1. Deploy Your flask backend and express frontend in amazon single ec2 instance.

2. Deploy Your flask backend and express frontend in separate ec2 instances.

3. Deploy Your flask backend and express frontend Docker Container using aws ecr, ecs and vpc services

EXPLANATION:-

# **1) Deploy Flask backend + Express frontend on a single EC2 instance**

**Assumptions**

* Ubuntu 22.04 (or similar) EC2 instance
* You have SSH key YOUR\_KEY.pem
* App code is in /home/ubuntu/app with frontend and backend folders as from your project

## **Steps**

### **A. Launch EC2**

1. Launch EC2 instance (t2.micro or t3.small). Choose your region (e.g. us-east-1).
2. Security Group: allow inbound 22 (SSH), 80 (HTTP) and 443 (HTTPS) from your IP/0.0.0.0/0 as needed.

### **B. Connect & update**

ssh -i YOUR\_KEY.pem ubuntu@EC2\_PUBLIC\_IP

sudo apt update && sudo apt upgrade -y

### **C. Install Node, Python, pip, Nginx, and build tools**

# Node (NodeSource)

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

sudo apt install -y nodejs build-essential

# Python & pip

sudo apt install -y python3 python3-venv python3-pip

# nginx

sudo apt install -y nginx

### **D. Copy code to server**

Option A: git clone your repo:

cd /home/ubuntu

git clone https://github.com/YOUR\_USERNAME/cloud-form-project.git

cd cloud-form-project

Or scp/upload zip and extract.

### **E. Setup backend (Flask)**

cd /home/ubuntu/cloud-form-project/backend

python3 -m venv venv

source venv/bin/activate

pip install -r requirements.txt

# Test run

python app.py # listens on 0.0.0.0:5000

Create a systemd service to run Flask with gunicorn:

**/etc/systemd/system/backend.service**

[Unit]

Description=Flask Backend

After=network.target

[Service]

User=ubuntu

WorkingDirectory=/home/ubuntu/cloud-form-project/backend

Environment="PATH=/home/ubuntu/cloud-form-project/backend/venv/bin"

ExecStart=/home/ubuntu/cloud-form-project/backend/venv/bin/gunicorn --bind 127.0.0.1:5000 app:app

Restart=always

[Install]

WantedBy=multi-user.target

Enable & start:

sudo systemctl daemon-reload

sudo systemctl enable backend

sudo systemctl start backend

sudo journalctl -u backend -f

### **F. Setup frontend (Express static)**

cd /home/ubuntu/cloud-form-project/frontend

npm install

# run once to test:

node index.js

Create systemd service:

**/etc/systemd/system/frontend.service**

[Unit]

Description=Node Frontend

After=network.target

[Service]

User=ubuntu

WorkingDirectory=/home/ubuntu/cloud-form-project/frontend

ExecStart=/usr/bin/node /home/ubuntu/cloud-form-project/frontend/index.js

Restart=always

[Install]

WantedBy=multi-user.target

Enable & start:

sudo systemctl daemon-reload

sudo systemctl enable frontend

sudo systemctl start frontend

### **G. Configure Nginx as reverse proxy (serve frontend on port 80 and proxy API)**

Create nginx site config /etc/nginx/sites-available/cloudapp:

server {

listen 80;

server\_name your\_domain\_or\_public\_ip;

# serve frontend static

location / {

proxy\_pass http://127.0.0.1:3000;

proxy\_http\_version 1.1;

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

}

# proxy API /submit to backend

location /submit {

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

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

}

location /health {

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

}

}

Enable and restart:

sudo ln -s /etc/nginx/sites-available/cloudapp /etc/nginx/sites-enabled/

sudo nginx -t

sudo systemctl restart nginx

### **H. Test**

* Visit http://EC2\_PUBLIC\_IP → frontend should load.
* Submit form → backend response echoed.
* Use sudo systemctl status frontend backend nginx to debug.

**Notes**

* For production, enable HTTPS with Let’s Encrypt certbot.
* Use UFW or security group rules to limit access if required.

# **2) Deploy frontend and backend on separate EC2 instances**

**Goal:** front and backend run on separate EC2 machines. Frontend calls backend by its private/public IP or DNS (secure via SG).

## **Steps**

### **A. Launch two EC2 instances**

* EC2-FRONT (for frontend): allow ports 22, 80
* EC2-BACK (for backend): allow ports 22, 5000 (or only permit traffic from frontend SG)

**Best practice:** create two security groups:

* sg-frontend: inbound 22,80/443 from your IP
* sg-backend: inbound 22 from your IP and inbound 5000 only from sg-frontend (not 0.0.0.0/0) — allows only frontend to talk to backend.

