Deploy App to EKS with GitHub CI.

This document will show you

* How to use GitHub actions to create a docker image out of your source code,
* Push it to Amazon Elastic Container Service (ECR).
* Use that image for deploying our application with the help of Amazon Elastic Kubernetes Service (EKS) on every code commit to GitHub.

Official Documentation Links:

<https://docs.aws.amazon.com/AmazonECR/latest/userguide/what-is-ecr.html>

<https://docs.aws.amazon.com/eks/latest/userguide/what-is-eks.html>

<https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions>

<https://aws.amazon.com/iam/>

<https://docs.docker.com/get-started/overview/>

<https://kubernetes.io/docs/concepts/>

Please note that AWS may charge you if you exceed the daily or monthly limits specified for free-tier accounts.

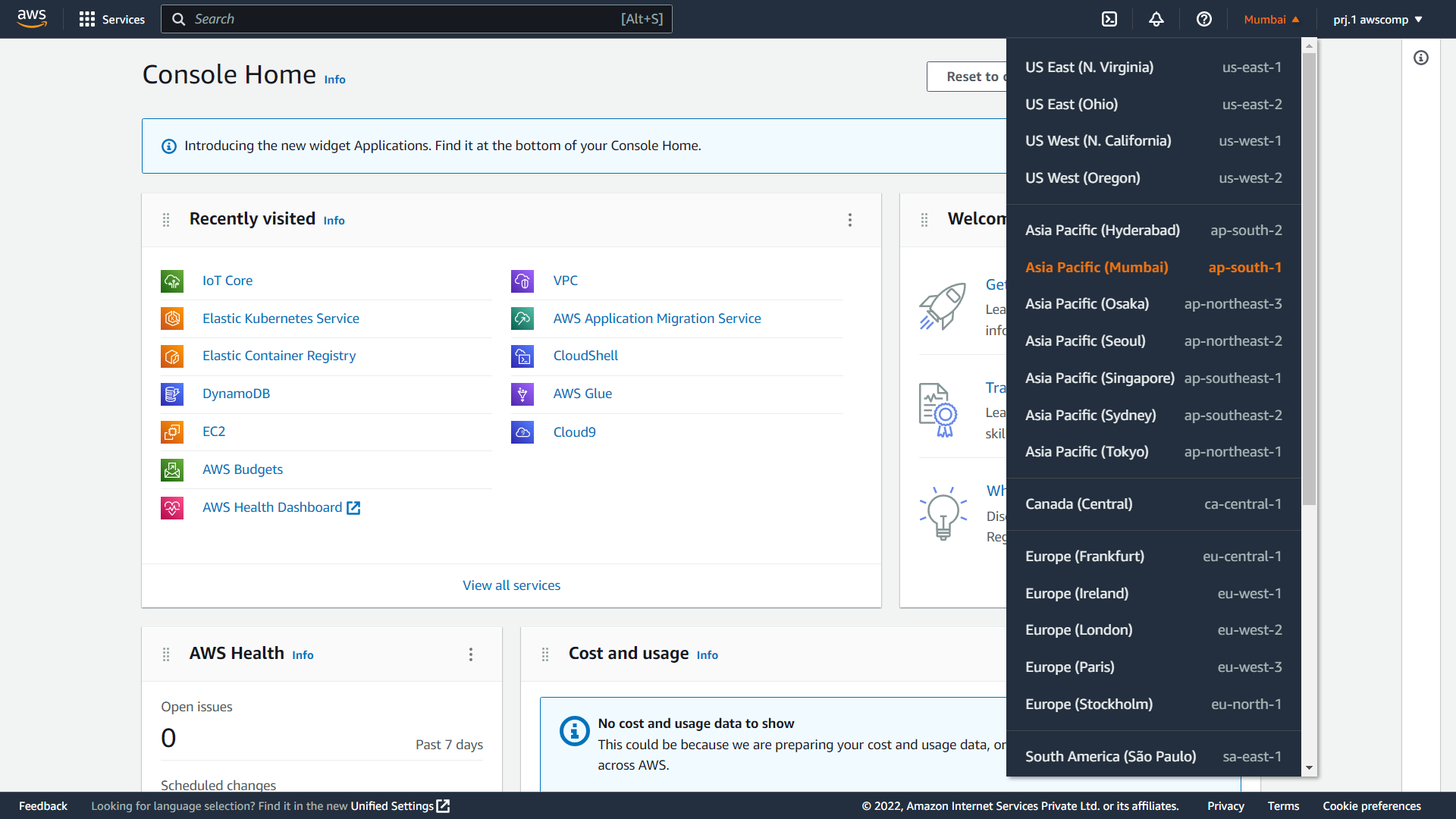
You may require IAM access for creations of roles, user and user groups which are then used for creation of Kubernetes clusters, Container repositories, etc.

Besides the configuration used for creating instances in this project may differ than what you might require according to your application needs.

**Part 1: On changes in master branch of GitHub, a workflow should be triggered that builds docker images and pushes to AWS ECR.**

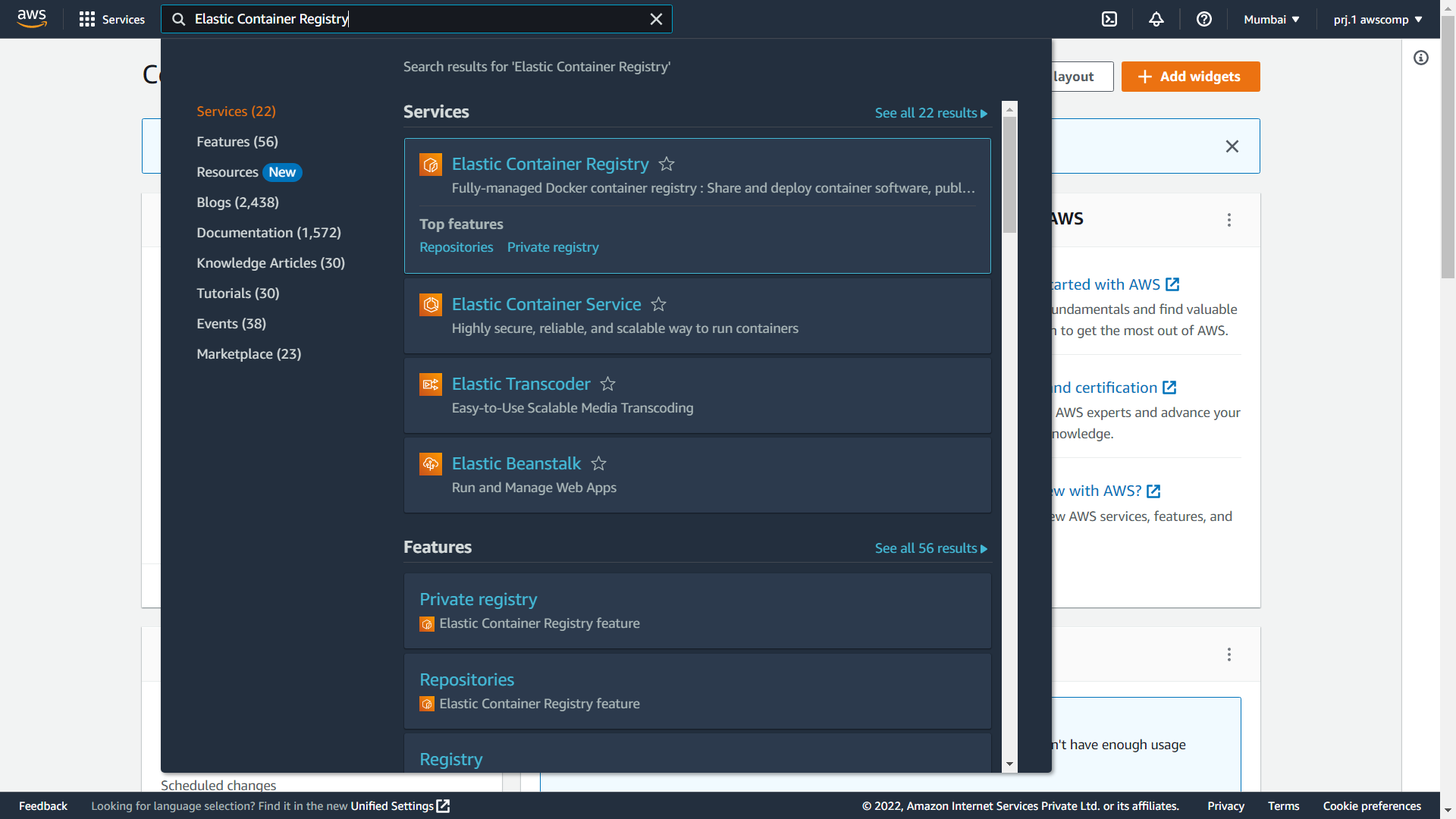
1. On AWS Dashboard, from the dropdown on top right, select region that is nearest to you.





*(Before we push images to Amazon ECR, i.e., Elastic Container Registry, we need to create a repository)*

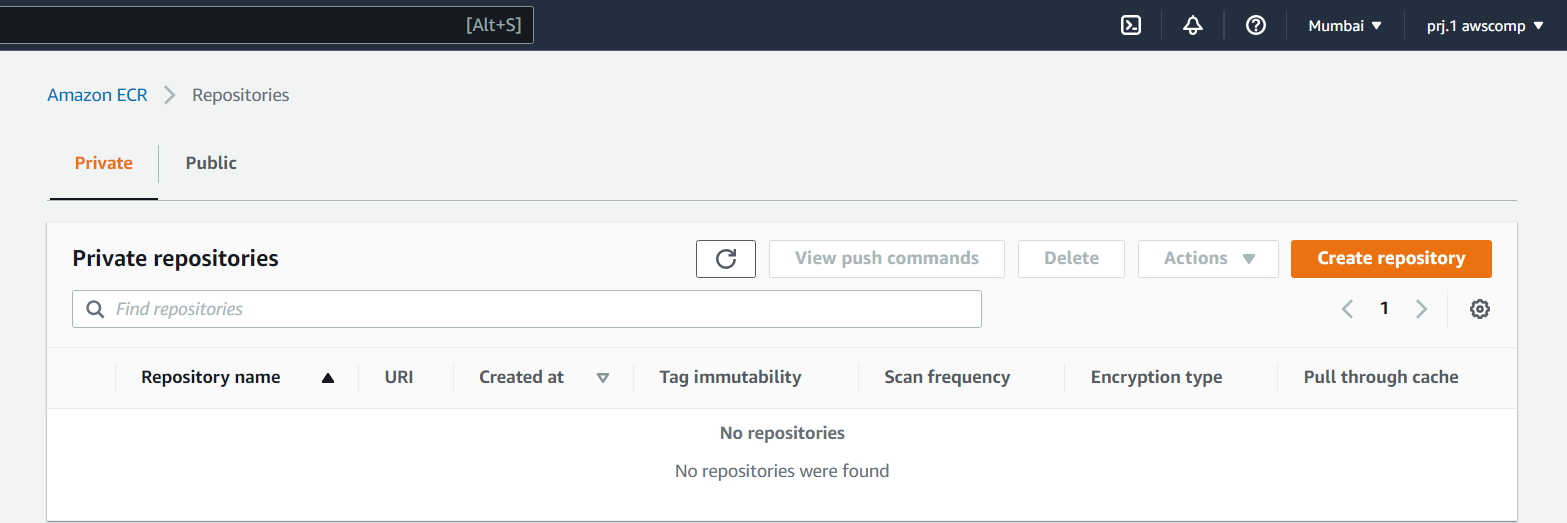
1. Go to Elastic Container Service Registry



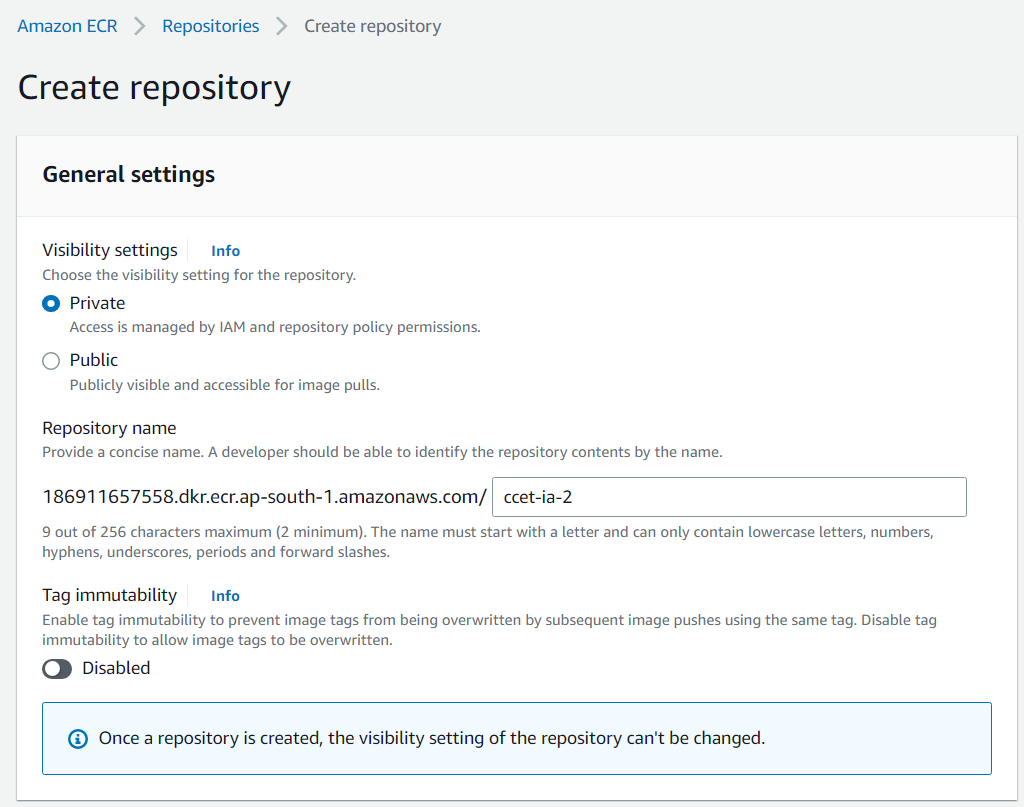
1. In ECR, go to repositories, from the navigation menu.

