**Experiment 10**

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

DOCKERFILE.

**Theory :**

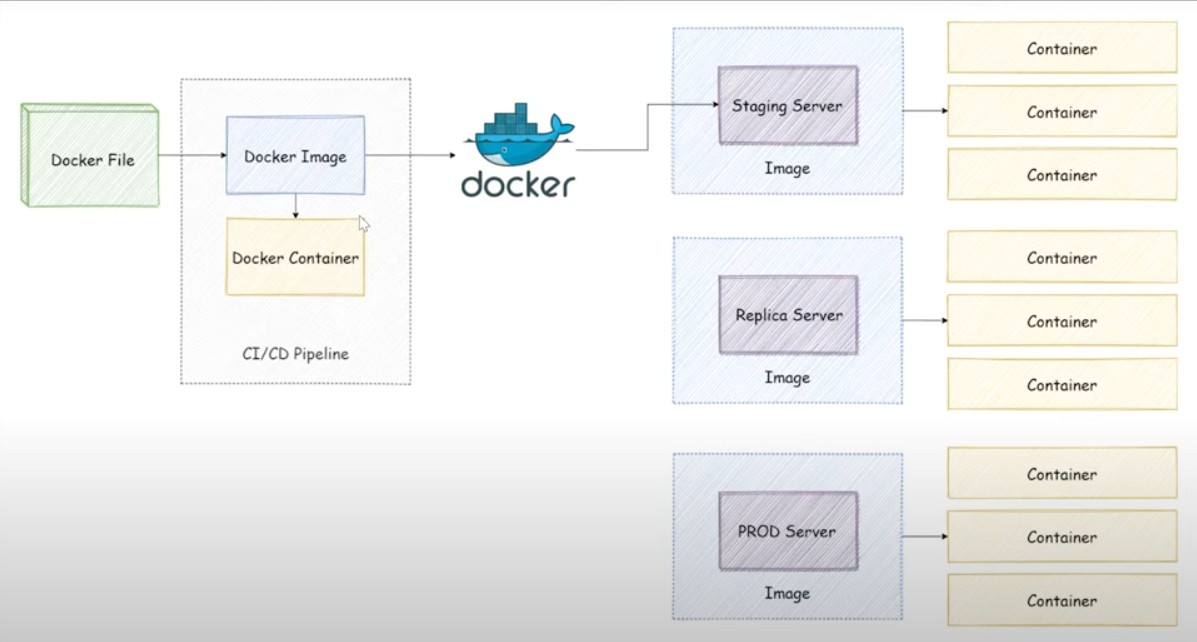
Dockerfiles are the cornerstone of creating Docker images. They contain a set of instructions that automate the process of building a Docker image, specifying everything from the base operating system to the application code, dependencies, and configuration settings.

**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.

1. **WORKDIR (Set Working Directory)** 
   * **Purpose**: Defines the directory inside the container where subsequent instructions will be executed.

**Example**:

WORKDIR /app

●

1. **COPY (Copy Files)** 
   * **Purpose**: Copies files or directories from the host system into the container.

**Example**:

COPY . /app

●

* + **Variants**:

* + 1. COPY <src> <dest>

○ ADD is similar but supports remote URLs and tar file extraction.

1. **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.

1. **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.

1. **ENV (Set Environment Variables)** 
   * **Purpose**: Sets environment variables inside the container.

**Example**:

ENV APP\_ENV=production

●

1. **CMD (Default Command)** 
   * **Purpose**: Specifies the default command to run when the container starts.

**Example**:

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

●

* + **Key Points**:

* + 1. Only one CMD is allowed.

○ It can be overridden by passing a command with docker run.

1. **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.

1. **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.

1. **Managing Docker Images List Images:** docker images

**Remove an Image:** docker rmi myapp:latest **Run a Container:**

docker run -p 8080:80 myapp:latest

1. **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).