### **B. Setup backend instance (EC2-BACK)**

Follow same backend steps from single EC2 (Python venv, gunicorn, systemd). Bind backend to private IP or localhost and allow traffic from frontend SG. If using private IP only, frontend must be in same VPC/subnet or use public IP.

Example: ensure backend is reachable on private IP 10.0.1.5:5000. Use ExecStart=... gunicorn --bind 0.0.0.0:5000 app:app.

### **C. Setup frontend instance (EC2-FRONT)**

Install Node, clone frontend repo. Edit frontend JS to point to backend address:

In frontend/public/index.html change:

const BACKEND\_URL = 'http://BACKEND\_PRIVATE\_IP:5000/submit';

If backend uses public IP, use that; better is backend private IP inside same VPC.

Run frontend as before and set up Nginx to serve frontend static and proxy only to backend if needed (or serve static files directly).

### **D. Security Groups (important)**

* Backend SG inbound rule: TCP 5000 source sg-frontend (or frontend private IP)
* Backend SG outbound: allow 0.0.0.0/0 (or VPC)
* Frontend DNS: you may use Elastic IP for frontend to keep stable public IP

### **E. Test**

* Visit http://FRONTEND\_PUBLIC\_IP and submit. Frontend should call backend private IP; verify sudo journalctl -u backend -f in backend instance shows request.

**Notes**

* Use internal DNS (e.g., use private hostnames or AWS Route53 private hosted zone) instead of hardcoding private IPs.
* For security, do not expose backend 5000 to internet.

# **3) Deploy both using Docker → ECR → ECS (Fargate) with VPC + ALB**

This is the modern containerized approach: build images, push to **ECR**, then run on **ECS (Fargate)** with an **Application Load Balancer (ALB)** inside a VPC.

## **Overview Steps**

1. Create ECR repositories (frontend & backend)
2. Build & push Docker images to ECR
3. Create an ECS cluster (Fargate) inside your VPC & subnets
4. Create Task Definitions for frontend & backend (use awsvpc networking)
5. Create Services for both tasks and attach to ALB target groups
6. Configure ALB listeners & health checks → route / to frontend, /submit proxied to backend or frontend serves client and calls backend via internal TG and DNS

I’ll provide CLI commands and sample task definition snippets. Replace REGION, AWS\_ACCOUNT\_ID, and network placeholders.

### **A. Create ECR repositories**

AWS\_REGION=us-east-1

ACCOUNT\_ID=AWS\_ACCOUNT\_ID

aws ecr create-repository --repository-name cloud-frontend --region $AWS\_REGION

aws ecr create-repository --repository-name cloud-backend --region $AWS\_REGION

### **B. Build & push images (local)**

Login, build, tag, push:

# login

aws ecr get-login-password --region $AWS\_REGION | docker login --username AWS --password-stdin $ACCOUNT\_ID.dkr.ecr.$AWS\_REGION.amazonaws.com

# build & tag frontend

docker build -t cloud-frontend:latest ./frontend

docker tag cloud-frontend:latest $ACCOUNT\_ID.dkr.ecr.$AWS\_REGION.amazonaws.com/cloud-frontend:latest

docker push $ACCOUNT\_ID.dkr.ecr.$AWS\_REGION.amazonaws.com/cloud-frontend:latest

# build & tag backend

docker build -t cloud-backend:latest ./backend

docker tag cloud-backend:latest $ACCOUNT\_ID.dkr.ecr.$AWS\_REGION.amazonaws.com/cloud-backend:latest

docker push $ACCOUNT\_ID.dkr.ecr.$AWS\_REGION.amazonaws.com/cloud-backend:latest

### **C. Prepare VPC & subnets**

You can use the default VPC or create a new VPC with at least two public subnets (for ALB) and two private subnets (for tasks). In console this can be done via VPC wizard. Note the subnet IDs.

### **D. Create ECS cluster (Fargate)**

aws ecs create-cluster --cluster-name cloud-cluster --capacity-providers FARGATE --region $AWS\_REGION

### **E. Create IAM execution role for ECS tasks**

ECS needs ecsTaskExecutionRole with AmazonECSTaskExecutionRolePolicy. AWS console often creates it automatically; otherwise create via CLI or console.

