AWS: EC-2 & ECS[Ecr,Vpc]

-----**Part 1: Single EC2 Instance Deployment**

Objective: Deploy both your Flask backend and Express frontend on a single Amazon EC2 instance.

1. **Launch an EC2 Instance**
   - Navigate to AWS Console → EC2 → Launch Instance.
   - **AMI:** Ubuntu 22.04 LTS
   - **Instance type:** t2.micro (Free tier eligible)
   - **Key pair:** Create or download one (e.g., aws-key.pem).
   - **Security Group:** Configure inbound rules to allow:
     - Port 22 (SSH)
     - Port 5000 (Flask)
     - Port 3000 (Express)
     - Port 80 (HTTP, if using Nginx)
2. **Connect to the Instance**
   - `chmod 400 aws-key.pem`
   - `ssh -i "aws-key.pem" ubuntu@<your-ec2-public-ip>`
3. **Install Dependencies**
   - `sudo apt update -y`
   - `sudo apt install python3-pip python3-venv git nginx -y`
   - `sudo apt install nodejs npm -y`
4. **Clone Your Project**
   - `git clone [https://github.com/](https://github.com/)<your-username>/<your-repo>.git`
   - `cd <your-repo>`
5. **Run Backend**
   - `cd backend`
   - `python3 -m venv venv`
   - `source venv/bin/activate`
   - `pip install -r requirements.txt`
   - `nohup python3 app.py &`
   - **Test:** `curl http://localhost:5000`
6. **Run Frontend**
   - `cd ../frontend`
   - `npm install`
   - `nohup node server.js &`

    - **Test:** `curl http://localhost:3000`
7. **Access via Browser**
    - Go to: `http://<EC2-PUBLIC-IP>:3000`
    - **Note:** Your Express frontend will call Flask via its internal URL. Update the API base URL to `http://<EC2-PUBLIC-IP>:5000`.

Optional: Use Nginx for Reverse Proxy

Configure `/etc/nginx/sites-available/default`:

```
server {
    listen 80;
    location / {
        proxy_pass http://localhost:3000;
    }
    location /api/ {
        proxy_pass http://localhost:5000/;
    }
}
```

- **Restart Nginx:** `sudo systemctl restart nginx`
- **Access:** `http://<EC2-PUBLIC-IP>/`

-----**Part 2: Separate EC2 Instances**

Objective: Deploy the backend and frontend on their own EC2 instances and connect them.

1. **Create Two EC2 Instances**
    - `flask-backend-instance`
    - `node-frontend-instance`
    - **Configuration for both:** Ubuntu 22.04, t2.micro, same key pair, same security group.
    - **Security Group Rules (for both instances):**
        - Port 22 (SSH)
        - Port 80 (HTTP)
        - Port 3000 (frontend)
        - Port 5000 (backend)
        - Allow inbound traffic from each other (add their private IPs if using a private VPC network).
2. **On the Backend Instance**
    - `ssh -i aws-key.pem ubuntu@<backend-ec2-ip>`
    - `sudo apt update && sudo apt install python3-pip python3-venv git -y`
    - `git clone [https://github.com/](https://github.com/)<your-username>/<your-repo>.git`
    - `cd <repo>/backend`
    - `python3 -m venv venv`
    - `source venv/bin/activate`
    - `pip install -r requirements.txt`
    - `nohup python3 app.py &`
3. **On the Frontend Instance**
    - `ssh -i aws-key.pem ubuntu@<frontend-ec2-ip>`
    - `sudo apt update && sudo apt install nodejs npm git -y`
    - `git clone [https://github.com/](https://github.com/)<your-username>/<your-repo>.git`
    - `cd <repo>/frontend`
    - **Update API endpoint in frontend code:**
        - Change `axios.post('http://<backend-ec2-public-ip>:5000/api/add_shop', {...})` to point to the backend instance's public IP.
    - `npm install`
    - `nohup node server.js &`
    - **Access Frontend via Browser:**

```
http://<frontend-ec2-public-ip>:3000
```

Notes:

- **To see live ports:** `sudo lsof -i:3000`
- **To kill a process:** `sudo kill -9 <PID NUMBER>`
- Change inbound rules in security groups.

-----**Part 3: Docker + AWS ECS + ECR + VPC**

Objective: To deploy both backend and frontend as Docker containers, managed by ECS, utilizing images stored in ECR.

Steps:

1. **ECR (Elastic Container Registry)**
    - Navigate to the ECR section within the AWS console.
    - Create a new repository.
    - Access the repository and retrieve the push commands provided by the AWS console.
    - Import all necessary Docker images into the repository.
2. **ECS (Elastic Container Service)**
    - Define a new task.
    - Populate the task definition with image details, ensuring to include port mappings and environment variables as required.
    - Create a cluster, using the family name of your task definition.
    - Modify the default security groups to add the necessary ports.
    - Use the public IP address combined with the relevant port to preview your application.

OUTPUT: