

Placement Empowerment Program

Cloud Computing and DevOps Centre

Set Up Environment Variables in Docker Containers:
Pass environment variables to your web app's container
to store configuration details (e.g., database
credentials).

Name: Hema S

Department: ECE

Introduction

In modern web applications, **environment variables** are crucial for managing configurations like database credentials, API keys, and application settings. Instead of hardcoding these values in the source code, environment variables allow flexibility and security.

In this POC, we will learn how to **pass environment variables** to a **Docker container** running an **Nginx web server** and use them in a static webpage.

Overview

This Proof of Concept (POC) demonstrates how to:

1. Create an **.env file** to store environment variables.
2. Use **Docker Compose** to pass environment variables to a container.
3. Deploy an **Nginx container** serving a static HTML page with dynamic environment variables.
4. Test the setup by running the container and checking if environment variables are displayed correctly.

Objective

1. Understand the concept of **environment variables** in Docker.
2. Learn how to pass environment variables using the `-e` flag and **.env file**.
3. Use **Docker Compose** to simplify container deployment with environment variables.
4. Deploy a static website inside an **Nginx container** that reads environment variables dynamically.

Importance

1. **Separation of Configuration & Code:** Avoids hardcoding sensitive information like database credentials.
2. **Security & Maintainability:** Sensitive data (like API keys) can be managed separately from the application code.
3. **Portability:** The same container image can be deployed in different environments (development, staging, production) with different configurations.
4. **Automation:** Using `.env` files and **Docker Compose** makes it easier to manage configurations in a DevOps workflow.

Step-by-Step Overview

Step 1:

Open **Command Prompt** (cmd) and run:

```
mkdir docker-env-poc
```

```
cd docker-env-poc
```

This creates a folder named **docker-env-poc** and moves into it.

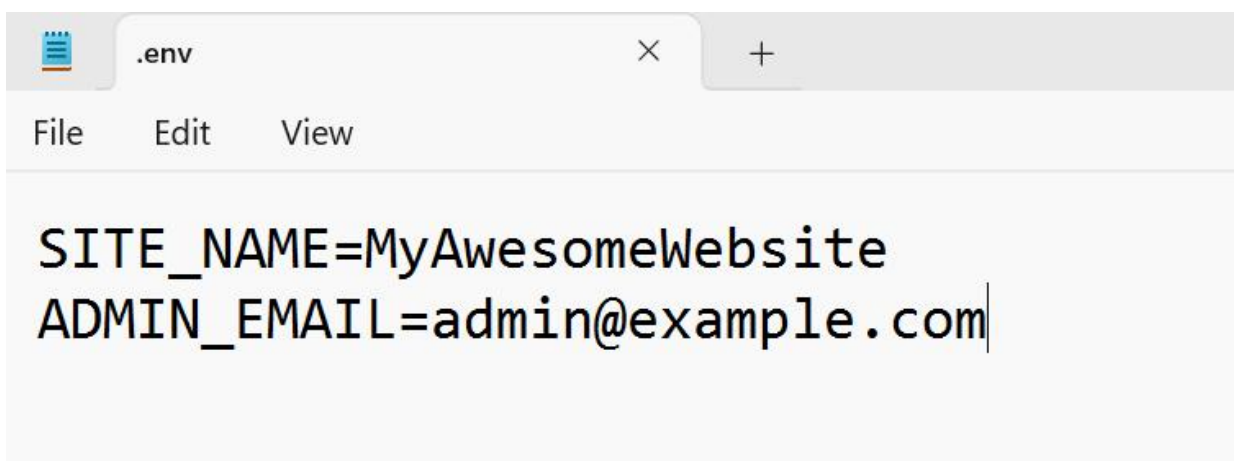
```
PS C:\Users\DELL> mkdir docker-env-poc
```

```
PS C:\Users\DELL> cd docker-env-poc|
```

Step 2:

Open Notepad and create a **.env** file (Stores environment variables)

