Name: Soham Attarde **Batch: T11** **Roll Number: 05**

Experiment 10

AIM: To learn Dockerfile instructions, build an image for a sample web application using DOCKERFILE.

Theory :

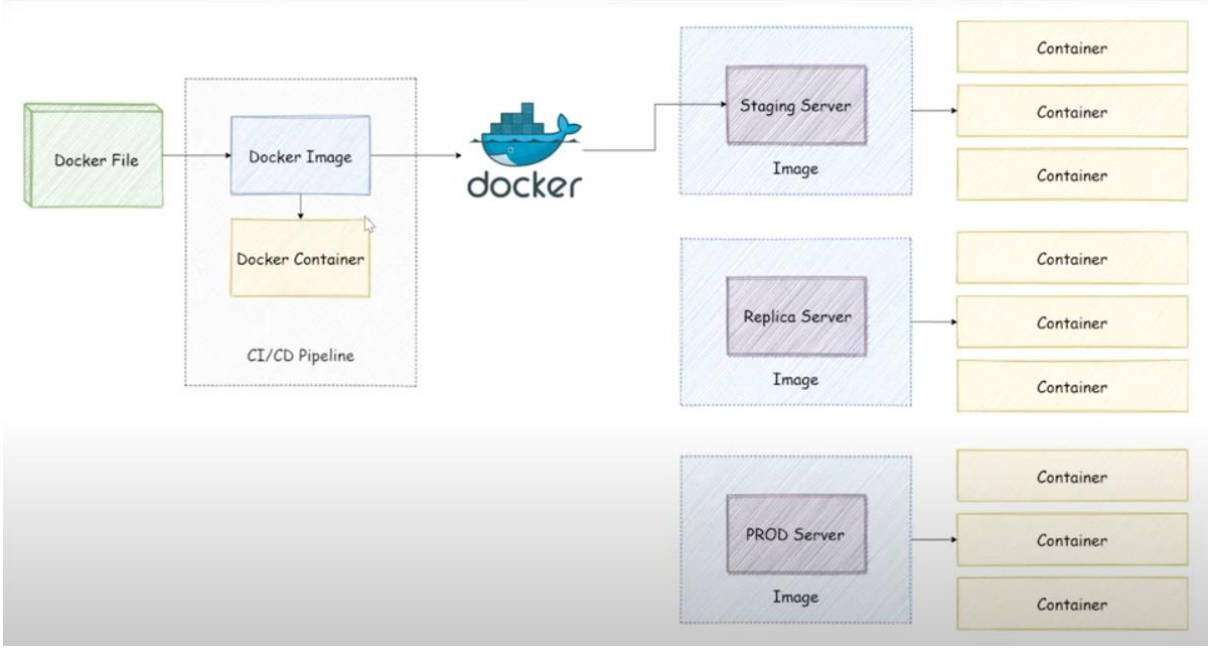
Dockerfiles are the cornerstone of creating Docker images. They contain a set of instructions that automate the process of building a Docker image, specifying everything from the base operating system to the application code, dependencies, and configuration settings.

---

1. What is a Dockerfile?

A Dockerfile is a plain text file that defines the steps required to build a Docker image. It contains a series of commands (or instructions) that specify how the image should be constructed.

- **Purpose**: Automate the creation of Docker images for reproducibility, scalability, and consistency.
- **Format**: Written in a simple scripting language, using instructions like `FROM`, `RUN`, `COPY`, `CMD`, etc.



---

2. Basic Structure of a Dockerfile

```
# Use an official Python runtime as a parent image
FROM python:3.9-slim

# Set the working directory inside the container
WORKDIR /app

# Copy the current directory contents into the container at /app
COPY . /app

# Install any necessary dependencies
RUN pip install --no-cache-dir -r requirements.txt

# Make port 80 available to the world outside this container
EXPOSE 80

# Define environment variable
ENV NAME World

# Run app.py when the container launches
CMD ["python", "app.py"]
```

---

3. Common Dockerfile Instructions

1. FROM (Base Image)

- **Purpose**: Specifies the base image for your Docker image.

Example:

```
FROM ubuntu:20.04
```

FROM node:14

FROM python:3.9-slim

-
- **Note**: This is the first instruction and is mandatory in most cases.

2. WORKDIR (Set Working Directory)

- **Purpose**: Defines the directory inside the container where subsequent instructions will be executed.

