

Building Application :

Click App (HTML,Node JS Frontend , Flask Backend)

aws instance --- > ubuntu 22 ---> create 2 folder

---> mkdir Backend , Frontend

---> apt install python3-pip -y

---> pip install flask

---> pip install flask-cors

---> apt install npm -y

---> sudo apt remove -y nodejs

---> sudo apt purge -y nodejs

---> sudo apt remove -y libnode-dev nodejs

---> sudo apt autoremove -y

---> sudo apt clean

---> curl -fsSL https://deb.nodesource.com/setup_20.x | sudo -E bash -

sudo apt install -y nodejs

---> cd Frontend

---> npm init -y

---> npm install express

---> cd Backend

---> touch app.py

---> touch business.py

---> touch names.txt

edit names.txt

----> nano names.txt

John

Mary

Alex

Sophia

David

Emma

Liam

Olivia

Noah

Ava

Ethan

Mia

Lucas

Isabella

James

Amelia

Benjamin

Charlotte

Henry

Harper

---> nano app.py

```
from flask import Flask
```

```
app = Flask(__name__)
```

```
@app.route('/')
```

```
def home():  
    return "Hello, World!"  
  
if __name__ == '__main__':  
    app.run(host='0.0.0.0', port=8000, debug=True)
```

---> python3 app.py

---> public-ip:8000

u can find hello world

modify business.py to fetch data from names.txt

---> nano business.py

```
def get_data():  
    try:  
        with open('names.txt', 'r') as f:  
            names = f.read().split()  
            return names  
    except FileNotFoundError:  
        return ["Error: names.txt not found"]
```

lets modify app.py in such a way that it fetches the data from names.txt using business.py

---> nano app.py

```
from flask import Flask, jsonify  
  
from flask_cors import CORS    # [] import CORS  
  
from business import get_data  
  
app = Flask(__name__)
```

```
CORS(app)          # enable CORS for all routes
```

```
@app.route('/')
```

```
def hello_world():
```

```
    return "Hello, World!"
```

```
@app.route('/api', methods=['GET'])
```

```
def api():
```

```
    data = get_data()
```

```
    response = {'data': data}
```

```
    return jsonify(response)
```

```
if __name__ == '__main__':
```

```
    app.run(host='0.0.0.0', port=8000, debug=True)
```

```
---> python3 app.py
```

```
---> public-ip:8000/api
```

here u get a json data . fine thats enough for backend now. Lets move with frontend application

Frontend is going to be a simple flutter application

Project Structure:

Frontend/

├─ package.json

├─ server.js

└─ public/

├─ index.html

├─ about.html

└─ css/

└─ styles.css

---> cd ..

---> cd Frontend

1) Create the files

---> npm init -y

this creates package.json similar to

```
{  
  "name": "my-express-site",  
  "version": "1.0.0",  
  "main": "server.js",  
  "scripts": {  
    "start": "node server.js"  
  },  
  "dependencies": {  
    "express": "^4.18.2"  
  }  
}
```

---> touch server.js

```
const express = require('express');
```

```
const path = require('path');
```

```
const app = express();
```

```
const PORT = process.env.PORT || 3000;

// 1) Serve everything in /public as static files (css, js, images, html)
app.use(express.static(path.join(__dirname, 'public')));

// 2) Optional: explicit routes that send specific HTML files
// (useful when you want server-side logic before serving)
app.get('/', (req, res) => {
  res.sendFile(path.join(__dirname, 'public', 'index.html'));
});

app.get('/about', (req, res) => {
  res.sendFile(path.join(__dirname, 'public', 'about.html'));
});

// 404 fallback for non-file routes (if you want)
app.use((req, res) => {
  res.status(404).sendFile(path.join(__dirname, 'public', '404.html'));
});

app.listen(PORT, () => {
  console.log(`Server running on http://localhost:${PORT}`);
});
```

---> cd public

---> touch index.html

```
<!doctype html>

<html>

<head>

  <meta charset="utf-8" />

  <title>Home — My Express Site</title>

  <link rel="stylesheet" href="/css/styles.css" />

</head>

<body>

  <h1>Welcome</h1>

  <p>This is the index page served by Express.</p>

  <button onclick="loadData()">Load Names</button>

  <ul id="namesList"></ul>

  <script>

    async function loadData() {

      const response = await fetch("http://<your-public-ip>:8000/api");

      const result = await response.json();

      const list = document.getElementById("namesList");

      list.innerHTML = "";

      result.data.forEach(name => {

        const li = document.createElement("li");

        li.textContent = name;

        list.appendChild(li);

      });

    }

  </script>

</body>

</html>
```

```
</script>
```

```
</body>
```

```
</html>
```

here in above index.html u have to replace ur instance public ip in above code

```
const response = await fetch("http://<your-public-ip>:8000/api");
```

