EXPERIMENT NO. 3

Title: Create a Dockerfile for a Web Application and Running it Inside a Docker Container.

Aim: To write a Dockerfile for a basic web application, build a Docker image from it, and run the web app inside a Docker container.

Theory:

Docker allows developers to package applications and their dependencies into a standardized unit called a container. A Dockerfile is a text file that contains instructions on how to build a Docker image. It defines the base image, application code, required dependencies, and the commands to run the application.

By creating a Dockerfile:

- We can automate the creation of a consistent environment.
- The image can be reused and deployed across different systems easily.
- Developers can ensure that the application runs the same everywhere.

In this experiment, we will:

- Create a basic web app using HTML and JavaScript.
- Write a Dockerfile for the web app.
- Build a Docker image.
- Run the app inside a Docker container using that image.

Steps of Execution:

- Create a simple web app using HTML, CSS, and JavaScript.
- Write a Dockerfile to containerize the web app using a base image (e.g., Nginx or Node.js).
- Build the Docker image.
- Run the Docker container from the image.
- Access the web app via browser.

Stepwise Procedure:

Step 1: Create the Project Directory

bash

CopyEdit

mkdir docker-web-app

cd docker-web-app

Step 2: Create the Web App Files

Create a file named index.html with the following content:

html

CopyEdit

```
<!DOCTYPE html>
<html>
<head>
  <title>Docker Web App</title>
</head>
<body>
  <h1>Hello from Dockerized Web App!</h1>
</body>
</html>
```

Step 3: Create the Dockerfile

Create a file named Dockerfile in the same folder:

Dockerfile

CopyEdit

```
# Use the official Nginx image as the base
FROM nginx:alpine


# Remove default Nginx static files
RUN rm -rf /usr/share/nginx/html/*
# Copy our web app files into the container
COPY index.html /usr/share/nginx/html/
# Expose port 80 to access the app
EXPOSE 80
```

Step 4: Build the Docker Image

bash

CopyEdit

```
docker build -t my-web-app .
```

Output:

Docker reads the Dockerfile, packages the files, and creates an image named my-web-app.

Step 5: Run the Docker Container

bash

CopyEdit

docker run -d -p 8080:80 --name web-container my-web-app

Explanation:

- -d: Runs the container in detached mode.
- -p 8080:80: Maps container's port 80 to host port 8080.
- --name web-container: Names the container.

Step 6: Open the Web App in Browser

Open your browser and visit:

arduino

CopyEdit

http://localhost:8080

OutPut:

csharp

CopyEdit

Hello from Dockerized Web App!

Key Points:

- Dockerfile defines the instructions to build a container image.
- FROM nginx:alpine uses a lightweight web server base image.
- COPY transfers application files into the image.
- EXPOSE and -p handle networking between host and container.
- docker build and docker run are core Docker commands for container lifecycle.

Some Examples:

Command	Purpose	Output
docker build -t my-web-app .	Builds the Docker image from Dockerfile	Creates an image named my-web-app
docker run -d -p 8080:80 my-web-app	Runs the container	Container runs and listens on localhost:8080
docker ps	Lists running containers	Shows active container ID and por mapping

Command	Purpose	Output
docker stop web-container	Stops the container	Container stops running
docker rm web-container	Removes the container	Container deleted

Conclusion:

This experiment demonstrated how to create a Dockerfile for a simple static web app, build a Docker image from it, and run the application inside a Docker container using Nginx. By doing this, students learned the basics of Dockerfile syntax, image creation, and container management—key skills for modern DevOps and cloud-native development.