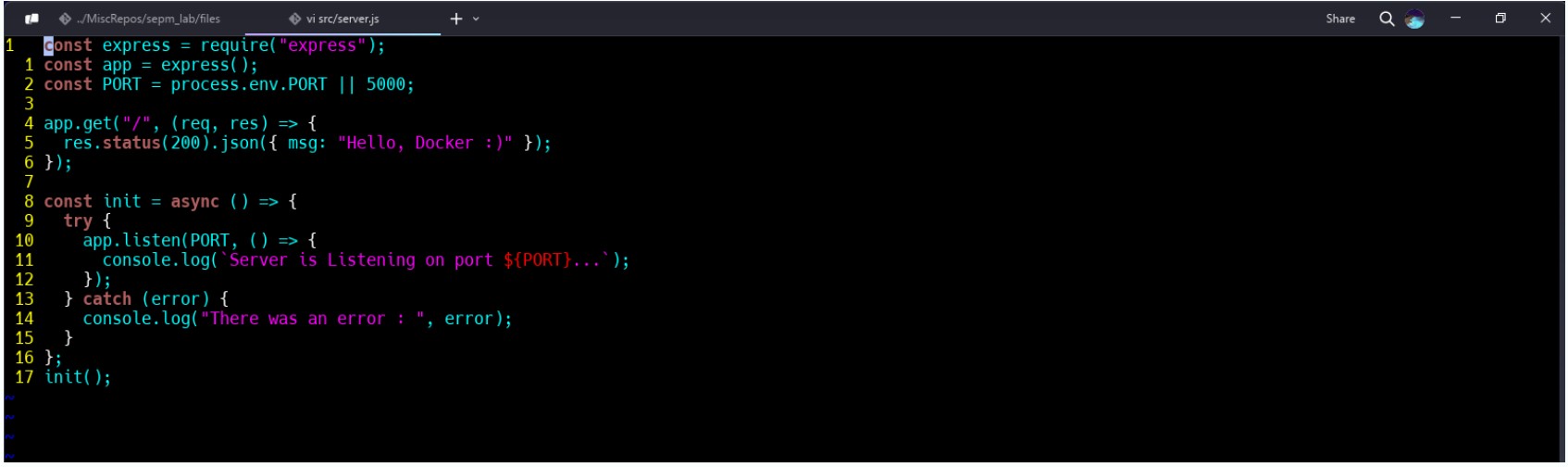
1. Order instructions from least to most frequently changing to leverage caching.