Example:

WORKDIR /app

-

3. COPY (Copy Files)

- **Purpose**: Copies files or directories from the host system into the container.

Example:

COPY . /app

-
- **Variants**:
    - `COPY <src> <dest>`
    - `ADD` is similar but supports remote URLs and tar file extraction.

4. RUN (Execute Commands)

- **Purpose**: Executes commands inside the container during the image build process.

Example:

RUN apt-get update && apt-get install -y curl

RUN pip install --no-cache-dir -r requirements.txt

-

- **Tip**: Each `RUN` creates a new layer in the image. Combine commands with `&&` to reduce image size.

5. EXPOSE (Expose Ports)

- **Purpose**: Informs Docker that the container will listen on the specified network ports at runtime.

Example:

```
EXPOSE 80
```

-
- **Note**: This does not publish the port; it's just for documentation.

6. ENV (Set Environment Variables)

- **Purpose**: Sets environment variables inside the container.

Example:

```
ENV APP_ENV=production
```

-

7. CMD (Default Command)

- **Purpose**: Specifies the default command to run when the container starts.

Example:

```
CMD ["python", "app.py"]
```

-
- **Key Points**:
    - Only one `CMD` is allowed.
    - It can be overridden by passing a command with `docker run`.

8. ENTRYPOINT (Set Entry Point)

- **Purpose**: Defines a command that will always be executed when the container starts.

Example:

ENTRYPOINT ["python"]

CMD ["app.py"]

-
- **Difference from CMD**: `ENTRYPOINT` is not overridden unless explicitly done with `--entrypoint`.

---

4. Building Images from a Dockerfile

To build an image:

docker build -t myapp:latest .

- `-t myapp:latest`: Tags the image.
- `.`: Refers to the current directory as build context.

Build Options:

- `-f <file>`: Specify a custom Dockerfile.
- `--no-cache`: Build without using the cache.
- `--build-arg <arg>`: Pass build-time arguments.

---

5. Managing Docker Images List

Images: docker images

Remove an Image: docker

rmi myapp:latest **Run a**

Container:

docker run -p 8080:80 myapp:latest

---

6. Multi-Stage Builds (Advanced)

Multi-stage builds help reduce image size by separating the build environment from runtime:

```
# Stage 1: Build stage
FROM node:14 AS build
WORKDIR /app
COPY package.json ./
RUN npm install
COPY . .

# Stage 2: Production stage
FROM node:14-slim
WORKDIR /app
COPY --from=build /app /app
CMD ["node", "server.js"]
```

This keeps the final image small and excludes unnecessary build tools.

---

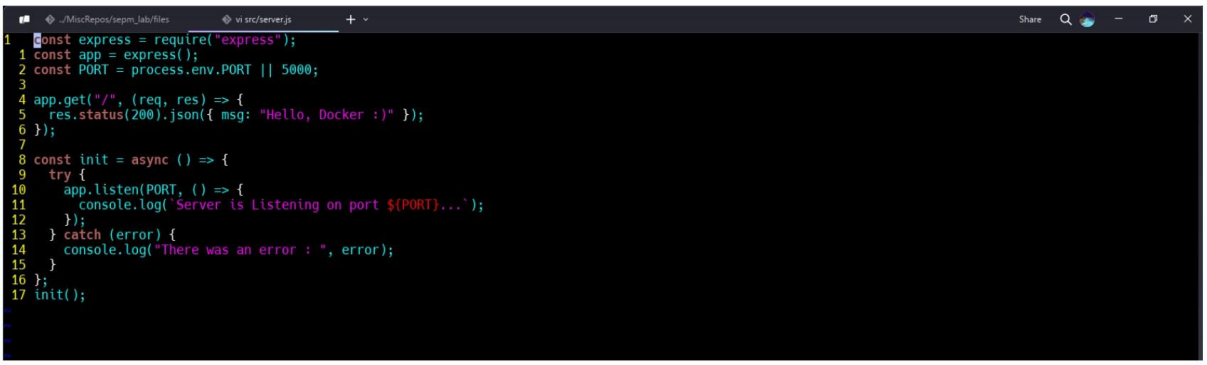
7. Best Practices for Dockerfiles

1. Use minimal base images (e.g., alpine).
2. Order instructions from least to most frequently changing to leverage caching.
3. Combine RUN commands with `&&`.

