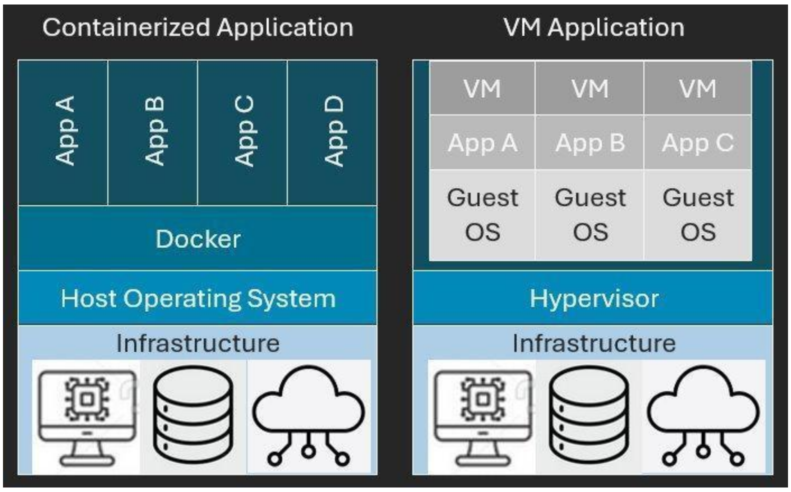
PRACTICAL NO. 5

Continuous Deployment

LOB3 To learn what containers are, using Docker, and the benefits they offer in terms of consistency, scalability, and efficiency.

LO3 Implement and evaluate containerized applications using Docker.

- A Virtual Machine (VM) is a software-based simulation of a physical computer that runs an operating system (OS) and applications just like a physical machine.
- It operates within a host machine but is completely isolated, meaning it has its own virtual CPU, memory, storage, and network resources.
- To host our apps, we need Infrastructure that may be hardware or cloud-based infrastructure.
- Virtual Machines (VMs) allow multiple applications to run on a single physical machine but require an OS for each instance. This adds OS cost, OS maintenance cost etc.
- Each service gets its own VM to ensure isolation, avoiding conflicts between applications.
- Running separate VMs for different services leads to increased resource consumption.
- Thus, Capital Expenditure (CapEx) and Operational Expenditure (OpEx) increases.
- To achive multiple services running in same OS but in isolatation we are using conatiners.
- **A container is a runtime instance of an image. When you run an image, it becomes a container. Containers are isolated environments that run applications with their own filesystem, networking, and processes, but share the host system's kernel.**
- **Image:** A lightweight, stand-alone, executable software package that includes everything needed to run a piece of software, including code, runtime, system tools, libraries, and settings.
- Images are like templates for creating containers or they are the building blocks of containers.
- Images are created using a Dockerfile, which defines the steps to build the image.
- Containers are instances of images that run in isolated environment, or they are the running instances of images.



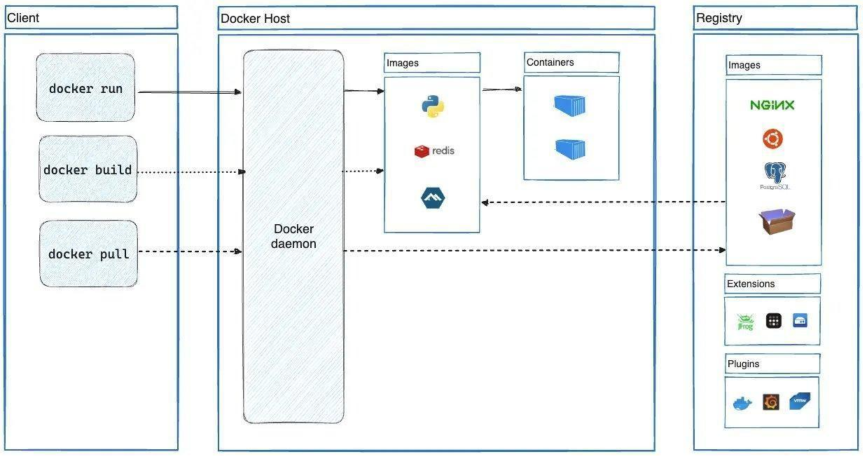
- **Containerization** is a lightweight alternative to full machine virtualization. It involves encapsulating an application and its dependencies into a container that can run on any system with a compatible container runtime (e.g., Docker).

Docker -

- Docker is a platform that enables developers to build, package, and deploy applications in containers.
- Docker is an open-source platform that automates the deployment, scaling, and management of applications using containerization.
- Docker enables you to separate your applications from your infrastructure so you can deliver software quickly.

Docker architecture

- Docker uses a client-server architecture.
- The Docker client talks to the Docker daemon, which does the heavy lifting of building, running, and distributing your Docker containers.
- The Docker client and daemon can run on the same system, or you can connect a Docker client to a remote Docker daemon.
- The Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface.



Docker Components

- Docker CLI - Command-line interface to interact with Docker.
- Docker Daemon - Background service managing containers.
- Dockerfile - A script/text file with instructions to build an image automatically.
- Docker Compose - Tool to define and run multi-container applications.
- Docker Hub - A cloud-based repository for sharing Docker images.
- Images: The read-only templates used to create containers.
- Containers: The running instances of images.

- Volumes: Persistent storage for containers.
- Networks: Isolated networks for communication between containers.

Docker Commands

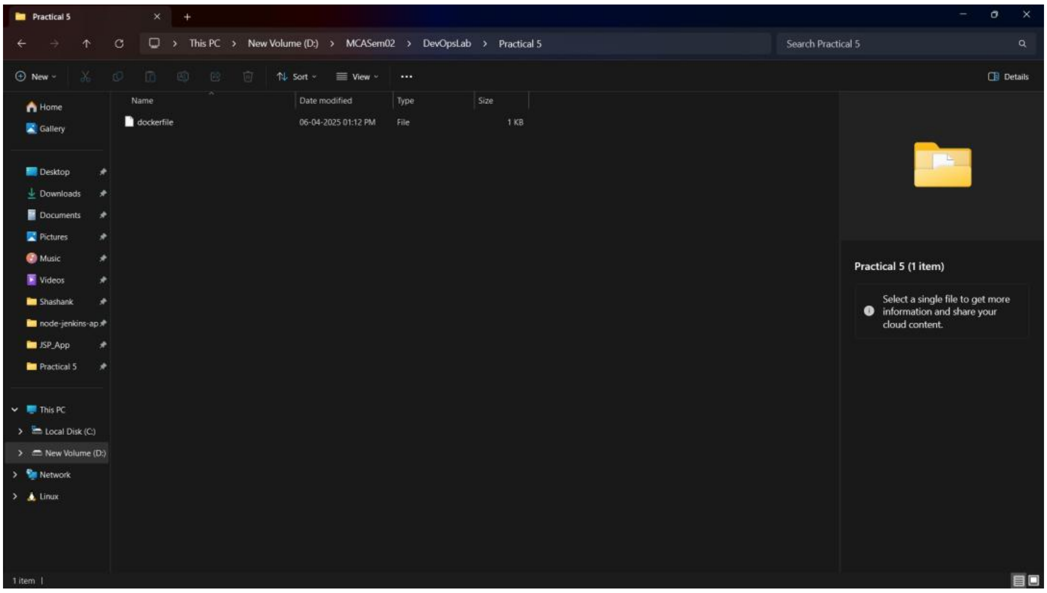
- build Command -
    - Docker build command is used to build an image using a Dockerfile.
    - Basic build command is as follows which will execute the Dockerfile in the project directory and do whatever instructions from the current directory files.
        - docker build .
    - To tag or name the image one can use -
        - docker build -t <docker_username>/<application_name>:version .
- Run command -
    - After building the image there are several options we must add when we run an image and basic command is as follows -
    - Docker run <image>
    - Image can be identified from the id or form the name if we tagged it.

