**AWS ANSWERS**

* EC2 Instance: Host both Flask and Express apps.
* Reverse Proxy (Nginx): Route traffic to backend or frontend.
* PM2 or systemd: Manage app processes.
* Gunicorn: WSGI server to serve Flask app.

🔧 Step-by-Step Deployment Guide

1. Launch an EC2 Instance

* Choose Amazon Linux 2 or Ubuntu 22.04.
* Allow SSH and HTTP/HTTPS in security group.
* Connect to EC2 via SSH:

bash

CopyEdit

ssh -i your-key.pem ec2-user@your-ec2-public-ip

2. Install Required Software

Ubuntu/Debian:

bash

CopyEdit

sudo apt update && sudo apt install -y nginx python3-pip nodejs npm git

sudo npm install -g pm2

Amazon Linux 2:

bash

CopyEdit

sudo yum update -y

sudo yum install -y nginx python3 git

curl -sL https://rpm.nodesource.com/setup\_18.x | sudo bash -

sudo yum install -y nodejs

sudo npm install -g pm2

3. Clone Your Code or Upload It

You can use git, scp, or a CI/CD pipeline to get your app code onto the instance.

bash

CopyEdit

git clone https://github.com/your-repo.git

cd your-repo

Assume:

* backend/ has Flask app (app.py)
* frontend/ has Express app (app.js)

4. Set Up Flask Backend

bash

CopyEdit

cd backend

python3 -m venv venv

source venv/bin/activate

pip install flask gunicorn

# Optionally install other dependencies

gunicorn -w 4 -b 127.0.0.1:5000 app:app

Or run with systemd or PM2:

bash

CopyEdit

pm2 start "gunicorn -b 127.0.0.1:5000 app:app" --name flask-backend

5. Set Up Express Frontend

bash

CopyEdit

cd ../frontend

npm install

pm2 start app.js --name express-frontend

Make sure Express listens on a non-conflicting port (e.g., 3000).

6. Configure Nginx as Reverse Proxy

bash

CopyEdit

sudo nano /etc/nginx/sites-available/default

Replace with something like this:

nginx

CopyEdit

server {

listen 80;

location /api/ {

proxy\_pass http://127.0.0.1:5000/;

proxy\_set\_header Host $host;

}

location / {

proxy\_pass http://127.0.0.1:3000/;

proxy\_set\_header Host $host;

}

}

Test and restart Nginx:

bash

CopyEdit

sudo nginx -t

sudo systemctl restart nginx

7. Enable Autostart on Reboot

bash

CopyEdit

pm2 startup

pm2 save

8. Firewall / Security Group Settings

Ensure these are open:

* Port 22 (SSH)
* Port 80 (HTTP)
* Port 443 (HTTPS if SSL is configured)

✅ Example Directory Structure

pgsql

CopyEdit

/home/ec2-user/

├── backend/

│ ├── app.py

│ └── requirements.txt

├── frontend/

│ ├── app.js

│ └── package.json

🛡️ (Optional) Set Up SSL with Let’s Encrypt

bash

CopyEdit

sudo apt install certbot python3-certbot-nginx

sudo certbot --nginx

🔄 Summary

| Component | Tech | Port | Role |
| --- | --- | --- | --- |
| Flask Backend | Gunicorn | 5000 | Handles /api requests |
| Express Front | Express/Node | 3000 | Serves web content |
| Reverse Proxy | Nginx | 80/443 | Routes traffic |

**Separate ec2 instances**

Step-by-Step Deployment Guide

🔁 Common Setup for Both EC2 Instances

1. Launch Two EC2 Instances

* Use Ubuntu 22.04 or Amazon Linux 2.
* Open required ports in the security group:
  + Backend: Allow SSH (22), HTTP (80), and optionally a custom port (like 5000).
  + Frontend: Allow SSH (22), HTTP (80).

2. Connect to EC2 Instances

bash

CopyEdit

ssh -i your-key.pem ec2-user@<BACKEND\_PUBLIC\_IP>

# Or

ssh -i your-key.pem ec2-user@<FRONTEND\_PUBLIC\_IP>

🖥️ Backend: Flask on EC2 Instance A

A. Install Dependencies

bash

CopyEdit

sudo apt update && sudo apt install -y python3-pip nginx git

sudo pip3 install virtualenv gunicorn flask

B. Upload or Clone Your Backend Code

bash

CopyEdit

git clone https://github.com/your-user/your-backend.git

cd your-backend

python3 -m venv venv

source venv/bin/activate

pip install -r requirements.txt

C. Run Flask with Gunicorn

bash

CopyEdit

gunicorn -w 4 -b 0.0.0.0:5000 app:app

Or manage with pm2 or systemd.

D. Allow Incoming Port 5000 in AWS Security Group

Or use Nginx reverse proxy to port 80 if preferred.

🌐 Frontend: Express on EC2 Instance B

A. Install Node.js & PM2

bash

CopyEdit

curl -fsSL https://deb.nodesource.com/setup\_18.x | sudo -E bash -

sudo apt install -y nodejs nginx git

sudo npm install -g pm2

B. Upload or Clone Your Express App

bash

CopyEdit

git clone https://github.com/your-user/your-frontend.git

cd your-frontend

npm install

C. Update API Calls in Express App

In your frontend code (where it fetches from Flask), replace:

js

CopyEdit

// Old (e.g., local)

fetch('http://localhost:5000/api/...')

// New

fetch('http://<BACKEND\_PUBLIC\_IP>:5000/api/...')