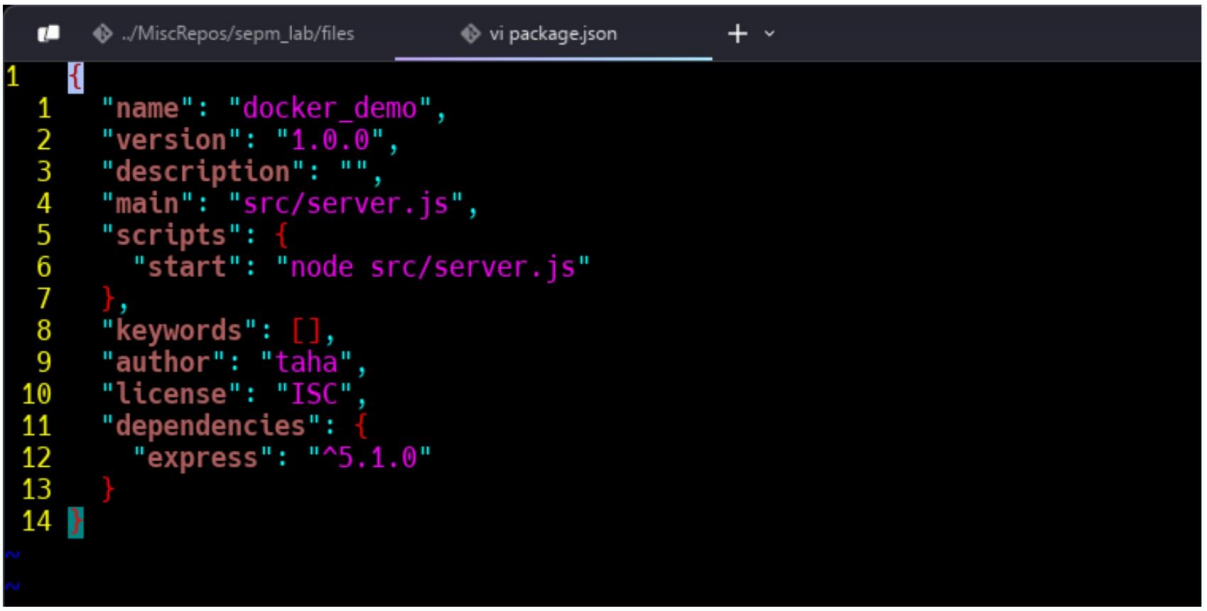
4. Avoid root – use non-root users.

5. Clean up unnecessary files to reduce image size.

OUTPUT:

