Deploying the EV Platform to AWS

Overview

The EV Platform is a Next.js frontend application that can be containerized using Docker (as it already includes a docker-compose.yml). We' can deploy it to AWS using a combination of services to ensure scalability, performance, and reliability:

- **EC2/ECS with Fargate**: To run the Docker containerized Next.js app.
- **S3 and CloudFront**: For hosting static assets and enabling a CDN for faster global access.
- **Route 53**: For domain management (optional).
- **Elastic Load Balancer (ELB)**: To distribute traffic to the ECS tasks.
- **IAM Roles**: To securely manage permissions for AWS services.

Alternatively, we can use **AWS Amplify** for a simpler deployment, which is ideal for Next.js applications but offers less control compared to ECS.

PRIMARY APPROACH (ECS WITH FARGATE FOR FRONTEND AND BACKEND):

Docker Images:

Build and push two Docker images to Amazon ECR: one for the frontend (ev-platform-frontend) and one for the backend (ev-platform-backend).

ECS Deployment:

Deploy the frontend and backend as separate services in an ECS cluster (ev-platform-cluster) using Fargate for serverless scaling.

Frontend Service: Runs the Next.js app, exposed on port 3001, with an Application Load Balancer (ALB) (ev-platform-frontend-alb) for public access.

Backend Service: Runs the API, exposed on port 5000 (or another port), with an internal ALB (ev-platform-backend-alb) for secure communication.

Networking:

Use a Virtual Private Cloud (VPC) with public and private subnets:

Frontend ALB in public subnets for public access.

Backend ALB and services in private subnets for security, accessible only by the frontend.

Configure security groups to allow traffic:

Frontend ALB: Allow HTTP/HTTPS (ports 80/443) from the internet.

Backend ALB: Allow traffic on port 5000 only from the frontend service.

Database (if applicable):

Use Amazon RDS (PostgreSQL) or DynamoDB for the backend database, deployed in private subnets.

Backend service connects to the database using environment variables (stored in AWS Secrets Manager for security).

Static Assets:

Host static assets (e.g., images in frontend/public) in an S3 bucket (ev-platform-static), served via CloudFront for CDN performance.

Domain and DNS:

Use Route 53 to configure a custom domain (e.g., www.evplatform.com for the frontend, api.evplatform.com for the backend).

Secure with HTTPS using AWS Certificate Manager (ACM) for SSL certificates.

Frontend-Backend Communication:

The frontend makes API calls to the backend using the backend ALB DNS or custom domain (e.g., https://api.evplatform.com).

Use environment variables in the frontend (via AWS Parameter Store) to configure the backend API URL.

Monitoring and Scaling:

Enable CloudWatch Logs for both services to monitor logs and metrics.

Set up auto-scaling for both the frontend and backend services based on CPU/memory usage (e.g., scale up if CPU > 70%).

Use CloudWatch Alarms to notify of issues (e.g., high error rates).

Alternative Approach (AWS Amplify for Frontend, ECS for Backend):

Frontend with Amplify:

Push the frontend code to GitHub and deploy it using AWS Amplify.

Amplify builds and hosts the Next.js app, providing automatic scaling, SSL, and a custom domain (e.g., www.evplatform.com).

Backend with ECS:

Deploy the backend API to ECS with Fargate, similar to the primary approach, using an internal ALB (ev-platform-backend-alb) in private subnets.

Expose the backend via a custom domain (e.g., api.evplatform.com).

Communication:

Configure the frontend in Amplify to call the backend API URL (https://api.evplatform.com) using environment variables.

Database:

Use RDS or DynamoDB for the backend, with secure access from the ECS service.

Benefits:

Amplify simplifies frontend deployment with built-in CI/CD and scaling.

ECS provides control over the backend for more complex API requirements.

Monitoring:

Use CloudWatch for the backend ECS service.

Amplify provides its own monitoring for the frontend.