- **The "dockerfile"**
    - A "dockerfile" consists of the following details:
    - FROM: This parameter enables you to specify the base image that will be used to build the container.
    - RUN: This command is used to install dependencies that are needed by your application in your container as specified by the package.js file.
    - COPY: This command replicates files from the current directory on your local machine into the Docker image.
    - CMD: This is short for command. This parameter is used to state the command that will be run when the container starts running.
    - ENV: This is used for setting environment variables.

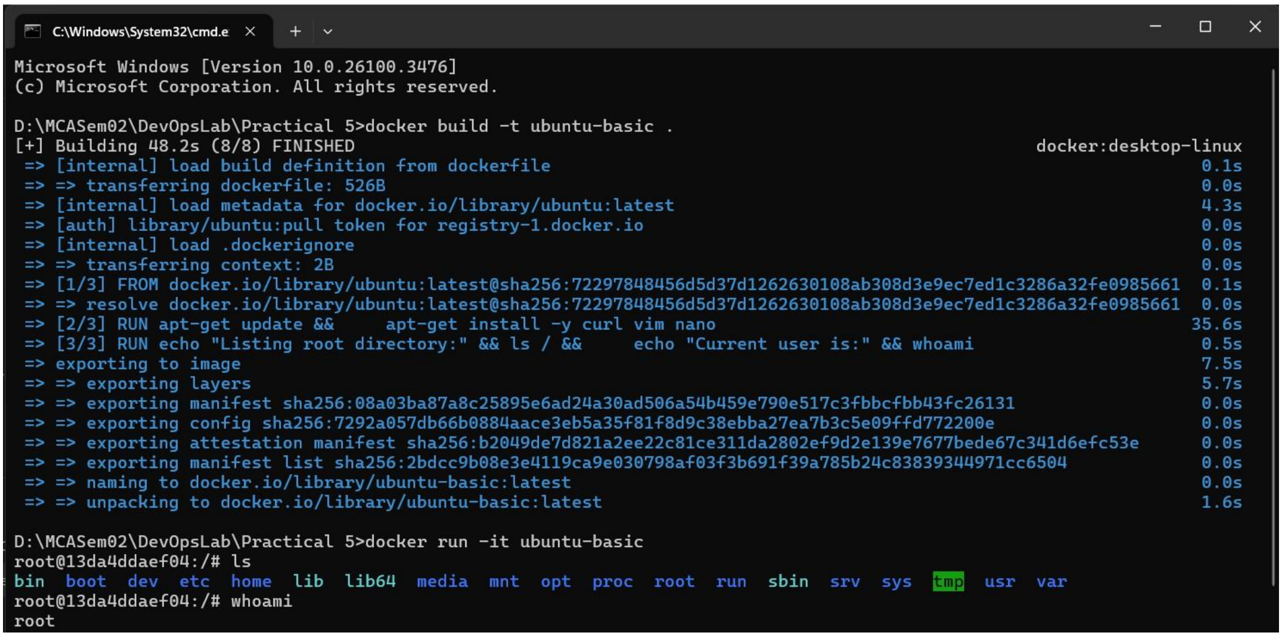
- Build, deploy and manage web application on Docker Engine
    - Create a working directory and index.html in it.
    - Select a base image and write a "dockerfile" file
    - Build the image using the Docker build command
    - Verify and Run the Docker image
    - Access the application

Exercise: -

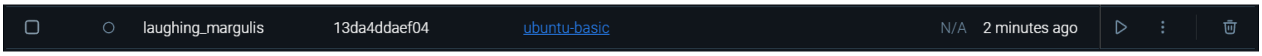
1. Write a dockerfile to create a containerized environment using an official base image of Ubuntu and execute basic commands inside it (like ls, whoami etc).



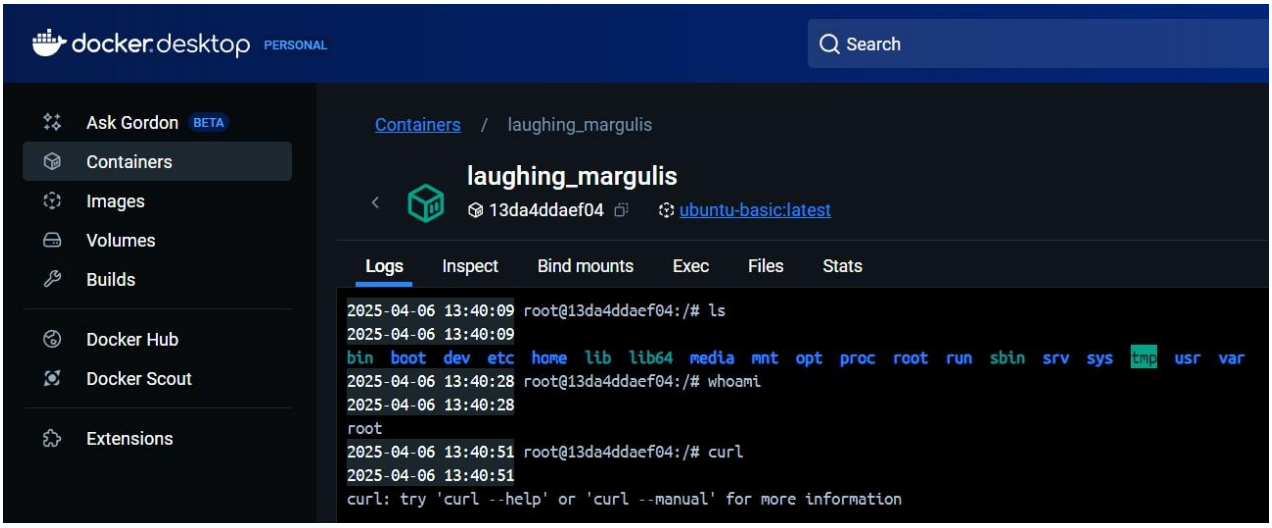
Name	Date modified	Type	Size
dockerfile	06-04-2025 01:12 PM	File	1 KB



```
Microsoft Windows [Version 10.0.26100.3476]
(c) Microsoft Corporation. All rights reserved.

D:\MCASem02\DevOpsLab\Practical 5>docker build -t ubuntu-basic .
[+] Building 48.2s (8/8) FINISHED                                                       docker:desktop-linux
 => [internal] load build definition from dockerfile                                                   0.1s
 => => transferring dockerfile: 526B                                                                   0.0s
 => [internal] load metadata for docker.io/library/ubuntu:latest                                       4.3s
 => [auth] library/ubuntu:pull token for registry-1.docker.io                                          0.0s
 => [internal] load .dockerignore                                                                      0.0s
 => => transferring context: 2B                                                                        0.0s
 => [1/3] FROM docker.io/library/ubuntu:latest@sha256:72297848456d5d37d1262630108ab308d3e9ec7ed1c3286a32fe0985661  0.1s
 => => resolve docker.io/library/ubuntu:latest@sha256:72297848456d5d37d1262630108ab308d3e9ec7ed1c3286a32fe0985661  0.0s
 => [2/3] RUN apt-get update &&     apt-get install -y curl vim nano                                  35.6s
 => [3/3] RUN echo "Listing root directory:" && ls / &&     echo "Current user is:" && whoami          0.5s
 => exporting to image                                                                                 7.5s
 => => exporting layers                                                                                5.7s
 => => exporting manifest sha256:08a03ba87a8c25895e6ad24a30ad506a54b459e790e517c3fbbcfbb43fc26131      0.0s
 => => exporting config sha256:7292a057db66b0884aace3eb5a35f81f8d9c38ebba27ea7b3c5e09ffd772200e        0.0s
 => => exporting attestation manifest sha256:b2049de7d821a2ee22c81ce311da2802ef9d2e139e7677bede67c341d6efc53e  0.0s
 => => exporting manifest list sha256:2bdcc9b08e3e4119ca9e030798af03f3b691f39a785b24c83839344971cc6504  0.0s
 => => naming to docker.io/library/ubuntu-basic:latest                                                 0.0s
 => => unpacking to docker.io/library/ubuntu-basic:latest                                              1.6s

D:\MCASem02\DevOpsLab\Practical 5>docker run -it ubuntu-basic
root@13da4ddaef04:/# ls
bin  boot  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
root@13da4ddaef04:/# whoami
root
```