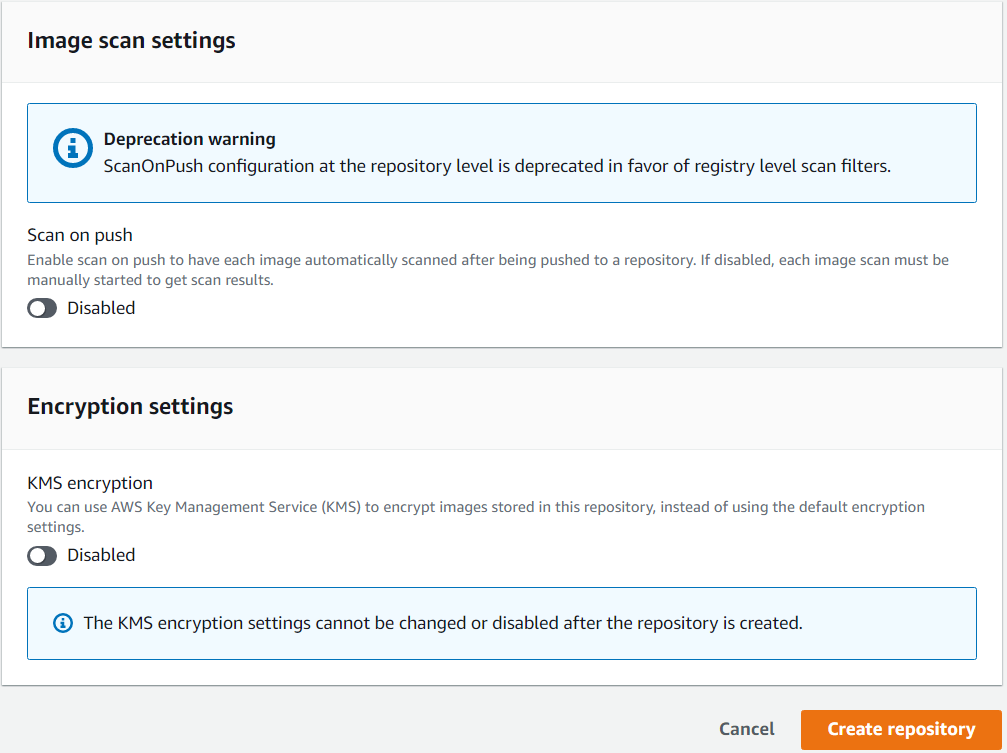


1. Next let’s create a repository so that we can push images to it. Creating a private repository prevents pushing or pulling of images by others except those who are given access to the repository.

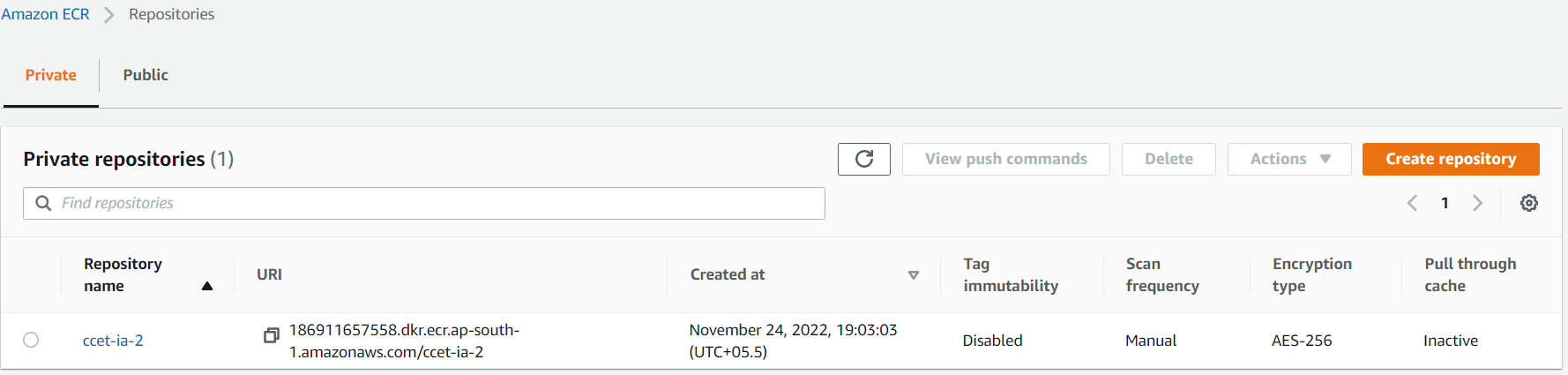


1. Name the repository and configure other settings according to your needs and click on create repository.



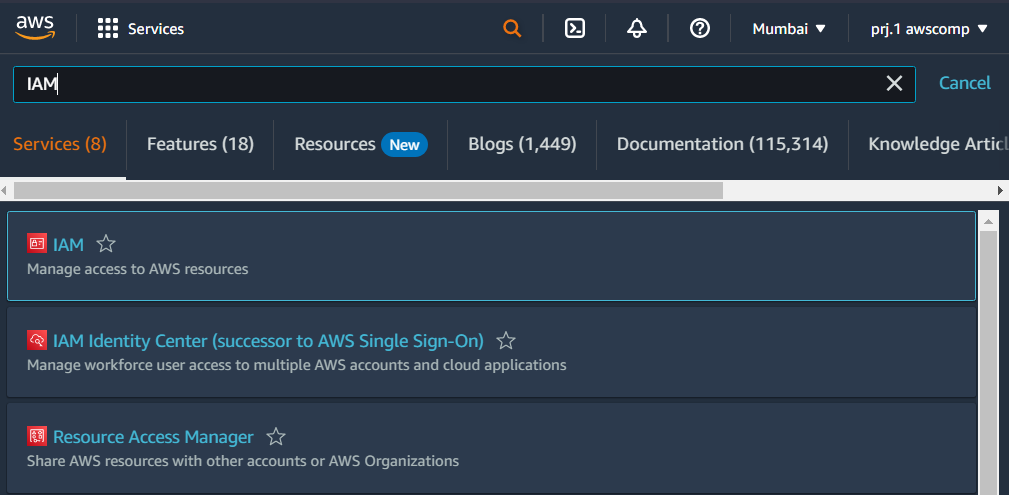


1. After successfully creating a repository, we can use the repo URI to push and pull images.

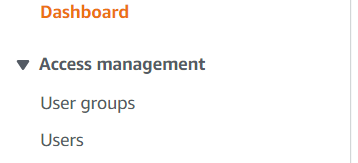


*(Now our next step is to make a workflow in GitHub actions so that on each push or pull request on master branch will trigger the workflow. But before that we need the computer node which runs GitHub workflow to login into AWS and for that we need to follow the below steps.)*

