Complete Deployment Guide: Creating ECS with Terraform and Deploying ECS Application

This guide includes step-by-step instructions for creating an Amazon ECR repository using Terraform, pushing Docker images via GitHub Actions, and deploying the application on AWS ECS.

Step 1: Prerequisites

1.1 AWS Account Setup

- **AWS Console Access**: Log in to the AWS Management Console.
- **IAM User**: Ensure your IAM user has permissions to manage ECS, ECR, RDS, and VPC resources.
    - Recommended IAM Policy: AdministratorAccess for initial setup.

1.2 Install Required Tools

Ensure the following tools are installed:

- **AWS CLI**: Install AWS CLI
- **Terraform**: Install Terraform
- **Docker**: Install Docker
- **Git**: Install Git

1.3 AWS CLI Configuration

Run the following command to configure your AWS CLI:

aws configure

Provide your **Access Key ID**, **Secret Access Key**, **Default Region** (e.g., us-east-1), and output format (json).

Step 2: Clone the Repository

1. Clone your fork:

git clone https://github.com/obiskomike/ipfs-metadata.git

cd ipfs-metadata

Step 2: Configure the Database

This application requires a PostgreSQL database to store data. You can choose between two options:

Option 1: Using AWS RDS (Recommended for Production)

AWS RDS (Relational Database Service) provides a managed PostgreSQL database.

- **Create an RDS Database Instance**:
  - Log in to the AWS Management Console.
  - Go to **RDS** and click **Create database**.
  - Choose **PostgreSQL** as the engine, and select the **Free Tier** option for testing.
  - Configure your database (e.g., name, username, password).
  - Once the instance is created, note the **Endpoint**, **Port**, **Username**, and **Password** for your database.
- **Configure the .env File**: In your project directory, create a .env file and add your database connection details:

DB_HOST=<RDS-endpoint>
DB_PORT=5432
DB_USER=<your-db-username>
DB_PASSWORD=<your-db-password>
DB_NAME=<your-db-name>

Option 2: Using Docker (for Testing Locally)

1. You can run PostgreSQL in a Docker container for local testing. Add the following command to run PostgreSQL:
   docker compose up postgres –d
2. Update your .env file to point to localhost for testing.

Step 3: Containerize the Application

3.1 Build and Test the Docker Image

Build the Docker image:

docker compose build

Run the Docker container locally:

docker compose up -d
Access
http://localhost:8080/metadata

Step 3: Terraform

3.1 Initialize and Apply Terraform

1. Initialize Terraform:

cd terraform
terraform init

2. Apply the configuration:

terraform apply

3. Confirm the changes by typing yes.
4. Terraform will create the all resources.

3.2 Verify Infrastructure

Terraform will create the following resources:

- **VPC**: Virtual Private Cloud for networking.
- **Subnets**: Public and private subnets for isolation.
- **ECS Cluster**: To host your application.
- **SECURITY GROUP:** For security
- **ALB (Application Load Balancer)**: For routing traffic.
- **IAM Roles**: For ECS tasks.

Step 4: Set Up CI/CD Pipeline with GitHub Actions

4.1 Create GitHub Secrets

1. Go to your GitHub repository and navigate to **Settings** > **Secrets and Variables** > **Actions**.
2. Add the following secrets:
    a. AWS_ACCESS_KEY_ID: The access key for your AWS IAM user.
    b. AWS_SECRET_ACCESS_KEY: The secret key for your AWS IAM user.
    c. AWS_REGION: The AWS region (e.g., us-east-1).
    d. ECR_REPOSITORY: Your ECR repository name (e.g., ipfs-metadata).

4.2 Create a GitHub Actions Workflow

1. Create a .github/workflows/aws.yml file in your repository with the following content:

name: Build and Push to Amazon ECR

```yaml
on:
  push:
    branches: [ "main" ]

env:
  AWS_REGION: us-east-1              # Replace with your AWS region
  ECR_REPOSITORY: ipfs-metadata

permissions:
  contents: read

jobs:
  build-and-push:
    name: Build and Push Docker Image
    runs-on: ubuntu-latest

    steps:
    - name: Checkout Code
      uses: actions/checkout@v4

    - name: Configure AWS Credentials
      uses: aws-actions/configure-aws-credentials@v1
      with:
        aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
        aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
        aws-region: ${{ env.AWS_REGION }}

    - name: Login to Amazon ECR
      id: login-ecr
      uses: aws-actions/amazon-ecr-login@v1

    - name: Set up Docker Buildx
      uses: docker/setup-buildx-action@v2

    - name: Build, Tag, and Push Docker Image using Buildx
      env:
        ECR_REGISTRY: ${{ steps.login-ecr.outputs.registry }}
        IMAGE_TAG: ${{ github.sha }}
```

```
run: |
  # Set up buildx builder if needed
  docker buildx create --use

  # Build and push multi-platform image (if needed)
  docker buildx build \
    --platform linux/amd64,linux/arm64 \
    -t $ECR_REGISTRY/$ECR_REPOSITORY:latest \
    -t $ECR_REGISTRY/$ECR_REPOSITORY:$IMAGE_TAG \
    --push .

  echo "Image URL: $ECR_REGISTRY/$ECR_REPOSITORY:latest"
```

4.3 Trigger the Workflow

- When you push changes to the main branch, GitHub Actions will automatically build the Docker image and push it to your container registry.

Step 5: Monitor and Access the Application

Once the infrastructure is provisioned and the container is running, the application will be accessible via the load balancer URL provided by Terraform.

Access the Application:

- Obtain the load balancer URL from the Terraform output or AWS Console.
- Visit the URL in a web browser to ensure the application is running.