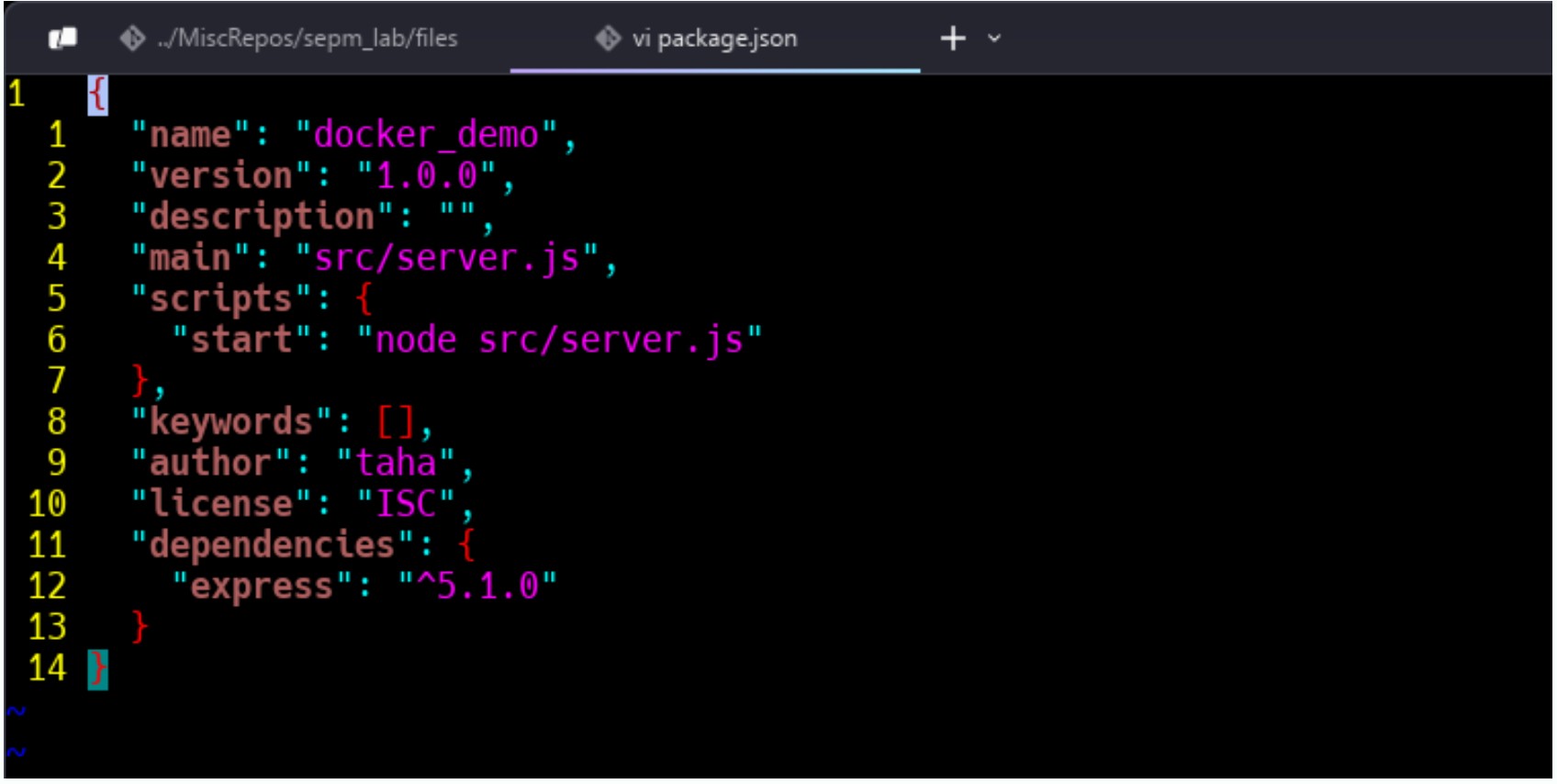
1. Combine RUN commands with &&.