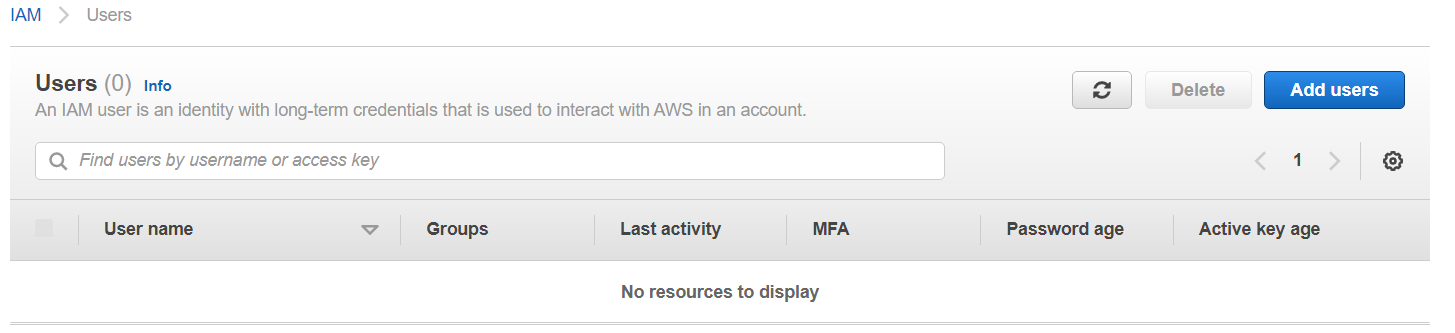
1. *Go to IAM Dashboard in AWS*

**

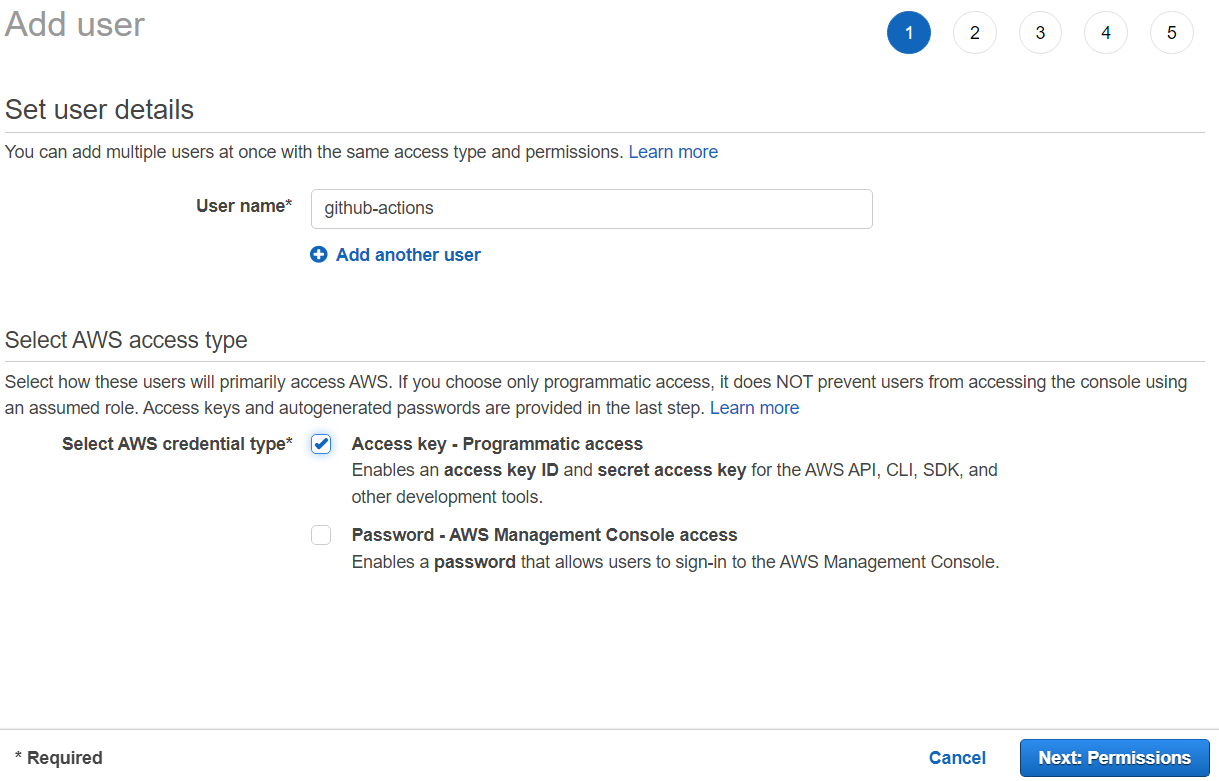
1. Click on Users



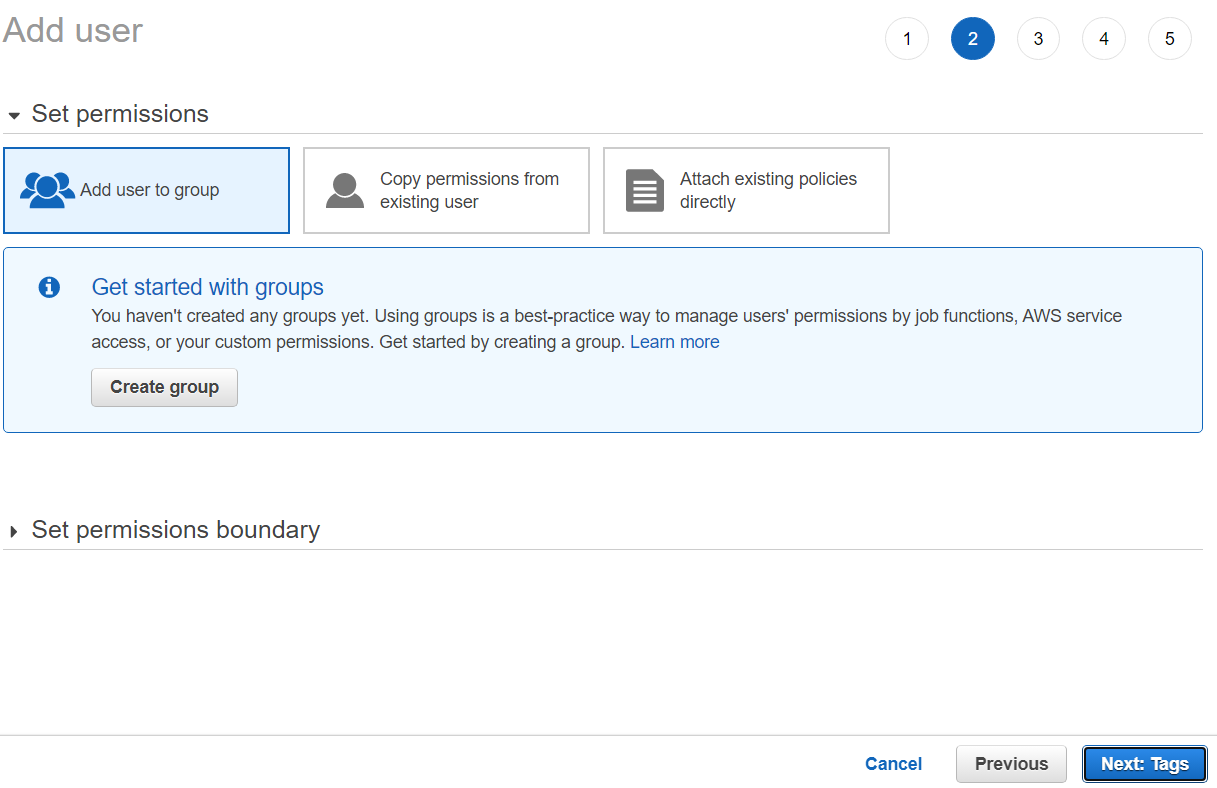
1. Add User



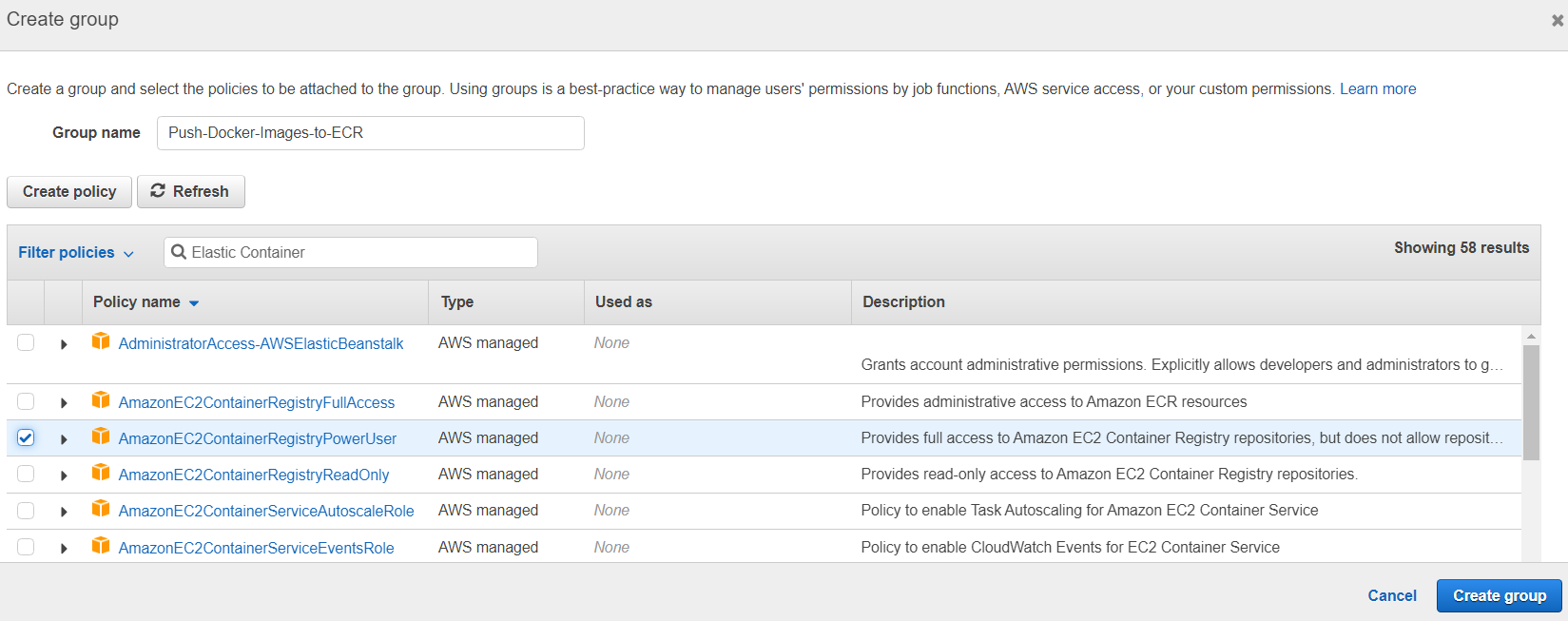
1. Name your user and select access type.

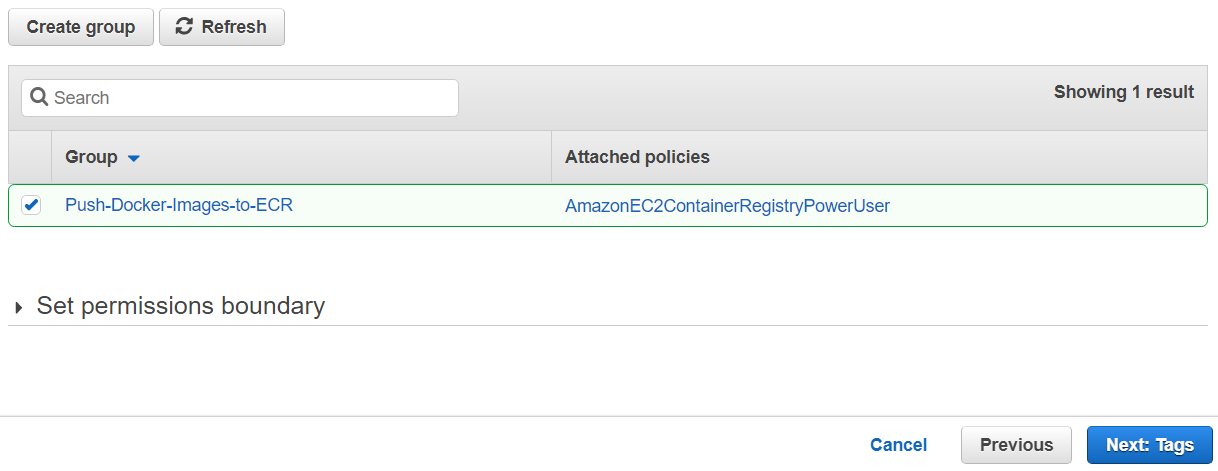


1. Next, create a group to add user to it.

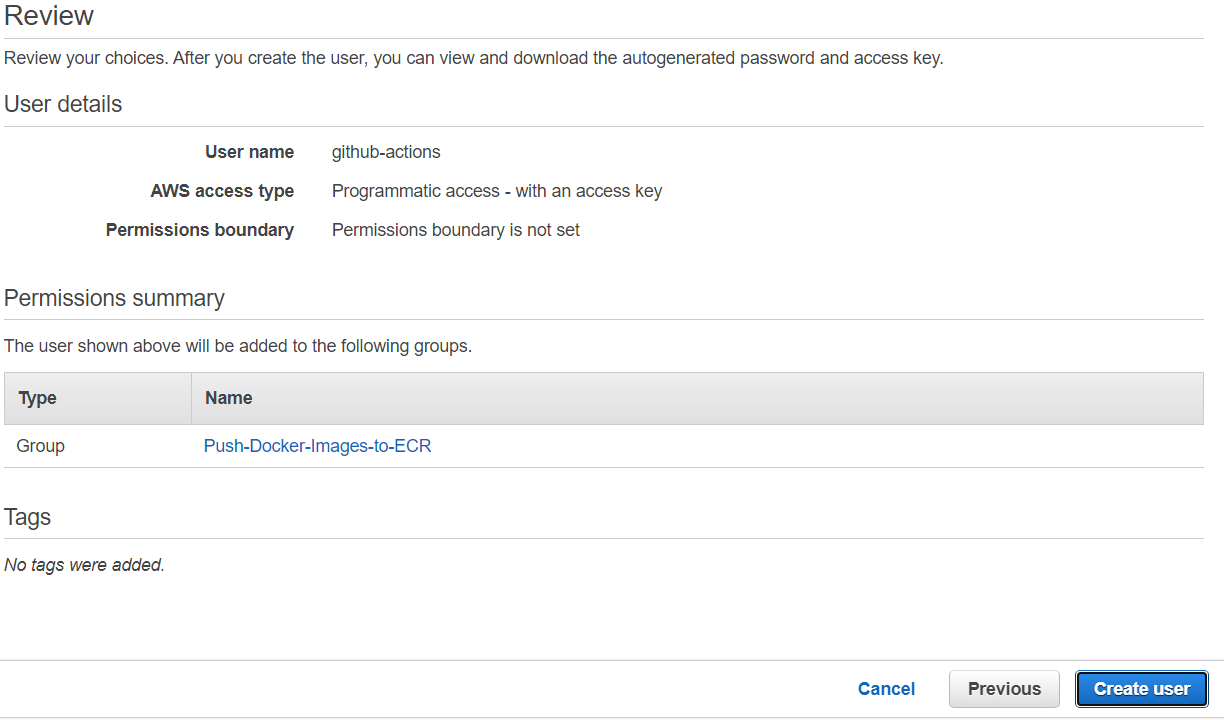


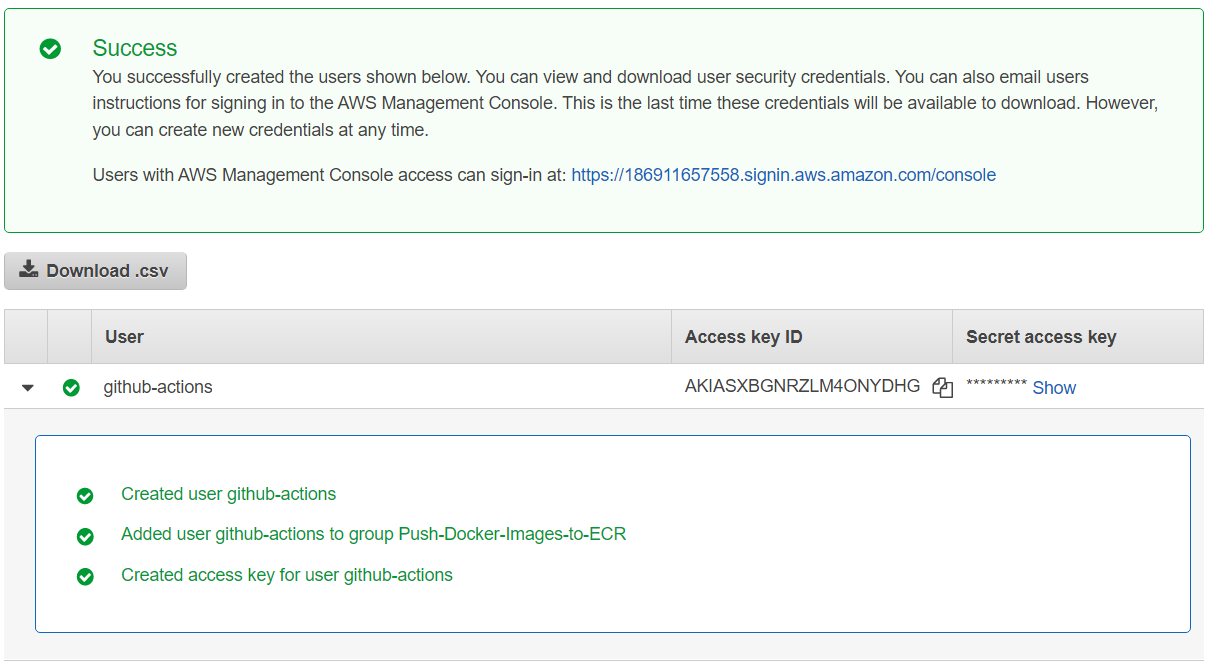
1. Name the group and select the policy.



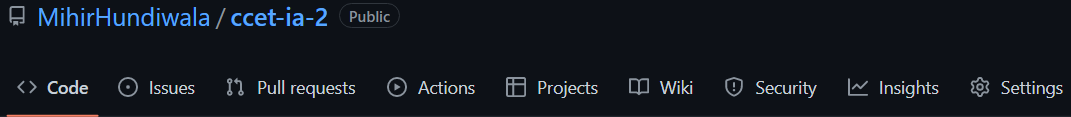


1. Adding tags is optional so we will click next and review the user before creating

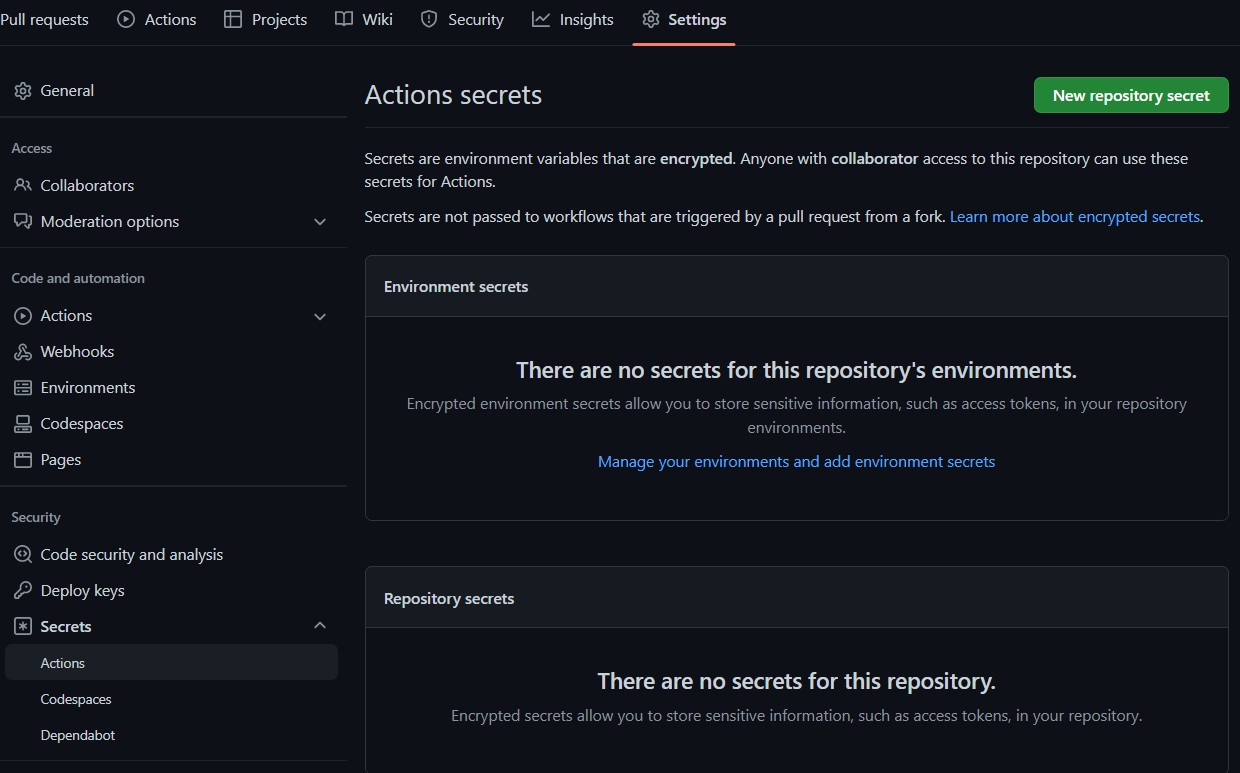




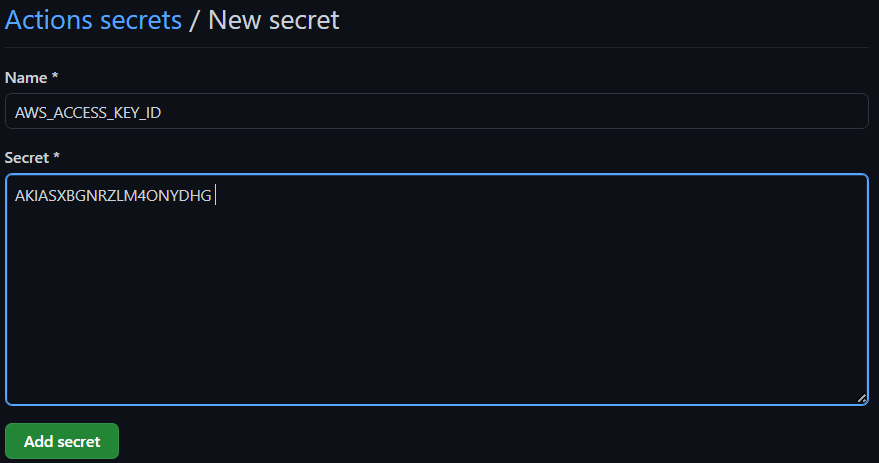
1. Let’s add these secrets to our GitHub repository, to do this we need to go to settings of our repository.



