4. Deploy a web application in the Kubernetes pod. And create a replica set. In any case load is going to increase on your replica set. increase the number of replicas of the pods.

1. Creating an eks cluster:
   We will use eks cluster here to make the k8s cluster

   Follow the below commands and paste it in command prompt

   apt-get update -y
   apt install unzip -y
   curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
   unzip awscliv2.zip
   sudo ./aws/install
   aws configure

   This will install aws cli in your machine and after hitting aws configure

```
AWS Access Key ID [****************QV5N]:
AWS Secret Access Key [****************h1iZ]:
Default region name [ap-south-1]:
Default output format [table]:
```

   Add your id and secrets

   Installl eksctl tool

   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

   This commands will install eksctl tool

   Now lets install kubectl

curl -LO https://storage.googleapis.com/kubernetes-release/release/$(curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt)/bin/linux/amd64/kubectl
sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl
kubectl version –client

This will install kubectl in your machine

2. Now run the following command
   eksctl create cluster --name my-cluster --region region-code --version 1.29 --vpc-public-subnets subnet-ExampleID1,subnet-ExampleID2 --without-nodegroup

   Here mention your subnets and region code

3. Lets create the node groups

```
eksctl create nodegroup \
  --cluster my-cluster \
  --region us-east-2 \
  --name my-node-group \
  --node-ami-family Ubuntu2004 \
  --node-type t2.small \
  --subnet-ids subnet-086ced1a84c94a342,subnet-01695faa5e0e61d97 \
  --nodes 3 \
  --nodes-min 2 \
  --nodes-max 4 \
  --ssh-access \
  --ssh-public-key /root/.ssh/id_rsa.pub
```

This will now create nodegroups

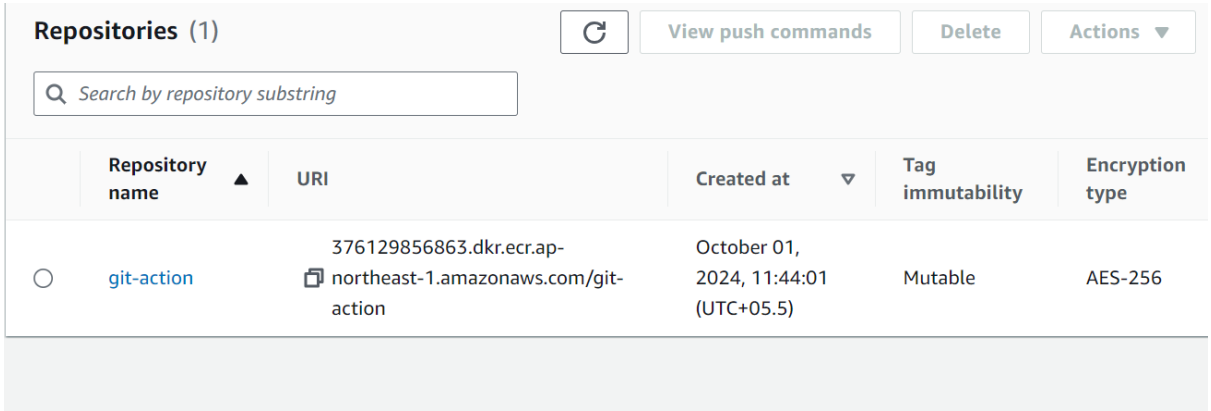
Note:- add you own region name of your cluster with proper subnet ids

```
[root@eks-manager ~]# eksctl get cluster
NAME            REGION          EKSCTL CREATED
git-action      ap-northeast-1  True
```

The cluster has being created

```
NAME                                                STATUS   ROLES    AGE   VERSION
ip-172-31-13-142.ap-northeast-1.compute.internal    Ready    <none>   2d4h  v1.29.6
ip-172-31-16-25.ap-northeast-1.compute.internal     Ready    <none>   2d4h  v1.29.6
ip-172-31-26-211.ap-northeast-1.compute.internal    Ready    <none>   2d4h  v1.29.6
```

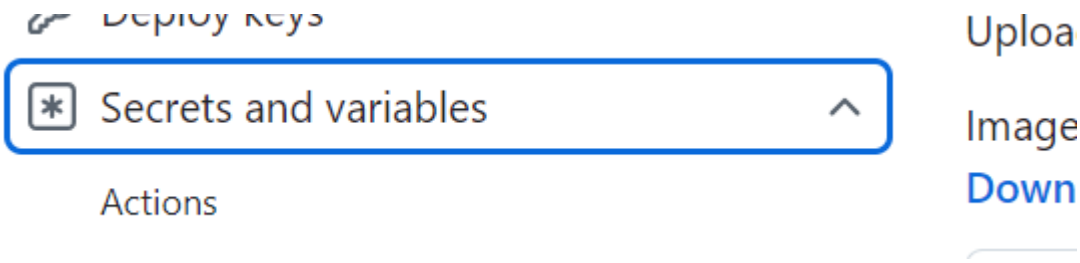
Create an ecr repo



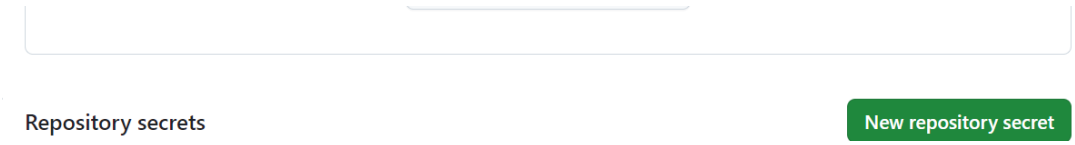
We will use git action to make our ci/cd pipeline