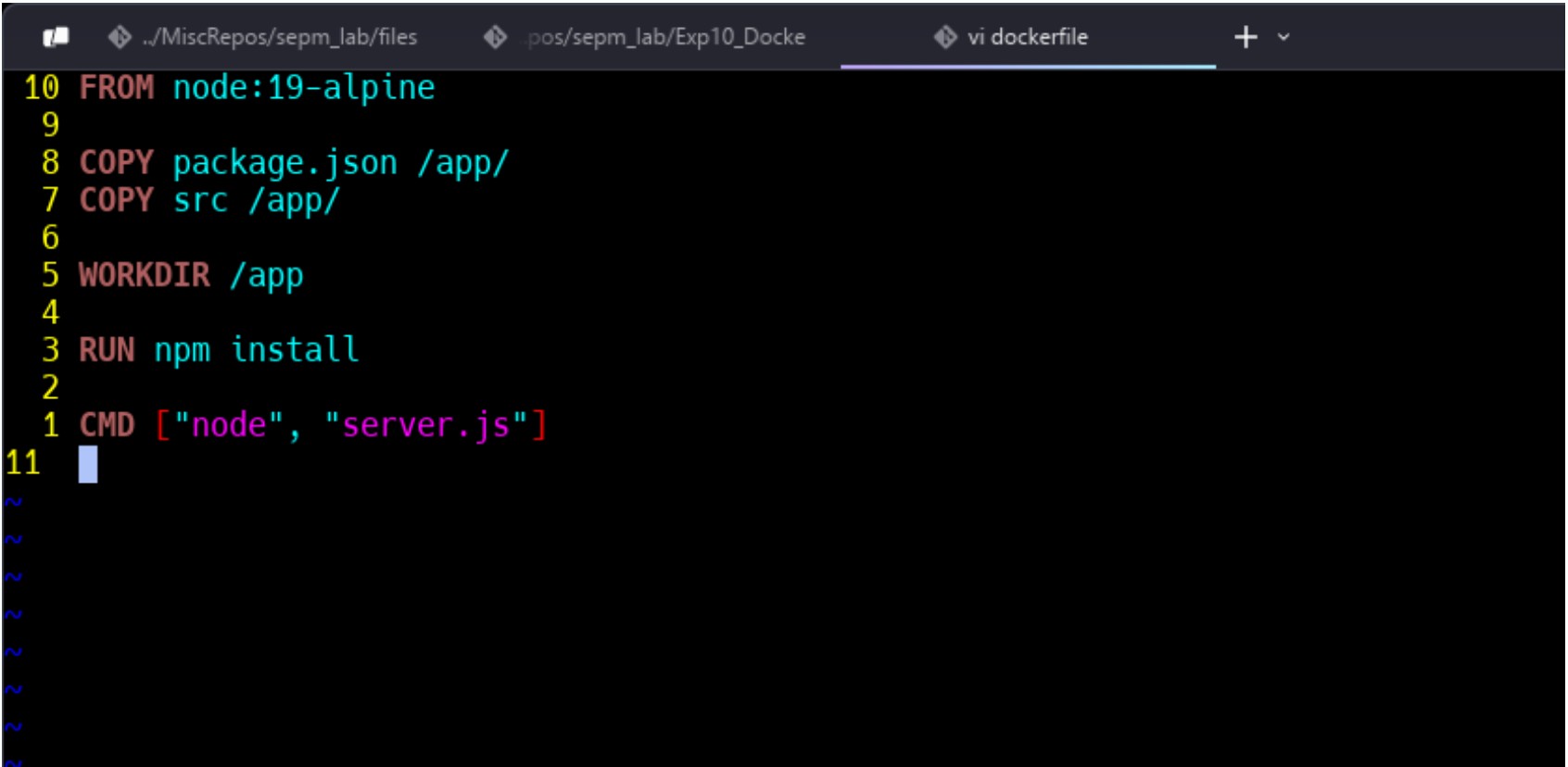
1. Avoid root – use non-root users.

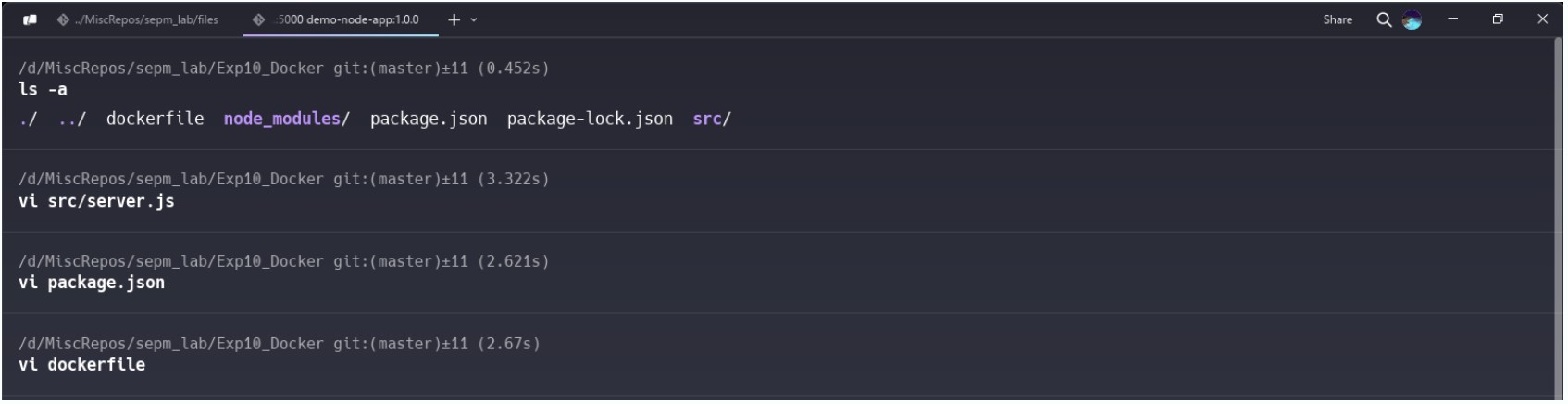
1. Clean up unnecessary files to reduce image size.