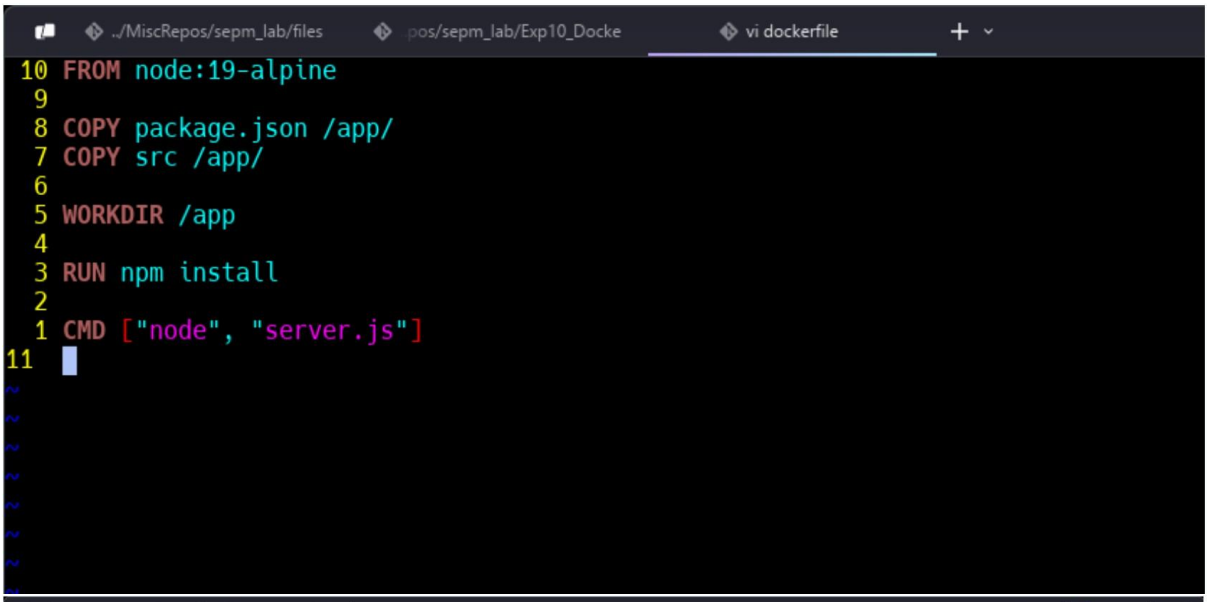


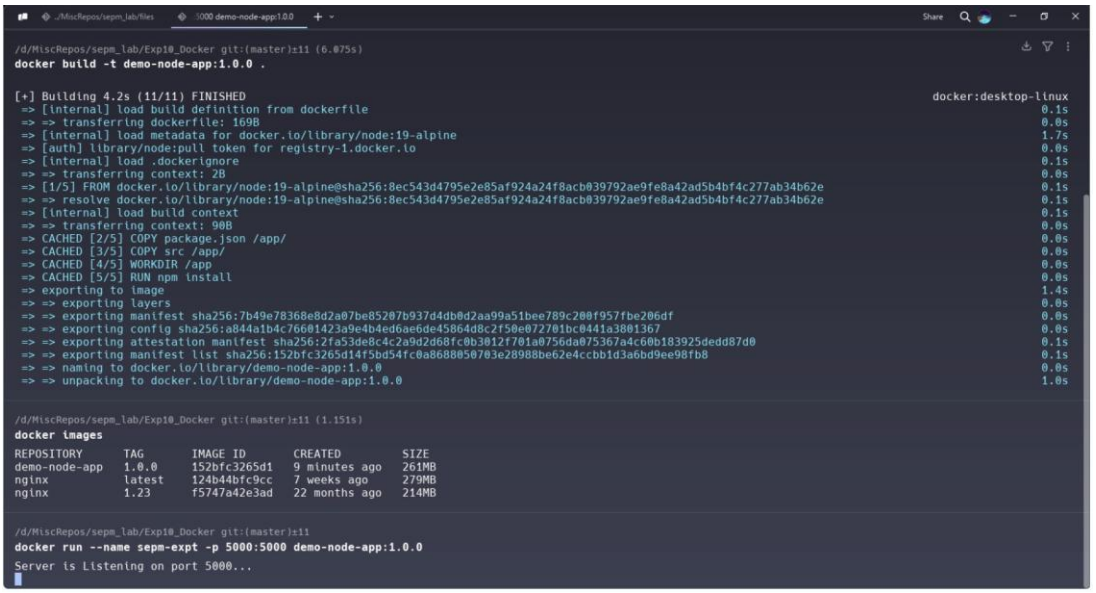
```
const express = require("express");
const app = express();
const PORT = process.env.PORT || 5000;

app.get("/", (req, res) => {
  res.status(200).json({ msg: "Hello, Docker :)" });
});

const init = async () => {
  try {
    app.listen(PORT, () => {
      console.log(`Server is Listening on port ${PORT}...`);
    });
  } catch (error) {
    console.log("There was an error : ", error);
  }
};
init();
```



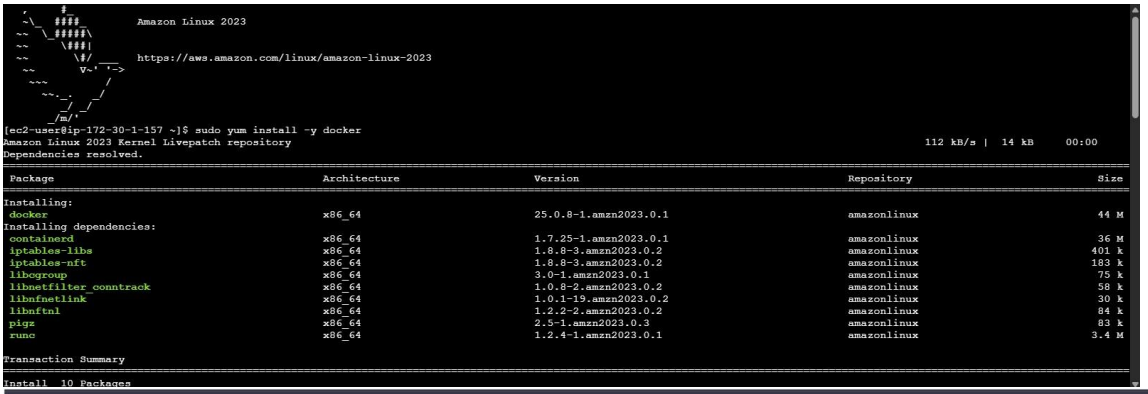
```
../MiscRepos/sepm_lab/files   vi package.json
{
  "name": "docker_demo",
  "version": "1.0.0",
  "description": "",
  "main": "src/server.js",
  "scripts": {
    "start": "node src/server.js"
  },
  "keywords": [],
  "author": "taha",
  "license": "ISC",
  "dependencies": {
    "express": "^5.1.0"
  }
}
```