□ ○ laughing_margulis 13da4ddaef04 ubuntu-basic N/A 2 minutes ago

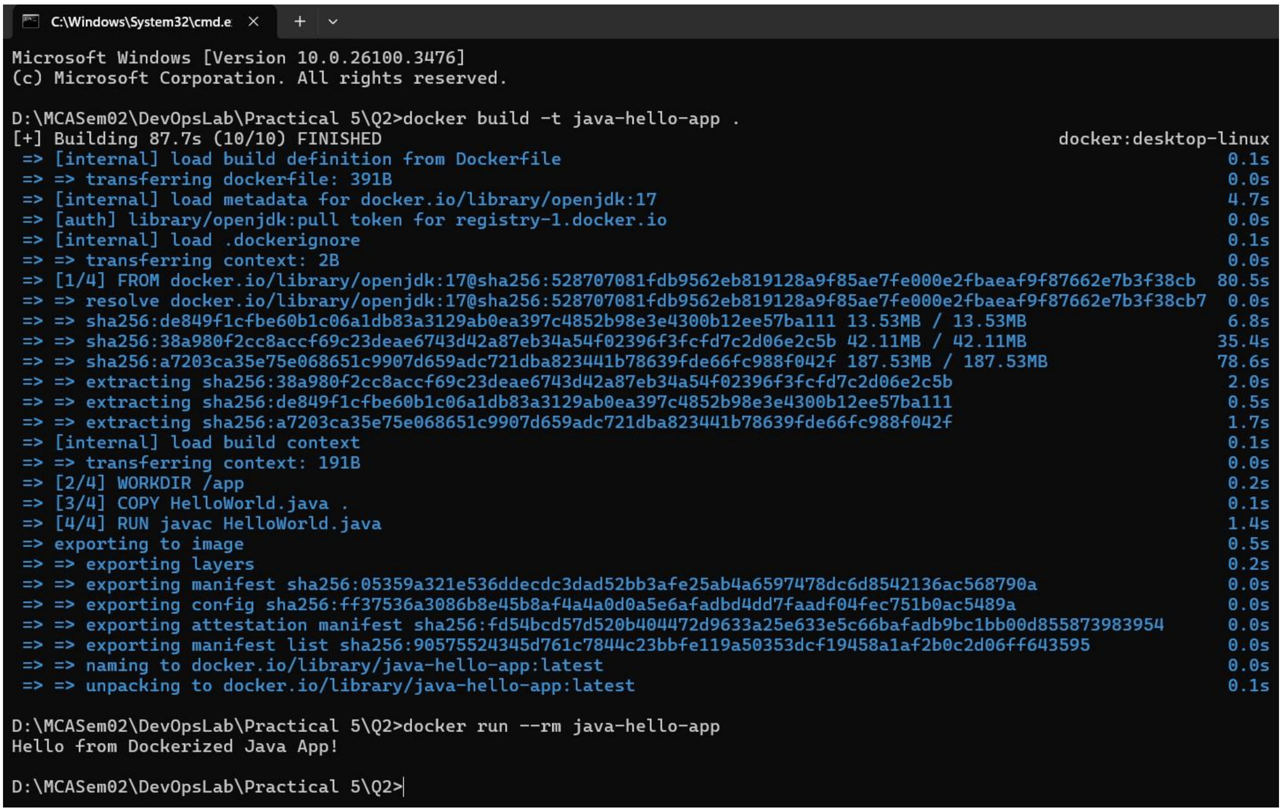


```
2025-04-06 13:40:09 root@13da4ddaef04:/# ls
2025-04-06 13:40:09
bin  boot  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
2025-04-06 13:40:28 root@13da4ddaef04:/# whoami
2025-04-06 13:40:28
root
2025-04-06 13:40:51 root@13da4ddaef04:/# curl
2025-04-06 13:40:51
curl: try 'curl --help' or 'curl --manual' for more information
```

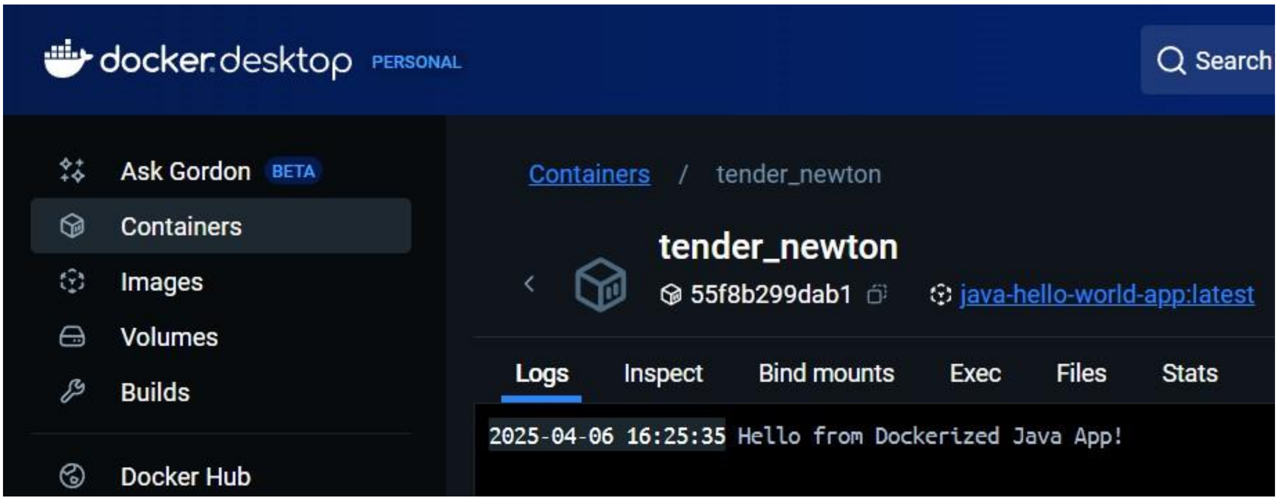
2. Write a Dockerfile that explicitly pulls an OpenJDK image before using it as a base, compiles HelloWorld.java, and runs the program inside a containerized environment.

