EKS Microservices Deployment with CI/CD

Overview

In this project, I set up a microservices architecture on Amazon EKS (Elastic Kubernetes Service) with a full CI/CD pipeline using Jenkins, Docker, and AWS services.

Key Components

- EKS: For orchestrating containerized microservices
- ECR: To store Docker images
- Jenkins: For CI/CD pipeline
- CodeCommit: For source control
- CloudWatch: For monitoring and logging

Implementation Highlights

Terraform for EKS Cluster

hcl
Copy

```hcl
module "eks" {
  source          = "terraform-aws-modules/eks/aws"
  cluster_name    = "my-eks-cluster"
  cluster_version = "1.21"
  subnets         = module.vpc.private_subnets

  node_groups = {
    eks_nodes = {
      desired_capacity = 2
      max_capacity     = 3
      min_capacity     = 1

      instance_type = "t3.medium"
    }
  }
}
```

Jenkinsfile for CI/CD Pipeline

groovy
Copy

```groovy
pipeline {
    agent any
    environment {
        AWS_ACCOUNT_ID="123456789012"
        AWS_DEFAULT_REGION="us-west-2"
        IMAGE_REPO_NAME="my-microservice"
        IMAGE_TAG="${BUILD_NUMBER}"
        REPOSITORY_URI =
"${AWS_ACCOUNT_ID}.dkr.ecr.${AWS_DEFAULT_REGION}.amazonaws.com/${IMAGE_REPO_NAME}"
    }

    stages {

        stage('Cloning Git') {
            steps {
                checkout([$class: 'CodeCommitSCM', credentialsId: 'aws-credentials', repoName: 'my-microservice-repo',
branchName: 'master'])
            }
        }

        stage('Building image') {
            steps{
                script {
                    dockerImage = docker.build "${IMAGE_REPO_NAME}:${IMAGE_TAG}"
                }
            }
        }

        stage('Pushing to ECR') {
            steps{
                script {
                    sh "aws ecr get-login-password --region ${AWS_DEFAULT_REGION} | docker login --username AWS --
password-stdin ${AWS_ACCOUNT_ID}.dkr.ecr.${AWS_DEFAULT_REGION}.amazonaws.com"
                    sh "docker push ${REPOSITORY_URI}:${IMAGE_TAG}"
                }
            }
        }

        stage('Deploy to EKS') {
            steps{
                script {
                    sh "aws eks get-token --cluster-name my-eks-cluster | kubectl apply -f k8s-deployment.yml"
                }
            }
        }
    }
}
```

Kubernetes Deployment YAML

yaml
Copy

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-microservice
spec:
  replicas: 3
```

```yaml
selector:
  matchLabels:
    app: my-microservice
template:
  metadata:
    labels:
      app: my-microservice
  spec:
    containers:
    - name: my-microservice
      image: ${REPOSITORY_URI}:${IMAGE_TAG}
      ports:
      - containerPort: 8080
```

Challenges and Solutions

- EKS cluster scaling: Implemented cluster autoscaler for dynamic node provisioning.
- Secret management: Used AWS Secrets Manager and Kubernetes secrets for secure configuration.

Outcome

This project showcased my skills in container orchestration, CI/CD implementation, and working with managed Kubernetes services on AWS.