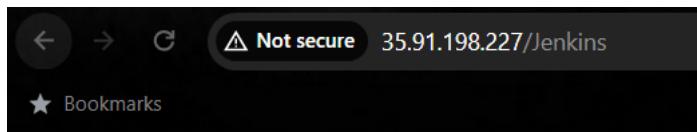
```
a34b395c0ca3: Layer already exists
b968c967e155: Layer already exists
28c7b3c0b176: Layer already exists
dd731ddf52be: Layer already exists
fbcd1f6990ee: Layer already exists
9fe9a137fd00: Layer already exists
ci-cd: digest: sha256:ded93dc858afb0de9455f0f22d918317f136f46a2eafb21c8dacbb06582bade size: 2199
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

→ Browser output of the container:





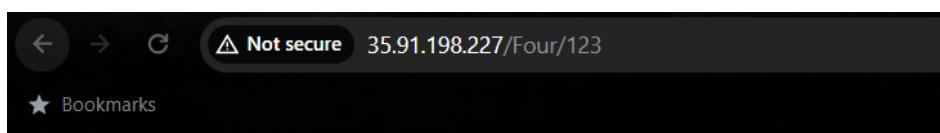
I am a Version Control Tool



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:

The screenshot shows the Docker Hub interface. At the top, there's a blue header with the Docker Hub logo, navigation links for 'Explore', 'Repositories' (which is underlined), and 'Organizations', and a search bar labeled 'Search Docker Hub'. Below the header, there are dropdown menus for 'ravivarman46' and 'Search by repository name' with a magnifying glass icon. To the right of the search bar is another dropdown for 'All Content' and a 'Create repository' button. The main content area displays a repository card for 'ravivarman46 / react-app'. The card shows the repository name, a note that it 'Contains: Image | Last pushed: a minute ago', and status indicators for 'Inactive' (with a gear icon), '0 stars' (with a star icon), '0 forks' (with a fork icon), and 'Public' (with a lock icon). The background of the page is white.

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 [Edit](#)

Last pushed: a minute ago

Docker commands
To push a new tag to this repository:

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

[Public View](#)

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:10 - 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:

 Status

 react-app-deployment

</> Changes

Build Now

 Configure

React-app-deployment

- 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) ?

 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:

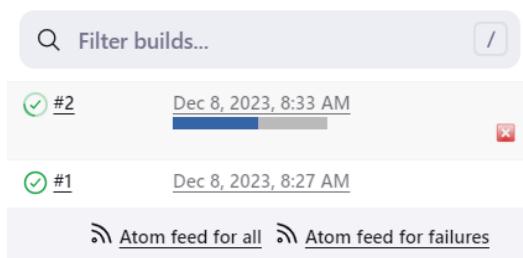
```
root@ip-172-31-19-184:/home/ubuntu/react-app-deployment-with-docker# ls
Jenkinsfile build.sh dockerfile package-lock.json package.json public src
root@ip-172-31-19-184:/home/ubuntu/react-app-deployment-with-docker# touch demo
root@ip-172-31-19-184:/home/ubuntu/react-app-deployment-with-docker# git 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:



→ Job executed successfully:

The screenshot shows a Jenkins pipeline configuration. On the left, there are various options like 'Changes', 'Build Now', 'Configure', 'Delete Pipeline', 'Full Stage View', 'Rename', 'Pipeline Syntax', 'GitHub Hook Log', and 'Build History'. The 'Build History' section shows two builds: #2 (Dec 8, 2023, 8:33 AM) and #1 (Dec 8, 2023, 8:27 AM). The main area is titled 'Stage View' and displays a table of stage times for both builds. The table has three columns: 'Declarative: Checkout SCM', 'changing the file permission', and 'executing the file'. For build #2, the times are 557ms, 335ms, and 12s respectively. For build #1, the times are 533ms, 346ms, and 12s respectively. The table also includes a header row with average stage times: 'Average stage times: (Average full run time: ~13s)'.

	Declarative: Checkout SCM	changing the file permission	executing the file
#2	557ms	335ms	12s
#1	533ms	346ms	12s

Docker Hub output:

ravivarman46 / [Repositories](#) / [react-app](#) / [General](#)

General Tags Builds Collaborators Webhooks Settings

ⓘ Add a short description for this repository
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 edit

Last pushed: a minute ago

Tags

This repository contains 1 tag(s).

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

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>