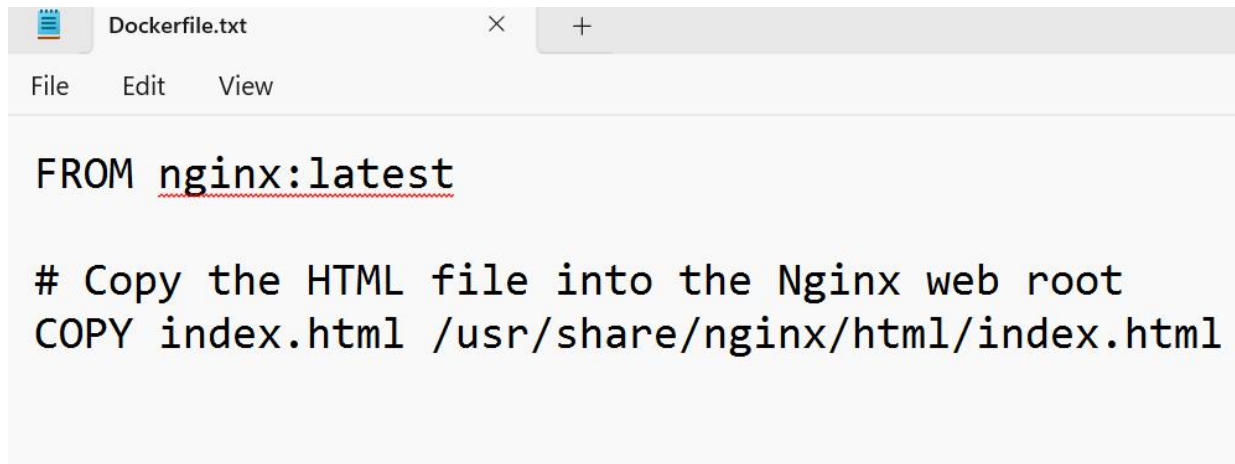
Make sure to save it as **.env** and not **.env.txt**



Step 3:

Dockerfile (Defines the Docker image)

Save it as **Dockerfile** (without any extension)

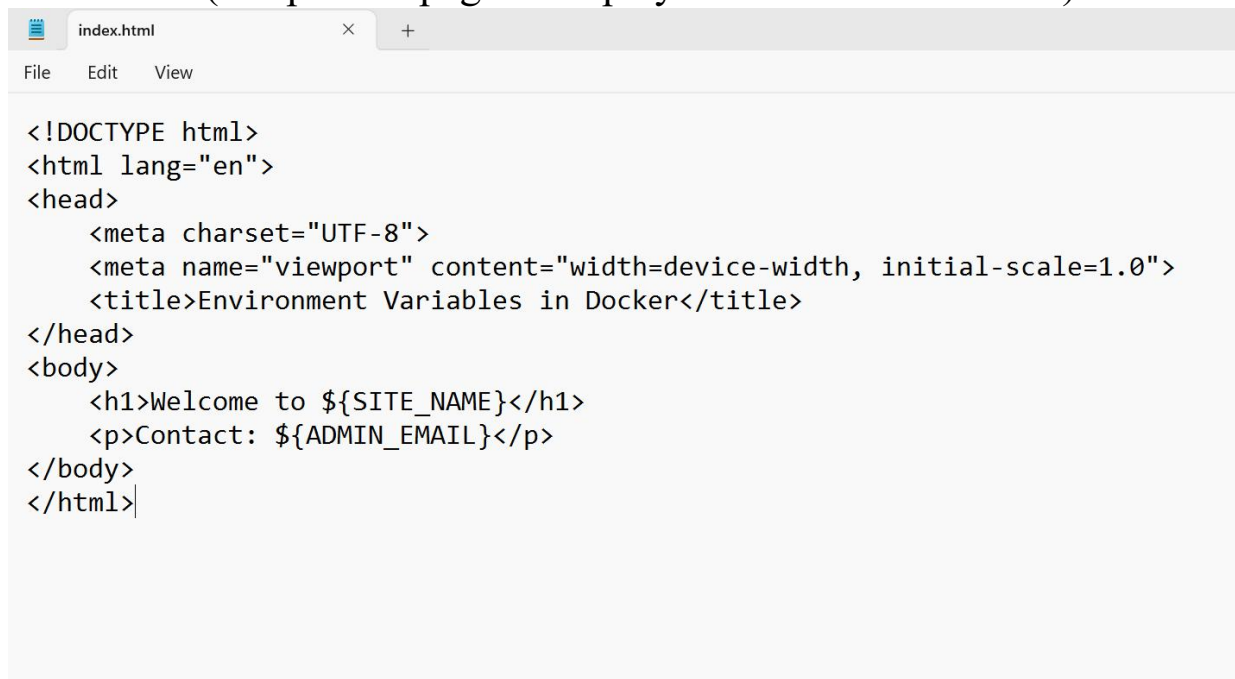
A screenshot of a text editor window titled 'Dockerfile.txt'. The window has a menu bar with 'File', 'Edit', and 'View'. The content of the file is as follows:

```
FROM nginx:latest

# Copy the HTML file into the Nginx web root
COPY index.html /usr/share/nginx/html/index.html
```

