Task 3 - Production Deployment of Flask & Express Using AWS ECR, ECS, and VPC



The goal of Task 3 is to deploy a Flask backend and an Express frontend in a production-grade environment using AWS cloud-native services:

Amazon ECR (Elastic Container Registry) for storing Docker images

Amazon ECS (Elastic Container Service) for container orchestration

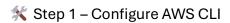
AWS VPC (Virtual Private Cloud) for secure networking

This approach represents a real-world, professional deployment pipeline used in production environments.

Backend (Flask) - runs on port 5000

Frontend (Express) - runs on port 3000

Communication: Frontend consumes backend API internally



Install AWS CLI and Docker if not already installed:

aws --version docker --version

Configure your AWS CLI:

aws configure

AWS Access Key ID: <your-access-key>

AWS Secret Access Key: <your-secret-key>

Default region name: ap-southeast-2

Default output format: json

Verification:

aws sts get-caller-identity

Step 2 – Create ECR Repositories

In AWS Console → ECR → Create repository:

flask-backend

express-frontend

Ensure both are private repositories.

Step 3 – Authenticate Docker with ECR

aws ecr get-login-password --region ap-southeast-2 | docker login --username AWS --password-stdin 215764923642.dkr.ecr.ap-southeast-2.amazonaws.com

Output:

Login Succeeded

Step 4 – Build and Tag Docker Images

Backend:

cd Backend docker build -t flask-backend . docker tag flask-backend:latest 215764923642.dkr.ecr.ap-southeast-2.amazonaws.com/flask-backend:latest

Frontend:

cd Frontend
docker build -t express-frontend .
docker tag express-frontend:latest 215764923642.dkr.ecr.ap-southeast2.amazonaws.com/express-frontend:latest

Step 5 – Push Images to ECR

docker push 215764923642.dkr.ecr.ap-southeast-2.amazonaws.com/flask-backend:latest docker push 215764923642.dkr.ecr.ap-southeast-2.amazonaws.com/express-frontend:latest

Verify uploaded images:

aws ecr describe-images --repository-name flask-backend --region ap-southeast-2 aws ecr describe-images --repository-name express-frontend --region ap-southeast-2

Step 6 – Create ECS Cluster

Go to ECS > Clusters > Create Cluster

Choose Fargate (Serverless) or EC2 launch type

Name: flask-express-cluster

Cluster created with 2 services and 2 running tasks

Step 7 – Create ECS Task Definitions

Create two task definitions:

Backend Task:

Image: 215764923642.dkr.ecr.ap-southeast-2.amazonaws.com/flask-backend:latest

Port: 5000

Frontend Task:

Image: 215764923642.dkr.ecr.ap-southeast-2.amazonaws.com/express-frontend:latest

Port: 3000

Step 8 - Create ECS Services & Networking

Launch both tasks as services inside the cluster

Create a VPC with at least 2 public subnets

Set up Security Groups:

Inbound: 5000 (backend), 3000 (frontend)

(Optional) Add an Application Load Balancer (ALB) for production scaling.

✓ Step 9 – Verify Deployment

Check running services:

curl http://<backend-public-ip>:5000

Response:

{"message": "Hello from Flask backend!"}

curl http://<frontend-public-ip>:3000

Response:

Express frontend is running 🚀

Using Flask backend: http://127.0.0.1:5000

Example verification:

Backend: http://54.252.159.32:5000 Frontend: http://16.176.232.29:3000

Final Verification in ECS Console

Cluster: flask-express-cluster

Tasks: 2 running

Status: Healthy

Logs: No errors

Network: VPC configured correctly

✓ Built and containerized Flask & Express apps

Pushed Docker images to Amazon ECR

Deployed containers via Amazon ECS

Secured them inside a VPC

✓ Verified production-level deployment with public endpoints