---> touch about.html

```
<!doctype html>
```

```
<html>
```

```
<head>
```

```
<meta charset="utf-8" />
```

```
<title>About — My Express Site</title>
```

```
<link rel="stylesheet" href="/css/styles.css" />
```

```
</head>
```

```
<body>
```

```
<h1>About</h1>
```

```
<p>About page served with res.sendFile()</p>
```

```
<a href="/">Home</a>
```

```
</body>
```

```
</html>
```

---> touch 404.html

```
<!doctype html>
```

```
<html>
```

```
<head><meta charset="utf-8" /><title>Not Found</title></head>

<body><h1>404 — Not Found</h1><a href="/">Go home</a></body>

</html>
```

---> cd css

---> touch style.css

```
body { font-family: Arial, sans-serif; margin: 2rem; }

h1 { color: #2a6; }

a { color: #06c; text-decoration: none; }
```

change the style as per ur requirement

2. Install and run

---> npm install

---> npm start

Visit <http://localhost:3000> and <http://localhost:3000/about>.

Now lets work on dockerfile for both frontend and backend

Final Project Structure:

AWS Instance

|— Backend/

| |— app.py

| |— business.py

| |— names.txt

| |— requirements.txt

| |— Dockerfile

|

|— Frontend/

| |— server.js

| |— package.json

| |— public/

| | |— index.html

| | |— about.html

| | |— 404.html

| | |— css/

| | | |— styles.css

|— Dockerfile

□ 1 Backend — Dockerfile

---> cd Backend

---> nano requirements.txt

flask

flask-cors

---> nano Dockerfile

FROM python:3.9-slim

WORKDIR /app

COPY . /app

```
RUN pip install --no-cache-dir -r requirements.txt
```

```
EXPOSE 8000
```

```
CMD ["python3", "app.py"]
```

---> Build and run

```
docker build -t flask-backend .
```

```
docker run --name backend -d -p 8000:8000 flask-backend
```

---> Access:

```
http://<public-ip>:8000/api
```

2. Frontend — Node.js Dockerfile

---> nano Dockerfile

```
FROM node:18-alpine
```

```
WORKDIR /usr/src/app
```

```
COPY package*.json ./
```

```
RUN npm install
```

```
COPY . .
```

```
EXPOSE 3000
```

```
CMD ["npm", "start"]
```

Build and run:

```
docker build -t express-frontend .
```

```
docker run --name frontend -d -p 3000:3000 express-frontend
```

Access:

```
http://<public-ip>:3000
```

Good now u can see both the containers are working well and good but is this how a application should run ? like manually initialising each module separately ? no

ur entire appplication should run in single go , that is ur frontend and backend container should run equally automaticaly at same time , not seprately , so here comes the concept

docker compose.

Let's now see how to combine them into a single, coordinated setup using Docker Compose.

What Docker Compose Does:

Docker Compose allows you to:

1. Define multiple services (like frontend, backend, and even db) in one YAML file.
2. Start/stop all containers together using simple commands.
3. Automatically create a shared network so containers can communicate by name instead of IP.

Install docker-compose:

--->

Step 1. Directory structure:

project/

|

|— backend/

| |— app.py

| |— business.py

| |— requirements.txt

| └─ Dockerfile

|

|— frontend/

| |— package.json

| |— server.js (or app.js)

| └─ Dockerfile

|
└─ docker-compose.yml

□ Step 2. Example docker-compose.yml

```
version: '3.8'
```

```
services:
```

```
  backend:
```

```
    build: ./backend
```

```
    container_name: flask-backend
```

```
    ports:
```

```
      - "8000:8000"
```

```
    networks:
```

```
      - app-network
```

```
  frontend:
```

```
    build: ./frontend
```

```
    container_name: express-frontend
```

```
    ports:
```

```
      - "3000:3000"
```

```
    depends_on:
```

```
      - backend
```

```
    environment:
```

```
      - REACT_APP_API_URL=http://backend:8000
```

networks:

- app-network

networks:

app-network:

driver: bridge

□ Step 3. Run everything:

---> docker-compose up -d

Check running containers:

docker ps

□ Step 4. Stop and clean up

---> docker-compose down

To rebuild (after code changes):

---> docker-compose up -d --build

Step 5. Inside Frontend Code

In your frontend JS code, don't use the public IP — use:

fetch('http://backend:8000/api')

Because Docker Compose provides DNS resolution so backend → container IP automatically.

