Experiment -8

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

Theory:-
In a Dockerfile, tags are used to add metadata or labels to various instructions. These tags serve different purposes and provide additional information about the Docker image being built. Here's a breakdown of the common tags used in a Dockerfile:

1. FROM:
   - The `FROM` instruction specifies the base image to use for subsequent instructions in the Dockerfile.
   - Example: `FROM node:14-alpine`
   - In this example, `node:14-alpine` is the base image tag, where `node` is the repository name, `14` is the version, and `alpine` is the operating system flavor.

2. LABEL:
   - The `LABEL` instruction adds metadata to the Docker image in the form of key-value pairs.
   - Example: `LABEL maintainer="John Doe <john@example.com>"`
   - In this example, the label `maintainer` is set to `"John Doe <john@example.com>"`.

3. COPY and ADD:
   - The `COPY` and `ADD` instructions are used to copy files and directories from the host machine into the Docker image.
   - Example: `COPY package*.json ./`
   - There are no specific tags associated with these instructions, but the source and destination paths can be considered as tags.

4. RUN:
   - The `RUN` instruction executes commands in the Docker container during the build process.
   - Example: `RUN npm install`
   - There are no specific tags associated with the `RUN` instruction, but the command being executed can be considered as a tag.

5. WORKDIR:
   - The `WORKDIR` instruction sets the working directory for subsequent instructions in the Dockerfile.
   - Example: `WORKDIR /usr/src/app`
   - There are no specific tags associated with the `WORKDIR` instruction, but the directory path can be considered as a tag.

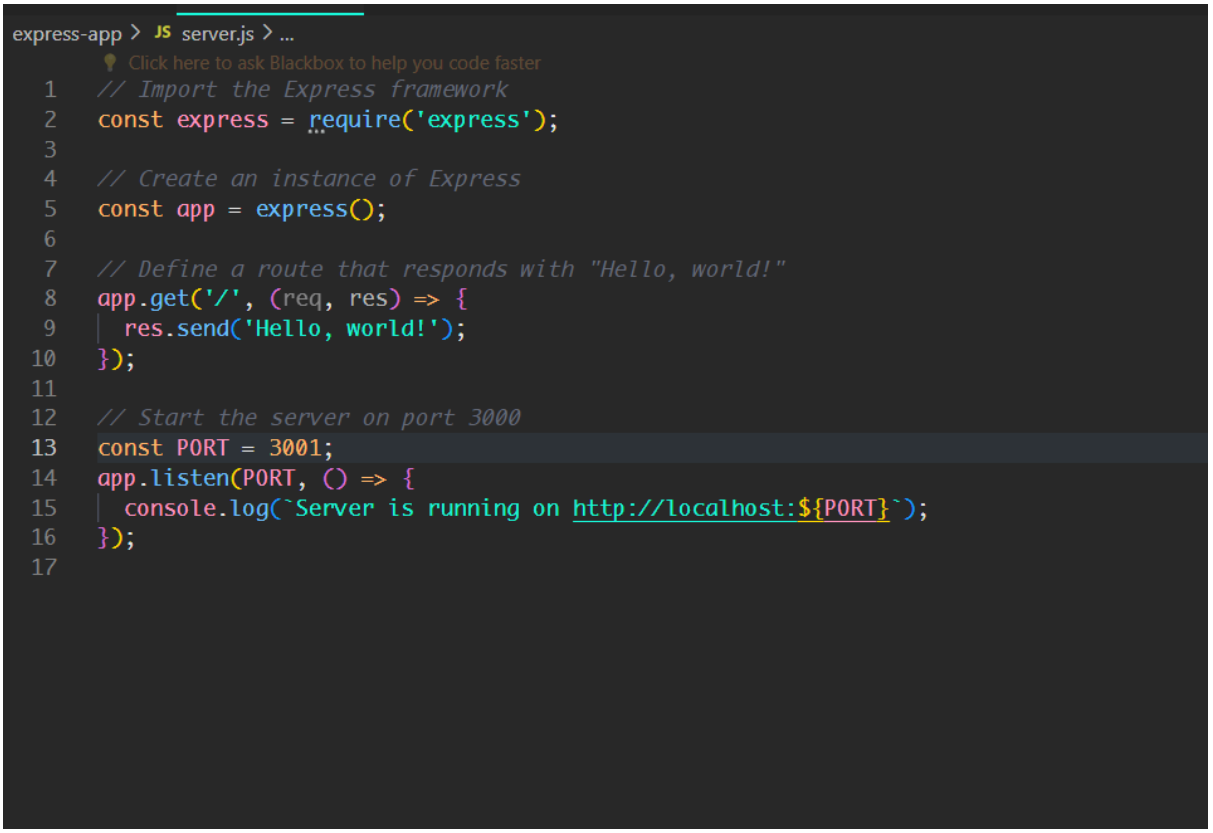
6. EXPOSE:
   - The `EXPOSE` instruction informs Docker that the container listens on specified network ports at runtime.

- Example: `EXPOSE 8080`
- There are no specific tags associated with the `EXPOSE` instruction, but the port number can be considered as a tag.

7. ENV:
- The `ENV` instruction sets environment variables in the Docker container.
- Example: `ENV NODE_ENV=production`
- There are no specific tags associated with the `ENV` instruction, but the environment variable names and values can be considered as tags.