Dockefile :-

```
# Step 1: Pull the official OpenJDK image FROM
openjdk:1
# Step 2: Set working directory inside the container
WORKDIR /ap
# Step 3: Copy the Java source file into the container
COPY HelloWorld.java
# Step 4: Compile the Java source file
RUN javac HelloWorld.ja
# Step 5: Run the compiled Java program
CMD ["java", "HelloWorld"]
```



```
Microsoft Windows [Version 10.0.26100.3476]
(c) Microsoft Corporation. All rights reserved.

D:\MCASem02\DevOpsLab\Practical 5\Q2>docker build -t java-hello-app .
[+] Building 87.7s (10/10) FINISHED                                                          docker:desktop-linux
 => [internal] load build definition from Dockerfile                                                         0.1s
 => => transferring dockerfile: 391B                                                                         0.0s
 => [internal] load metadata for docker.io/library/openjdk:17                                                4.7s
 => [auth] library/openjdk:pull token for registry-1.docker.io                                               0.0s
 => [internal] load .dockerignore                                                                            0.1s
 => => transferring context: 2B                                                                              0.0s
 => [1/4] FROM docker.io/library/openjdk:17@sha256:528707081fdb9562eb819128a9f85ae7fe000e2fbaeaf9f87662e7b3f38cb  80.5s
 => => resolve docker.io/library/openjdk:17@sha256:528707081fdb9562eb819128a9f85ae7fe000e2fbaeaf9f87662e7b3f38cb7  0.0s
 => => sha256:de849f1cfbe60b1c06a1db83a3129ab0ea397c4852b98e3e4300b12ee57ba111 13.53MB / 13.53MB              6.8s
 => => sha256:38a980f2cc8accf69c23deae6743d42a87eb34a54f02396f3fcfd7c2d06e2c5b 42.11MB / 42.11MB             35.4s
 => => sha256:a7203ca35e75e068651c9907d659adc721dba823441b78639fde66fc988f042f 187.53MB / 187.53MB           78.6s
 => => extracting sha256:38a980f2cc8accf69c23deae6743d42a87eb34a54f02396f3fcfd7c2d06e2c5b                    2.0s
 => => extracting sha256:de849f1cfbe60b1c06a1db83a3129ab0ea397c4852b98e3e4300b12ee57ba111                    0.5s
 => => extracting sha256:a7203ca35e75e068651c9907d659adc721dba823441b78639fde66fc988f042f                    1.7s
 => [internal] load build context                                                                            0.1s
 => => transferring context: 191B                                                                            0.0s
 => [2/4] WORKDIR /app                                                                                       0.2s
 => [3/4] COPY HelloWorld.java .                                                                             0.1s
 => [4/4] RUN javac HelloWorld.java                                                                          1.4s
 => exporting to image                                                                                       0.5s
 => => exporting layers                                                                                      0.2s
 => => exporting manifest sha256:05359a321e536ddecdc3dad52bb3afe25ab4a6597478dc6d8542136ac568790a            0.0s
 => => exporting config sha256:ff37536a3086b8e45b8af4a4a0d0a5e6afadbd4dd7faadf04fec751b0ac5489a              0.0s
 => => exporting attestation manifest sha256:fd54bcd57d520b404472d9633a25e633e5c66bafadb9bc1bb00d855873983954  0.0s
 => => exporting manifest list sha256:90575524345d761c7844c23bbfe119a50353dcf19458a1af2b0c2d06ff643595        0.0s
 => => naming to docker.io/library/java-hello-app:latest                                                     0.0s
 => => unpacking to docker.io/library/java-hello-app:latest                                                  0.1s

D:\MCASem02\DevOpsLab\Practical 5\Q2>docker run --rm java-hello-app
Hello from Dockerized Java App!

D:\MCASem02\DevOpsLab\Practical 5\Q2>
```



```
2025-04-06 16:25:35 Hello from Dockerized Java App!
```

3. Create a Jenkins pipeline to build and deploy a Dockerized Java application (like Exercise-2 mentioned above) automatically. Use GitHub repo to maintain the dockerfile, Jenkinsfile and source code.

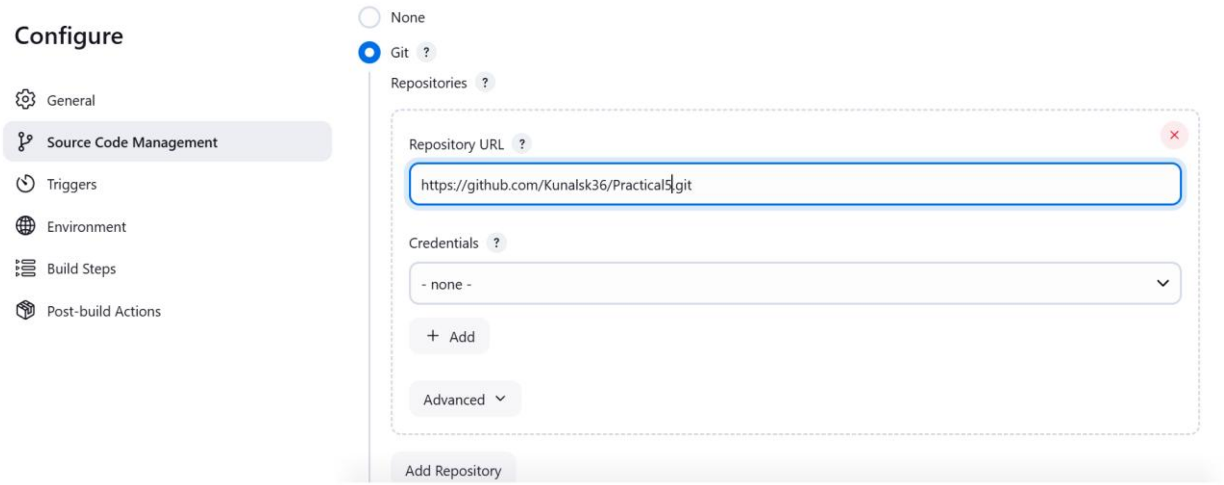
Dockerfile : -

```
# Pull the OpenJDK base image
FROM openjdk:17-slim

# Set working directory
WORKDIR /app

# Copy Java file into the container
COPY HelloWorld.java .

# Compile the Java file
RUN javac HelloWorld.java

# Run the Java program
CMD ["java", "HelloWorld"]
```

Jenkinsfile : -

```
pipeline {
  agent any

  stages {
    stage('Clone Repository') {
  steps {
    git branch: 'main', url: 'https://github.com/TEJASBHIDE/Practical5.git'
  }
}
    stage('Build Docker Image') {
      steps {
        script {
          docker.build("java-hello-world-app")
```

```
                }
            }
        }

        stage('Run Docker Container') {
            steps {
                script {
                    docker.image("java-hello-world-app").run()
                }
            }
        }
    }
}
```



○ None

● Git ?

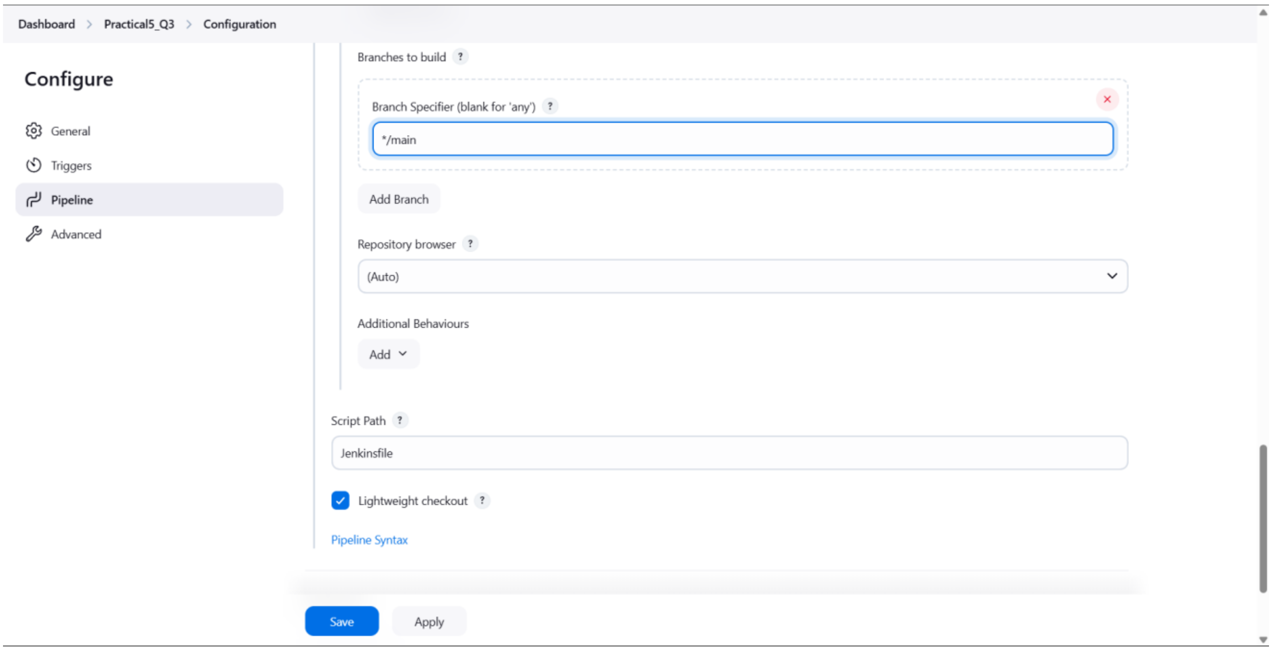
Repositories ?

