**Github\_Actions\_Deployment\_on\_AWS :**

Create a eks cluster using below script:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#!/bin/bash

aws --version

if [ $? -eq 0 ]

then

echo -e "plain \e[0;31maws cli is already installed \e[0m reset"

else

curl "https://awscli.amazonaws.com/awscli-exe-linux-x86\_64.zip" -o "awscliv2.zip"

sudo apt install unzip

unzip awscliv2.zip

sudo ./aws/install

aws --version

echo -e "plain \e[0;31mAWSCLI is installed \e[0m reset"

fi

curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"

curl -LO "https://dl.k8s.io/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl.sha256"

echo "$(cat kubectl.sha256) kubectl" | sha256sum --check

sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl

kubectl version --client

echo -e "plain \e[0;31mkubectl is installed \e[0m reset"

curl --silent --location "https://github.com/weaveworks/eksctl/releases/latest/download/eksctl\_$(uname -s)\_amd64.tar.gz" | tar xz -C /tmp

sudo mv /tmp/eksctl /usr/local/bin

eksctl version

echo -e "plain \e[0;31mEKSCTL is installed \e[0m reset"

aws configure

eksctl create cluster --name eksdemo --version 1.26 --region us-east-1 --nodegroup-name eksdemo-ng --node-type t3.medium --nodes 2 –managed

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Connect eksctl terminal using putty or MobaXterm

**On eksctl terminal do the following:**

**1.Create pv.yml**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

apiVersion: v1

kind: PersistentVolume

metadata:

name: mysql-pv

spec:

capacity:

storage: 2Gi # Adjust the storage capacity as needed

accessModes:

- ReadWriteOnce

persistentVolumeReclaimPolicy: Delete # This means the volume will be deleted when the PVC is deleted

storageClassName: gp2 # Ensure this matches the storage class you intend to use

hostPath: # For local testing, replace with AWS EBS or your cloud provider's specification

path: /mnt/data/mysql # Adjust this path according to your environment

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**2.create pvc.yml**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: mysql-pvc

spec:

accessModes:

- ReadWriteOnce

resources:

requests:

storage: 2Gi # The requested storage size must match or be less than the PV size

storageClassName: gp2 # Ensure this matches the storage class of your PV

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**3.create mysql-deployment.yml**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

apiVersion: apps/v1

kind: Deployment

metadata:

name: mysql

spec:

replicas: 1

selector:

matchLabels:

app: mysql

template:

metadata:

labels:

app: mysql

spec:

containers:

- name: mysql

image: mysql:latest

env:

- name: MYSQL\_ROOT\_PASSWORD

value: root

- name: MYSQL\_DATABASE

value: automation5

ports:

- containerPort: 3306

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**4.create mysql-service.yml**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

apiVersion: v1

kind: Service

metadata:

name: mysql

spec:

type: ClusterIP

ports:

- port: 3306

targetPort: 3306

selector:

app: mysql

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Run the above created file on eksctl:

Kubectl apply -f pv.yml

Kubectl apply -f pvc.yml

Kubectl apply -f mysql-deployment.yml

Kubectl apply -f mysql-service.yml

**4.Go to the github repository :**

1.Add the MySql loadbalancer to the persistence.xml and application.properties before building.

2.create the deployment and service file for the application :

**Deployment.yml**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

apiVersion: apps/v1

kind: Deployment

metadata:

name: automation-deployment

spec:

replicas: 2

selector:

matchLabels:

app: automation

template:

metadata:

labels:

app: automation

spec:

containers:

- name: automation-container

image: shahnawaz312/java-app:latest

ports:

- containerPort: 9082

resources:

requests:

memory: "256Mi" # Adjust the memory request

cpu: "500m" # Adjust the CPU request

limits:

memory: "512Mi" # Adjust the memory limit

cpu: "1" # Adjust the CPU limit

env:

- name: SPRING\_DATASOURCE\_URL

value: jdbc:mysql://mysql:3306/automation5?createDatabaseIfNotExist=true&useUnicode=true&useJDBCCompliantTimezoneShift=true&useLegacyDatetimeCode=false&serverTimezone=UTC

- name: SPRING\_DATASOURCE\_USERNAME

value: root

- name: SPRING\_DATASOURCE\_PASSWORD

value: root

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**service.yml**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

apiVersion: v1

kind: Service

metadata:

name: automation-service

spec:

type: LoadBalancer

selector:

app: automation

ports:

- protocol: TCP

port: 80

targetPort: 9082

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**5.Now create github actions pipeline:**

Workflow file:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

name: Build and Deploy to EKS

on:

push:

branches:

- main # Trigger workflow on pushes to the main branch

jobs:

build:

runs-on: ubuntu-latest

steps:

- name: Checkout the code

uses: actions/checkout@v2

- name: Set up JDK 17

uses: actions/setup-java@v2

with:

distribution: 'temurin' # Specify Java distribution

java-version: '17'

- name: Build with Maven

run: mvn clean install -DskipTests # Skip tests since no database is used

- name: Build Docker image

run: docker build -t shahnawaz312/java-app:latest .

