End-to-End CI/CD Pipeline using Jenkins, Docker & Amazon EKS

📌Project Overview

This project demonstrates a **complete CI/CD workflow** where application code changes pushed to GitHub automatically trigger a Jenkins pipeline that: - Builds a Docker image - Pushes the image to Docker Hub - Performs a rolling update on an Amazon EKS cluster

The infrastructure includes **Amazon EKS**, **EC2 (Jenkins server)**, **Docker**, **Kubernetes**, **GitHub Webhooks**, and **AWS IAM**.

🧱Architecture

📐CI/CD Architecture Diagram

📌**Diagram file:** `docs/architecture.png`

CI/CD Architecture Diagram

Flow Explanation: - Developer pushes code to GitHub - GitHub Webhook triggers Jenkins pipeline - Jenkins (running on EC2 with minimum 4 GiB RAM) builds Docker image - Image is pushed to Docker Hub - Jenkins updates Kubernetes deployment in Amazon EKS using kubectl - Kubernetes Service exposes application via AWS Load Balancer - End users access the application through the Load Balancer DNS

```
Developer
   |
   v
GitHub Repository
   |
   v
GitHub Webhook
   |
   v
Jenkins (EC2, 4 GiB RAM)
   |
   | \--> Docker Build & Push
   |       |
   |       v
   |   Docker Hub
   |       |
   |       v
   |   Amazon EKS Cluster
   |        |   |   |
```

```
   |    Frontend Backend Database
   |                |
   v                v
kubectl        Kubernetes Service
                    |
                    v
             AWS Load Balancer
                    |
                    v
                End Users
```

```
**Explanation:**
- Developers push code to GitHub
- Webhook triggers Jenkins automatically
- Jenkins builds Docker image and pushes to Docker Hub
- Jenkins updates EKS deployment using kubectl
- EKS exposes application using AWS Load Balancer

---

## 🛠️ Technologies Used
- AWS EKS
- AWS EC2 (Amazon Linux 2023)
- IAM Roles & Policies
- Jenkins
- Docker
- Kubernetes (kubectl)
- Git & GitHub Webhooks

---

## ✂️ EKS Cluster Setup

### IAM Roles
#### EKS Cluster Role
- **Role Name:** eks-cluster
- **Policy Attached:**
  - AmazonEKSClusterPolicy

#### EKS Node Role
- **Role Name:** EKS-Node-Role
- **Policies Attached:**
  - AmazonEC2ContainerRegistryReadOnly
  - AmazonEKS_CNI_Policy
  - AmazonEKSWorkerNodePolicy

### Cluster Configuration
- VPC: Default
- Subnets: Default
- Add-ons:
```

```
  - kube-proxy
  - CoreDNS
  - Amazon VPC CNI
  - Node Monitoring Agent
  - Amazon EKS Pod Identity Agent

### Node Group
- Desired size: 1
- Min size: 1
- Max size: 1
- IAM Role: EKS-Node-Role

---

## ✒️ Jenkins Server (EC2)

### Instance Details
- AMI: Amazon Linux 2023 (Kernel 6.1)
- Instance Type: **t2.medium (Minimum 4 GiB RAM required for Jenkins)**
- vCPU: 2
- Memory: 4 GiB
- Storage: 25 GiB gp3
- Security Group:
  - Port 8080 (Jenkins)
  - Port 443 (EKS API)

---

## 🛏️ Jenkins Installation

> 💡 **Tip:** All command blocks below automatically show a **Copy** button on GitHub for easy execution.

```bash
sudo dnf install java-17-amazon-corretto -y
sudo wget -O /etc/yum.repos.d/jenkins.repo https://pkg.jenkins.io/redhat-stable/jenkins.repo
sudo rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.io-2023.key
sudo yum install jenkins -y
sudo systemctl start jenkins
sudo systemctl enable jenkins
```
```

```
sudo dnf install java-17-amazon-corretto -y
sudo wget -O /etc/yum.repos.d/jenkins.repo https://pkg.jenkins.io/redhat-stable/jenkins.repo
sudo rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.io-2023.key
sudo yum install jenkins -y
sudo systemctl start jenkins
sudo systemctl enable jenkins
```

Access Jenkins:

```
http://<EC2-PUBLIC-IP>:8080
```

Retrieve Admin Password:

```
sudo cat /var/lib/jenkins/secrets/initialAdminPassword
```

🛠️ Install Required Tools

Git

```
sudo yum install git -y
```

Docker

```
sudo yum install docker -y
sudo systemctl start docker
sudo systemctl enable docker
```

kubectl

```
curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/
stable.txt)/bin/linux/amd64/kubectl"
chmod +x ./kubectl
sudo mv ./kubectl /usr/local/bin/kubectl
kubectl version --client
```

🌡️ AWS CLI Configuration

```
aws configure
```

- Access Key ID: ** - Secret Access Key: ** - Region: ap-south-1 - Output format: default

> 🔒 **Security Note:** Access keys are sensitive credentials and must never be exposed in GitHub, screenshots, or code. Always mask them using `****` or environment variables.