Repository URL ?

https://github.com/Kunalsk36/Practical5.git

Credentials ?

- none -



Branches to build ?

Branch Specifier (blank for 'any') ?

*/main

Repository browser ?

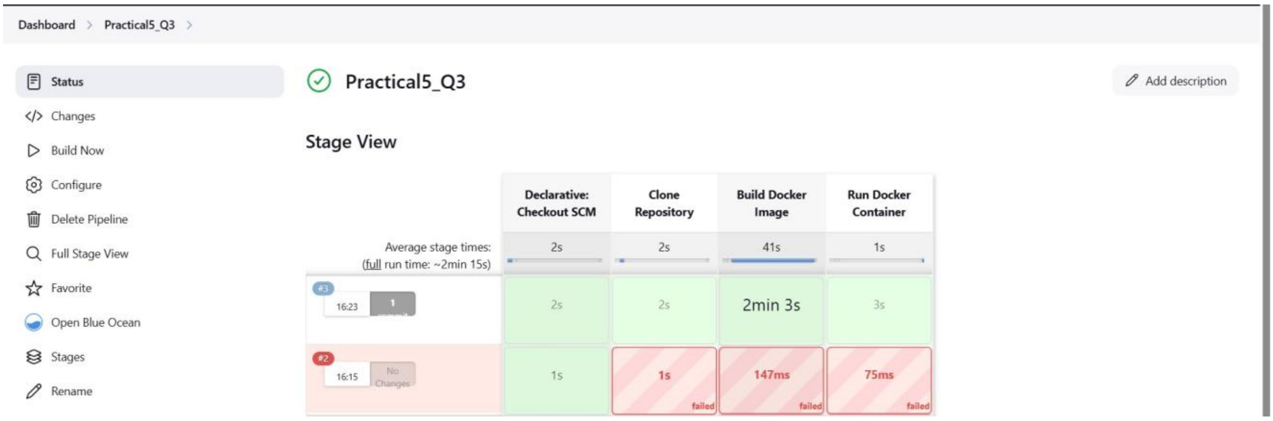
(Auto)

Additional Behaviours

Script Path ?

Jenkinsfile

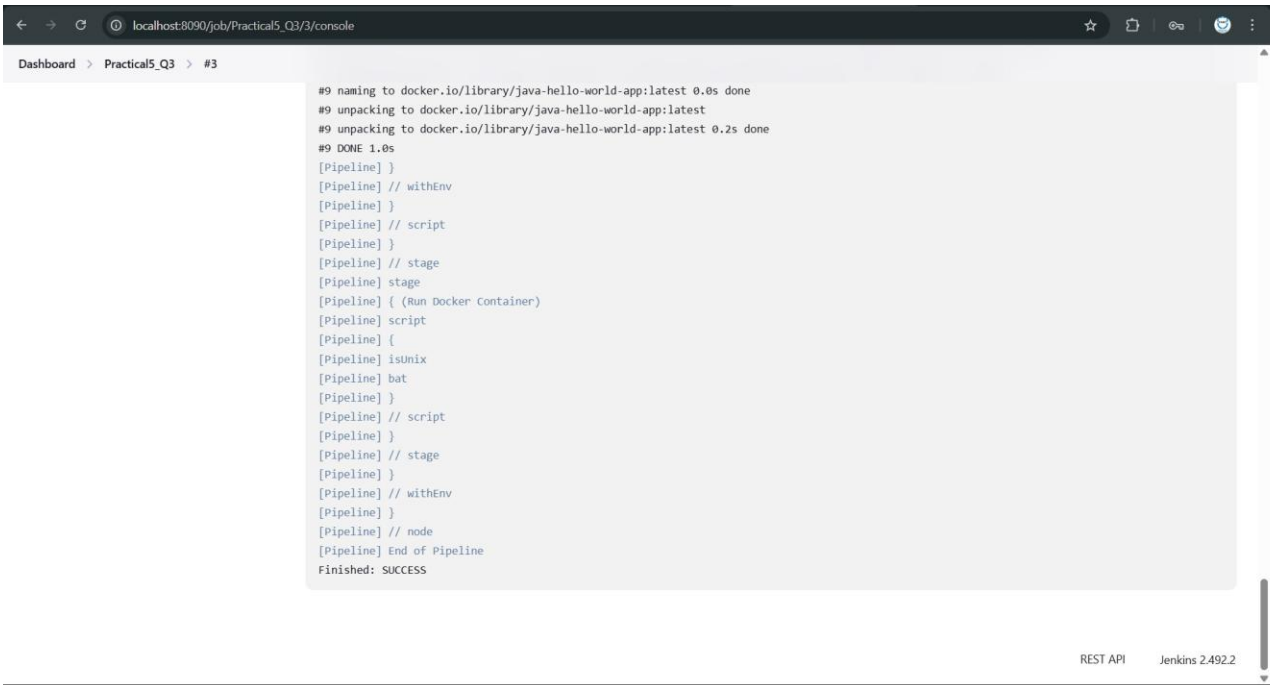
☑ Lightweight checkout ?



Practical5_Q3

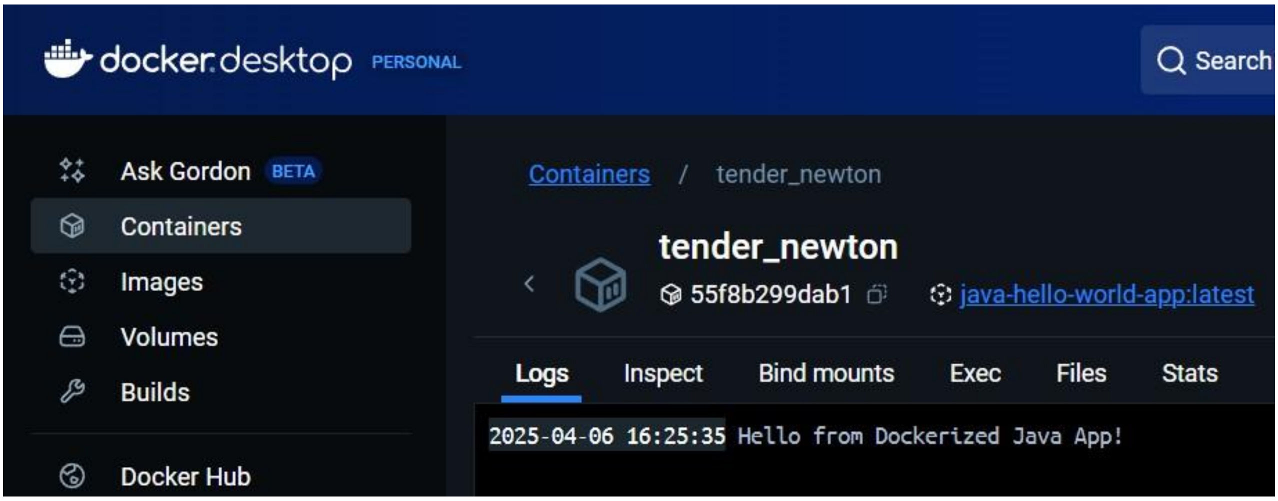
Stage View

	Declarative: Checkout SCM	Clone Repository	Build Docker Image	Run Docker Container
Average stage times: (full run time: ~2min 15s)	2s	2s	41s	1s
#3 16:23 1 commit	2s	2s	2min 3s	3s
#2 16:15 No Changes	1s	1s failed	147ms failed	75ms failed