Name: Soham Attarde **Batch: T11** **Roll Number: 05**





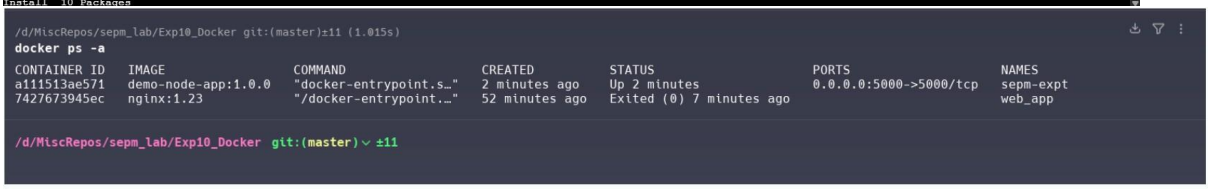
Name: Soham Attarde **Batch: T11** **Roll Number: 05**



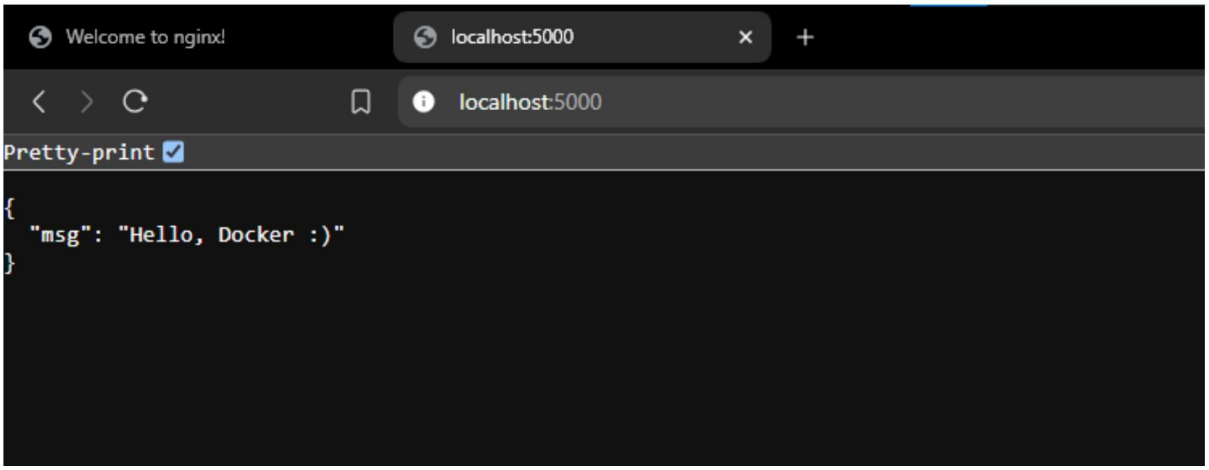
```
       _  #_
       ~\_  ####_        Amazon Linux 2023
      ~~  \_#####\
      ~~     \###|
      ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
       ~~       V~' '->
        ~~~         /
          ~~._.   _/
             _/ _/
           _/m/'
[ec2-user@ip-172-30-1-157 ~]$ sudo yum install -y docker
Amazon Linux 2023 Kernel Livepatch repository                                                    112 kB/s |  14 kB     00:00
Dependencies resolved.
```

