



## Project: Deploy a Secure & Scalable Static Website on AWS with ECS Fargate & ECR (Terraform + Docker)

---

### 1. Objective

- Deploy a **static website** (HTML + Nginx container).
  - Make it **publicly accessible** via the internet.
  - Ensure **automatic scaling** based on CPU/Memory.
  - Manage everything using **Infrastructure as Code (IaC) with Terraform**.
  - Use **ECS Fargate** (serverless containers) and **ECR** (private image registry).
  - Use **Application Load Balancer (ALB)** for traffic routing.
- 

### 2. Architecture Overview

#### 1. Developer Workflow

- Build Docker image → Push to **Amazon ECR**.
- Terraform provisions AWS resources.

#### 2. AWS Components

- **ECR (Elastic Container Registry)**: Stores Docker images securely.
- **ECS (Elastic Container Service) Fargate**: Runs containers without managing servers.
- **Application Load Balancer (ALB)**: Provides internet-facing DNS & distributes traffic.
- **VPC + Subnets**: Secure networking for containers.
- **Security Groups**: Control inbound/outbound traffic.
- **Auto Scaling**: ECS scales tasks up/down based on CPU/Memory usage.

#### 3. End Result

- User opens browser → hits ALB DNS → ALB forwards request → ECS task (container) serves **static site**.
- 

### 3. Workflow

#### ◆ Step 1: Containerization

- A **Dockerfile** was created:
  - Based on nginx:alpine (lightweight web server).
  - Added index.html.template file.
  - Used an **entrypoint script** to dynamically inject the container's IP (SITE\_HOSTNAME) at runtime.

◆ **Step 2: Push Image to ECR**

- docker build → built the static-site image.
- docker tag & docker push → pushed to **ECR repo** (127898337602.dkr.ecr.us-west-2.amazonaws.com/static-site).

◆ **Step 3: Infrastructure Provisioning (Terraform)**

- **ecr.tf** → Creates ECR repo (force\_delete = true for cleanup).
- **vpc.tf** → Creates VPC, subnets, internet gateway, and route tables.
- **alb.tf / service-and-alb.tf** → Creates ALB, Target Group (target\_type = ip for Fargate), and Listener.
- **ecs.tf** → Defines ECS cluster, task definition, and service using the ECR image.
- **autoscaling.tf** → Adds scaling policy (CPU utilization target).
- **security-groups.tf** → Allows **TCP:80** inbound from anywhere for ALB, and secure rules for ECS tasks.

◆ **Step 4: Access the Website**

- Terraform outputs the **ALB DNS name**.
- Example:
- <http://static-site-alb-xxxxxxxxx.us-west-2.elb.amazonaws.com>
- Website loads → Displays welcome message with container IP.

#### **4. Scalability & High Availability**

- ECS service runs tasks in **multiple subnets** across AZs → high availability.
- **Auto Scaling Policies:**
  - If CPU > 60% → launch more tasks.
  - If CPU < 30% → scale down tasks.
- Load Balancer distributes traffic evenly.

---

## 5. Security

- Containers run in **private subnets**; only ALB is internet-facing.
  - **Security Groups:**
    - ALB SG → Allows inbound HTTP (TCP:80) from anywhere.
    - ECS Task SG → Allows traffic **only from ALB SG**.
  - ECR images are private → requires IAM authentication.
- 

## 6. Benefits of this Setup

- Serverless** — No EC2 management (thanks to Fargate).
  - Secure** — Private networking, restricted SGs, private container images.
  - Scalable** — Auto scaling ensures cost-efficiency and performance.
  - Automated** — Entire infra built/destroyed using Terraform.
  - Portable** — Same setup works in dev, staging, and prod environments.
-