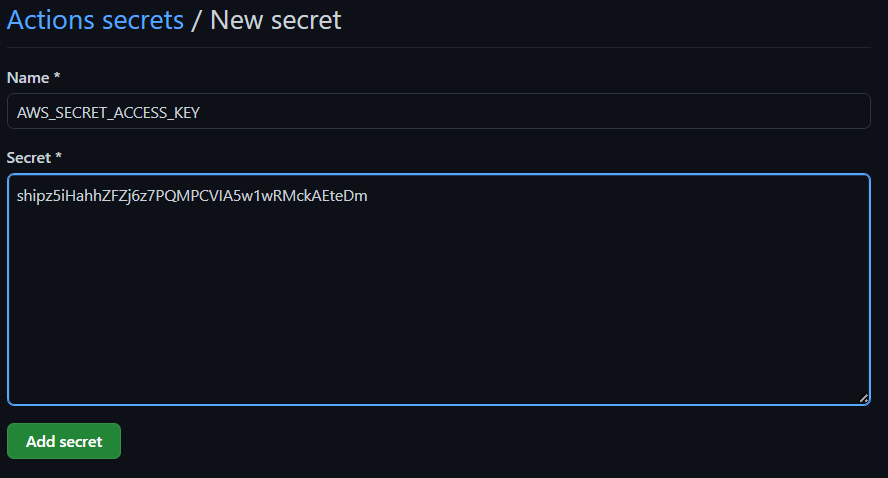
1. Go to secrets 🡪 actions 🡪 and create new repository secret



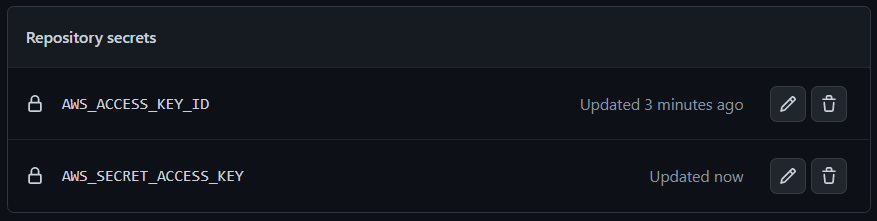
1. Copy AWS\_ACCESS\_KEY\_ID from the user we just created and create a secret in GitHub



1. Do the same for AWS\_SECRET\_ACCESS\_KEY



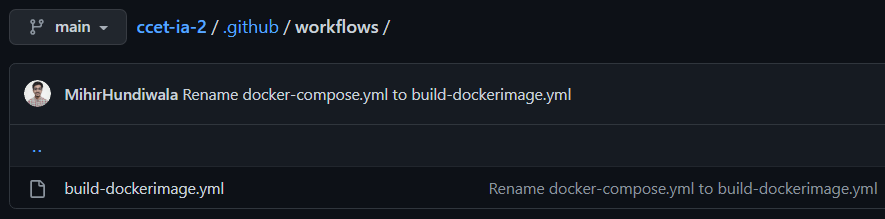




1. Let’s head to create our workflows for GitHub Actions. Workflow files are created inside

.github 🡪 workflow 🡪 *filename*.yml in your repository.

We have build-dockerimage.yml which will build a docker image out of our source code and push it to the AWS Elastic Container Registry.



**docker-compose.yml**

name: Build and Push image to Amazon ECR

on:

  push:

    branches: [ main ]

  pull\_request:

    branches: [ main ]

jobs:

  build:

    name: Build and Push Image to Amazon ECR

    runs-on: ubuntu-latest

    steps:

      - name: Check out code for building image

        uses: actions/checkout@v1

      - 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-south-1

      - name: Login to Amazon ECR

        id: login-ecr

        uses: aws-actions/amazon-ecr-login@v1

      - name: Build, tag, and push docker image to Amazon ECR

        env:

          ECR\_REGISTRY: ${{ steps.login-ecr.outputs.registry }}

          ECR\_REPOSITORY: ccet-ia-2

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

        run: |

          docker build -t $ECR\_REGISTRY/$ ECR\_REPOSITORY:$IMAGE\_TAG .

          docker push $ ECR\_REGISTRY/$ ECR\_REPOSITORY:$IMAGE\_TAG

1. Let’s, create a Dockerfile without which we cannot create docker image of our app. Dockerfile should be situated in the root folder.

FROM python:3.8.10

COPY ./src /app

WORKDIR /app

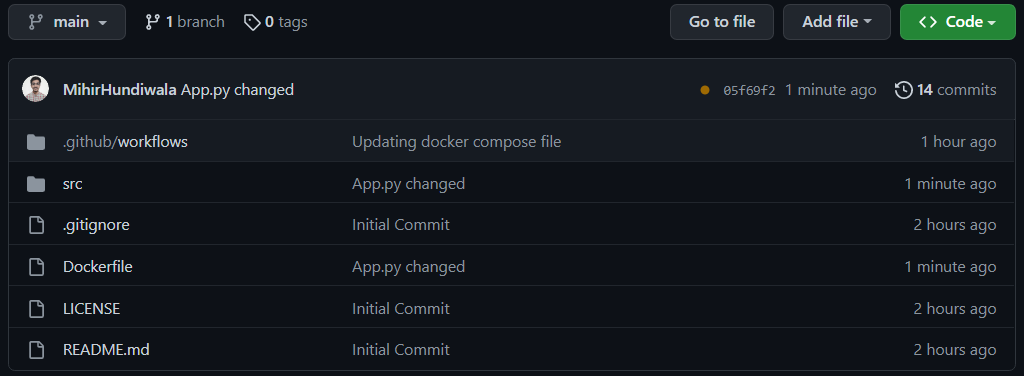
RUN pip install -r requirements.txt

EXPOSE 5000

ENTRYPOINT [ "flask"]

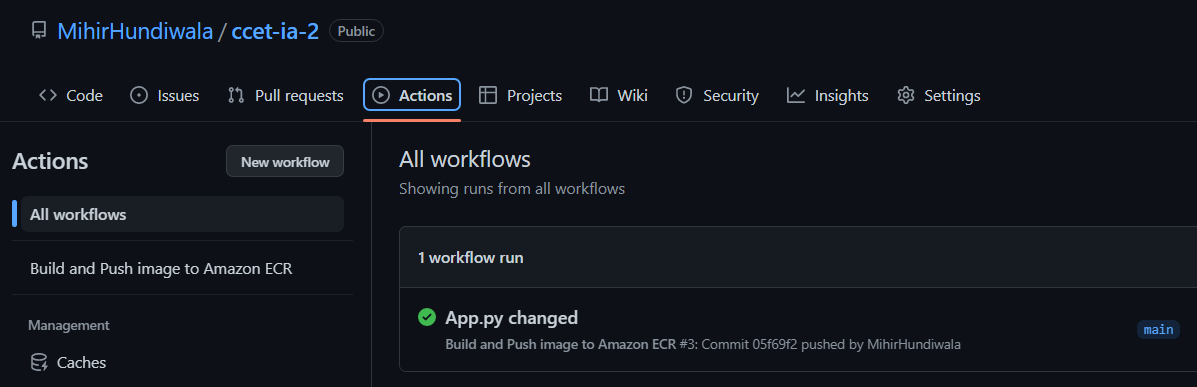
CMD ["run", "-h", "0.0.0.0", "-p", "5000"]

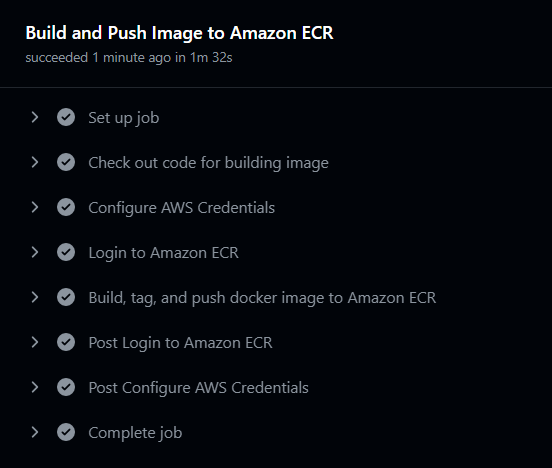
Now we can push all the changes to repository.



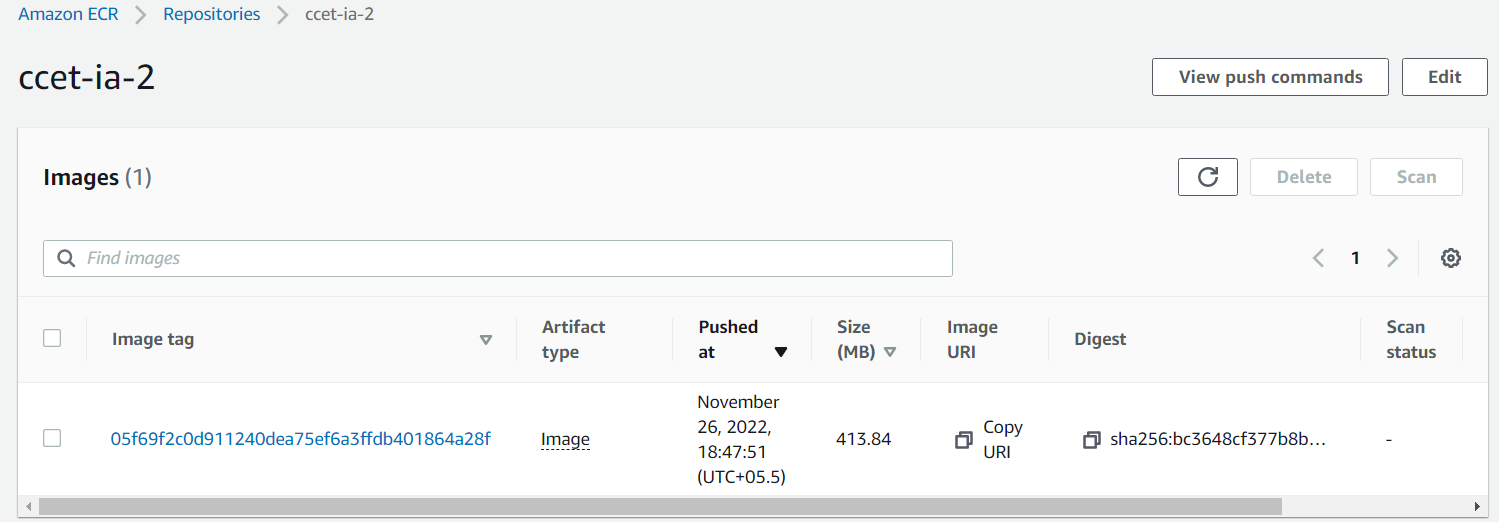


1. Let’s try to make some changes on our master branch and let’s see if our workflow triggers or not. (Note in the above image that I have already pushed some code and that should have already triggered a workflow. Go to actions to look for workflows.





1. Go to AWS ECR Repository to check if the image has been pushed.





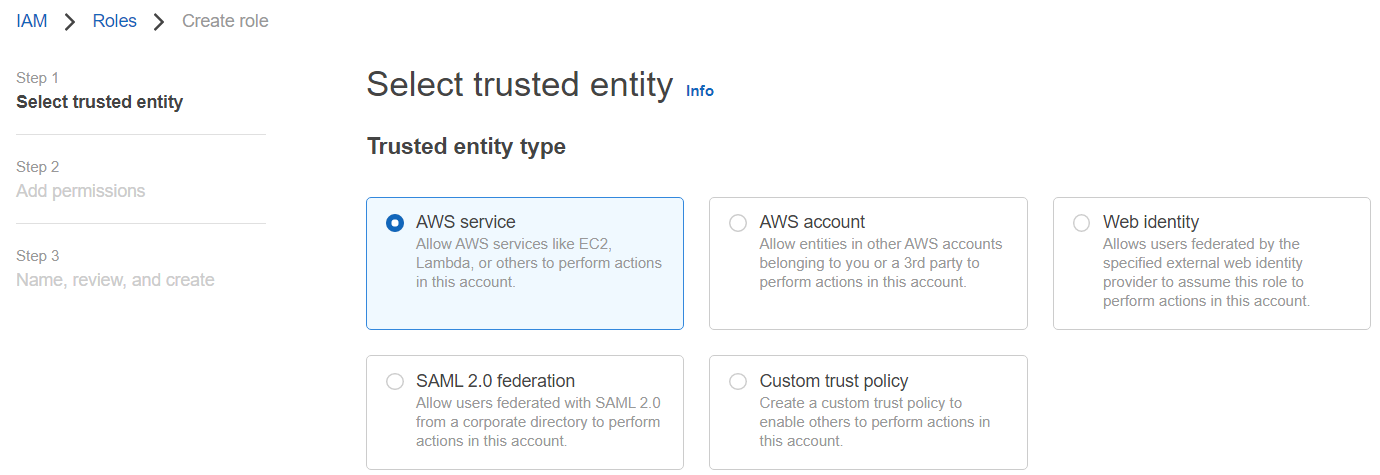
*Congratulations, Part 1 is completed!*

**Part 2: Create Kubernetes Cluster using EKS.**

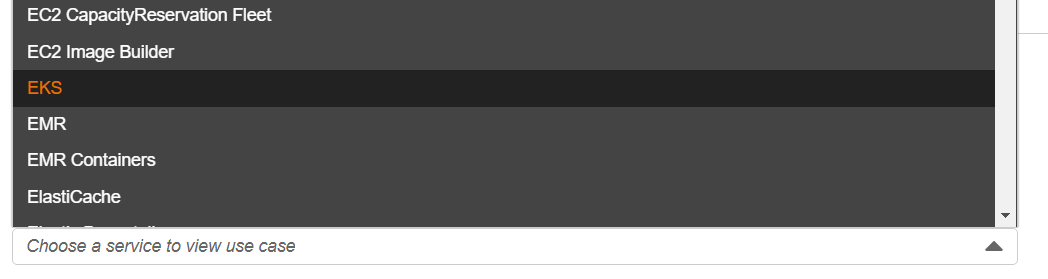
*(First, we need to create Kubernetes cluster before deploying the application.)*