8. CMD and ENTRYPOINT:
- The `CMD` and `ENTRYPOINT` instructions specify the command to run when the container starts.
- Example: `CMD ["npm", "start"]`
- There are no specific tags associated with these instructions, but the command being executed can be considered as a tag.

These are some of the common tags used in a Dockerfile. Each tag serves a specific purpose in defining the behavior and configuration of the Docker image and container.



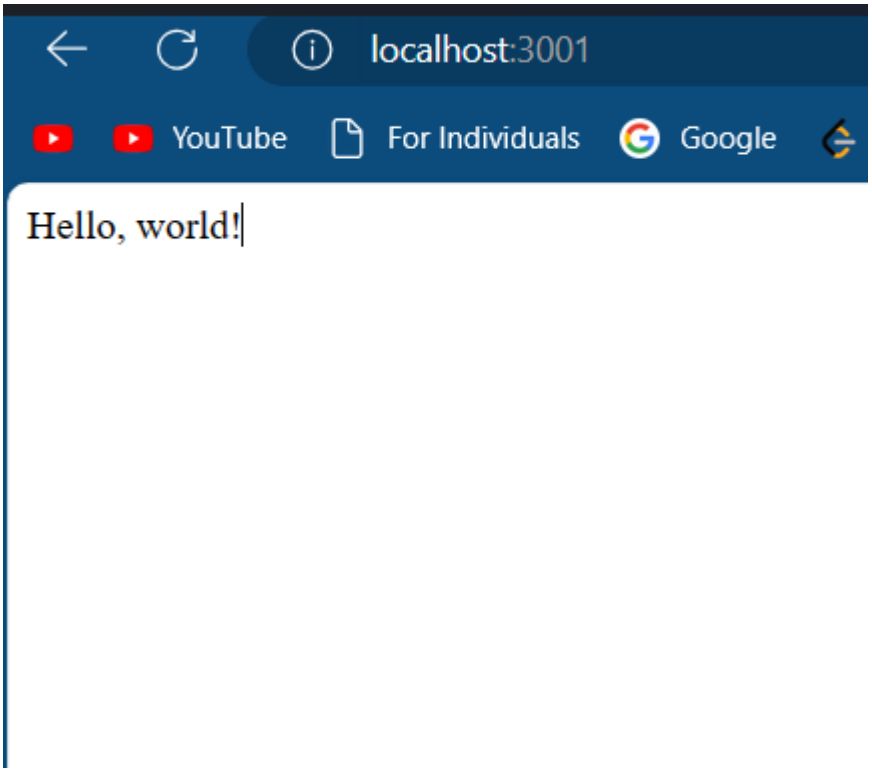
```
Click here to ask Blackbox to help you coc
# Use an official Node.js run
FROM node:latest

# Set the working directory
WORKDIR express-app/

# Copy package.json and packa
COPY package*.json ./

# Install dependencies
RUN npm install

# Copy the rest of the applic
COPY . .

# Expose port 3000 to the out
EXPOSE 8081

# Command to run the server
CMD ["node", "server.js"]
```

```
PS C:\Users\ASHIS\Desktop\docker> docker build -t nodeserver1.0 .
2024/03/25 19:34:40 http2: server: error reading preface from client //./pipe/docker_engine: fi
le has already been closed
[+] Building 13.9s (9/9) FINISHED                                          docker:default
 => [internal] load build definition from Dockerfile                                  0.0s
 => => transferring dockerfile: 157B                                                  0.0s
 => [internal] load metadata for docker.io/library/node:19-alpine                     2.7s
 => [internal] load .dockerignore                                                     0.0s
 => => transferring context: 2B                                                       0.0s
 => [1/4] FROM docker.io/library/node:19-alpine@sha256:8ec543d4795e2e85af924a24f8acb039  10.0s
 => => resolve docker.io/library/node:19-alpine@sha256:8ec543d4795e2e85af924a24f8acb0397  0.0s
 => => sha256:8ec543d4795e2e85af924a24f8acb039792ae9fe8a42ad5b4bf4c277ab 1.43kB / 1.43kB  0.0s
 => => sha256:d0ba7111bc031323ce2706f8e424afc868db289ba40ff55b05561cf59c 1.16kB / 1.16kB  0.0s
 => => sha256:e2a8cc97f817417787050d381376568c494547f9af9decfca6463dee6d 6.73kB / 6.73kB  0.0s
 => => sha256:8a49fdb3b6a5ff2bd8ec6a86c05b2922a0f7454579ecc07637e94dfd1d 3.40MB / 3.40MB  0.8s
 => => sha256:1197750296b3abe1d21ffbb3d3ea76df5ba887cf82c8e3284d267cbb 48.15MB / 48.15MB  7.8s
```



```
0.0.0.0:3000: bind: address already in use.
PS C:\Users\ASHIS\Desktop\docker> docker run -d -p 3001:3001 no
deserver1.0
a00a3d4bcaa412c6f69f8fb8fa22364de047155b4dfad3a6bbb5edb4986a268
6
```

```
PS C:\Users\ASHIS\Desktop\docker> docker ps
CONTAINER ID   IMAGE             COMMAND                  CREATED          STATUS          PORTS                     NAMES
f2048488c3f9   getting-started   "docker-entrypoint.s…"   36 minutes ago   Up 36 minutes   127.0.0.1:3000->3000/tcp   focused_hermann
PS C:\Users\ASHIS\Desktop\docker>
```



Conclusion : We have successfully conatinerized a webapp application consisting of a web server using Docker.