11)You can still create the static website and Dockerize it without using **VS Code**. Here's how you can do everything using just the command line and a text editor of your choice (e.g., Notepad on Windows, Nano on Linux/macOS, or any editor you prefer).

2. Create the `index.html` File

Create the `index.html` file with a simple HTML structure. You can use **Notepad** (Windows) or **nano** (Linux/macOS) to create and edit the file.

- **On Windows (Notepad):**
  1. Open **Notepad** and paste the following HTML content:

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>My Static Website</title>
</head>
<body>
    <h1>Welcome to My Static Website</h1>
    <p>This is a simple static website served using Nginx inside a Docker containe
</body>
</html>
```

  2. Save the file as `index.html` in the `STATIC_WEBSITE` directory.

3. Create the `Dockerfile`

Next, create a `Dockerfile` that will instruct Docker to serve the static website using Nginx.

- **On Windows (Notepad):**
  1. Open **Notepad** and paste the following content:

```dockerfile
# Use the official Nginx image from Docker Hub
FROM nginx:alpine

# Copy the static website files to the Nginx web directory
COPY index.html /usr/share/nginx/html/index.html

# Expose port 80 to the outside world
EXPOSE 80
```

  2. Save the file as `Dockerfile` (no extension) in the `STATIC_WEBSITE` directory.

4. Build the Docker Image

Now that you have your `index.html` and `Dockerfile`, build the Docker image using the `docker build` command.

- Open a terminal/command prompt in the `STATIC_WEBSITE` directory and run the following:

  bash Copy code

  ```bash
  docker build -t static-website .
  ```

  This command will create a Docker image named `static-website` using the `Dockerfile` in the current directory (denoted by `.`).

4. Verify the Image

After building the image, you can verify that the image has been created by running:

bash Copy code

```bash
docker images
```

Look for the `static-website` image in the list.

5. Run the Docker Container

Once the image is successfully built, run it with the following command:

bash Copy code

```bash
docker run -d -p 8080:80 static-website
```

This command will start a container in detached mode (`-d`) and bind port `80` inside the container to port `8080` on your local machine.

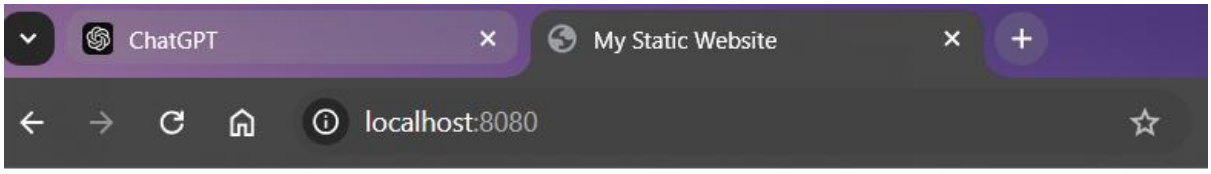
6. Access the Website

Open your browser and navigate to:

arduino Copy code

```arduino
http://localhost:8080
```

You should see the static website that you created!



Welcome to My Static Website

This is a simple static website served using Nginx inside a Docker container.

12)

1. Create a Simple Flask API

1. Create a Python file named `app.py` in your project directory:

bash Copy code

```bash
touch app.py
```

2. Add the following code to `app.py` using a text editor:

python Copy code

```python
from flask import Flask, jsonify

app = Flask(__name__)

@app.route('/')
def home():
    return jsonify({"message": "Welcome to the Flask API!"})

@app.route('/health')
def health_check():
    return jsonify({"status": "healthy"})

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

3. Verify it runs locally:

bash Copy code

```bash
python app.py
```

Python-

```
from flask import Flask, jsonify


app = Flask(__name__)


@app.route('/')
def home():
```

```
    return jsonify({"message": "Welcome to the Flask API!"})

@app.route('/health')
def health_check():
    return jsonify({"status": "healthy"})

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

2. Containerize the Flask App with a Dockerfile

1. Create a `Dockerfile` in the same directory as `app.py`:

bash Copy code

```bash
touch Dockerfile
```

2. Add the following content to the `Dockerfile`:

Dockerfile Copy code

```Dockerfile
# Use Python base image
FROM python:3.9-slim

# Set the working directory
WORKDIR /app

# Copy application code
COPY app.py /app/

# Install Flask
RUN pip install flask

# Expose port 5000
EXPOSE 5000

# Run the Flask application
CMD ["python", "app.py"]
```

```
# Use Python base image
FROM python:3.9-slim

# Set the working directory
WORKDIR /app

# Copy application code
COPY app.py /app/

# Install Flask
RUN pip install flask

# Expose port 5000
EXPOSE 5000

# Run the Flask application
CMD ["python", "app.py"]
```

3. Build the Docker Image

1. Build the Docker image:

```bash
docker build -t <your-dockerhub-username>/flask-api:latest .
```

2. Verify the image is built:

```bash
docker images
```

---

4. Push the Docker Image to Docker Hub

1. Push the image to Docker Hub:

```bash
docker push <your-dockerhub-username>/flask-api:latest
```

2. Verify the image is available on Docker Hub by logging into your Docker Hub account and checking the repository.

5. Create Kubernetes Manifests

Create a Deployment YAML File

1. Create `deployment.yaml`:

bash Copy code

```bash
touch deployment.yaml
```

2. Add the following content:

yaml Copy code

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: flask-api-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: flask-api
  template:
    metadata:
      labels:
        app: flask-api
    spec:
      containers:
      - name: flask-api
        image: <your-dockerhub-username>/flask-api:latest
        ports:
        - containerPort: 5000
```

```yaml
  replicas: 2
  selector:
    matchLabels:
      app: flask-api
  template:
    metadata:
      labels:
        app: flask-api
    spec:
      containers:
      - name: flask-api
        image: <your-dockerhub-username>/flask-api:latest
        ports:
        - containerPort: 5000
```

Create a Service YAML File

1. Create `service.yaml`:

```bash
touch service.yaml
```

2. Add the following content:

```yaml
apiVersion: v1
kind: Service
metadata:
  name: flask-api-service
spec:
```

apiVersion: apps/v1

kind: Deployment

metadata:

  name: flask-api-deployment

spec:

```
  replicas: 2
  selector:
    matchLabels:
      app: flask-api
  template:
    metadata:
      labels:
        app: flask-api
    spec:
      containers:
      - name: flask-api
        image: <your-dockerhub-username>/flask-api:latest
        ports:
        - containerPort: 5000
```

2 yaml

```
apiVersion: v1
kind: Service
metadata:
  name: flask-api-service
spec:
  selector:
    app: flask-api
  ports:
    - protocol: TCP
      port: 80
      targetPort: 5000
  type: NodePort
```

6. Deploy the Application to Kubernetes

1. Apply the Deployment and Service manifests:

   bash Copy code

   ```bash
   kubectl apply -f deployment.yaml
   kubectl apply -f service.yaml
   ```

2. Verify the Deployment and Pods:

   bash Copy code

   ```bash
   kubectl get deployments
   kubectl get pods
   ```

3. Verify the Service:

   bash Copy code

   ```bash
   kubectl get services
   ```

   Note the `NodePort` under the `PORT(S)` column.

7. Access the Flask API

1. Get the Kubernetes Node IP (if using Minikube):

   bash Copy code

   ```bash
   minikube ip
   ```

2. Access the API in your browser or using `curl`:

   bash Copy code

   ```bash
   curl http://<node-ip>:<node-port>/health
   ```

   Example:

   arduino Copy code

   ```arduino
   http://192.168.49.2:30007/health
   ```