**Ruby on Rails Application - End-to-End Deployment Documentation (Docker, ECR, ECS, Terraform)**

**Objective**

Dockerize a Ruby on Rails application and deploy it using AWS ECS (Fargate) behind an Application Load Balancer (ALB), with PostgreSQL hosted on Amazon RDS, and infrastructure managed by Terraform.

**EC2-Based Development (Docker Compose on Cloud Ubuntu)**

*This Docker Compose-based setup was executed on a* ***cloud-hosted Ubuntu EC2 instance****, simulating local development but enabling external access and testing.*

**Services Setup**

* **Rails App Container** (Port: 3000)
* **PostgreSQL Container** (Port: 5432)
* **NGINX Container** (Port: 80)

**Issues Resolved**

* Corrected PostgreSQL hostname (from db to postgres) to align with container name
* Updated .env variables for correct DB hostname and port
* Fixed NGINX reverse proxy configuration
* Allowed port 80 in EC2 security group for HTTP access

**Configuration Snippets**

**docker-compose.yml**:

services:

rails\_app:

build:

context: .

dockerfile: docker/app/Dockerfile

ports:

- "3000:3000"

env\_file: rails\_app.env

webserver:

build:

context: .

dockerfile: docker/nginx/Dockerfile

ports:

- "80:80"

postgres:

image: postgres:13.3

ports:

- "5432:5432"

**rails\_app.env**:

RDS\_DB\_NAME=rails\_test\_db

RDS\_USERNAME=postgres

RDS\_PASSWORD=password

RDS\_HOSTNAME=postgres

RDS\_PORT=5432

**Nginx default.conf**:

upstream rails\_app {

server rails\_app:3000;

}

**Verified With:**

sudo docker-compose up --build -d

sudo docker exec -it rails\_app bash

curl http://localhost:3000

**Docker Images → Pushed to ECR**

**Dockerized Two Services:**

* **Rails App Image**
* **NGINX Reverse Proxy Image**

**Steps:**

1. **Build Images**:

# Rails App

sudo docker build -t rails-app-v1 -f docker/app/Dockerfile .

# NGINX

sudo docker build -t nginx-proxy-v1 -f docker/nginx/Dockerfile .

1. **Tag Images**:

# Rails App

sudo docker tag rails-app-v1:latest <account>.dkr.ecr.us-east-1.amazonaws.com/ruby\_rails\_project\_ecr:v1

# NGINX

sudo docker tag nginx-proxy-v1:latest <account>.dkr.ecr.us-east-1.amazonaws.com/ruby\_rails\_project\_nginx:v1

1. **Authenticate with ECR**:

aws ecr get-login-password --region us-east-1 | \

docker login --username AWS --password-stdin <account>.dkr.ecr.us-east-1.amazonaws.com

**Note**: AWS account ID is masked for security purposes.

1. **Push to ECR**:

# Rails App

docker push <account>.dkr.ecr.us-east-1.amazonaws.com/ruby\_rails\_project\_ecr:v1

# NGINX

docker push <account>.dkr.ecr.us-east-1.amazonaws.com/ruby\_rails\_project\_nginx:v1

**AWS ECS Fargate + ALB Deployment**

**Problem Encountered**

nginx: [emerg] host not found in upstream "rails\_app:3000"

**Root Cause & Fix**

In ECS Fargate, containers run under awsvpc networking mode and share the same ENI (Elastic Network Interface). However, unlike Docker Compose, containers **do not resolve each other by name**.

Therefore, we updated the NGINX configuration:

upstream rails\_app {

server 127.0.0.1:3000;

}

This allowed NGINX to talk to the Rails app via localhost, which works because both containers are in the **same task** and share the network namespace.

**ECS Task Definition:**

* 2 Containers: NGINX, Rails App
* Network Mode: awsvpc
* CPU: 0.5 vCPU, Memory: 1GB

**ECS Concepts**

* **ECS Cluster**: Logical grouping of ECS resources
* **Task Definition**: Blueprint of containers (Rails + NGINX), memory, CPU, and networking config
* **Service**: Maintains desired task count, handles failover and restarts

**Load Balancing**

* **ALB (Application Load Balancer)** created in **public subnet**
* **Target Group** registered with ECS service
* ALB forwards port 80 → ECS task → NGINX container
* NGINX forwards internally to **Rails app on 127.0.0.1:3000**

**Terraform Infrastructure**

**Modules Created:**

1. **VPC & Networking**
   * VPC with CIDR: 10.0.0.0/16
   * 2 Public Subnets → ALB & NAT Gateway
   * 2 Private Subnets → ECS Tasks & RDS
   * Internet Gateway for public traffic
   * NAT Gateway for private ECS outbound access
2. **Security Groups**
   * ALB SG: Allows HTTP (port 80) from internet
   * ECS SG: Allows traffic only from ALB SG
   * RDS SG: Allows traffic only from ECS SG (port 5432)
3. **IAM Roles**
   * ECS Task Execution Role → Pull image from ECR, log to CloudWatch
   * ECS App Task Role → Access S3 securely (no access keys)
4. **ECS Resources**
   * ECS Cluster
   * Task Definition (2 containers: NGINX + Rails)
   * ECS Service (attached to target group)
5. **ALB Setup**
   * ALB with listener (port 80)
   * Target Group with ECS tasks as targets
6. **RDS (PostgreSQL)**
   * Hosted in private subnets
   * Not publicly accessible
7. **S3 Access**
   * Rails app uploads/downloads via IAM Role

**Final Outcome**

* Rails app & NGINX containerized and pushed to ECR
* ECS Fargate deployed via Terraform using secure networking
* ALB → ECS → NGINX → Rails → RDS flow working
* Infrastructure is modular and production-ready
* S3 used via IAM role (no static credentials)