Step 4:

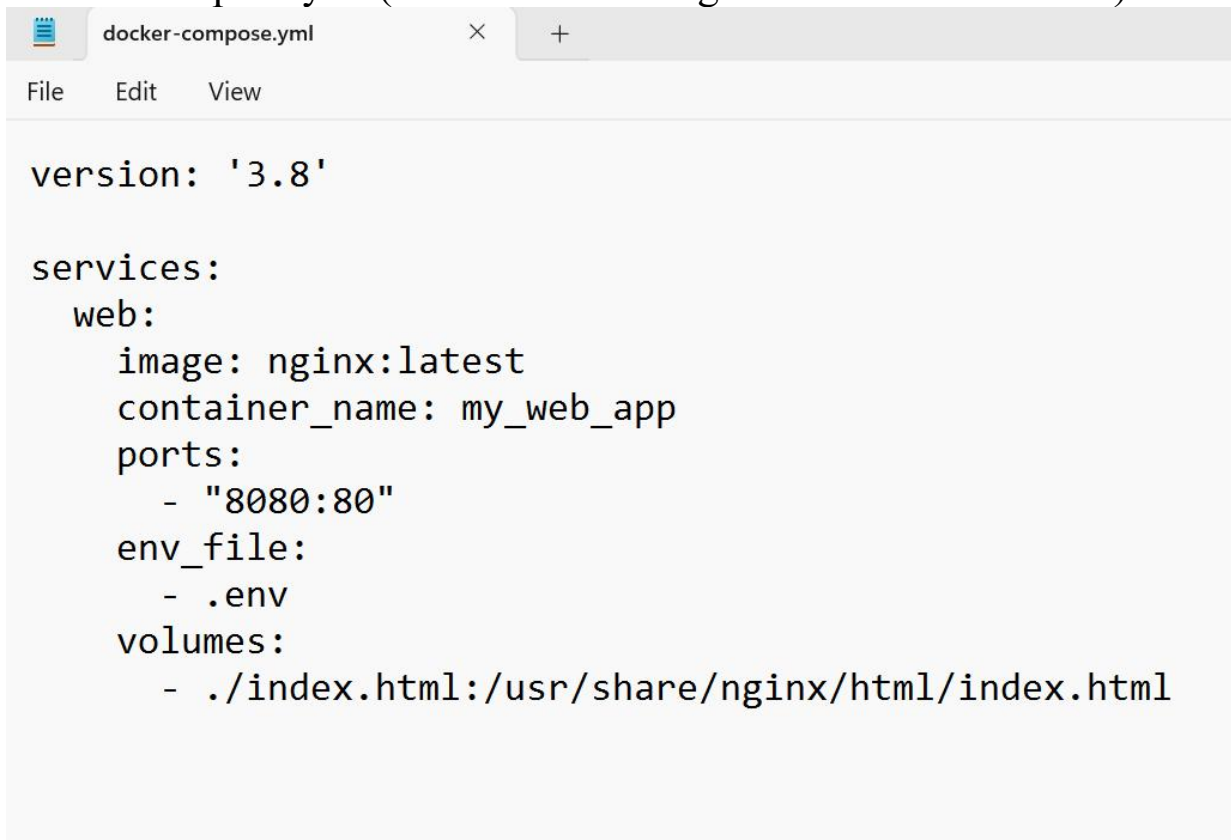
index.html (Simple webpage to display environment variables)