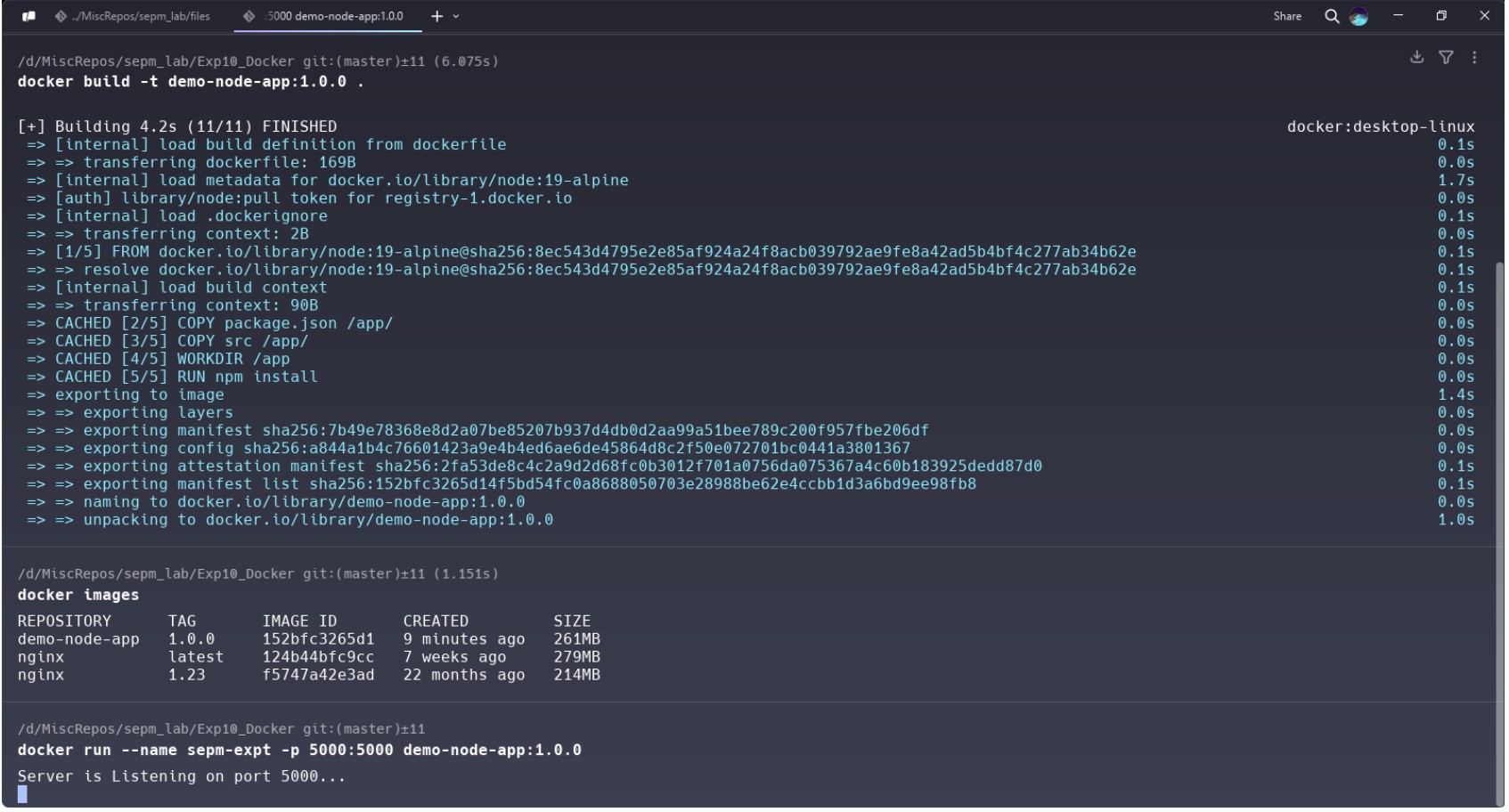
**OUTPUT:**

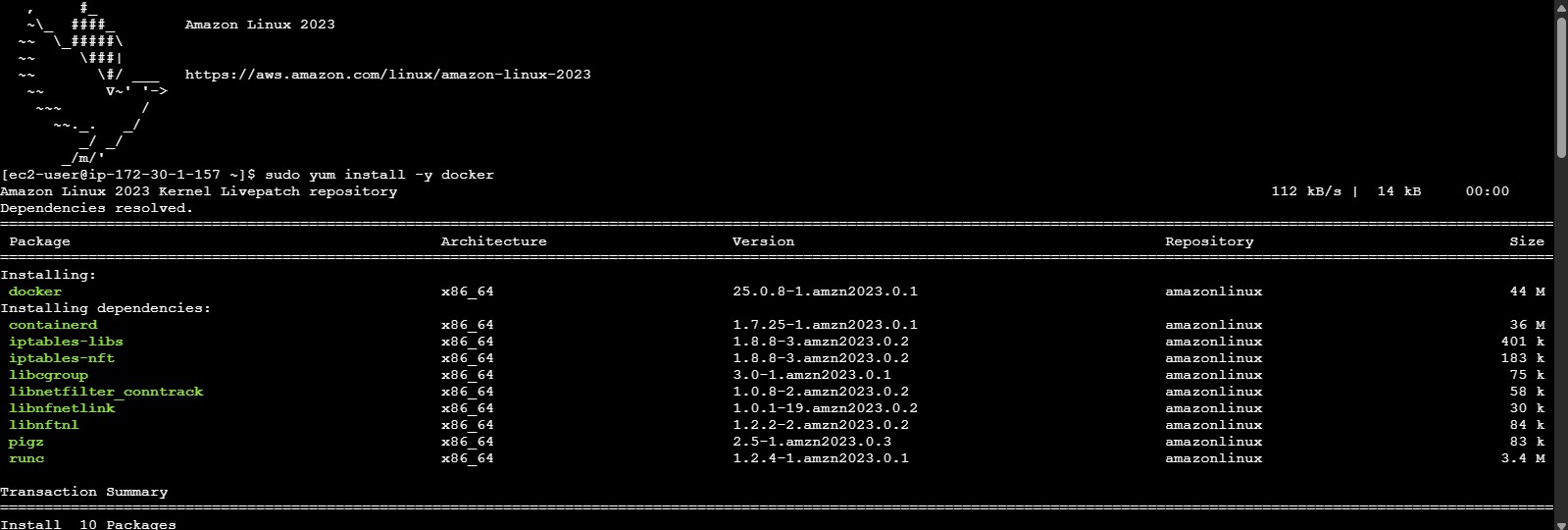