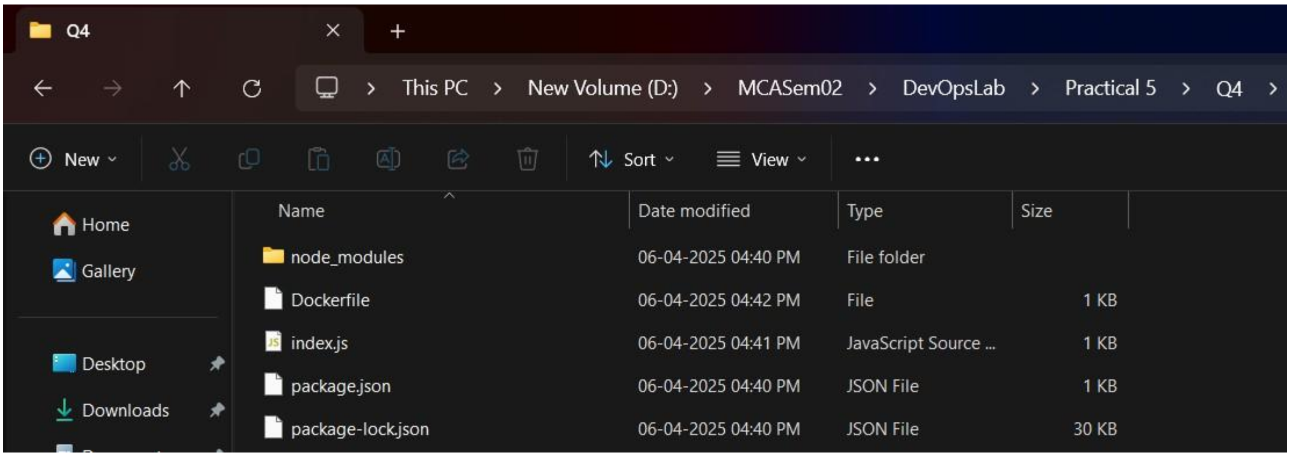
localhost:8090/job/Practical5_Q3/3/console
Dashboard > Practical5_Q3 > #3

```
#9 naming to docker.io/library/java-hello-world-app:latest 0.0s done
#9 unpacking to docker.io/library/java-hello-world-app:latest
#9 unpacking to docker.io/library/java-hello-world-app:latest 0.2s done
#9 DONE 1.0s
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // script
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Run Docker Container)
[Pipeline] script
[Pipeline] {
[Pipeline] isUnix
[Pipeline] bat
[Pipeline] }
[Pipeline] // script
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```



```
2025-04-06 16:25:35 Hello from Dockerized Java App!
```

4. Develop a simple containerized application using Docker for NodeJS web application.

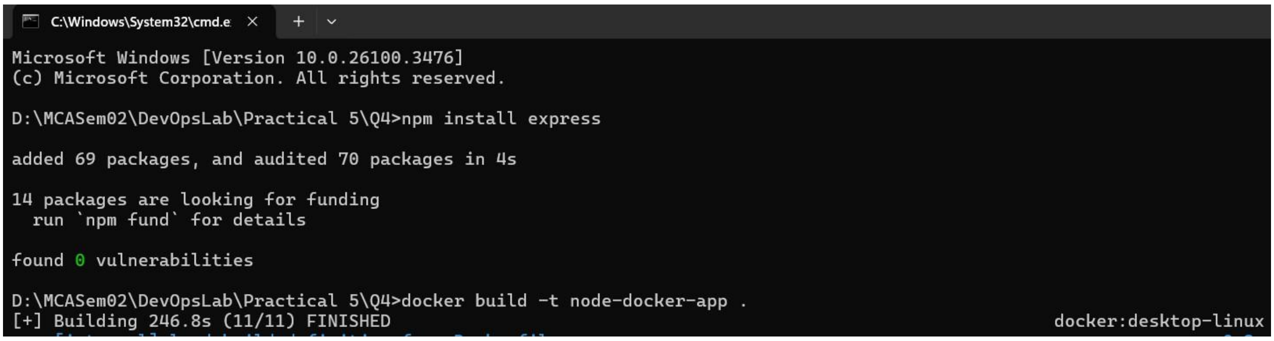


This PC > New Volume (D:) > MCASem02 > DevOpsLab > Practical 5 > Q4 >

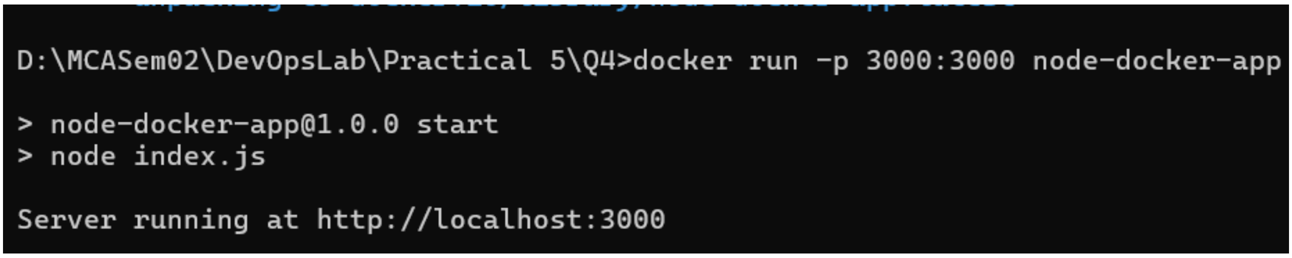
Name	Date modified	Type	Size
node_modules	06-04-2025 04:40 PM	File folder	
Dockerfile	06-04-2025 04:42 PM	File	1 KB
index.js	06-04-2025 04:41 PM	JavaScript Source ...	1 KB
package.json	06-04-2025 04:40 PM	JSON File	1 KB
package-lock.json	06-04-2025 04:40 PM	JSON File	30 KB

Dockerfile :-

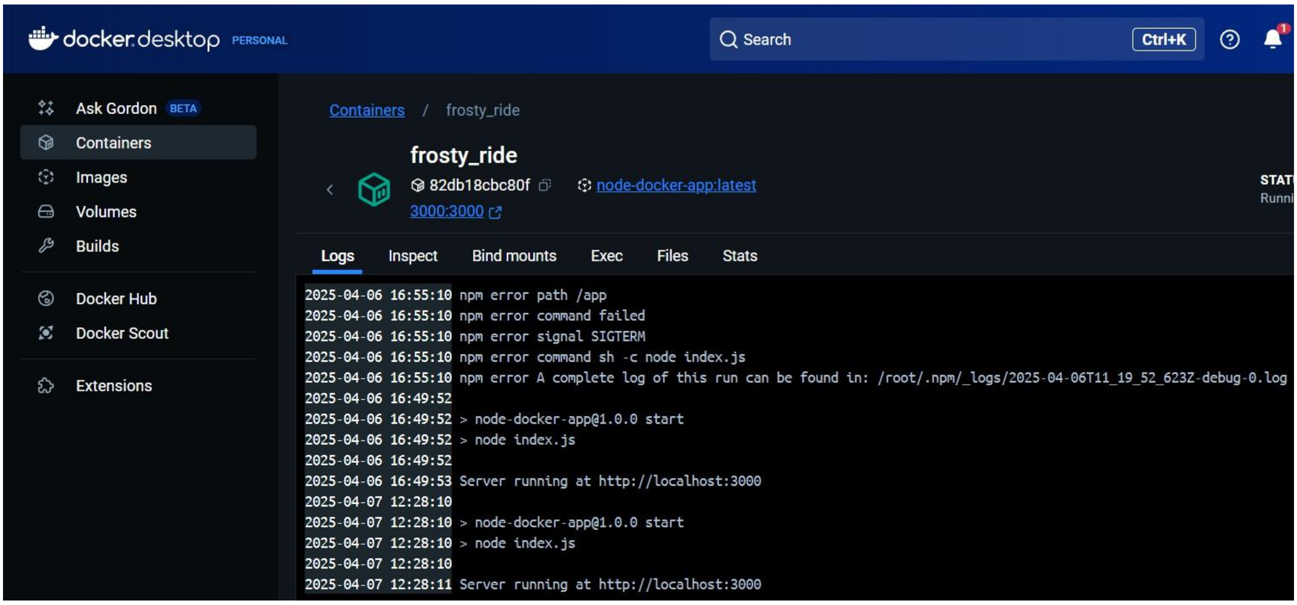
```
# Use NodeJS base image
FROM node:18

# Set working directory
WORKDIR /app

# Copy package.json and install dependencies
COPY package*.json ./
RUN npm install

# Copy the rest of the app
COPY . .

# Expose the port
EXPOSE 3000

# Command to run the app
CMD ["npm", "start"]
```



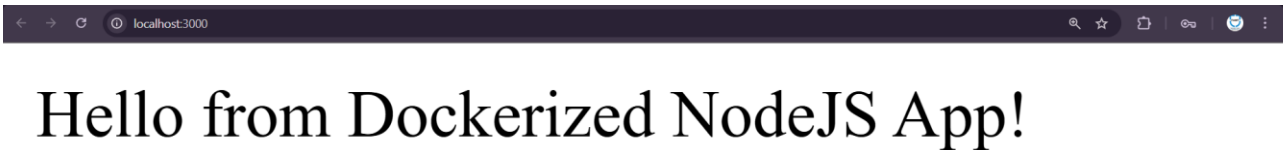
```
Microsoft Windows [Version 10.0.26100.3476]
(c) Microsoft Corporation. All rights reserved.

