**Assignment:**

Create a CICD pipeline in GitHub Actions to fetch Java code from GitHub repository, containerize using Docker, push to ECR and deploy to EKS so that users can access the application viz URL.

---------------------------------------------------------------------------------------------------------------------------------------

**Technology Used:**

Docker, ECR, EKS, GitHub Actions

---------------------------------------------------------------------------------------------------------------------------------------

**Folder structure:**

**|\_** src

**|\_** .github

**|\_**workflows

**|\_** deploy.yml

**|\_** Dockerfile

**|\_** pom.xml

|\_ k8s/

**|\_** deployment.yaml

**|\_** service.yaml

---------------------------------------------------------------------------------------------------------------------------------------

**Java Project**

* Install JDK & verify

$ java -version

* Install Apache Maven, Add path in environment variable (Environment variable > Path > C:\Program Files\apache-maven-3.9.9\bin) & verify

$ mvn -version

* Install Intel J
* Spring Initializr

Browse – <https://start.spring.io/>

Project – Maven

Language – Java

Spring Boot – 3.5.3

Group – com.example

Artifact – java-hello-world

Name – java-hello-world

Description – Demo project for Spring Boot

Package name – com.example.java-hello-world

Packaging – Jar

Java – 17

Dependencies – Spring Web

Generate

* Extract downloaded zip file and drag into Intel J
* Verify project java version in Intel J

Settings > Project structure > Project > SDK – 17 > Apply > OK

* Navigate into source folder "src/main/java/com.example.java\_hello\_world".
* JavaHelloWorldApplication.java

package com.example.java\_hello\_world;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class JavaHelloWorldApplication {

public static void main(String[] args) {

SpringApplication.run(JavaHelloWorldApplication.class, args);

}

}

* HomeController.java

package com.example.java\_hello\_world;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.GetMapping;

@Controller

public class HomeController {

@GetMapping("/")

public String display() {

return "index";

}

}

* src/main/resources/templates/index.html

<!DOCTYPE html>

<html xmlns:th="http://www.thymeleaf.org">

<head>

<title>Homepage</title>

<link rel="stylesheet" href="/style.css">

</head>

<body>

<div class="container">

<h1>Hello World</h1>

<h2>LTIMindtree</h2>

<h3>Shivam Thakur</h3>

</div>

</body>

</html>

* src/main/resources/static/style.css

body {

font-family: Arial, sans-serif;

background-color: #f0f8ff;

text-align: center;

padding-top: 50px;

}

.container h1 {

color: #2e8b57;

font-size: 3em;

}

.container h2 {

color: #4682b4;

font-size: 2.5em;

}

.container h3 {

color: #8a2be2;

font-size: 2em;

}

* pom.xml

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-thymeleaf</artifactId>

</dependency>

* Run below commands

$ mvn clean package

$ mvn spring-boot:run or java -jar target/<jar-file-name>.jar

* Verify

Browse – http://localhost:8080/ > "It will show Hello World

---------------------------------------------------------------------------------------------------------------------------------------

**Setup**

* Create new repository in GitHub.
* Git Configuration

1. Configure Git user info in local

$ git config --global user.name "shivam"

$ git config --global user.email [shivamthakur0567@gmail.com](mailto:shivamthakur0567@gmail.com)

1. Initialize Git Repository

$ git init

1. Add files to Git:

$ git add .

1. Commit changes:

$ git commit -m "Initial commit"

1. Check current local branch

$ git branch

Master

1. Rename local branch to main

$ git branch -m main

1. Add our remote repository

$ git remote add origin https://github.com/shivam-th/java-hello-world.git

1. Push the code from local to remote repository

$ git push -u origin main --force

* AWS Configuration

$ aws configure

AWS Access Key ID [None]:

AWS Secret Access Key [None]:

Default region name [None]:

Default output format [None]:

* Create OIDC Identity Provider

IAM > Identity providers > Add provider

Provider type – OpenID Connect

Provider URL – <https://token.actions.githubusercontent.com>

Audience – sts.amazonaws.com

* Create IAM Role for GitHub OIDC

IAM > Roles > Create role

Trusted entity type – Web identity > Identity provider – <https://token.actions.githubusercontent.com> > Audience – sts.amazonaws.com

GitHub organization – shivam-th

Permission Policies –

AmazonEC2ContainerRegistryFullAccess (for pushing/pulling images from ECR)

AmazonEKSWorkerNodePolicy (for EKS node permissions)

AmazonEKS\_CNI\_Policy (for networking in EKS.