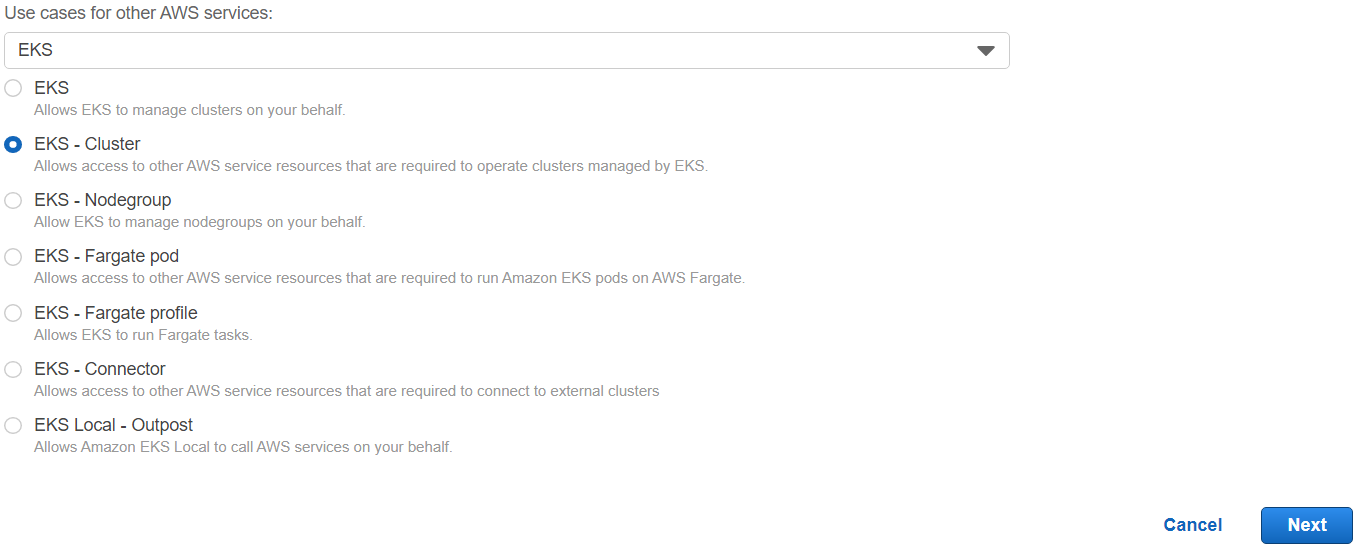
*(Before creating cluster, we need to create an IAM role to manage Kubernetes resources.)*

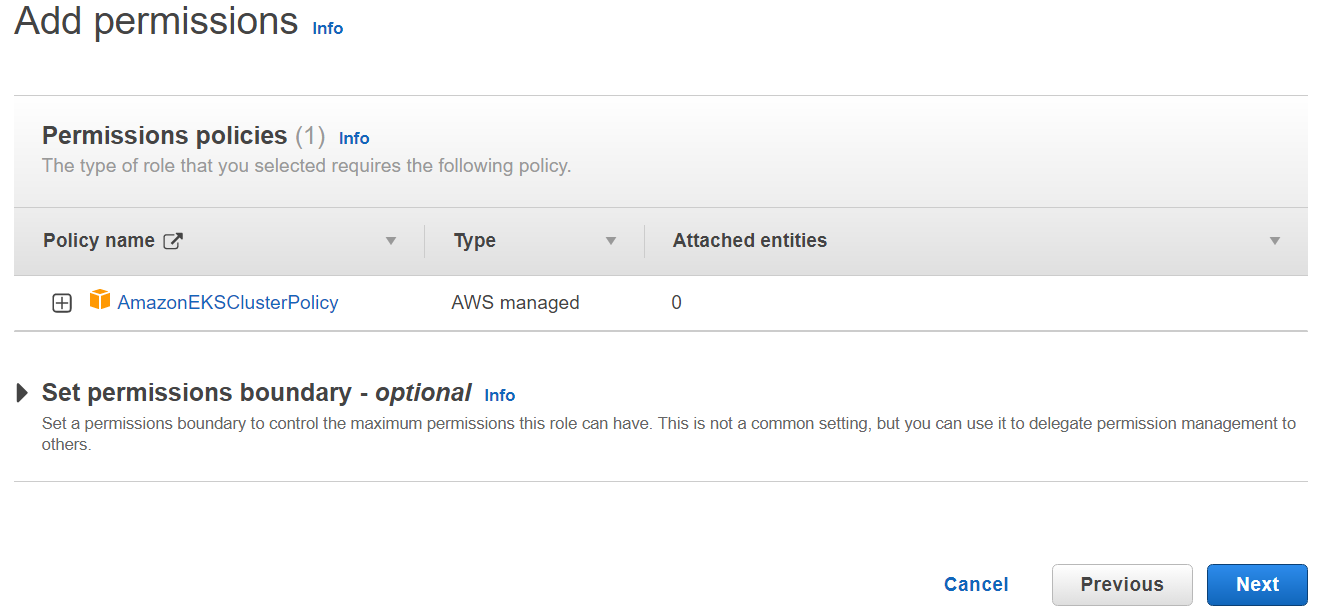
1. Go to IAM 🡪 Roles

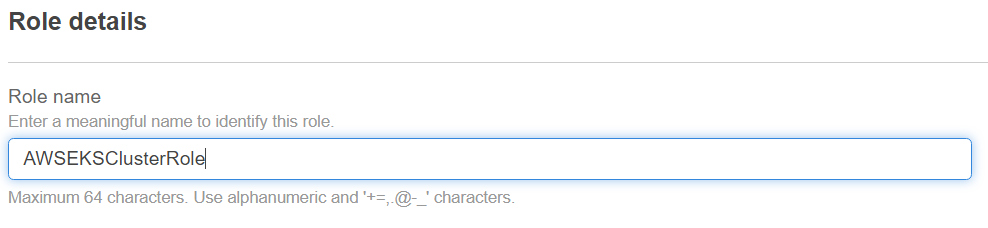


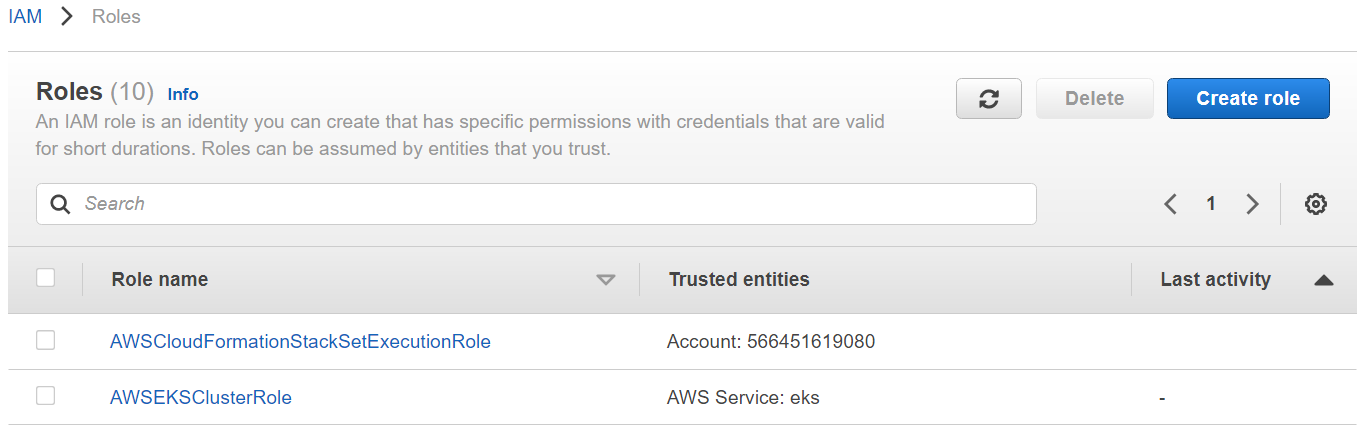
1. For use case select EKS Cluster.