a.)Go to settings of repository

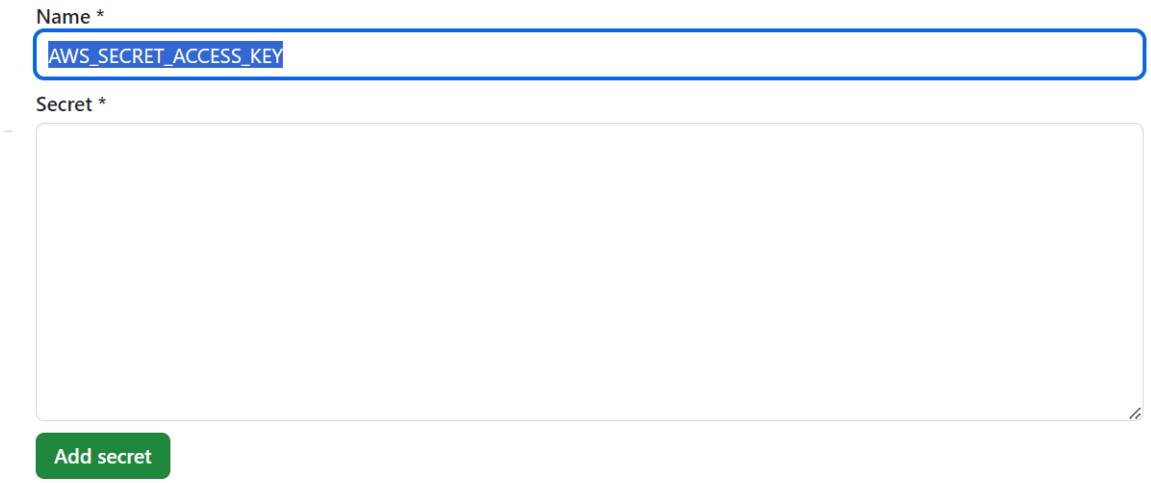
Settings->Secrets&Variable->Actions



b.) Go to new repository secrets

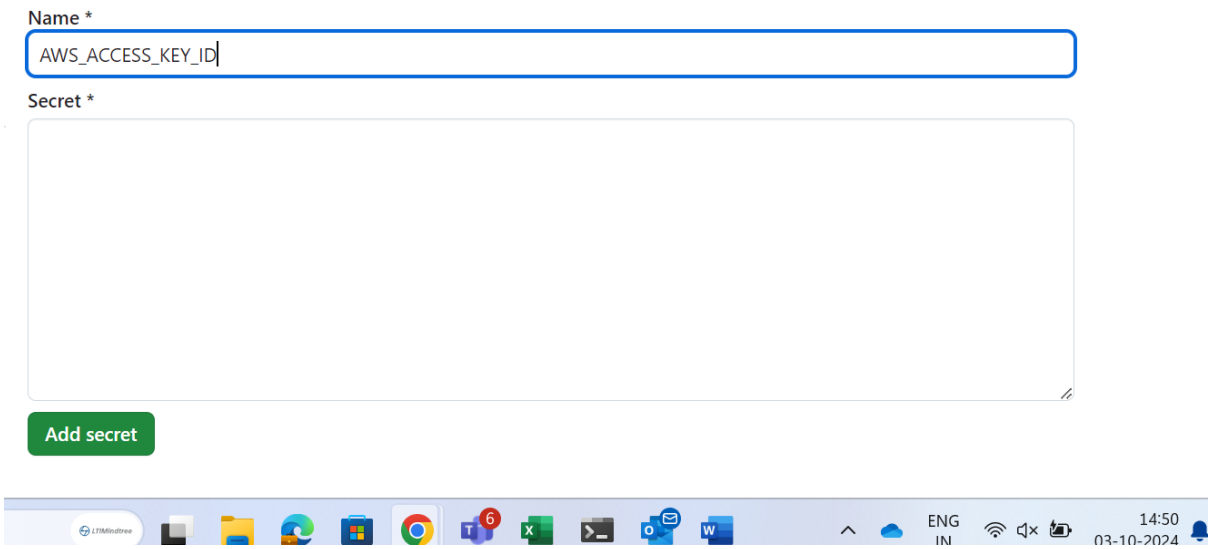


Add AWS_SECRET_ACCESS_KEY



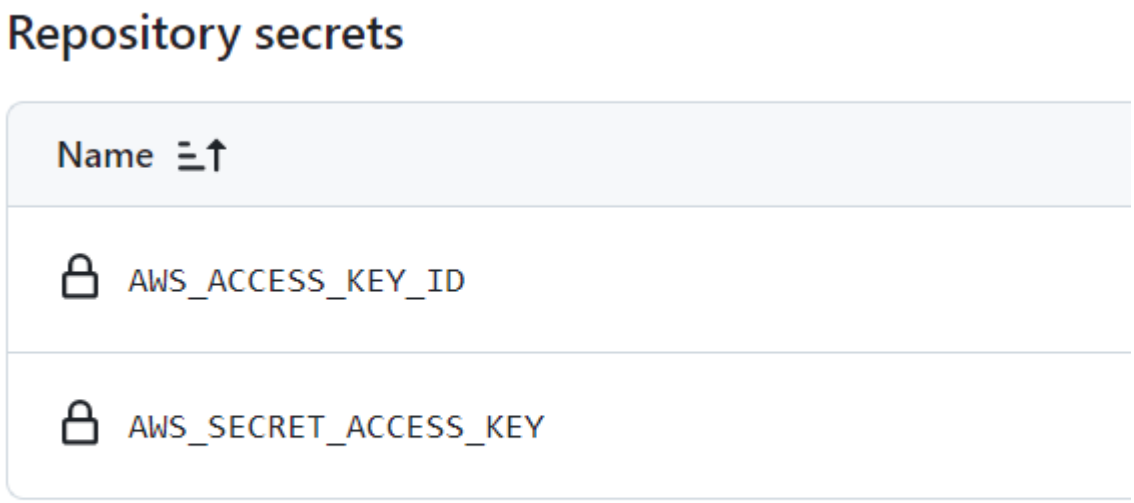
Create one more new secret variable

AWS_ACCESS_KEY_ID



Not:Add your respective key id and secret that you get from creating an IAM user

Now you will be able to see secrets and id in you repository settings



3. Now create a folder named .github and inside it workflows

This folder will consist of your workflow file which will trigger the action

Now write the following script inside .github/workflows

```
name: Java CI with Maven

on:
  push:
    branches: [ "main" ]
  pull_request:
    branches: [ "main" ]

jobs:
  build:
```