AmazonEKSServicePolicy (for managing EKS services)

Role name – GitHubActionsOIDCRole

* Create ECR

$ aws ecr create-repository \

    --repository-name my-java-app \

    --region us-east-1

* Install EKS

Browse – <https://eksctl.io/installation/> > download required zip file

Create "eksctl" folder in C drive, extract the downloaded zip file and place .exe file in a "C:\eksctl".

Add folder path to Environment Variable

Edit Environment Variables > Path

Verify

$ eksctl version

* Create EKS cluster

$ eksctl create cluster \

--name my-cluster \

--region us-east-1 \

--nodegroup-name linux-nodes \

--node-type t3.medium \

--nodes 2 \

--nodes-min 1 \

--nodes-max 3 \

--managed

* Allow GitHub Actions to authenticate with AWS EKS using OIDC

EKS > mycluster > CloudShell > $ kubectl edit configmap aws-auth -n kube-system

- rolearn: arn:aws:iam::471112745159:role/GitHubActionsOIDCRole

      username: github-actions

      groups:

        - system:masters

---------------------------------------------------------------------------------------------------------------------------------------

**Containerization**

* Dockerfile

# Use maven image to build the application

FROM maven:3.8.5-openjdk-17 AS builder

# Set the working directory

WORKDIR /app

# Copy the source code

COPY . .

# Build the application

RUN mvn clean package

##################################################

# Use a JDK image to run the application

FROM openjdk:17-jdk-slim

# Set the working directory

WORKDIR /app

# copy build file from stage1

COPY --from=builder /app/target/\*.jar app.jar

# Expose the port

EXPOSE 8080

# Run the application

ENTRYPOINT ["java", "-jar", "app.jar"]

**Kubernetes Manifests**

* k8s/deployment.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

name: java-app-deployment

spec:

replicas: 2

selector:

matchLabels:

app: java-app

template:

metadata:

labels:

app: java-app

spec:

containers:

- name: java-app

image: <ECR\_IMAGE\_URI>

ports:

- containerPort: 8080

* k8s/service.yaml

apiVersion: v1

kind: Service

metadata:

name: java-app-service

spec:

type: LoadBalancer

selector:

app: java-app

ports:

- port: 80

targetPort: 8080

---------------------------------------------------------------------------------------------------------------------------------------

**GitHub Action Workflow**

* .github/workflows/ci-cd.yml

name: CI/CD Pipeline

# Trigger the workflow on push to the main branch

on:

push:

branches: [main]

# Define environment variables

env:

AWS\_REGION: us-east-1

ECR\_REGISTRY: 471112745159.dkr.ecr.us-east-1.amazonaws.com

ECR\_REPOSITORY: my-java-app

CLUSTER\_NAME: my-cluster

IMAGE\_TAG: ${{ github.sha }}

# for OIDC

permissions:

id-token: write

contents: read

jobs:

deploy:

runs-on: ubuntu-latest

steps:

# Checkout the source code from the repository

- name: Checkout code

uses: actions/checkout@v4

# Configure AWS credentials using GitHub Secrets

- name: Configure AWS credentials using OIDC

uses: aws-actions/configure-aws-credentials@v2

with:

role-to-assume: arn:aws:iam::471112745159:role/GitHubActionsOIDCRole

aws-region: ${{ env.AWS\_REGION }}

# Log in to Amazon ECR to allow pushing Docker images

- name: Login to Amazon ECR

id: login-ecr

uses: aws-actions/amazon-ecr-login@v2

# Build the Docker image and push it to ECR

- name: Build and Push Docker image

run: |

ECR\_URI=$ECR\_REGISTRY/$ECR\_REPOSITORY:$IMAGE\_TAG

docker build -t $ECR\_URI .

docker push $ECR\_URI

# Update kubeconfig to interact with the EKS cluster

- name: Update kubeconfig

run: |

aws eks update-kubeconfig --name $CLUSTER\_NAME --region $AWS\_REGION

# Deploy the application to EKS using Kubernetes manifests

- name: Deploy to EKS

run: |

ECR\_URI=$ECR\_REGISTRY/$ECR\_REPOSITORY:$IMAGE\_TAG

sed "s|<ECR\_IMAGE\_URI>|$ECR\_URI|g" k8s/deployment.yaml| kubectl apply -f -

kubectl apply -f k8s/service.yaml

# Fetch Pod status and LoadBalancer IP

- name: Fetch details

run: |

echo "Fetching pod details.."

sleep 10

kubectl get svc java-app-service

kubectl get pods

---------------------------------------------------------------------------------------------------------------------------------------

**Cleanup**

* Delete node groups

$ aws eks delete-nodegroup --cluster-name my-cluster --nodegroup-name linux-nodes --region us-east-1

* Delete cluster

$ aws eks delete-cluster --name my-cluster