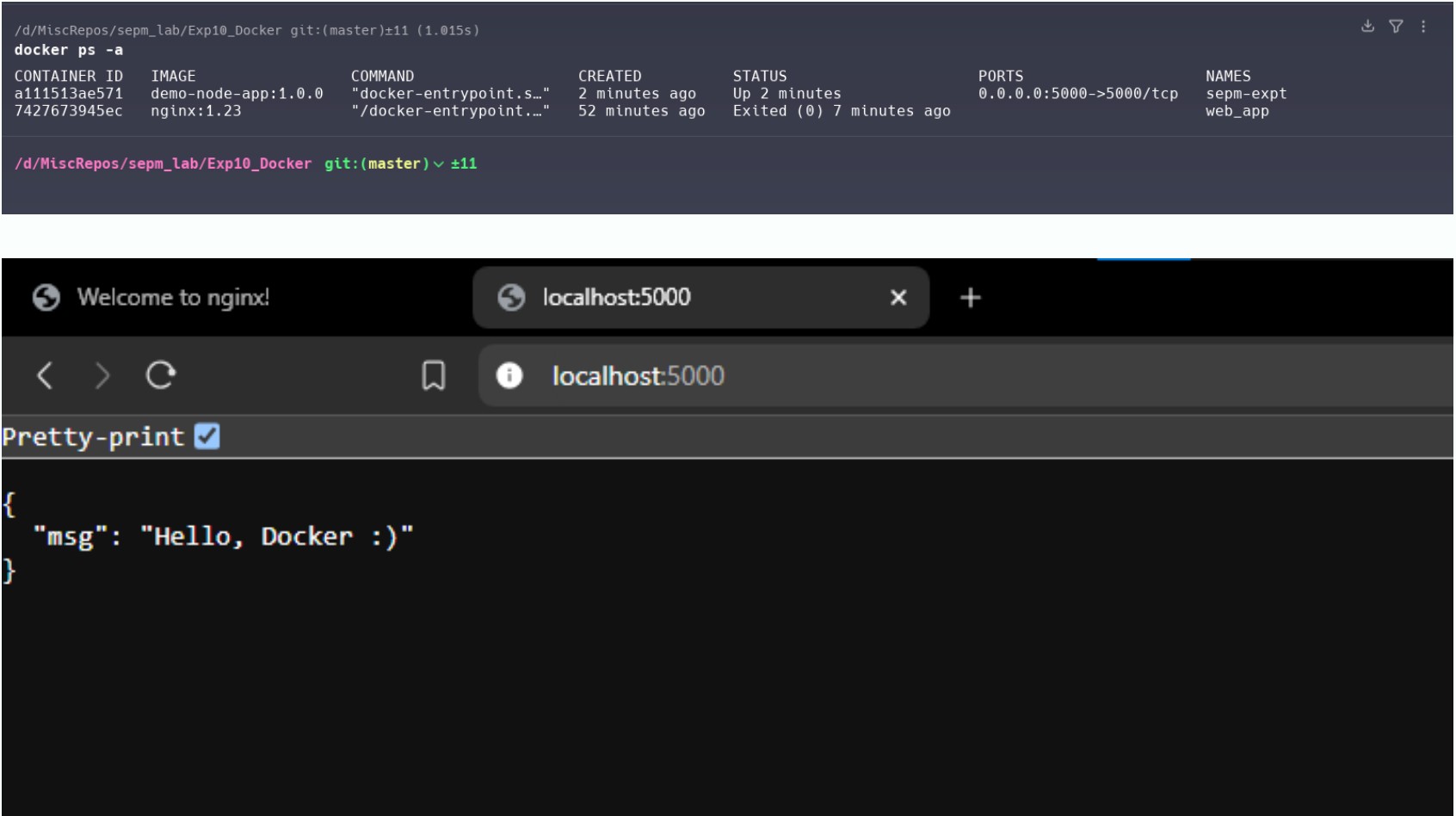


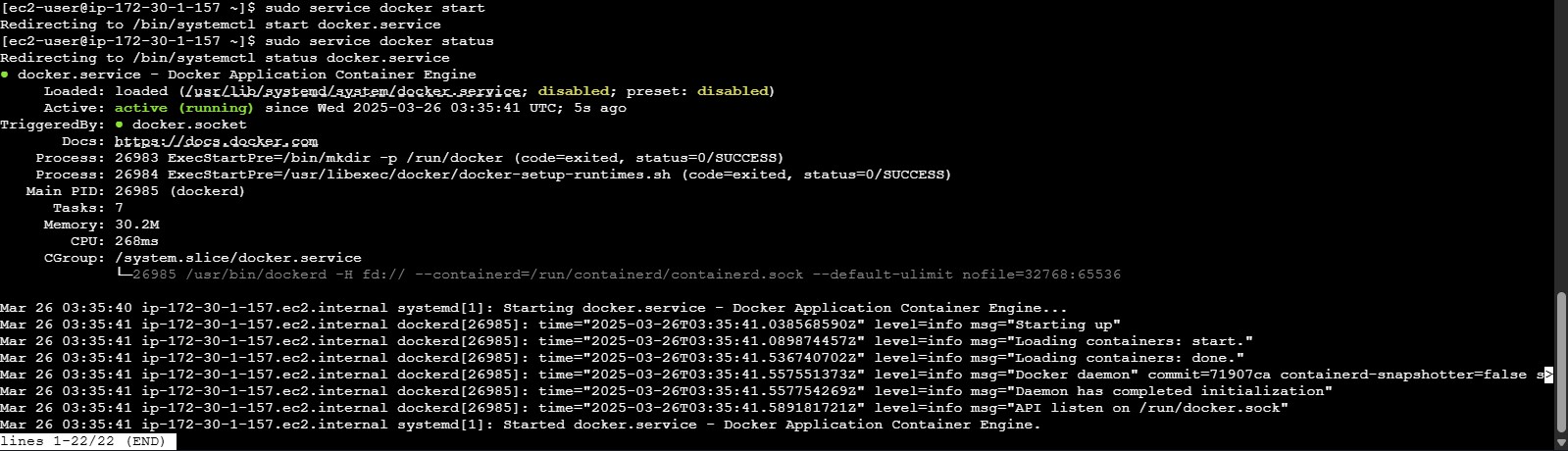