A screenshot of a text editor window titled 'index.html'. The window has a menu bar with 'File', 'Edit', and 'View'. The content of the file is as follows:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Environment Variables in Docker</title>
</head>
<body>
  <h1>Welcome to ${SITE_NAME}</h1>
  <p>Contact: ${ADMIN_EMAIL}</p>
</body>
</html>
```

Step 5:

docker-compose.yml (Automates running the container with .env)

A screenshot of a code editor window titled 'docker-compose.yml'. The editor has a menu bar with 'File', 'Edit', and 'View'. The code content is as follows:

```
version: '3.8'

services:
  web:
    image: nginx:latest
    container_name: my_web_app
    ports:
      - "8080:80"
    env_file:
      - .env
    volumes:
      - ./index.html:/usr/share/nginx/html/index.html
```

Step 6:

Run the following command in the **same directory** where your Dockerfile is located:

docker build -t my-static-site .

This will create a Docker image named my-static-site.

Once the image is built, start the container:

This maps **port 8080** on your system to **port 80** inside the container.

```
PS C:\Users\DELL\docker-env-poc> docker build -t my-static-site .
```

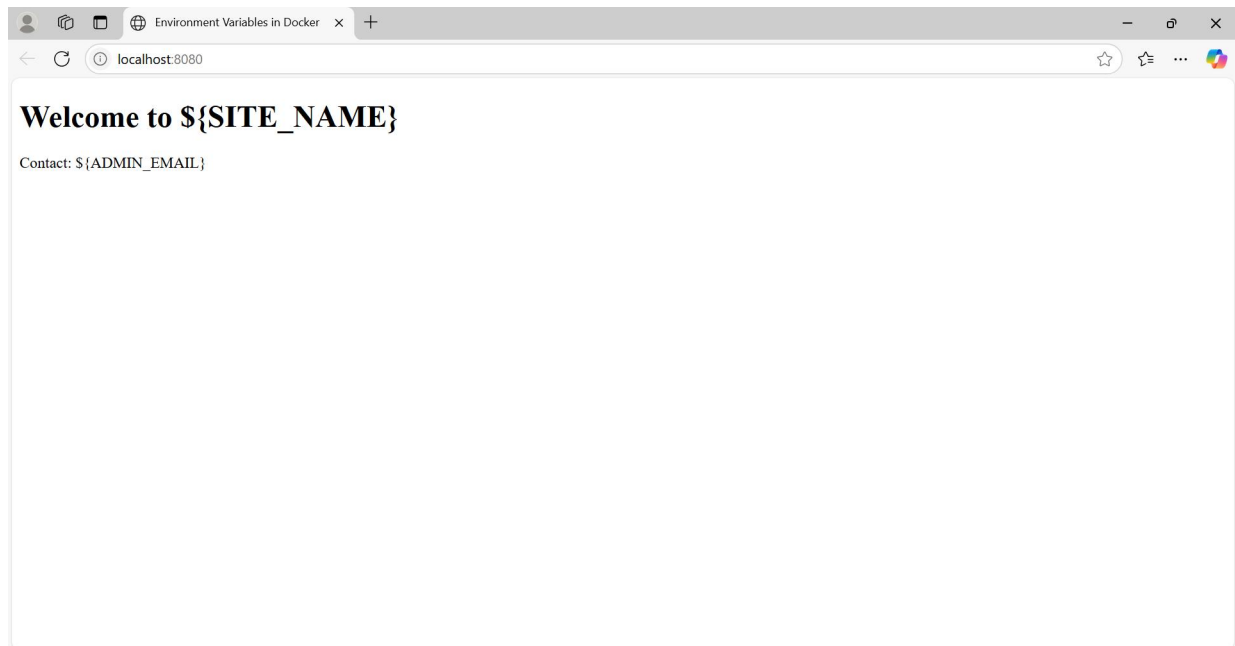
```
PS C:\Users\DELL\docker-env-poc> docker run -d --name my-nginx -p 8080:80 my-static-site
f06da6727ccc6651f1686123f9f846525d587b0673469cf8e48504d501fd8894
```

Step 9:

Open your browser and go to:

http://localhost:8080

If everything is correct, your static website should load inside the Docker container!



Step 8:

Check if your container is running:

docker ps

It should show my-nginx running.

