🚀 Deploying a Frontend Application on AWS using ECS, ECR, and Fargate

This document explains **step by step**, in a **simple and clear way**, how to deploy a frontend application on **AWS ECS using Fargate**, with the container image stored in **Amazon ECR**.

The guide is written for beginners and focuses on **manual AWS Console steps**, which is the best way to understand the architecture before automation.

Architecture Overview

```
Local Machine
   ↓ (Docker build & push)
Amazon ECR (Image Registry)
   ↓
Amazon ECS (Fargate)
   ↓
Application Load Balancer
   ↓
Browser (Public URL)
```

🏠 Prerequisites

Before starting, make sure you have:

- An AWS account
- IAM user with permissions for:
- ECR
- ECS
- EC2 (for Load Balancer & Security Groups)
- Docker installed locally **or** on an EC2 instance
- A simple frontend app (HTML / React / Angular / Vue)

Step 1: Create an ECR Repository

1. Open **AWS Console → ECR**
2. Click **Create repository**
3. Choose:

4. Visibility: **Private**
5. Repository name: `frontend-app`
6. Click **Create repository**

📌 Copy the **repository URI**, you will need it later.

🐳 Step 2: Build and Push Docker Image to ECR

Authenticate Docker to ECR

```
aws ecr get-login-password --region us-east-1
| docker login --username AWS --password-stdin <ACCOUNT_ID>.dkr.ecr.us-
east-1.amazonaws.com
```

Build the Docker image

```
docker build -t frontend-app .
```

Tag the image

```
docker tag frontend-app:latest <ECR_URI>:latest
```

Push image to ECR

```
docker push <ECR_URI>:latest
```

🔗 Image is now stored in ECR.

Step 3: Create an ECS Cluster (Fargate)

1. Go to **ECS → Clusters**
2. Click **Create cluster**
3. Choose:
4. Cluster name: `frontend-cluster`
5. Infrastructure: **AWS Fargate**
6. Click **Create**

Step 4: Create Task Definition

1. ECS → **Task Definitions** → **Create new**
2. Choose:
3. Launch type: **Fargate**

Task configuration

- Task definition family: `frontend-task`
- CPU: `0.5 vCPU`
- Memory: `1 GB`
- Network mode: `awsvpc`
- Execution role: `ecsTaskExecutionRole`

Container configuration

- Container name: `frontend-container`
- Image URI: `<ECR_URI>:latest`
- Port mappings:
- Container port: `80`
- Protocol: TCP

🔗 Save the task definition.

Step 5: Create Application Load Balancer

1. Go to **EC2** → **Load Balancers**
2. Click **Create Load Balancer**
3. Choose **Application Load Balancer**
4. Configuration:
5. Scheme: Internet-facing
6. Listener: HTTP : 80
7. VPC: Default VPC

Target Group

- Target type: **IP**
- Protocol: HTTP
- Port: 80
- Health check path: `/`

🔗 Create Load Balancer and Target Group.

🚚 Step 6: Configure Security Groups

Load Balancer Security Group

- Inbound:
- HTTP (80) → `0.0.0.0/0`

ECS Task Security Group

- Inbound:
- HTTP (80) → **Load Balancer Security Group**

🚀 Step 7: Create ECS Service

1. ECS → Clusters → `frontend-cluster`
2. Click **Create service**
3. Configuration:
4. Launch type: **Fargate**
5. Task definition: `frontend-task`
6. Service name: `frontend-service`
7. Desired tasks: `1`

Networking

- VPC: Default
- Subnets: Public subnets
- Auto-assign public IP: **Enabled**
- Security group: ECS task SG

Load balancing

- Enable load balancing
- Type: Application Load Balancer
- Target group: previously created TG

🔗 Create service.

🛩️ Step 8: Verify Health Check

1. EC2 → Target Groups
2. Check targets
3. Status should be:

```
Healthy
```

If unhealthy: - Confirm app listens on port 80 - Confirm health check path `/`

🚀 Step 9: Access the Application

1. EC2 → Load Balancers
2. Copy **DNS name**
3. Open it in your browser

🎉 Frontend app is now live!

Cleanup (Avoid Cost)

To delete everything safely:

1. Delete ECS service
2. Delete ECS cluster
3. Delete Load Balancer & Target Group
4. Delete ECR repository (optional)

🔍 If cluster was created by CloudFormation, delete the **CloudFormation stack**, not the cluster directly.

Key Takeaways

- ECR stores container images
- ECS Fargate runs containers without managing servers
- Load Balancer exposes the app publicly
- Health checks are critical
- Empty ECS clusters are **free**

Next Steps (Optional)

- Automate deployment using **GitHub Actions**
- Add HTTPS using **ACM**
- Use **CloudFormation or Terraform** for IaC

🧱 End of documentation