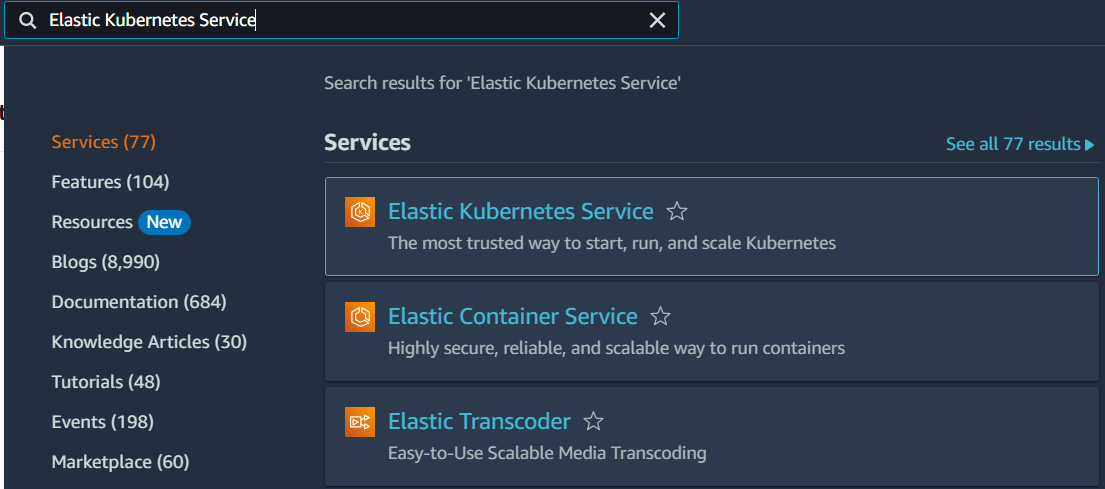




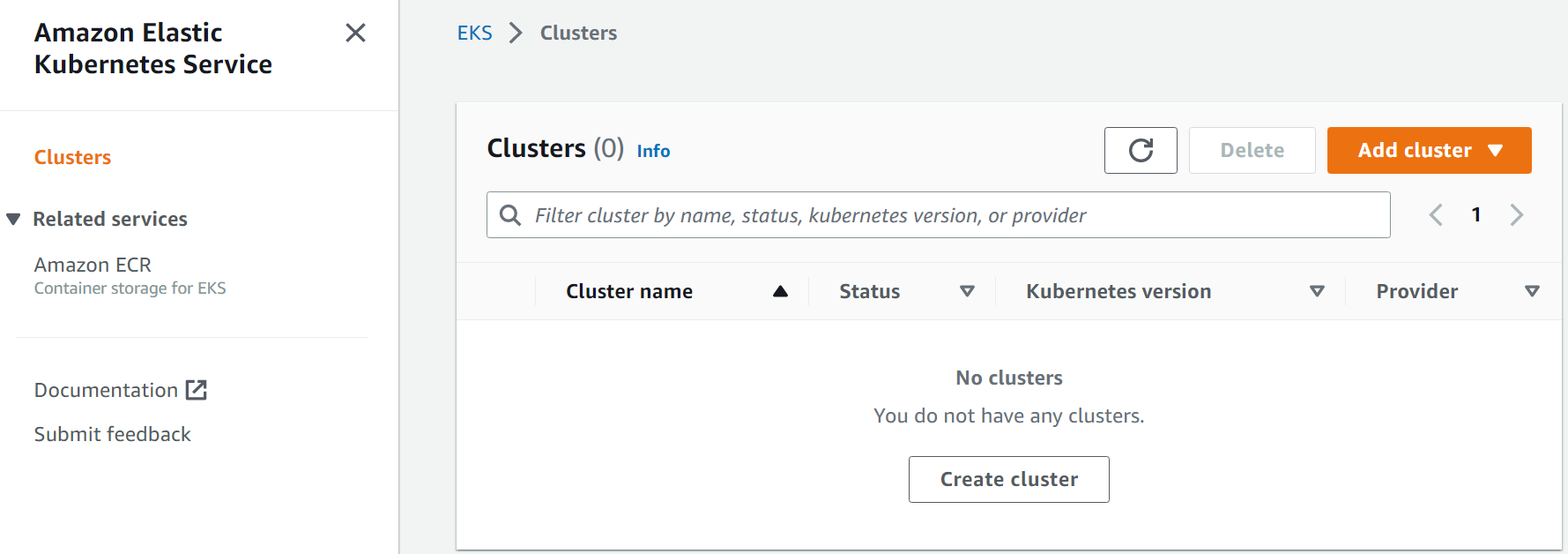
*(Click on Create Role)*



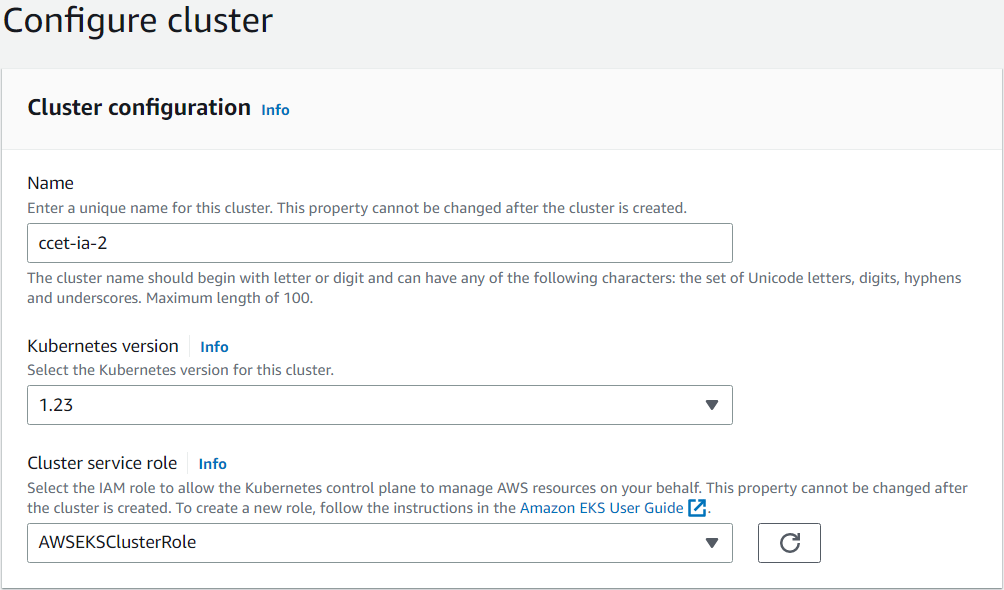
1. Go to Amazon EKS

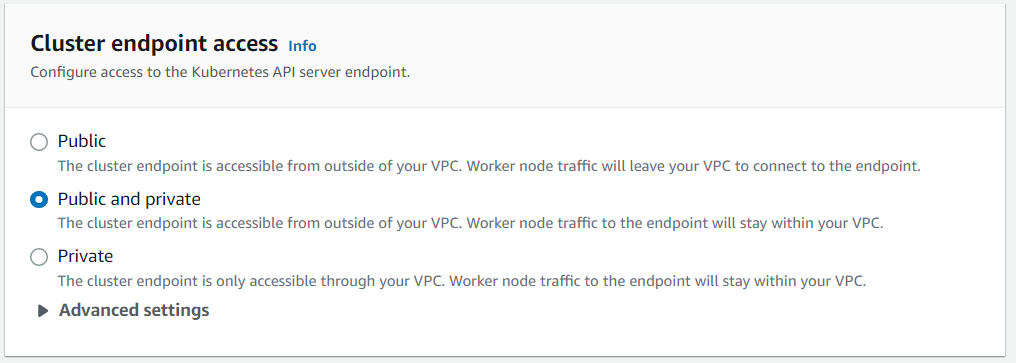


1. Create a cluster

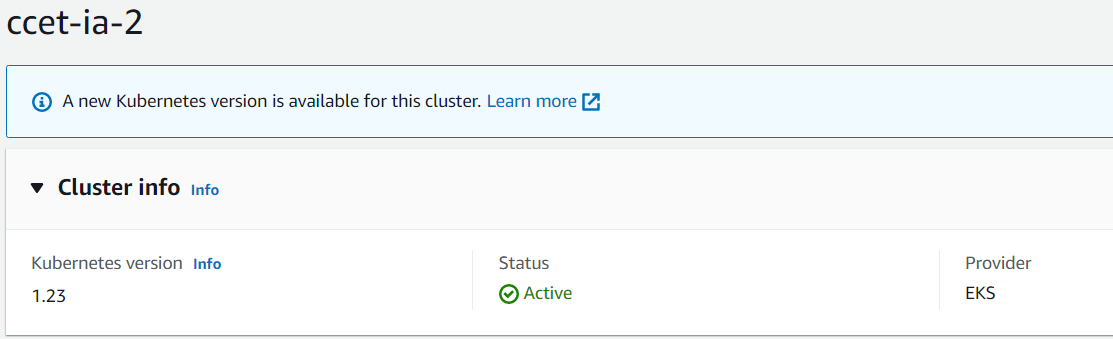


1. Configure the cluster



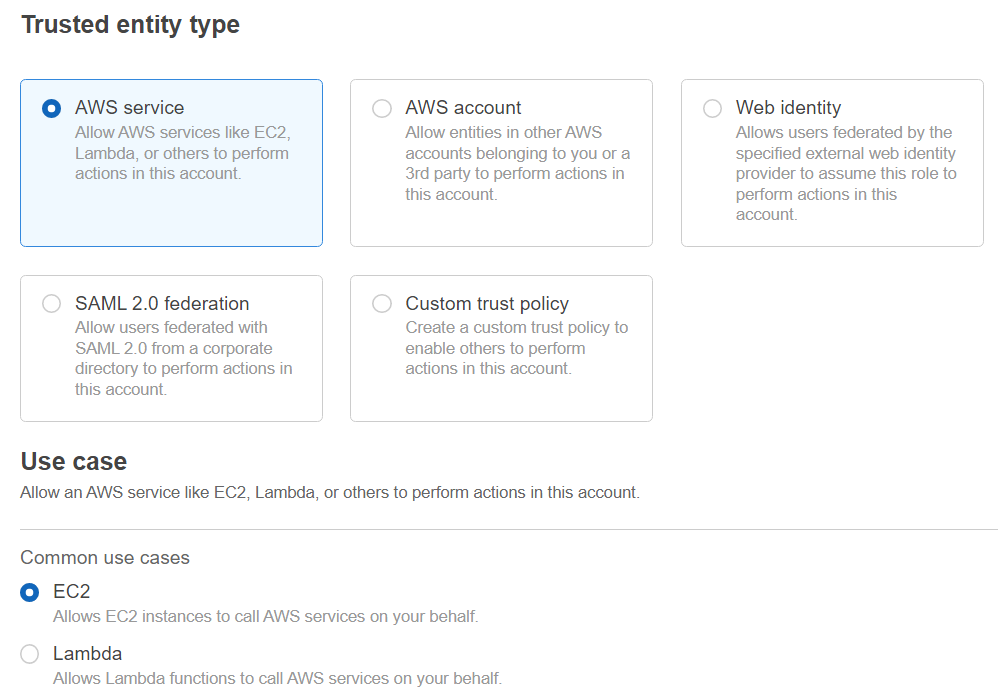


Leave all the rest of the settings as default and create cluster.

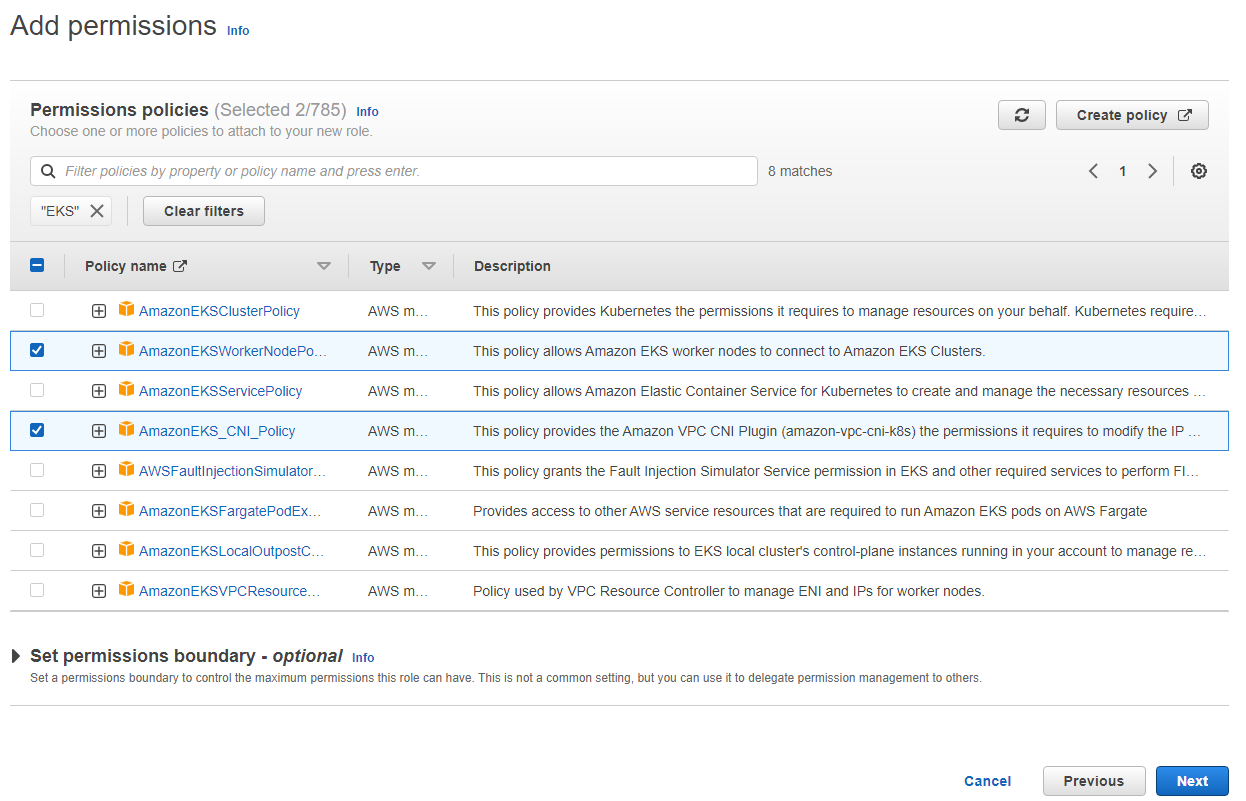


(Again, we will require an IAM Role which has enough permissions required for adding node group.)