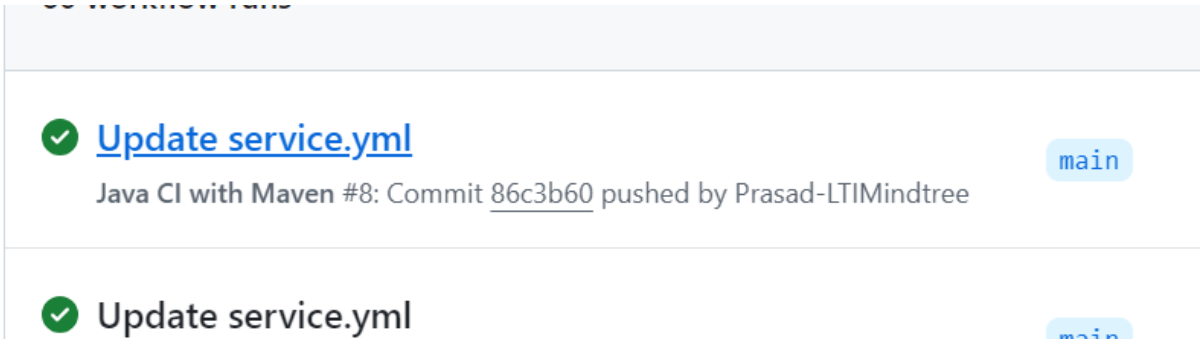
```
runs-on: ubuntu-latest

steps:
- uses: actions/checkout@v4
- name: Set up JDK 17
  uses: actions/setup-java@v4
  with:
    java-version: '17'
    distribution: 'temurin'
    cache: maven
- name: Build with Maven
  run: mvn -B package --file pom.xml

- name: Upload build artifact
  uses: actions/upload-artifact@v4
  with:
    name: war-file
    path: /home/runner/work/project-maven/project-maven/webapp/target/*.war

- name: List build directory contents
  run: ls -la /home/runner/work/project-maven/project-maven/webapp/target

- name: Download build artifact
  uses: actions/download-artifact@v4
  with:
    name: war-file
    path: ./artifact
```

```
  - name: List build directory contents
    run: ls -la /home/runner/work/project-maven/project-maven/artifact

  - 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: ap-northeast-1
  - name: Log in to Amazon ECR
    id: login-ecr
    uses: aws-actions/amazon-ecr-login@v1
  - name: Build Docker image
    run: |
      docker build -t git-action .
      docker tag git-action:latest 376129856863.dkr.ecr.ap-northeast-1.amazonaws.com/git-action:latest

  - name: Push Docker image to ECR
    run: |
      docker push 376129856863.dkr.ecr.ap-northeast-1.amazonaws.com/git-action:latest
  - name: Set up kubectl
    uses: azure/setup-kubectl@v1
    with:
      version: v1.31.0
  - name: Update kubeconfig
    run: |
      aws eks update-kubeconfig --name git-action --region ap-northeast-1
```

```
- name: Deploy to EKS
  env:
    ECR_REGISTRY: ${{ steps.login-ecr.outputs.registry }}
    ECR_REPOSITORY: git-action
    IMAGE_TAG: ${{ github.sha }}
  run: |
    kubectl apply -f  webapp/deployment.yml
    kubectl apply -f webapp/service.yml
```

This scriot will build the image push it to the ecr and deploy to an pod of our cluster

Opnce you create a commit build will run



Now that our build is done lets see them in pods

Note: don't forget to write following deployment.yml and service.yml file inside webapp folder of your git repo

