Barista Cafe - Dockerized Static Website Deployment to AWS EKS

Overview

This repository contains a static website (Barista Cafe) that is served using **Nginx** within a Docker container. The project is configured to be built and deployed to **AWS Elastic Kubernetes Service (EKS)** using **GitHub Actions** and **AWS Elastic Container Registry (ECR)**.

Features

- **Dockerized Static Website**: Uses Nginx to serve static content.
- **AWS ECR for Image Storage**: Stores the Docker image in AWS Elastic Container Registry.
- **AWS EKS for Deployment**: Runs the website on Kubernetes.
- **GitHub Actions for CI/CD**: Automates the build and deployment process.

Folder Structure

```
2137_BARISTA_CAFE
├── css
├── fonts
├── images
├── js
├── videos
├── index.html
├── reservation.html
├── Dockerfile
└── ABOUT THIS TEMPLATE.txt
```

How the Docker Image is Built & Deployed

Step 1: Clone the Repository

```
git clone https://github.com/your-repository.git
cd your-repository
```

Step 2: Build the Docker Image

To create a Docker image from the `Dockerfile`, run the following command:

```
docker build -t barista_cafe .
```

Step 3: Run the Docker Container Locally (Optional)

```
docker run -d -p 80:80 --name barista_cafe barista_cafe
```

Access the website at `http://localhost/`.

Step 4: Push the Image to AWS ECR

1. Authenticate Docker with AWS ECR

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

2. Create an ECR Repository (if not already created)

```
aws ecr create-repository --repository-name barista-cafe
```

3. Tag and Push the Image to ECR

```
ACCOUNT_ID=$(aws sts get-caller-identity --query "Account" --output text)
IMAGE_URI="$ACCOUNT_ID.dkr.ecr.us-east-1.amazonaws.com/barista-cafe:latest"
docker tag barista_cafe $IMAGE_URI
docker push $IMAGE_URI
```

Step 5: Deploy to AWS EKS

1. Update kubeconfig to access EKS Cluster

```
aws eks update-kubeconfig --region us-east-1 --name my-eks-cluster
```

2. Deploy to Kubernetes

```
kubectl set image deployment/nginx-deployment nginx=$IMAGE_URI -n default
kubectl rollout status deployment/nginx-deployment -n default
```

Automated Deployment with GitHub Actions

GitHub Secrets Required

Go to `Settings > Secrets and Variables > Actions` and add:

- `AWS_ACCESS_KEY_ID`
- `AWS_SECRET_ACCESS_KEY`
- `AWS_REGION` (e.g., `us-east-1`)
- `ECR_REPOSITORY` (e.g., `barista-cafe`)

GitHub Actions Workflow

Whenever code is pushed to the `main` branch, GitHub Actions:

1. **Builds the Docker image**
2. **Pushes it to ECR**
3. **Deploys it to EKS**

Accessing the Deployed Website

Once deployed, you can access the website using the external URL of the **LoadBalancer** service:

```
kubectl get svc -n default
```

Copy the `EXTERNAL-IP` and open it in your browser.

Cleanup

To delete the deployment from EKS:

```
kubectl delete deployment nginx-deployment -n default
```

To delete the ECR repository:

```
aws ecr delete-repository --repository-name barista-cafe --force
```

To delete the EKS cluster:

```
eksctl delete cluster --name my-eks-cluster
```

Conclusion

This setup automates the deployment of a static website using **Docker, AWS ECR, EKS, and GitHub Actions**. The process ensures seamless CI/CD while hosting the website on a scalable Kubernetes environment.

🚀 **Happy Deploying!**