Placement Empowerment Program

Cloud Computing and DevOps Centre

BUILD AND RUN A CUSTOM DOCKER IMAGE

NAME: RENITA A DEPT: ECE

St. JOSEPH'S COLLEGE OF ENGINEERING | St. JOSEPH'S INSTITUTE OF TECHNOLOGY
AUTONOMOUS INSTITUTIONS, AFFILIATED TO ANNA UNIVERSITY

Introduction

Containerization has revolutionized the way applications are developed, deployed, and managed. Docker provides a lightweight and efficient way to package applications with all dependencies, ensuring consistency across different environments. In this POC, we will build and run a custom Docker image that serves a simple HTML web page using NGINX.

Overview

This POC demonstrates how to:

1. Create a Dockerized static website using NGINX.
2. Build a custom Docker image with an HTML file.
3. Run a Docker container to serve the HTML content.
4. Access the webpage via http://localhost:8080.

By the end of this exercise, you will have a functional web server running inside a Docker container.

Objectives

Understand the Dockerfile and key Docker commands (FROM, COPY, CMD).

- Learn to build and run a Docker containerized application.
- Gain hands-on experience in port mapping and container management.
- Demonstrate how Docker simplifies deployment compared to traditional setups

Importance

- Portability – Docker ensures the application runs identically across different environments.
- Efficiency – Containers are lightweight and require minimal system resources.
- Scalability – Containers make it easier to scale applications without dependency conflicts.
- Faster Deployment – Eliminates the need for manual configurations and installations.

Step-by-Step Overview

Step 1:

Install Docker (If Not Installed)

1. Open Command Prompt (cmd) and check

if Docker is installed

docker - -version

2. If not installed, download and install Docker from Docker's official website.

3. Ensure Docker Desktop is running before proceeding

```
C:\Users\renit>docker --version
Docker version 27.5.1, build 9f9e405
```

Step 2:

Create a Project Directory

Open Command Prompt and run:

mkdir my-docker-html && cd my-docker-html

mkdir my-docker-html → Creates a new directory.

cd my-docker-html → Moves inside the directory.

```
C:\Users\renit>mkdir my-docker-html && cd my-docker-html

C:\Users\renit\my-docker-html>
```

Step 3:

Create an HTML File (index.html)

Run this command to create an empty
index.html file:

type nul > index.html

Open the file in Notepad:

notepad index.html

Add the following content and save the file

```
C:\Users\renit\my-docker-html>type nul > index.html
```

```
C:\Users\renit\my-docker-html>notepad index.html
```

File Edit View

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Docker HTML App</title>
</head>
<body>
    <h1>Hello from Docker!</h1>
    <p>This HTML page is served from inside a Docker container.</p>
</body>
</html>
```

Step 4:

Create a Dockerfile

Run this command to create an empty

Dockerfile:

type nul > Dockerfile

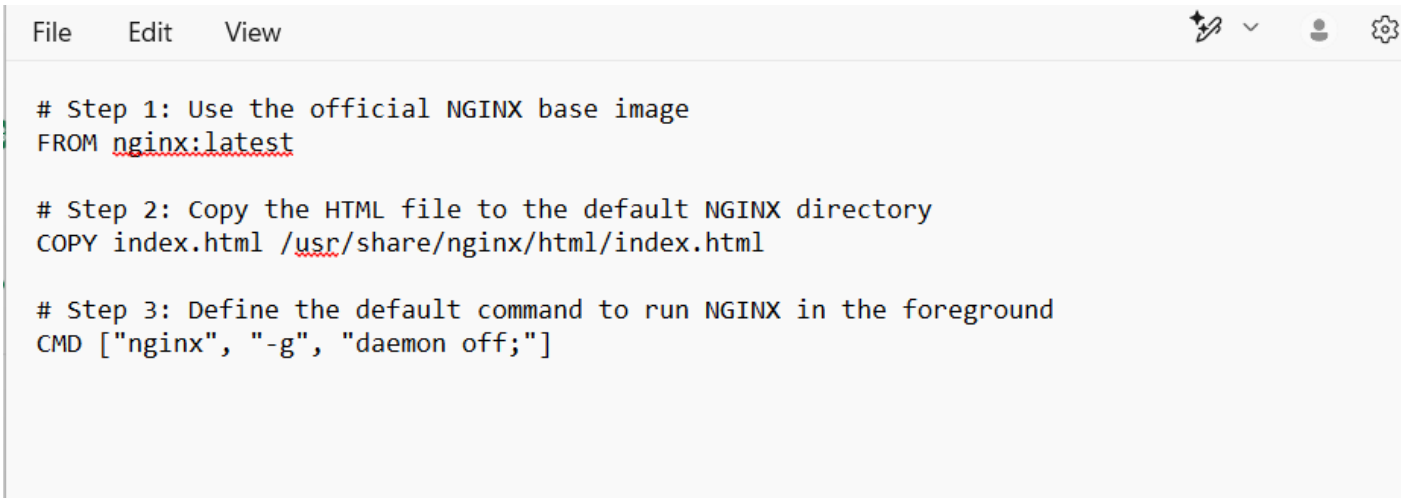
Open the file in Notepad:

notepad Dockerfile

Add the following content and save the file:

```
C:\Users\renit\my-docker-html>type nul >Dockerfile
```

```
C:\Users\renit\my-docker-html>notepad Dockerfile
```



Step 5:

Build the Docker Image

Run the following command inside the my-docker-html directory:

docker build -t my-html-image .

-t my-html-image → Names the image my-html-image.

. → Uses the current directory (where the Dockerfile is located).