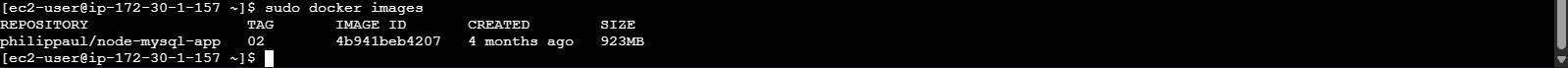


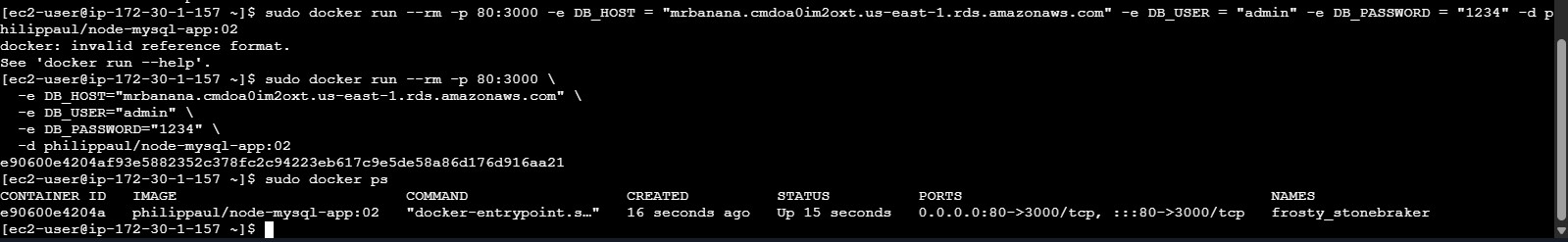












**Conclusion :**

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

DOCKERFILE.