Or better, use an environment variable.

D. Run Express App

bash

CopyEdit

pm2 start app.js --name express-frontend

📦 Optional: Set Up Nginx for Both

On each instance, install Nginx and use it as a reverse proxy to expose your apps on port 80 instead of 5000 or 3000.

Example Nginx Config on Backend EC2:

nginx

CopyEdit

server {

listen 80;

location / {

proxy\_pass http://127.0.0.1:5000;

}

}

bash

CopyEdit

sudo nginx -t

sudo systemctl restart nginx

🔐 Security Group Rules Summary

| Instance | Port(s) to Open | Notes |
| --- | --- | --- |
| Flask (A) | 22, 5000 or 80 | Expose 5000 or reverse proxy via Nginx |
| Express (B) | 22, 80 | Standard HTTP traffic |

✅ Summary of URLs

| Component | Hosted On | Example URL |
| --- | --- | --- |
| Flask Backend | EC2 A (e.g., 3.1.2.3) | http://3.1.2.3:5000/api/endpoint |
| Express Front | EC2 B (e.g., 44.3.2.1) | http://44.3.2.1/ (makes API calls to backend) |

📦 Optional Enhancements

* Use domains via Route 53 or external DNS
* Add SSL/TLS with Certbot (https://)
* Use Docker for easier deployment
* Use environment variables to manage configs
* Set up systemd or PM2 with auto-restart on reboot

**Docker Container using aws ecr, ecs and vpc services**

**🔧 Prerequisites**

* AWS CLI configured (aws configure)
* Docker installed
* Two apps: Flask backend and Express frontend
* IAM permissions to create ECR, ECS, VPC, etc.

**🧱 Project Structure**

pgsql

CopyEdit

project/

├── flask-backend/

│ ├── app.py

│ └── Dockerfile

├── express-frontend/

│ ├── server.js

│ └── Dockerfile

**🐳 Step 1: Build and Push Docker Images to ECR**

**1.1 Create ECR Repositories**

bash

CopyEdit

aws ecr create-repository --repository-name flask-backend

aws ecr create-repository --repository-name express-frontend

**1.2 Authenticate Docker to ECR**

bash

CopyEdit

aws ecr get-login-password --region us-east-1 | \

docker login --username AWS --password-stdin <your-aws-account-id>.dkr.ecr.us-east-1.amazonaws.com

**1.3 Build and Push Images**

**Flask Backend**

bash

CopyEdit

cd flask-backend

docker build -t flask-backend .

docker tag flask-backend:latest <aws-id>.dkr.ecr.us-east-1.amazonaws.com/flask-backend

docker push <aws-id>.dkr.ecr.us-east-1.amazonaws.com/flask-backend

**Express Frontend**

bash

CopyEdit

cd express-frontend

docker build -t express-frontend .

docker tag express-frontend:latest <aws-id>.dkr.ecr.us-east-1.amazonaws.com/express-frontend

docker push <aws-id>.dkr.ecr.us-east-1.amazonaws.com/express-frontend

**🌐 Step 2: Create VPC and Networking Components**

You can do this manually via AWS Console or use the CLI:

bash

CopyEdit

aws ec2 create-vpc --cidr-block 10.0.0.0/16

# (then create subnets, route tables, IGW, etc.)

Alternatively, use **VPC Wizard** in the AWS Console to create a VPC with public subnets.

**🛳️ Step 3: Deploy Containers using ECS**

**3.1 Create ECS Cluster**

bash

CopyEdit

aws ecs create-cluster --cluster-name flask-express-cluster

Or use the ECS console to create a new cluster with EC2 or Fargate (we'll use **Fargate**).

**3.2 Create Task Definitions**

Define two task definitions (or one with both containers):

json

CopyEdit

{

"family": "flask-express-task",

"networkMode": "awsvpc",

"requiresCompatibilities": ["FARGATE"],

"cpu": "256",

"memory": "512",

"containerDefinitions": [

{

"name": "flask-backend",

"image": "<aws-id>.dkr.ecr.us-east-1.amazonaws.com/flask-backend",

"essential": true,

"portMappings": [{ "containerPort": 5000 }]

},

{

"name": "express-frontend",

"image": "<aws-id>.dkr.ecr.us-east-1.amazonaws.com/express-frontend",

"essential": true,

"portMappings": [{ "containerPort": 3000 }]

}

]

}

Save as task-def.json and register:

bash

CopyEdit

aws ecs register-task-definition --cli-input-json file://task-def.json

**3.3 Create ECS Service**

bash

CopyEdit

aws ecs create-service \

--cluster flask-express-cluster \

--service-name flask-express-service \

--task-definition flask-express-task \

--desired-count 1 \

--launch-type FARGATE \

--network-configuration "awsvpcConfiguration={subnets=[subnet-xxxx],securityGroups=[sg-xxxx],assignPublicIp=ENABLED}"

**🌍 Step 4: Expose Services**

* Ensure your **security group** allows inbound traffic on ports 80, 3000, and 5000.
* Use an **Application Load Balancer** (ALB) for routing traffic if needed.
* Map frontend to root / and backend to /api for example.

**✅ Done!**

You’ve now:

* Pushed Docker images to **ECR**
* Deployed containers using **ECS Fargate**
* Networked them via **VPC**

**🧩 Optional Enhancements**

* Use Route 53 for custom domain
* Add HTTPS with ACM and ALB
* Use Terraform/CDK for infrastructure as code