```
PS C:\Users\DELL\docker-env-poc> docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
f06da6727ccc	my-static-site	"/docker-entrypoint..."	3 minutes ago	Up 3 minutes	0.0.0.0:8080->80/tcp	my-nginx

Step 9:

Run the following command to see CPU, memory, and network usage:

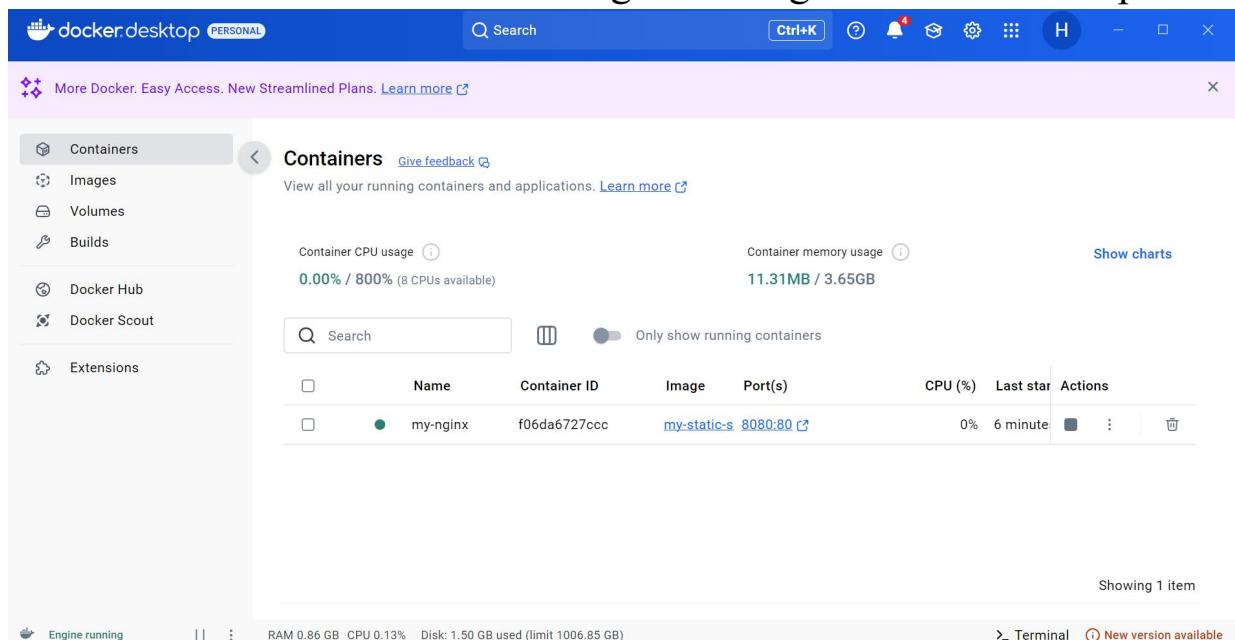
docker stats

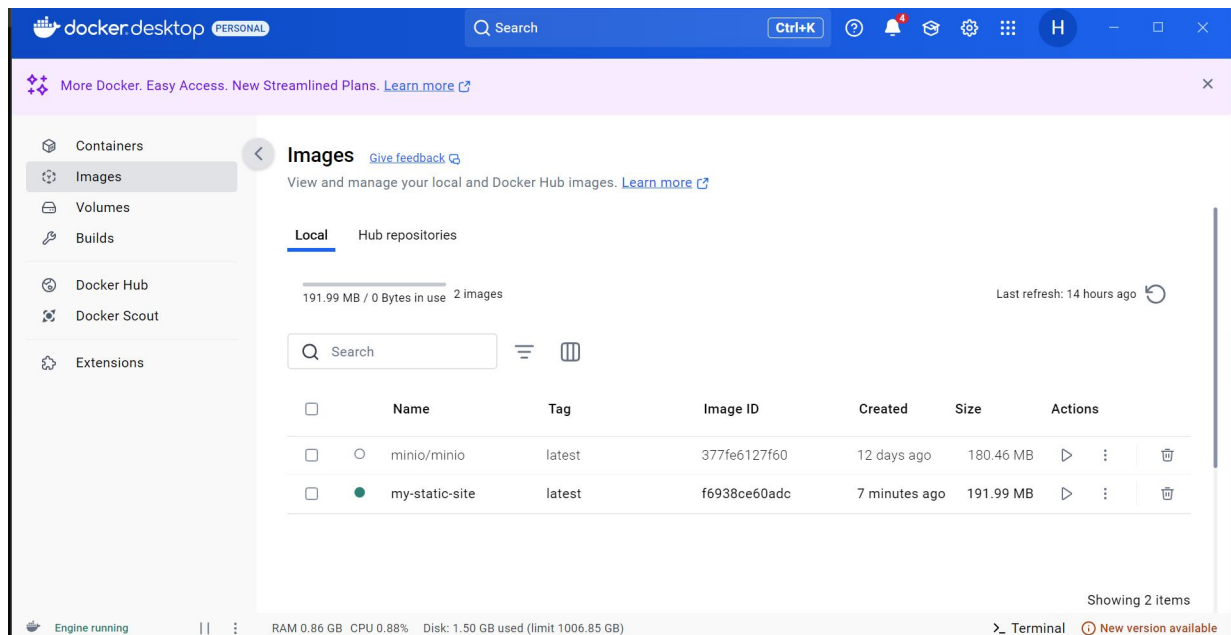
```
PS C:\Users\DELL\docker-env-poc> docker stats
```

CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O	PIDS
f06da6727ccc	my-nginx	0.00%	11.26MiB / 3.734GiB	0.29%	4.23kB / 2.56kB	0B / 0B	9

Step 10:

You can see the containers and images running in Docker Desktop.





Step 11:

Stop and Remove the Container (When Done)

docker stop my-nginx

docker rm my-nginx

```
PS C:\Users\DELL\docker-env-poc> docker stop my-nginx
my-nginx
PS C:\Users\DELL\docker-env-poc> docker rm my-nginx
my-nginx
```

Outcomes

By completing this POC, you will:

1. **Understand Environment Variables in Docker** – Learn how to pass configuration values dynamically to a container instead of hardcoding them in the application.
2. **Work with .env Files** – Gain hands-on experience in creating and managing .env files to store and retrieve environment variables securely.
3. **Use Docker Compose for Configuration Management** – Learn how to use **Docker Compose** to automatically load environment variables and simplify container deployment.
4. **Deploy an Nginx Web Server with Dynamic Content** – Set up an **Nginx container** that reads and displays environment variables inside a static webpage.
5. **Improve Docker Command Proficiency** – Enhance your skills with essential Docker commands like `docker run -e`, `docker ps`, `docker stop`, and `docker rm` for managing containerized applications efficiently.
6. **Troubleshoot and Debug Environment Variables** – Learn to verify environment variables inside a running container using `docker exec`, `docker logs`, and `printenv` commands.