D:\MCASem02\DevOpsLab\Practical 5\Q4>npm install express

added 69 packages, and audited 70 packages in 4s

14 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities

D:\MCASem02\DevOpsLab\Practical 5\Q4>docker build -t node-docker-app .
[+] Building 246.8s (11/11) FINISHED                                         docker:desktop-linux
```



```
D:\MCASem02\DevOpsLab\Practical 5\Q4>docker run -p 3000:3000 node-docker-app

> node-docker-app@1.0.0 start
> node index.js

Server running at http://localhost:3000
```

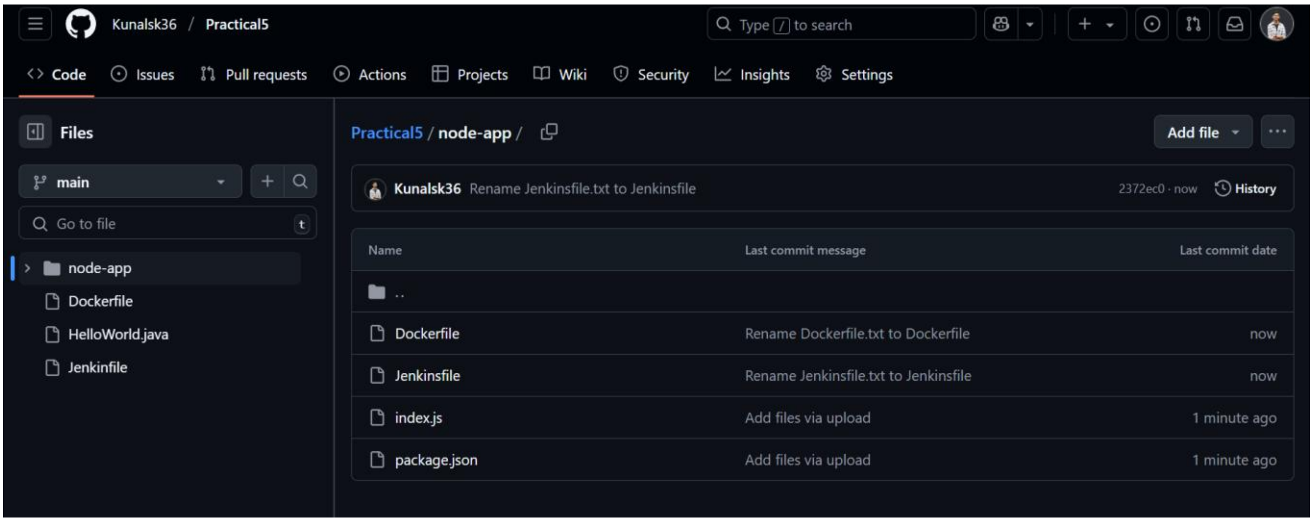


```
2025-04-06 16:55:10 npm error path /app
2025-04-06 16:55:10 npm error command failed
2025-04-06 16:55:10 npm error signal SIGTERM
2025-04-06 16:55:10 npm error command sh -c node index.js
2025-04-06 16:55:10 npm error A complete log of this run can be found in: /root/.npm/_logs/2025-04-06T11_19_52_623Z-debug-0.log
2025-04-06 16:49:52
2025-04-06 16:49:52 > node-docker-app@1.0.0 start
2025-04-06 16:49:52 > node index.js
2025-04-06 16:49:52
2025-04-06 16:49:53 Server running at http://localhost:3000
2025-04-07 12:28:10
2025-04-07 12:28:10 > node-docker-app@1.0.0 start
2025-04-07 12:28:10 > node index.js
2025-04-07 12:28:10
2025-04-07 12:28:11 Server running at http://localhost:3000
```



Hello from Dockerized NodeJS App!

5. Create a Jenkins pipeline script to build and deploy a Dockerized NodeJS application (like Exercise-4 mentioned above) automatically. Use GitHub repo to maintain the dockerfile and source code.



Kunalsk36 Rename Jenkinsfile.txt to Jenkinsfile 2372ec0 · now

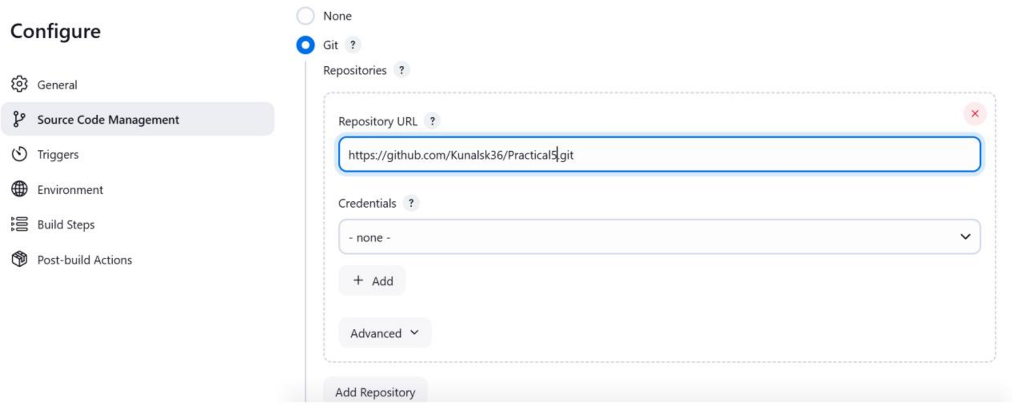
Name	Last commit message	Last commit date
..		
Dockerfile	Rename Dockerfile.txt to Dockerfile	now
Jenkinsfile	Rename Jenkinsfile.txt to Jenkinsfile	now
index.js	Add files via upload	1 minute ago
package.json	Add files via upload	1 minute ago

Dockerfile :-

```
FROM node:14
WORKDIR /app
COPY . .
RUN npm install
EXPOSE 3000
CMD ["node", "index.js"]
```

Jenkinsfile :-