```
C:\Users\renit\my-docker-html>docker build -t my-html-image .
[+] Building 67.4s (8/8) FINISHED                                                                                docker:desktop-linux
 => [internal] load build definition from Dockerfile                                                                             0.2s
 => => transferring dockerfile: 321B                                                                                             0.1s
 => [internal] load metadata for docker.io/library/nginx:latest                                                                  5.1s
 => [auth] library/nginx:pull token for registry-1.docker.io                                                                     0.0s
 => [internal] load .dockerignore                                                                                                0.0s
 => => transferring context: 2B                                                                                                  0.0s
 => [internal] load build context                                                                                                0.1s
 => => transferring context: 357B                                                                                                0.0s
 => [1/2] FROM docker.io/library/nginx:latest@sha256:9d6b58feebd2dbd3c56ab5853333d627cc6e281011cfd6050fa4bcf2072                61.2s
 => => resolve docker.io/library/nginx:latest@sha256:9d6b58feebd2dbd3c56ab5853333d627cc6e281011cfd6050fa4bcf2072c                0.1s
 => => sha256:943ea0f0c2e42ccacc72ac65701347eadb2b0cb22828fac30f1400bba3d37088 1.21kB / 1.21kB                                   0.8s
 => => sha256:103f50cb3e9f200431b555078cce5e8df3db6ddc2e54d714a10b994e430e98a3 1.40kB / 1.40kB                                   0.6s
 => => sha256:9dd21ad5a4a6a856d82bb6bb6147c30ad90a9768c3651c55775354e7649bc74d 406B / 406B                                       1.2s
 => => sha256:d014f92d532d416c7b9eadb244f14f73fdb3d2ead120264b749e342700824f3c 957B / 957B                                       1.2s
 => => sha256:513c3649bb1480ca9a04c73f320b6b5a909e24e4ac18ae72fd56b818241d6730 626B / 626B                                       0.7s
 => => sha256:bf9acace214a6c23630803d90911f1fd7d1ba06a3083f0a62fd036a6d1d8e274 43.95MB / 43.95MB                                59.0s
 => => sha256:7cf63256a31a4cc44f6defe8e1af95363aee5fa75f30a248d95cae684f87c53c 28.22MB / 28.22MB                                48.4s
 => => extracting sha256:7cf63256a31a4cc44f6defe8e1af95363aee5fa75f30a248d95cae684f87c53c                                         1.4s
 => => extracting sha256:bf9acace214a6c23630803d90911f1fd7d1ba06a3083f0a62fd036a6d1d8e274                                        0.9s
 => => extracting sha256:513c3649bb1480ca9a04c73f320b6b5a909e24e4ac18ae72fd56b818241d6730                                        0.0s
 => => extracting sha256:d014f92d532d416c7b9eadb244f14f73fdb3d2ead120264b749e342700824f3c                                        0.0s
 => => extracting sha256:9dd21ad5a4a6a856d82bb6bb6147c30ad90a9768c3651c55775354e7649bc74d                                        0.0s
 => => extracting sha256:943ea0f0c2e42ccacc72ac65701347eadb2b0cb22828fac30f1400bba3d37088                                        0.0s
 => => extracting sha256:103f50cb3e9f200431b555078cce5e8df3db6ddc2e54d714a10b994e430e98a3                                        0.0s
 => [2/2] COPY index.html /usr/share/nginx/html/index.html                                                                       0.3s
 => exporting to image                                                                                                           0.3s
 => => exporting layers                                                                                                          0.1s
 => => exporting manifest sha256:01584c9e7138adb2ab421238219a9d7050470ad65e10eeba3514cccdf3f7e03c                                0.0s
 => => exporting config sha256:1aa9071d12c9b09d93ab9ed31aa0036670194326fbb076314119213O7f8870c0                                  0.0s
 => => exporting attestation manifest sha256:5ec143ca1769af4f531fcfbdb3b3f017deed9df04aea956a66d818e96f5c5d05                    0.0s
 => => exporting manifest list sha256:cb4dfcdd35519b91beb92b528b068609b6e425d8470a39cd6e6a36cfae5a60f6                           0.0s
 => => naming to docker.io/library/my-html-image:latest                                                                          0.0s
 => => unpacking to docker.io/library/my-html-image:latest                                                                       0.0s
```

Step 6:

Run the Docker Container

Run this command to start a container from the image and expose it on port 8080:

docker run -d -p 8080:80 my-html-image

-d → Runs the container in detached mode (background).

-p 8080:80 → Maps port 8080 on your computer to port 80 inside the container.

```
C:\Users\renit\my-docker-html>docker run -d -p 8080:80 my-html-image
a49546f6c302160429ee11857676002127de6f336b465e396c1285c2f9e7dd17
```

Step 7:

View the Web Page in a Browser

Open a browser and go to:

http://localhost:8080

You should see the "Hello from Docker!" message.

Outcome

🎯 Successfully built and ran a Docker container hosting an HTML webpage.

🎯 Accessed the webpage using http://localhost:8080.

🎯 Understood how Docker images and containers work for web applications.

🎯 Gained practical experience in containerization using Docker.