📁 Project Structure and Explanation – aws-flask-express-deploy

The project consists of a Flask backend API and an Express.js frontend deployed using Docker and Docker Compose.
Each folder and file plays a crucial role in the containerized deployment pipeline.

🗂️ Directory Overview

```
aws-flask-express-deploy/
├── Backend/
│   ├── app.py
│   ├── Dockerfile
│   └── requirements.txt
├── Frontend/
│   ├── index.js
│   ├── package.json
│   ├── Dockerfile
│   └── docker-compose.yml
├── docker-compose.yml       # Combined deployment for both backend & frontend
└── readme.md                # Documentation
```

🧠 1. Backend/ – Flask API Service

The Backend folder contains the Python REST API built with Flask. It provides endpoints consumed by the frontend.

📄 app.py

The main Flask application exposing REST API routes.

```
from flask import Flask, jsonify

app = Flask(__name__)

@app.route("/")
def home():
    return jsonify({"message": "Hello from Flask backend!"})
```

```
@app.route("/api/data")
def data():
    return jsonify({"status": "success", "data": "This is data from Flask backend."})

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

Explanation:

Flask() creates the backend application.

/ – A simple root endpoint to test connectivity.

/api/data – A sample API endpoint returning JSON.

Runs on port 5000 to be accessible from Docker and EC2.

requirements.txt

This file lists Python dependencies required for the backend to run.

```
flask==2.2.5
gunicorn==21.2.0
```

Explanation:

flask – Web framework.

gunicorn – Production-ready WSGI server for deploying Flask apps in Docker.

Dockerfile

This Dockerfile builds the Flask backend image.

```
# Use official Python base image
FROM python:3.10-slim

# Set working directory
WORKDIR /app
```

```
# Copy dependency list and install
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Copy the application
COPY . .

# Expose port 5000 for the backend
EXPOSE 5000

# Use Gunicorn as production server
CMD ["gunicorn", "-w", "3", "-b", "0.0.0.0:5000", "app:app"]
```

Explanation:

Creates a lightweight Python container.

Installs dependencies from requirements.txt.

Runs Flask using Gunicorn with 3 workers.

Exposes port 5000 for backend service.

🧠 2. Frontend/ – Express.js Web Server

The Frontend directory contains a simple Node.js + Express server acting as the frontend, which calls the backend API.

📄 index.js

The entry point of the Express server.

```
const express = require("express");
const axios = require("axios");

const app = express();
const PORT = 3000;
```

```
// Read backend URL from environment variable
const BACKEND_URL = process.env.BACKEND_URL || "http://localhost:5000";

app.get("/", async (req, res) => {
 try {
  const response = await axios.get(`${BACKEND_URL}/`);
  res.send(`
   <h1>🚀 Express Frontend</h1>
   <p>✅ Backend says: ${response.data.message}</p>
  `);
 } catch (error) {
  res.status(500).send("❌ Error connecting to backend.");
 }
});

app.listen(PORT, () => {
 console.log(`✅ Frontend running on port ${PORT}`);
});
```

Explanation:

express – Creates a web server on port 3000.

axios – Makes HTTP requests to the Flask backend.

Reads BACKEND_URL from environment variables for container linking.

Displays backend response on the browser.

📄 package.json

Defines the frontend dependencies and scripts.

```
{
 "name": "express-frontend",
 "version": "1.0.0",
 "main": "index.js",
 "scripts": {
  "start": "node index.js"
```

```
 },
 "dependencies": {
  "axios": "^1.4.0",
  "express": "^4.18.2"
 }
}
```

Explanation:

Lists project metadata and dependencies.

npm start runs the server using node index.js.

Dockerfile

The Dockerfile builds the Express frontend container.

```
FROM node:18-alpine

WORKDIR /app

COPY package*.json ./
RUN npm install

COPY . .

EXPOSE 3000
CMD ["npm", "start"]
```

Explanation:

Uses lightweight node:18-alpine base image.

Installs dependencies.

Exposes port 3000.

Starts the server with npm start.

📄 docker-compose.yml (inside Frontend)

This is an optional standalone docker-compose.yml for frontend deployment (when running separately):

```
version: "3.8"
services:
 frontend:
  build: .
  ports:
   - "3000:3000"
  environment:
   BACKEND_URL: "http://<BACKEND_PUBLIC_IP>:5000"
  restart: always
```

Explanation:

Builds the frontend service from the Dockerfile.

Sets the backend URL dynamically.

Exposes port 3000.

🐬 Root-Level Files

📄 docker-compose.yml (Combined Deployment)

This is used in Task 1 when deploying both backend and frontend on the same EC2 instance.

```
version: "3.8"

services:
 backend:
  build: ./Backend
  container_name: flask_backend
  ports:
   - "5000:5000"
  restart: always
```

```
 frontend:
   build: ./Frontend
   container_name: express_frontend
   ports:
    - "3000:3000"
   environment:
    BACKEND_URL: "http://backend:5000"
   depends_on:
    - backend
   restart: always
```

Explanation:

Defines two services: backend (Flask) and frontend (Express).

Uses Docker network to communicate via service name (backend).

Automatically restarts containers if they fail.

readme.md

This file contains the project documentation — including:

How to build and run the containers.

Deployment steps for AWS EC2.

Commands for debugging and testing.

Project structure explanation.