AWS Assignments:

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

SOLUTION:

## PART 1 — Deploy both Flask (backend) and Express (frontend) on a single EC2 instance

SOLUTION

1)Launch EC2 instances

2)SSH to instances

chmod 400 YOUR\_KEY.pem

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

3)Install dependencies and tools

# Run as ubuntu

sudo apt update -y

sudo apt install -y git nginx python3 python3-venv python3-pip nodejs npm build-essential

# Optional: pm2 for node

sudo npm install -g pm2

# Optional: gunicorn installed later per venv

4)Backend Setup

cd ~/backend

python3 -m venv venv

source venv/bin/activate

pip install -r requirements.txt

pip install gunicorn

# test run

python app.py # or gunicorn --bind 0.0.0.0:5000 app:app

CREATE SYSTEMD UNIT /etc/systemd/system/backend.service:

[Unit]

Description=Flask Backend

After=network.target

[Service]

User=ubuntu

WorkingDirectory=/home/ubuntu/backend

Environment="PATH=/home/ubuntu/backend/venv/bin"

ExecStart=/home/ubuntu/backend/venv/bin/gunicorn --workers 3 --bind 127.0.0.1:5000 app:app

Restart=always

[Install]

WantedBy=multi-user.target

6)FRONTEND SETUP;

cd ~/frontend

npm install

# set backend URL used by frontend to call backend, e.g., http://127.0.0.1:5000

# start with pm2

pm2 start index.js --name frontend

pm2 save

pm2 startup # follow printed command

7)NGINIX REVERSE PROXY (SERVE ON PORT 80)

server {

listen 80;

server\_name \_;

location / {

proxy\_pass http://127.0.0.1:3000;

proxy\_set\_header Host $host;

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

}

location /api/ {

# if your backend endpoints are prefixed with /api

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

proxy\_set\_header Host $host;

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

}

}

8)TEST

[http://EC2\_PUBLIC\_IP](http://ec2_public_ip)

PART 2 — Deploy Flask and Express on two separate EC2 instances

SOLUTION

## 1)CREATE A VPC

2)CREATE two security groups

Sg-frontend

Sg-backend

3)Launch EC2 instances

Instance A

Instance B

FRONTEND INSTANCE(EC2-FRONT)

const BACKEND\_URL = 'http://10.0.1.25:5000'; // backend private IP

4)TEST

[http://FRONTEND\_PUBLIC\_IP](http://frontend_public_ip)

PART 3 — Deploy Flask & Express as Docker containers using ECR, ECS (Fargate), VPC & ALB (recommended production)

SOLUTION

A)Build and push docker images to ECR

1)create ECR repos:

AWS\_REGION=us-east-1

ACCOUNT\_ID=AWS\_ACCOUNT\_ID

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

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

2)Authenticate Docker to ECR

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

3)Build & tag images (run from project root):

# frontend

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

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

# backend

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

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

4)push

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

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

B)CREATE VPC AND SUBNETS

C)CREATE ECS CLUSTER (FRAGATE)

aws ecs create-cluster --cluster-name zenfit-cluster --region $AWS\_REGION

D)IAM ROLE FOR TASK EXECUTION

aws iam create-role --role-name ecsTaskExecutionRole --assume-role-policy-document file://assume-role.json

aws iam attach-role-policy --role-name ecsTaskExecutionRole --policy-arn arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy

Assume-role.json

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Principal": {"Service": "ecs-tasks.amazonaws.com"},

"Action": "sts:AssumeRole"

}

]

}

E)ALB AND TARGATE GROUPS

F)REGISTER TASK DEFINITIONS (EX JSON)

FRONTEND-TASK.JSON

{

"family": "zenfit-frontend",

"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/zenfit-frontend:latest",

"essential": true,

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

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

}

]

}

BACKENDS-TASK.JSON

{

"family": "zenfit-backend",

"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/zenfit-backend:latest",

"essential": true,

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

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

}

]

}

G)CREATE ECS SERVICES

aws ecs create-service \

--cluster zenfit-cluster \

--service-name frontend-service \

--task-definition zenfit-frontend \

--desired-count 2 \

--launch-type FARGATE \

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

--load-balancers "targetGroupArn=FRONTEND\_TG\_ARN,containerName=frontend,containerPort=3000" \

--region $AWS\_REGION

H)SECURITY GROUPS

I)TEST

Open ALB DNS in browser → should show frontend.  
Submit form → ALB forwards /api to backend TG → backend responds.  
Use CloudWatch logs (enable awslogs in task def) to debug.