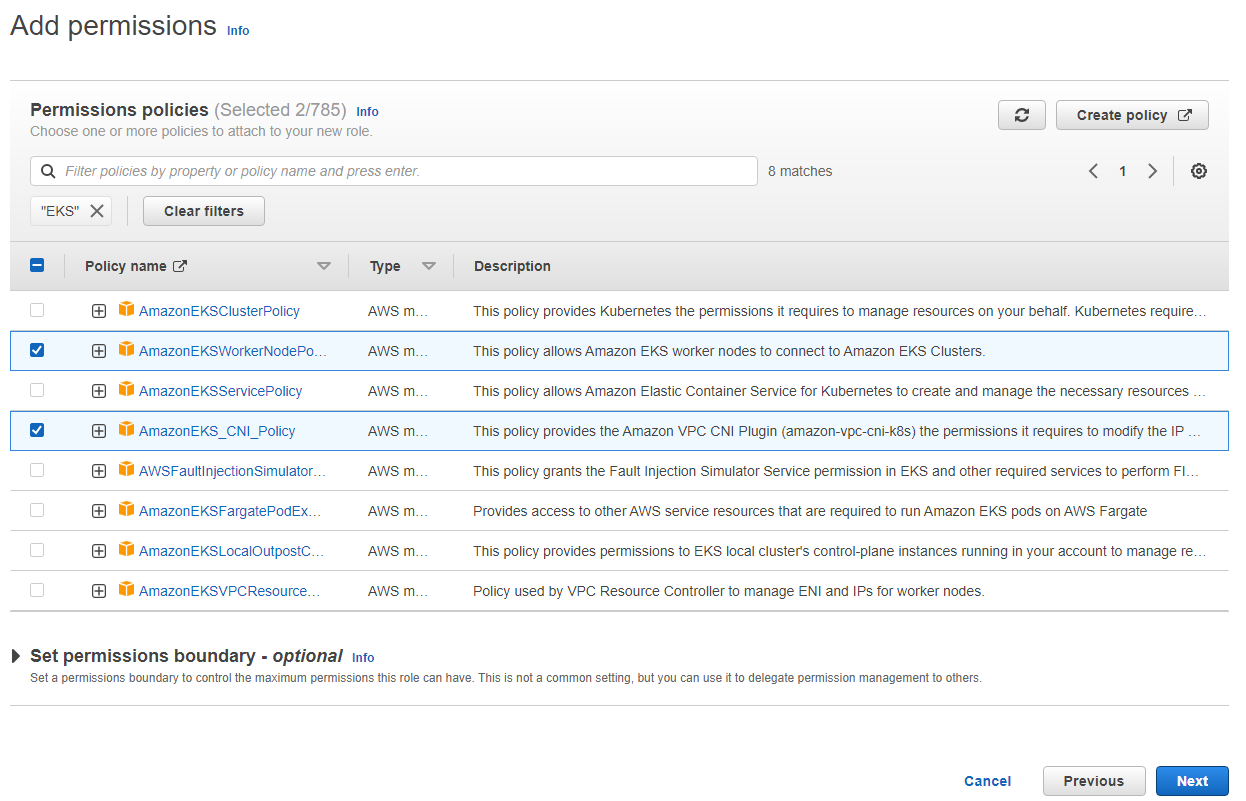
1. Go to IAM 🡪 Roles 🡪 Create Role

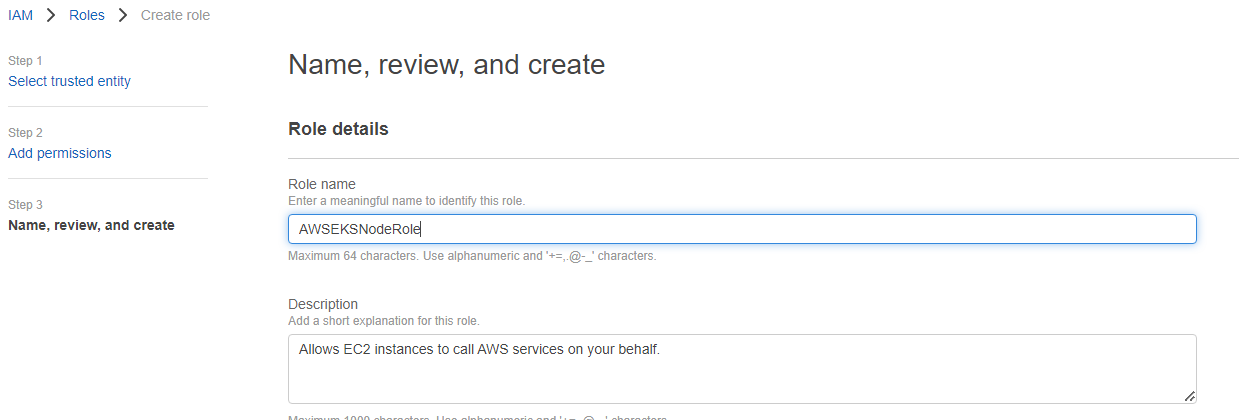


Click Next



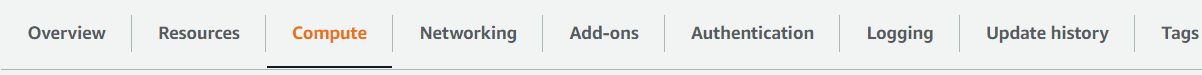


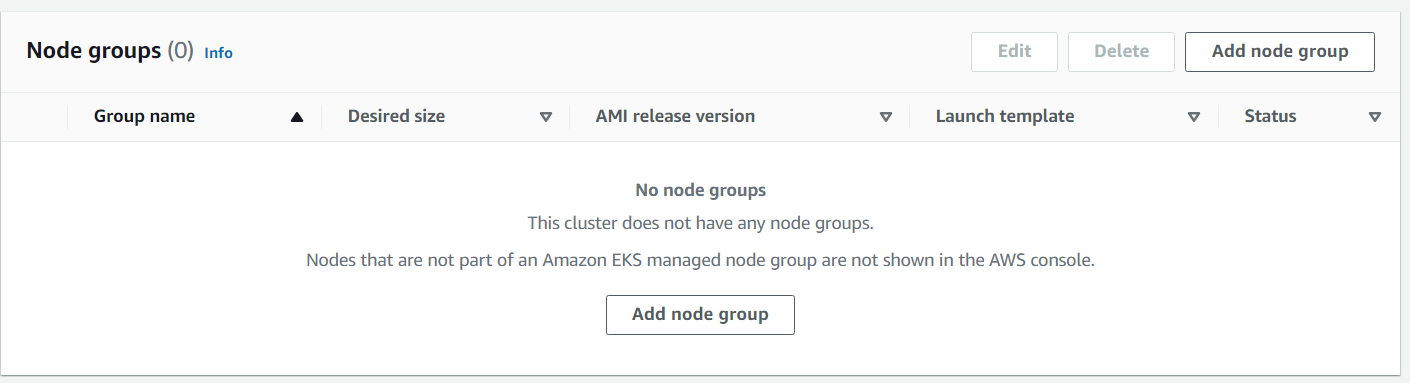


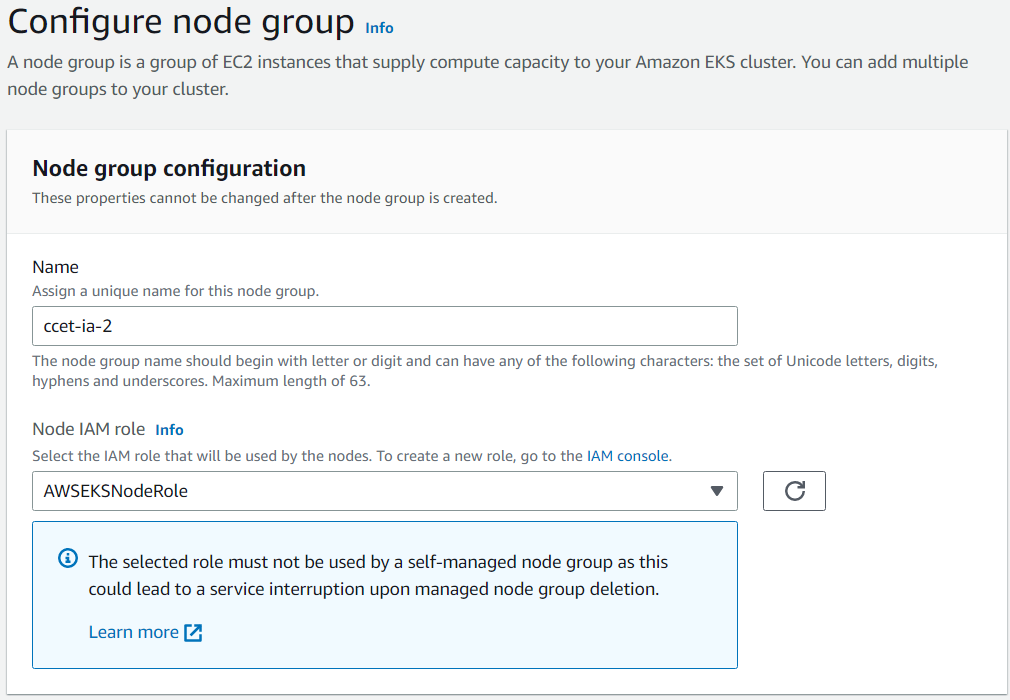


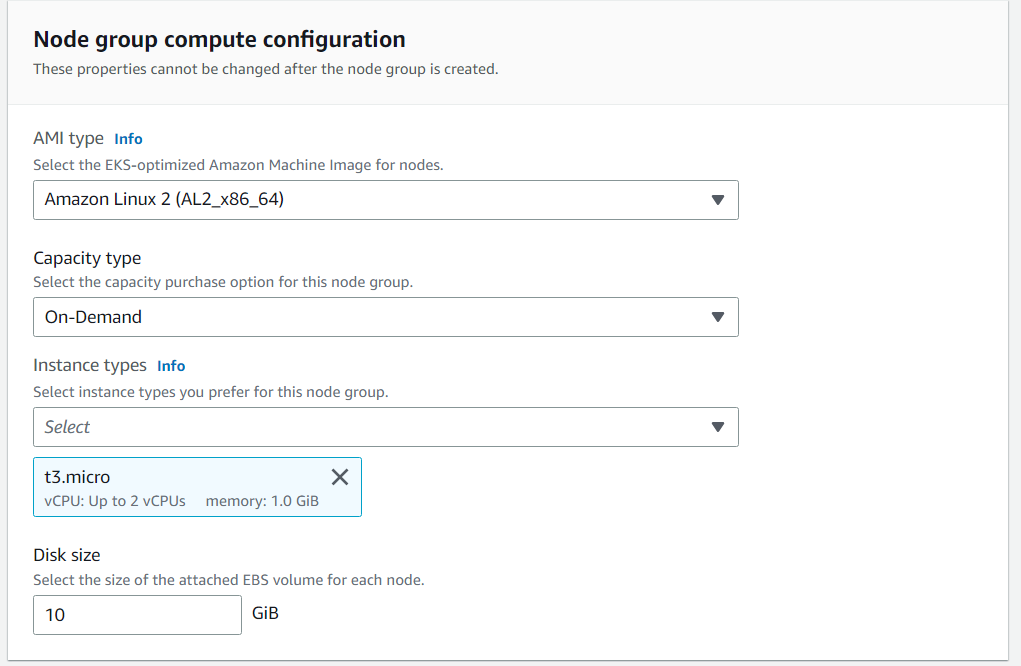
Finally, click on create role.

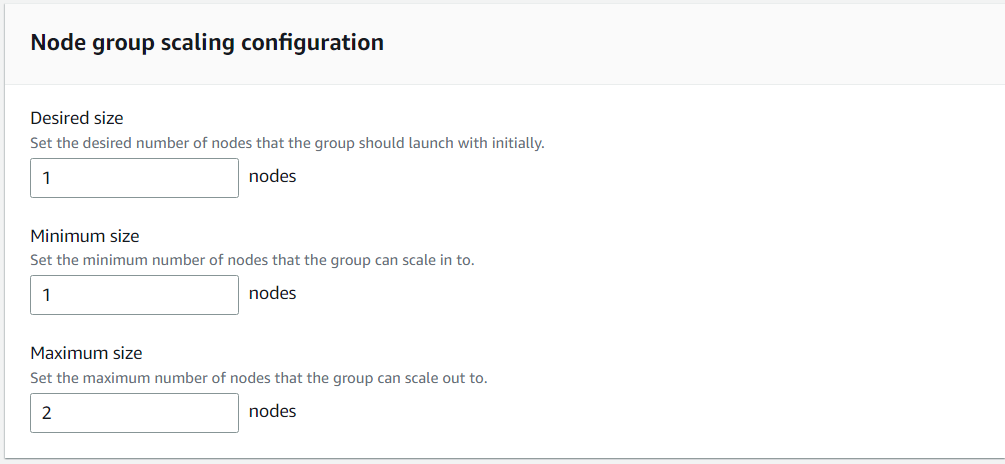
1. Coming back to Kubernetes cluster. Add Node Group.



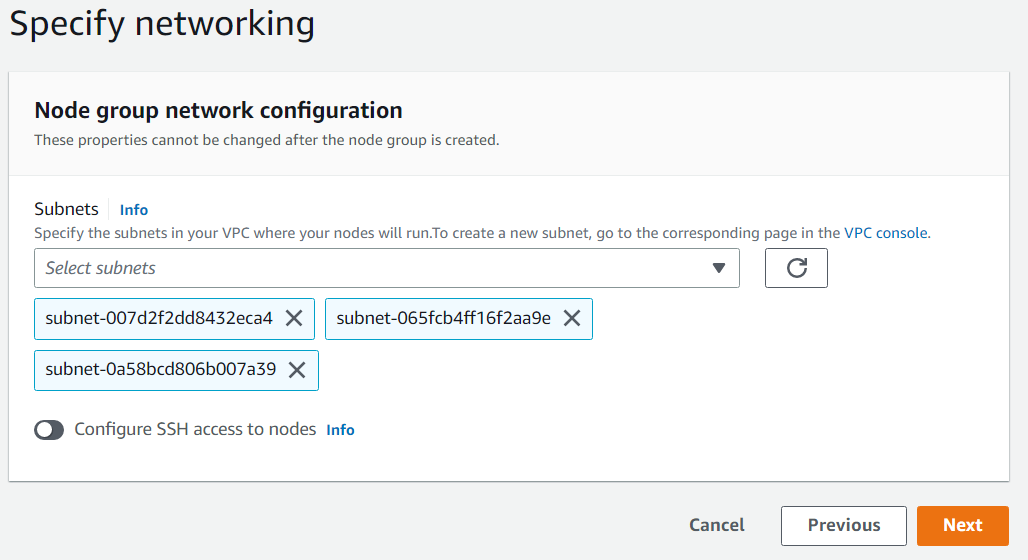




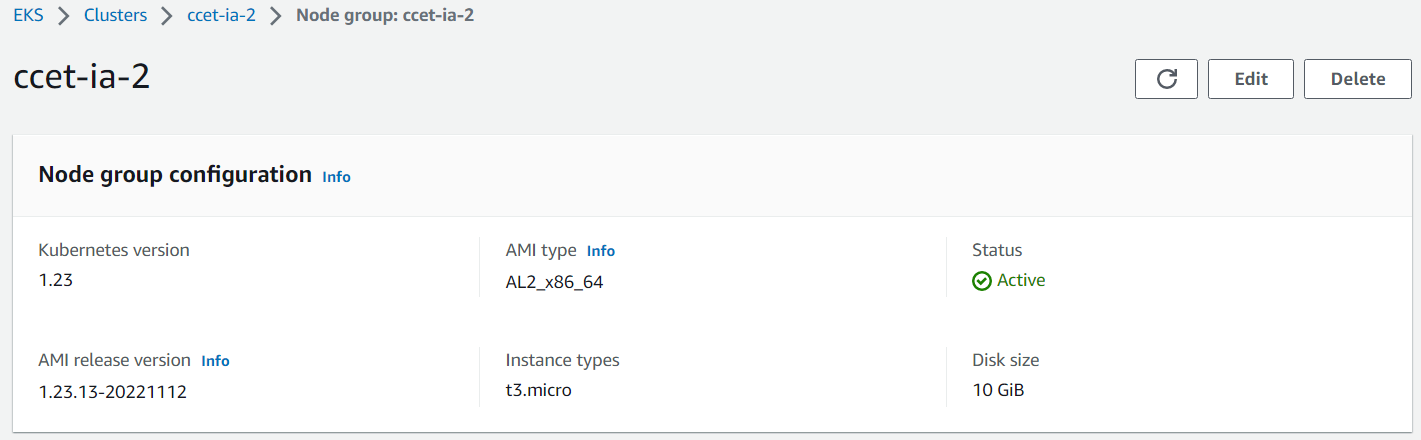


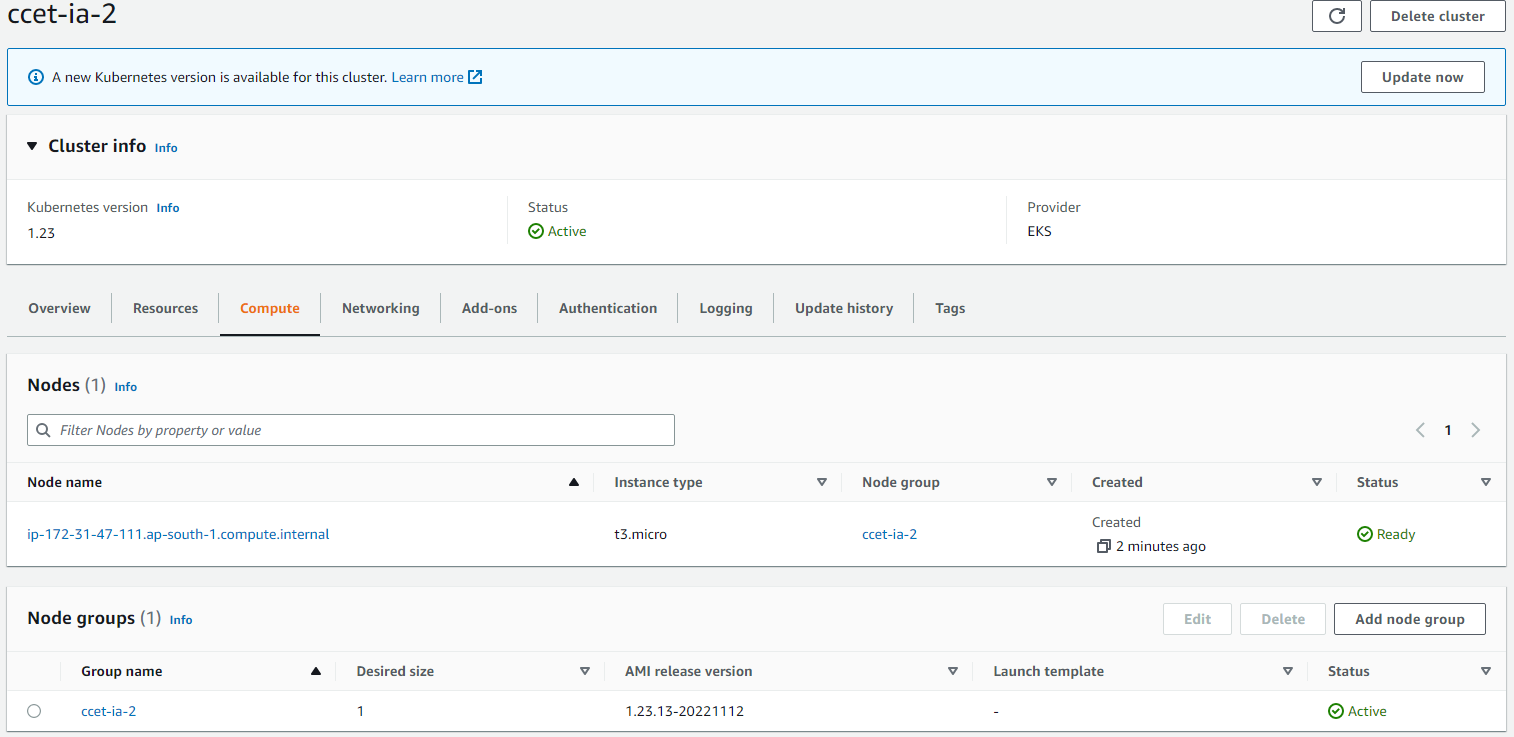






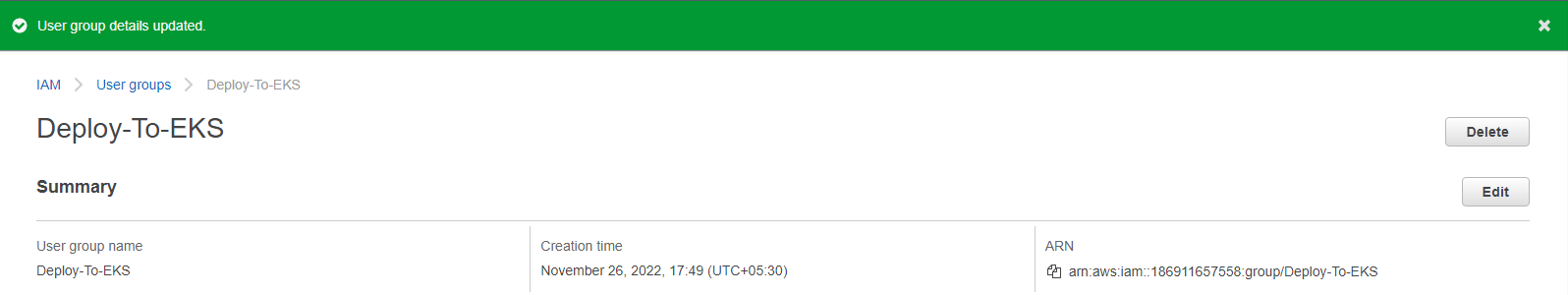
Finally, click on create.