Deployment.yml:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: regapp-deployment
```

```
  labels:
    app: regapp

spec:
  replicas: 2
  selector:
    matchLabels:
      app: regapp

  template:
    metadata:
      labels:
        app: regapp
    spec:
      containers:
      - name: regapp
        image: 376129856863.dkr.ecr.ap-northeast-1.amazonaws.com/git-action:latest
        imagePullPolicy: Always
        ports:
        - containerPort: 8080
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 1
```

Service.yml

```
apiVersion: v1
kind: Service
metadata:
  name: regapp-deployment
  labels:
    app: regapp
spec:
  selector:
    app: regapp
  ports:
    - port: 8080
      targetPort: 8080
  type: LoadBalancer
```

Lets check the pods

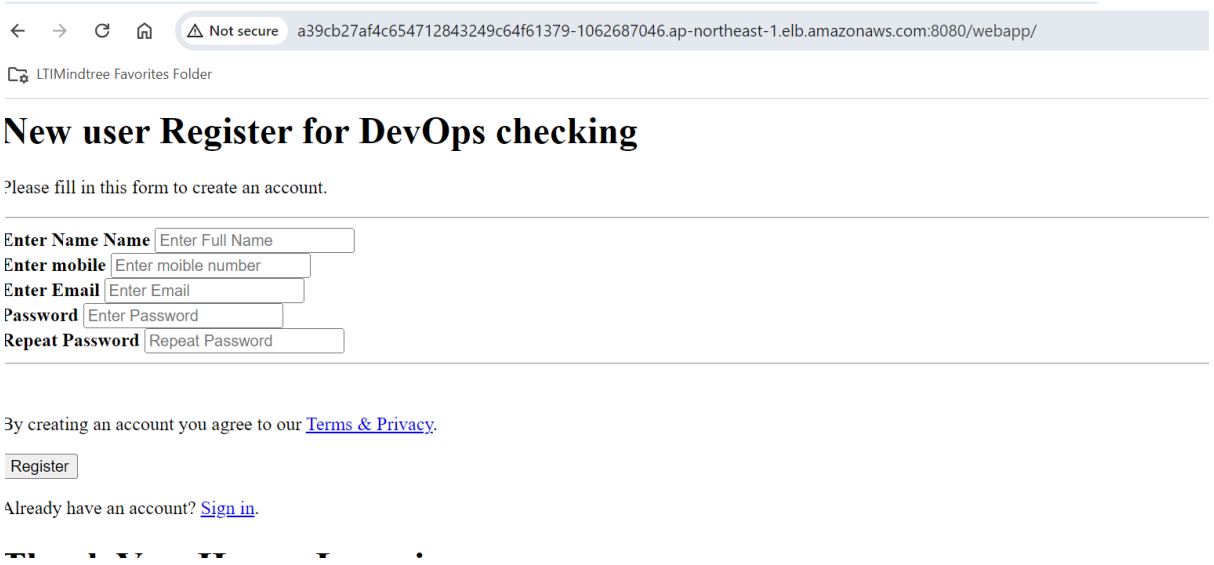
You can see regapp pods has being successfully created

```
[root@eks-manager ~]# kubectl get pods
NAME                                  READY   STATUS    RESTARTS
regapp-deployment-7b57fd57b8-tmxr8    1/1     Running   0
regapp-deployment-7b57fd57b8-xzl4q    1/1     Running   0
```

Now to chek the application

```
[root@eks-manager ~]# kubectl get svc
NAME                TYPE           CLUSTER-IP       EXTERNAL-IP
RT(S)          AGE
kubernetes          ClusterIP      10.100.0.1       <none>
3/TCP          2d5h
regapp-deployment   LoadBalancer   10.100.157.235   a39cb27af4c654712843249c64f61379-1062687046.ap-northeast-1.elb.amazonaws.com
80:31032/TCP   2d3h
```

Copy the external ip and chek in browser



Our web app is live