```
pipeline {
  agent any

  stages {
    stage('Clone') {
      steps {
        echo 'Cloning the repository...'
        // Repo already cloned by Jenkins
      }
    }

    stage('Build Docker Image') {
      steps {
        script {
          echo 'Building Docker image...'
          dir('node-app') {
            bat 'docker build -t node-jenkins-app .'
          }
        }
      }
    }

    stage('Run Docker Container') {
            steps {
                script {
                    echo 'Running Docker container...'
                    bat 'docker run -d --name node-jenkins-container -p 3000:3000 node-jenkins-app'
                }
            }
        }
    }

    post {
        success {
            echo 'Pipeline completed successfully.'
        }
        failure {
            echo 'Pipeline failed.'
        }
    }
}
```



Configure

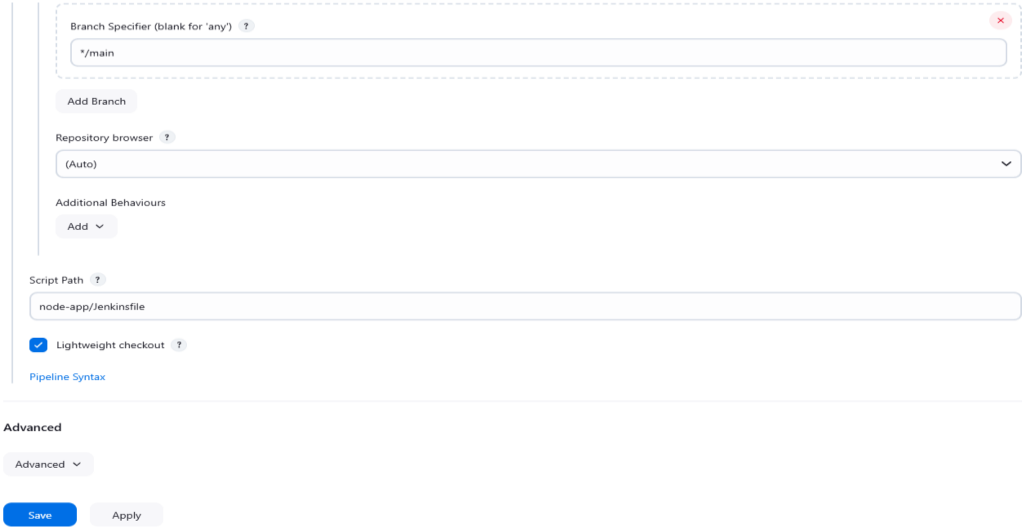
○ None

● Git ?

Repositories ?

Repository URL ?
https://github.com/Kunalsk36/Practical5.git

Credentials ?
- none -



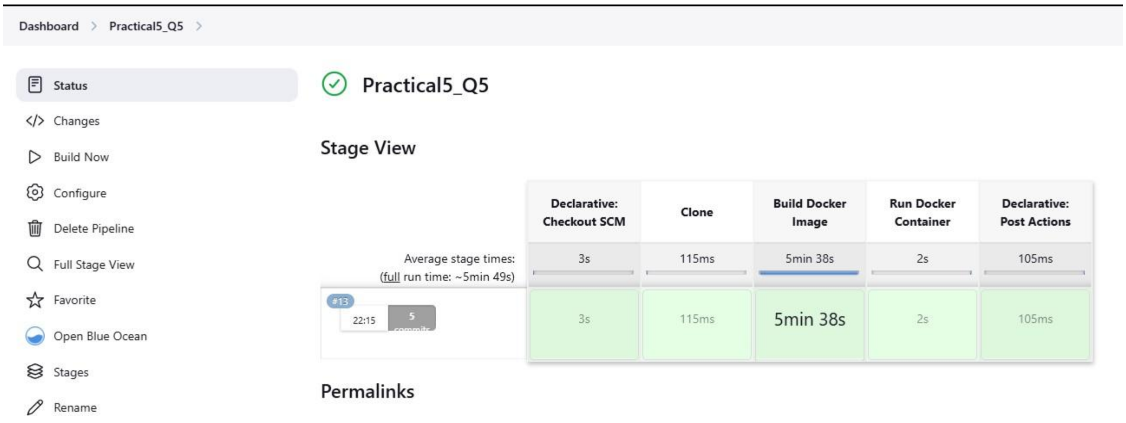
Branch Specifier (blank for 'any') ?
*/main

Repository browser ?
(Auto)

Additional Behaviours

Script Path ?
node-app/Jenkinsfile

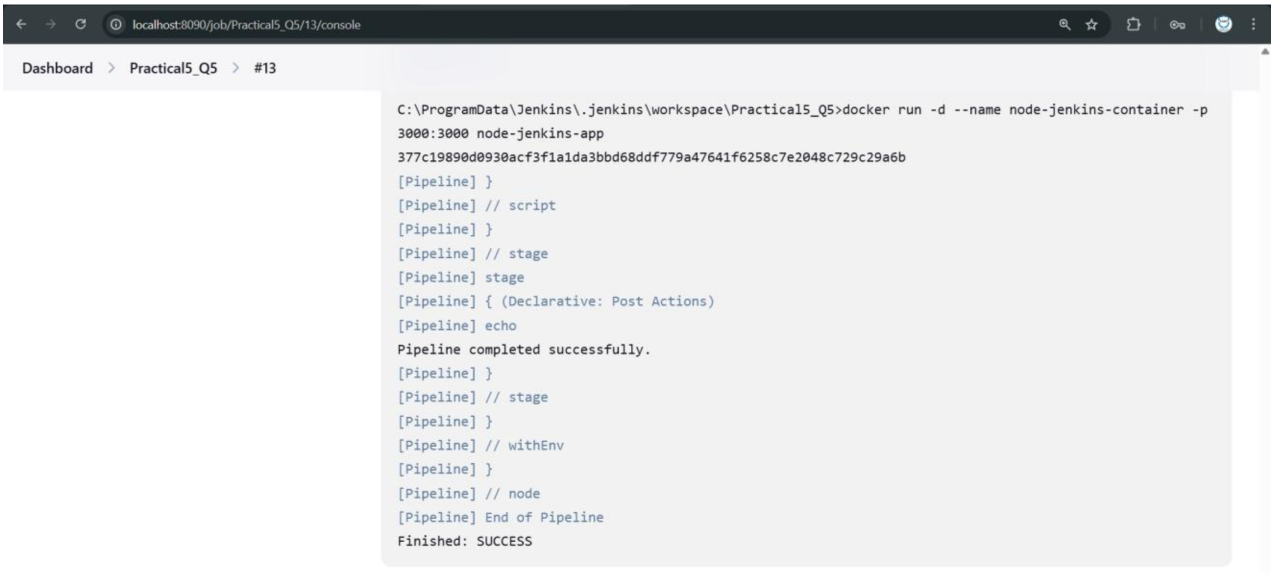
☑ Lightweight checkout ?



Practical5_Q5

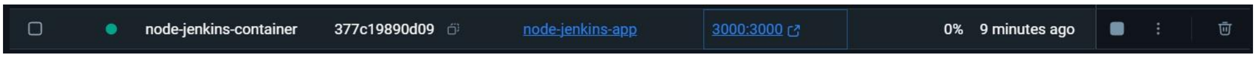
Stage View

	Declarative: Checkout SCM	Clone	Build Docker Image	Run Docker Container	Declarative: Post Actions
Average stage times: (full run time: ~5min 49s)	3s	115ms	5min 38s	2s	105ms
#13 22:15 5 commits	3s	115ms	5min 38s	2s	105ms

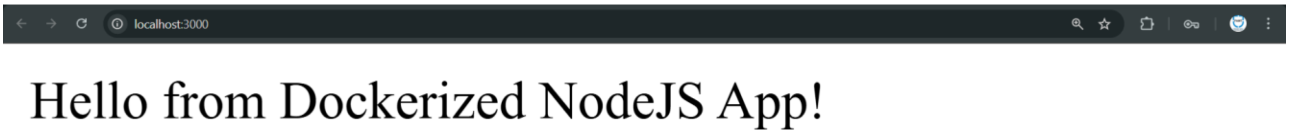

localhost:8090/job/Practical5_Q5/13/console
Dashboard > Practical5_Q5 > #13

```
C:\ProgramData\Jenkins\.jenkins\workspace\Practical5_Q5>docker run -d --name node-jenkins-container -p
3000:3000 node-jenkins-app
377c19890d0930acf3f1a1da3bbd68ddf779a47641f6258c7e2048c729c29a6b
[Pipeline] }
[Pipeline] // script
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Declarative: Post Actions)
[Pipeline] echo
Pipeline completed successfully.
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

Docker image Running:-



□ ● node-jenkins-container 377c19890d09 node-jenkins-app 3000:3000 0% 9 minutes ago



Hello from Dockerized NodeJS App!