- name: Login to DockerHub

run: echo "${{ secrets.DOCKER\_PASSWORD }}" | docker login -u "${{ secrets.DOCKER\_USERNAME }}" --password-stdin

- name: Push Docker image to DockerHub

run: docker push shahnawaz312/java-app:latest

deploy:

needs: build

runs-on: ubuntu-latest

steps:

- name: Checkout the repository

uses: actions/checkout@v2

- 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: us-east-1

- name: Update kubeconfig to access EKS cluster

run: aws eks update-kubeconfig --name eksdemo

- name: Deploy the application

run: |

kubectl apply -f deployment.yaml

kubectl apply -f service.yaml

- name: Wait for 10 seconds

run: sleep 10

- name: Verify Deployment

run: kubectl get all

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**6.After running the pipeline :**

1.create a new connection in workbench using the loadbalancer ip of aws.

2. Create a database automation5.

3.run the dump file.

**Description of the pipeline:**

**Pipeline Trigger (on: push)**

* **Trigger Condition**: This pipeline triggers on any push event to the main branch of the repository.

**Jobs Section**

This pipeline consists of two main jobs:

1. **Build**: Compiles and packages the Java application, builds a Docker image, and pushes it to Docker Hub.
2. **Deploy**: Deploys the application to the Amazon EKS cluster.

**Job 1: Build**

The build process runs on the latest version of Ubuntu.

**Steps:**

**1. Checkout the Code**

- name: Checkout the code

uses: actions/checkout@v2

* This action checks out the repository's code so that subsequent steps can use it.

**2. Set up JDK 17**

- name: Set up JDK 17

uses: actions/setup-java@v2

with:

distribution: 'temurin'

java-version: '17'

* This step sets up **JDK 17** using the temurin distribution (OpenJDK-based).
* It ensures that the correct version of Java is available for the build process.

**3. Build with Maven**

- name: Build with Maven

run: mvn clean install -DskipTests

* The Maven command mvn clean install -DskipTests compiles the Java project and packages it into a JAR/WAR file. Tests are skipped to speed up the build process because the database is not needed for this step.

**4. Build Docker Image**

- name: Build Docker image

run: docker build -t shahnawaz312/java-app:latest .

* The docker build command creates a Docker image for the application, tagging it as shahnawaz312/java-app:latest.
* This command uses the Dockerfile in the current directory (.) to build the image.

**5. Login to DockerHub**

- name: Login to DockerHub

run: echo "${{ secrets.DOCKER\_PASSWORD }}" | docker login -u "${{ secrets.DOCKER\_USERNAME }}" --password-stdin

* This step logs into DockerHub using credentials stored in the GitHub repository's secrets (DOCKER\_USERNAME and DOCKER\_PASSWORD).
* It pipes the DockerHub password to the docker login command for secure authentication.

**6. Push Docker Image to DockerHub**

- name: Push Docker image to DockerHub

run: docker push shahnawaz312/java-app:latest

* This command pushes the built Docker image (shahnawaz312/java-app:latest) to the DockerHub repository so that it can be pulled and deployed later.

**Job 2: Deploy**

This job deploys the Dockerized Java application to the EKS cluster.

**Steps:**

**1. Checkout the Repository**

- name: Checkout the repository

uses: actions/checkout@v2

* This action checks out the code repository again so that the Kubernetes configuration files (deployment.yaml and service.yaml) can be accessed for the deployment.

**2. Configure AWS Credentials**

- 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: us-east-1

* This step sets up AWS credentials, required to interact with AWS services (like EKS).
* The credentials (AWS\_ACCESS\_KEY\_ID and AWS\_SECRET\_ACCESS\_KEY) are securely stored in GitHub secrets.
* It also configures the AWS region to us-east-1.

**3. Update kubeconfig to Access EKS Cluster**

- name: Update kubeconfig to access EKS cluster

run: aws eks update-kubeconfig --name eksdemo

* This command updates the local kubeconfig file to allow kubectl to communicate with the specified EKS cluster (eksdemo). The EKS cluster must already exist in AWS.

**4. Deploy the Application**

- name: Deploy the application

run: |

kubectl apply -f deployment.yaml

kubectl apply -f service.yaml

* This step applies the Kubernetes configuration files (deployment.yaml and service.yaml).
  + **deployment.yaml** defines the deployment, including the Docker image, replicas, and other configurations.
  + **service.yaml** defines how the application is exposed to the network (e.g., LoadBalancer, ClusterIP, etc.).