### **F. Create ALB (Application Load Balancer)**

Use console or CLI. Example CLI (simplified):

# create security group for alb (allow 80)

# create ALB

aws elbv2 create-load-balancer --name cloud-alb --subnets subnet-1 subnet-2 --security-groups sg-alb --region $AWS\_REGION

# create target groups for frontend and backend

aws elbv2 create-target-group --name tg-frontend --protocol HTTP --port 80 --vpc-id vpc-xxxx --target-type ip

aws elbv2 create-target-group --name tg-backend --protocol HTTP --port 5000 --vpc-id vpc-xxxx --target-type ip

# create listeners

aws elbv2 create-listener --load-balancer-arn ALB\_ARN --protocol HTTP --port 80 --default-actions Type=forward,TargetGroupArn=TG\_FRONTEND\_ARN

# You may add path-based rules to forward /api or /submit to backend TG

### **G. Task Definition (Fargate) — examples**

**frontend-task.json** (snippet)

{

"family": "frontend-task",

"networkMode": "awsvpc",

"requiresCompatibilities": ["FARGATE"],

"cpu": "256",

"memory": "512",

"executionRoleArn": "arn:aws:iam::ACCOUNT\_ID:role/ecsTaskExecutionRole",

"containerDefinitions": [

{

"name": "frontend",

"image": "ACCOUNT\_ID.dkr.ecr.REGION.amazonaws.com/cloud-frontend:latest",

"portMappings": [{ "containerPort": 3000, "protocol": "tcp" }],

"essential": true,

"healthCheck": { "command": ["CMD-SHELL","curl -f http://localhost:3000/health || exit 1"], "interval": 30, "timeout": 5, "retries": 3 }

}

]

}

**backend-task.json** (snippet)

{

"family": "backend-task",

"networkMode": "awsvpc",

"requiresCompatibilities": ["FARGATE"],

"cpu": "256",

"memory": "512",

"executionRoleArn": "arn:aws:iam::ACCOUNT\_ID:role/ecsTaskExecutionRole",

"containerDefinitions": [

{

"name": "backend",

"image": "ACCOUNT\_ID.dkr.ecr.REGION.amazonaws.com/cloud-backend:latest",

"portMappings": [{ "containerPort": 5000, "protocol": "tcp" }],

"essential": true,

"healthCheck": { "command": ["CMD-SHELL","curl -f http://localhost:5000/health || exit 1"], "interval": 30, "timeout": 5, "retries": 3 }

}

]

}

Register task definitions:

aws ecs register-task-definition --cli-input-json file://frontend-task.json --region $AWS\_REGION

aws ecs register-task-definition --cli-input-json file://backend-task.json --region $AWS\_REGION

### **H. Create ECS Services (attach to ALB target groups)**

Use aws ecs create-service for each:

Example frontend service:

aws ecs create-service \

--cluster cloud-cluster \

--service-name frontend-service \

--task-definition frontend-task \

--desired-count 2 \

--launch-type FARGATE \

--network-configuration "awsvpcConfiguration={subnets=[subnet-1,subnet-2],securityGroups=[sg-ecs],assignPublicIp=ENABLED}" \

--load-balancers "targetGroupArn=TG\_FRONTEND\_ARN,containerName=frontend,containerPort=3000" \

--region $AWS\_REGION

Example backend service:

aws ecs create-service \

--cluster cloud-cluster \

--service-name backend-service \

--task-definition backend-task \

--desired-count 2 \

--launch-type FARGATE \

--network-configuration "awsvpcConfiguration={subnets=[subnet-priv-1,subnet-priv-2],securityGroups=[sg-ecs-backend],assignPublicIp=DISABLED}" \

--load-balancers "targetGroupArn=TG\_BACKEND\_ARN,containerName=backend,containerPort=5000" \

--region $AWS\_REGION

**Important networking notes**

* Use awsvpc so each task gets an elastic network interface in the VPC.
* Place frontend tasks in public subnets (or with NAT & ALB) and backend in private subnets. Backend SG must allow inbound from ALB or frontend SG.
* ALB target type ip + ECS awsvpc works well.

### **I. Configure ALB routing**

* Set ALB listener rules: /submit\* or /api/\* → backend target group. Root / → frontend target group.
* Health checks: set to /health endpoint for both TGs.

### **J. Test**

* Visit ALB DNS (found in EC2 console under Load Balancers) → should show frontend.
* Submit form → frontend (in browser) will make requests to backend path. If you route /submit to backend, frontend can call /submit relative path and ALB will forward accordingly.

### **K. CI/CD (optional)**

* Use GitHub Actions to build Docker images and push to ECR on push to main.
* Use ECS Deploy actions to update ECS service after push.