🎒Connect Jenkins to EKS

```
aws eks --region ap-south-1 update-kubeconfig --name cluster1
kubectl get nodes
```

Jenkins Access to kubeconfig & AWS

```
sudo mkdir -p /var/lib/jenkins/.kube
sudo cp -R /home/ec2-user/.kube/config /var/lib/jenkins/.kube/config
sudo chown -R jenkins:jenkins /var/lib/jenkins/.kube

sudo cp -R /home/ec2-user/.aws /var/lib/jenkins/.aws
sudo chown -R jenkins:jenkins /var/lib/jenkins/.aws

sudo usermod -aG docker jenkins
sudo systemctl restart jenkins
```

🎓Kubernetes Deployment

Repository:

```
https://github.com/DevRahul16/999-proj1.git
```

```
git clone https://github.com/DevRahul16/999-proj1.git
cd 999-proj1/k8
kubectl apply -f fe.yaml
kubectl apply -f be.yaml
kubectl apply -f db.yaml
```

Check Services:

```
kubectl get svc
```

Access application using Load Balancer DNS (HTTP).

🎩Jenkins Pipeline

Required Plugins

- Docker Pipeline
- Kubernetes CLI

DockerHub Credentials

- ID: dockerhub
- Username: devrahul16

Jenkinsfile

```
pipeline {
    agent any
    environment {
        DOCKERHUB_CREDENTIALS = 'dockerhub'
        DOCKER_IMAGE = 'devrahul16/fe'
        KUBERNETES_DEPLOYMENT = 'frontend'
    }
    stages {
        stage('Clone Repository') {
            steps {
                git branch: 'master', url: 'https://github.com/devrahul16/
fe1.git'
            }
        }
        stage('Build Docker Image') {
            steps {
                script {
                    env.TAG = "${env.BUILD_NUMBER}"
                    sh "docker build -t $DOCKER_IMAGE:$TAG ."
                }
            }
        }
        stage('Push Docker Image') {
            steps {
                withDockerRegistry([credentialsId: 'dockerhub', url:
'https://index.docker.io/v1/']) {
                    sh "docker push $DOCKER_IMAGE:$TAG"
                }
            }
        }
        stage('Rolling Update Kubernetes Deployment') {
            steps {
                sh "kubectl set image deployment/$KUBERNETES_DEPLOYMENT
frontend=$DOCKER_IMAGE:$TAG"
            }
        }
    }
}
```

🎽 GitHub Webhook Configuration

- Payload URL:

```
http://<JENKINS-IP>:8080/github-webhook/
```

- Content Type: application/json
- SSL Verification: Disabled
- Trigger Event: Push

Enable **GitHub hook trigger for GITScm polling** in Jenkins job.

🔗 Final Outcome

- Code push to GitHub automatically triggers Jenkins
- Docker image is built & pushed
- Kubernetes deployment is updated with zero downtime

📏 Screenshots to Add

Place screenshots inside the `docs/` folder: - `iam-eks-role.png` - `eks-cluster.png` - `node-group.png` - `jenkins-dashboard.png` - `pipeline-success.png` - `kubectl-get-svc.png` - `architecture.png`

🗜️ Improvements (Optional)

- Use IAM Role for Service Account (IRSA)
- Replace root access keys with IAM user / role-based access
- Use HTTPS with ACM + ALB Ingress Controller
- Store secrets in AWS Secrets Manager or Kubernetes Secrets
- Use Jenkinsfile from SCM instead of inline pipeline
- Add monitoring using Prometheus & Grafana

📁 Recommended Repository Structure

```
EKS-Jenkins-CICD/
├── docs/
│   ├── architecture.png
│   ├── eks-cluster.png
│   ├── jenkins-dashboard.png
│   └── pipeline-success.png
├── k8s/
│   ├── fe.yaml
```

```
│   ├── be.yaml
│   └── db.yaml
├── jenkins/
│   └── Jenkinsfile
└── README.md
```

Author: Rahul Kumar
GitHub: https://github.com/DevRahul16