This GitHub Actions pipeline is designed to **build** a Java application, **package** it into a Docker image, **push** it to Docker Hub, and then **deploy** it to an **EKS** (Elastic Kubernetes Service) cluster. Let’s break down each section of the pipeline:

**Pipeline Trigger (on: push)**

* **Trigger Condition**: This pipeline triggers on any push event to the main branch of the repository.

**Jobs Section**

This pipeline consists of two main jobs:

1. **Build**: Compiles and packages the Java application, builds a Docker image, and pushes it to Docker Hub.
2. **Deploy**: Deploys the application to the Amazon EKS cluster.

**Job 1: Build**

The build process runs on the latest version of Ubuntu.

**Steps:**

**1. Checkout the Code**

yaml

Copy code

- name: Checkout the code

uses: actions/checkout@v2

* This action checks out the repository's code so that subsequent steps can use it.

**2. Set up JDK 17**

yaml

Copy code

- name: Set up JDK 17

uses: actions/setup-java@v2

with:

distribution: 'temurin'

java-version: '17'

* This step sets up **JDK 17** using the temurin distribution (OpenJDK-based).
* It ensures that the correct version of Java is available for the build process.

**3. Build with Maven**

yaml

Copy code

- name: Build with Maven

run: mvn clean install -DskipTests

* The Maven command mvn clean install -DskipTests compiles the Java project and packages it into a JAR/WAR file. Tests are skipped to speed up the build process because the database is not needed for this step.

**4. Build Docker Image**

yaml

Copy code

- name: Build Docker image

run: docker build -t shahnawaz312/java-app:latest .

* The docker build command creates a Docker image for the application, tagging it as shahnawaz312/java-app:latest.
* This command uses the Dockerfile in the current directory (.) to build the image.

**5. Login to DockerHub**

yaml

Copy code

- name: Login to DockerHub

run: echo "${{ secrets.DOCKER\_PASSWORD }}" | docker login -u "${{ secrets.DOCKER\_USERNAME }}" --password-stdin

* This step logs into DockerHub using credentials stored in the GitHub repository's secrets (DOCKER\_USERNAME and DOCKER\_PASSWORD).
* It pipes the DockerHub password to the docker login command for secure authentication.

**6. Push Docker Image to DockerHub**

yaml

Copy code

- name: Push Docker image to DockerHub

run: docker push shahnawaz312/java-app:latest

* This command pushes the built Docker image (shahnawaz312/java-app:latest) to the DockerHub repository so that it can be pulled and deployed later.

**Job 2: Deploy**

This job deploys the Dockerized Java application to the EKS cluster.

**Steps:**

**1. Checkout the Repository**

yaml

Copy code

- name: Checkout the repository

uses: actions/checkout@v2

* This action checks out the code repository again so that the Kubernetes configuration files (deployment.yaml and service.yaml) can be accessed for the deployment.

**2. Configure AWS Credentials**

yaml

Copy code

- 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: us-east-1

* This step sets up AWS credentials, required to interact with AWS services (like EKS).
* The credentials (AWS\_ACCESS\_KEY\_ID and AWS\_SECRET\_ACCESS\_KEY) are securely stored in GitHub secrets.
* It also configures the AWS region to us-east-1.

**3. Update kubeconfig to Access EKS Cluster**

yaml

Copy code

- name: Update kubeconfig to access EKS cluster

run: aws eks update-kubeconfig --name eksdemo

* This command updates the local kubeconfig file to allow kubectl to communicate with the specified EKS cluster (eksdemo). The EKS cluster must already exist in AWS.

**4. Deploy the Application**

yaml

Copy code

- name: Deploy the application

run: |

kubectl apply -f deployment.yaml

kubectl apply -f service.yaml

* This step applies the Kubernetes configuration files (deployment.yaml and service.yaml).
  + **deployment.yaml** defines the deployment, including the Docker image, replicas, and other configurations.
  + **service.yaml** defines how the application is exposed to the network (e.g., LoadBalancer, ClusterIP, etc.).

**5. Wait for 10 Seconds**

- name: Wait for 10 seconds

run: sleep 10

* This step introduces a brief pause to give Kubernetes time to spin up the deployment.

**6. Verify Deployment**

- name: Verify Deployment

run: kubectl get all

* This step verifies that the deployment was successful by listing all Kubernetes resources (pods, services, etc.) currently running in the cluster.

**Conclusion**

This pipeline performs continuous integration and continuous deployment (CI/CD) by:

1. **Building** the Java application.
2. **Packaging** it into a Docker image.
3. **Pushing** the image to DockerHub.
4. **Deploying** the application to an AWS EKS cluster.