Package	Architecture	Version	Repository	Size
Installing:				
docker	x86_64	25.0.8-1.amzn2023.0.1	amazonlinux	44 M
Installing dependencies:				
containerd	x86_64	1.7.25-1.amzn2023.0.1	amazonlinux	36 M
iptables-libs	x86_64	1.8.8-3.amzn2023.0.2	amazonlinux	401 k
iptables-nft	x86_64	1.8.8-3.amzn2023.0.2	amazonlinux	183 k
libcgroup	x86_64	3.0-1.amzn2023.0.1	amazonlinux	75 k
libnetfilter_conntrack	x86_64	1.0.8-2.amzn2023.0.2	amazonlinux	58 k
libnfnetlink	x86_64	1.0.1-19.amzn2023.0.2	amazonlinux	30 k
libnftnl	x86_64	1.2.2-2.amzn2023.0.2	amazonlinux	84 k
pigz	x86_64	2.5-1.amzn2023.0.3	amazonlinux	83 k
runc	x86_64	1.2.4-1.amzn2023.0.1	amazonlinux	3.4 M

```
Transaction Summary
Install  10 Packages
```



```
/d/MiscRepos/sepm_lab/Exp10_Docker git:(master)±11 (1.015s)
docker ps -a
CONTAINER ID   IMAGE                COMMAND                  CREATED          STATUS                      PORTS                    NAMES
a111513ae571   demo-node-app:1.0.0  "docker-entrypoint.s…"   2 minutes ago    Up 2 minutes                0.0.0.0:5000->5000/tcp   sepm-expt
7427673945ec   nginx:1.23           "/docker-entrypoint.…"   52 minutes ago   Exited (0) 7 minutes ago                             web_app

/d/MiscRepos/sepm_lab/Exp10_Docker git:(master) ±11
```



Welcome to nginx!

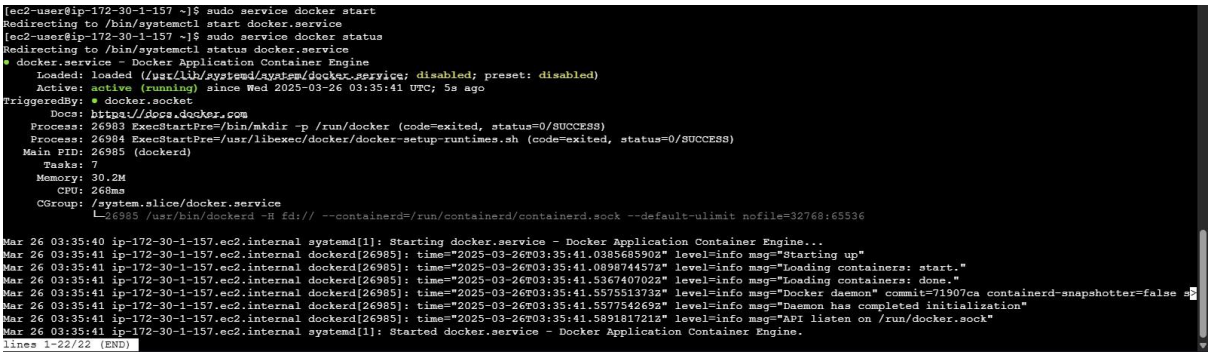
localhost:5000

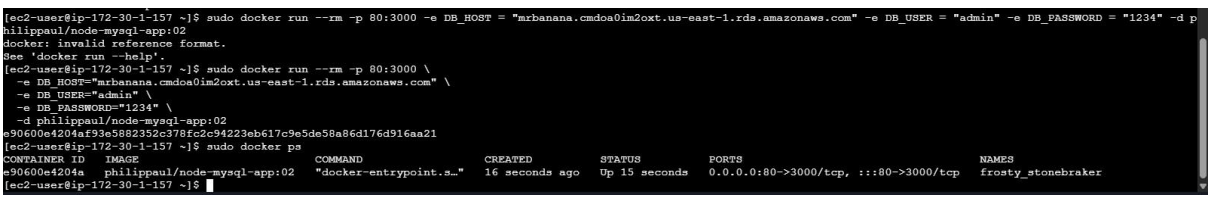
localhost:5000

Pretty-print

```
{
  "msg": "Hello, Docker :)"
}
```

Name: Soham Attarde **Batch: T11** **Roll Number: 05**





Conclusion :

We have learnt Dockerfile instructions, built an image for a sample web application using DOCKERFILE.