**Part 3: Deploy Container Application to EKS cluster using GitHub Actions.**

1. Edit User group that we created before so that users can have access to EKS as well.



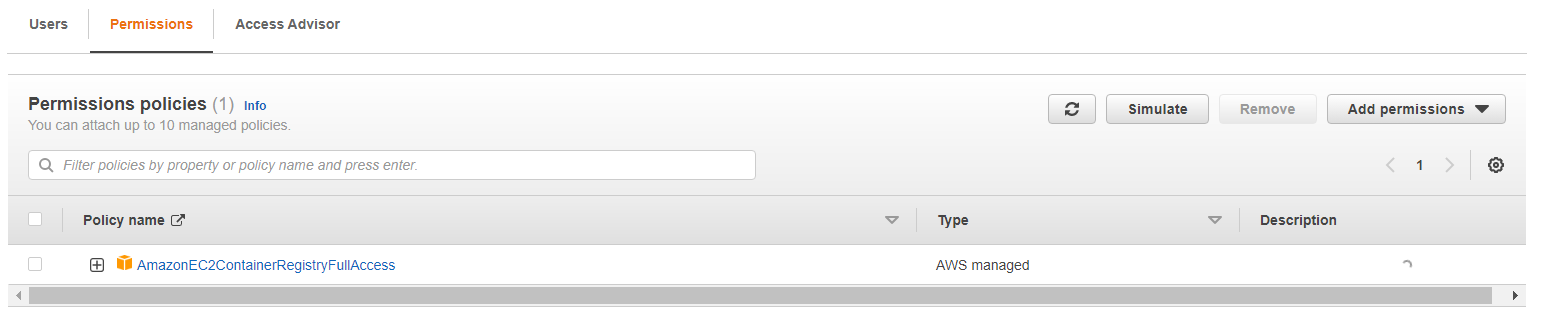
Let’s the change the name to Deploy-To-EKS from Push-Docker-Images-To-ECR, for convenience.

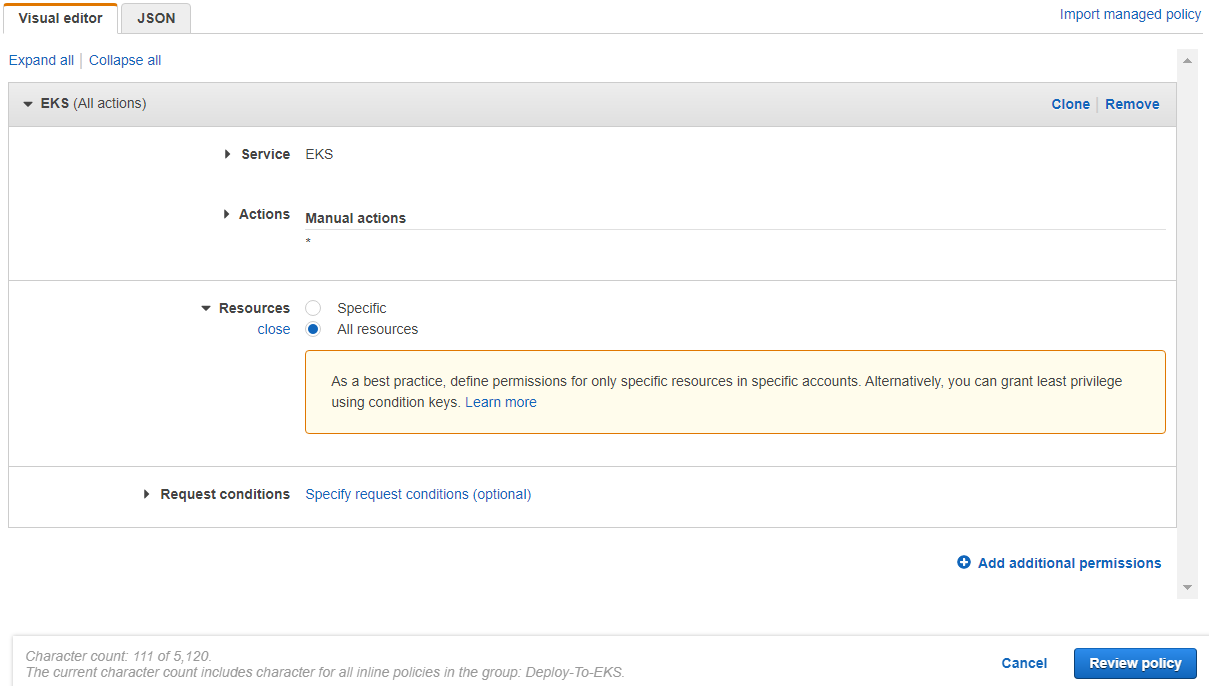
Go to permissions 🡪 add permission 🡪 create inline policy.

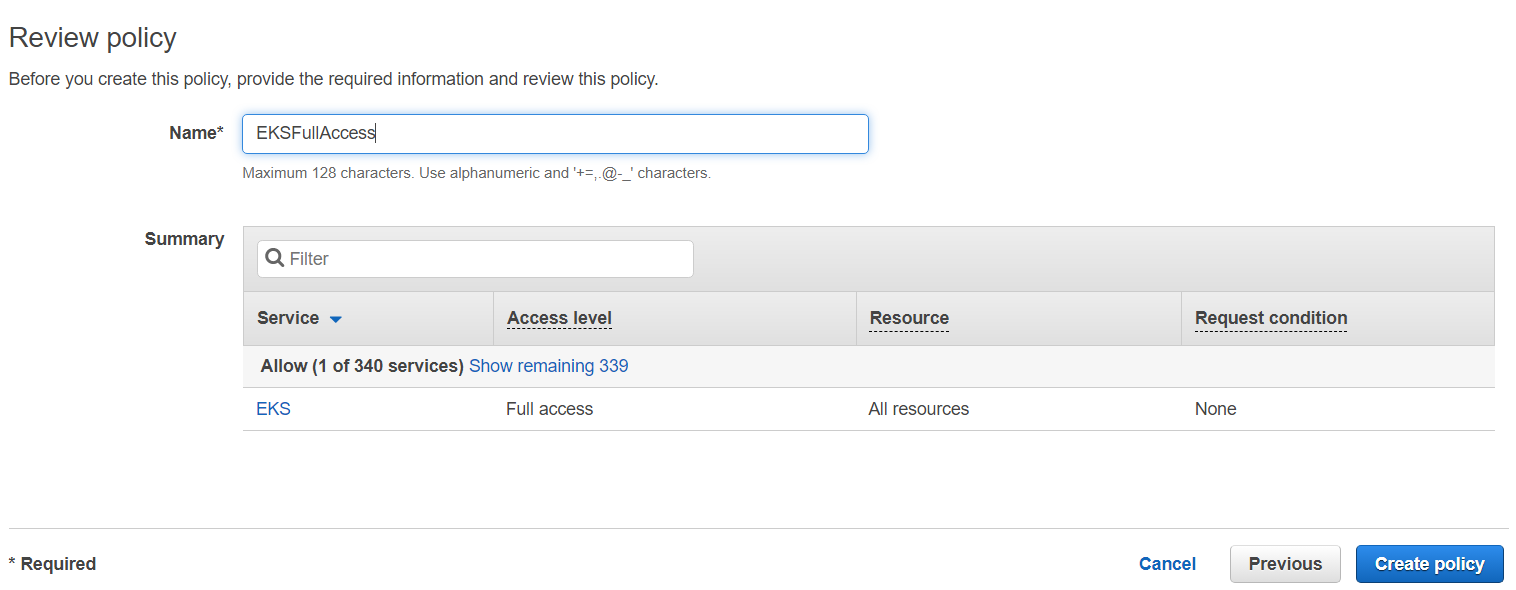
Select EKS as service, and in manual actions select all actions for now.

In Resources select all resources for now.

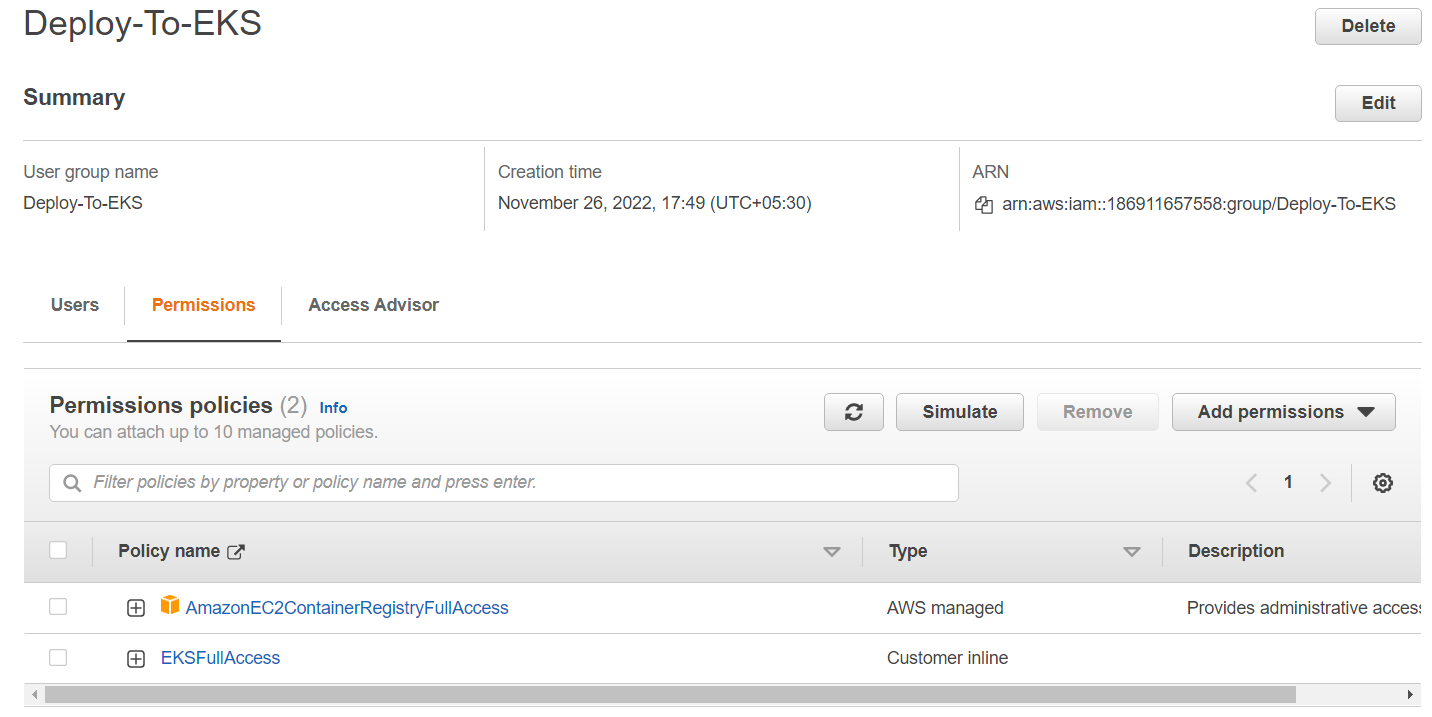
Click on review, name it and create policy.



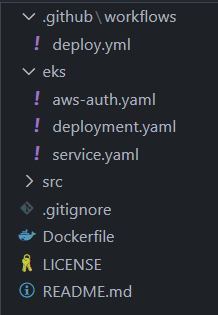




User group looks like this:



1. Let’s edit our workflow file that we create to build docker image out of our app so that it can also deploy it to EKS. Our workflow file will be named as deploy.yml.



Files in the eks folder are required to configure our cluster.

Deploy.yml

name: Deploy docker container to EKS

on:

  push:

    branches: [ main ]

  pull\_request:

    branches: [ main ]

jobs:

  build:

    name: Deploy docker container to EKS

    runs-on: ubuntu-latest

    steps:

      - name: Check out code for building image

        uses: actions/checkout@v1

      - name: Install kubectl

        uses: azure/setup-kubectl@v3

      - 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-south-1

      - name: Login to Amazon ECR

        id: login-ecr

        uses: aws-actions/amazon-ecr-login@v1

      - name: Build, tag, and push docker image to Amazon ECR

        env:

          ECR\_REGISTRY: ${{ steps.login-ecr.outputs.registry }}

          ECR\_REPOSITORY: ccet-ia-2

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

        run: |

          docker build -t $ECR\_REGISTRY/$ECR\_REPOSITORY:$IMAGE\_TAG -t $ECR\_REGISTRY/$ECR\_REPOSITORY:latest .

          docker push -a $ECR\_REGISTRY/$ECR\_REPOSITORY

      - name: Update kube config

        run: aws eks update-kubeconfig --name ccet-ia-2 --region ap-south-1

      - name: Deploy image to Amazon EKS

        run: |

          kubectl apply -f eks/aws-auth.yaml

          kubectl apply -f eks/deployment.yaml

          kubectl apply -f eks/service.yaml

aws-auth.yaml

apiVersion: v1

kind: ConfigMap

metadata:

  name: aws-auth

  namespace: kube-system

data:

  mapUsers: |

    - userarn: arn:aws:iam::186911657558:user/github-actions

      username: github-actions

      groups:

        - system:masters

deployment.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

  name: t-space-deployment

  labels:

    app: t-space

spec:

  replicas: 5

  selector:

    matchLabels:

      app: t-space

  strategy:

    type: RollingUpdate

  template:

    metadata:

      labels:

        app: t-space

    spec:

      containers:

      — name: t-space

        image: 186911657558.dkr.ecr.ap-south-1.amazonaws.com/ccet-ia-2:latest

        imagePullPolicy: Always

        ports:

        — containerPort: 5000

service.yaml

apiVersion: v1

kind: Service

metadata:

  name: t-space-service

spec:

  selector:

    app: t-space

  ports:

  - protocol: TCP

    port: 80

    targetPort: 5000

1. Now we first need to apply config in aws-auth.yaml to our eks cluster, doing this will give the computer node in github actions access to EKS.

To do that install kubectl and aws cli on your computer.

In your command prompt, type ‘aws configure’ to set your AWS\_ACCESS\_KEY\_ID and AWS\_SECRET\_ACCESS\_KEY which will help us to login to AWS.

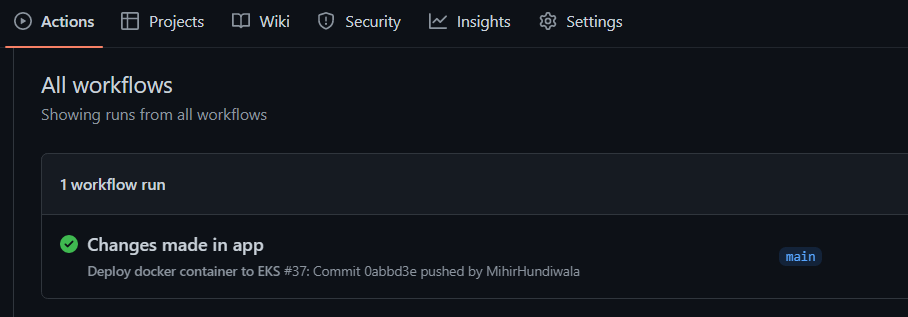
Then, update the kubectl config by typing the below command:

‘aws eks update-kubeconfig --name ccet-ia-2 --region ap-south-1’

Then, copy the aws-auth.yaml file into your current working directory and type the command: ‘kubectl apply -f aws-auth.yaml’

Since in aws-auth.yaml, we have mentioned ‘github-actions’ in username, this will give that user the permission to access aws eks resources.

1. Commit and push these files to GitHub. From now on, every change in code to master branch will automatically create docker image push it to ECR and will deploy the image to EKS.



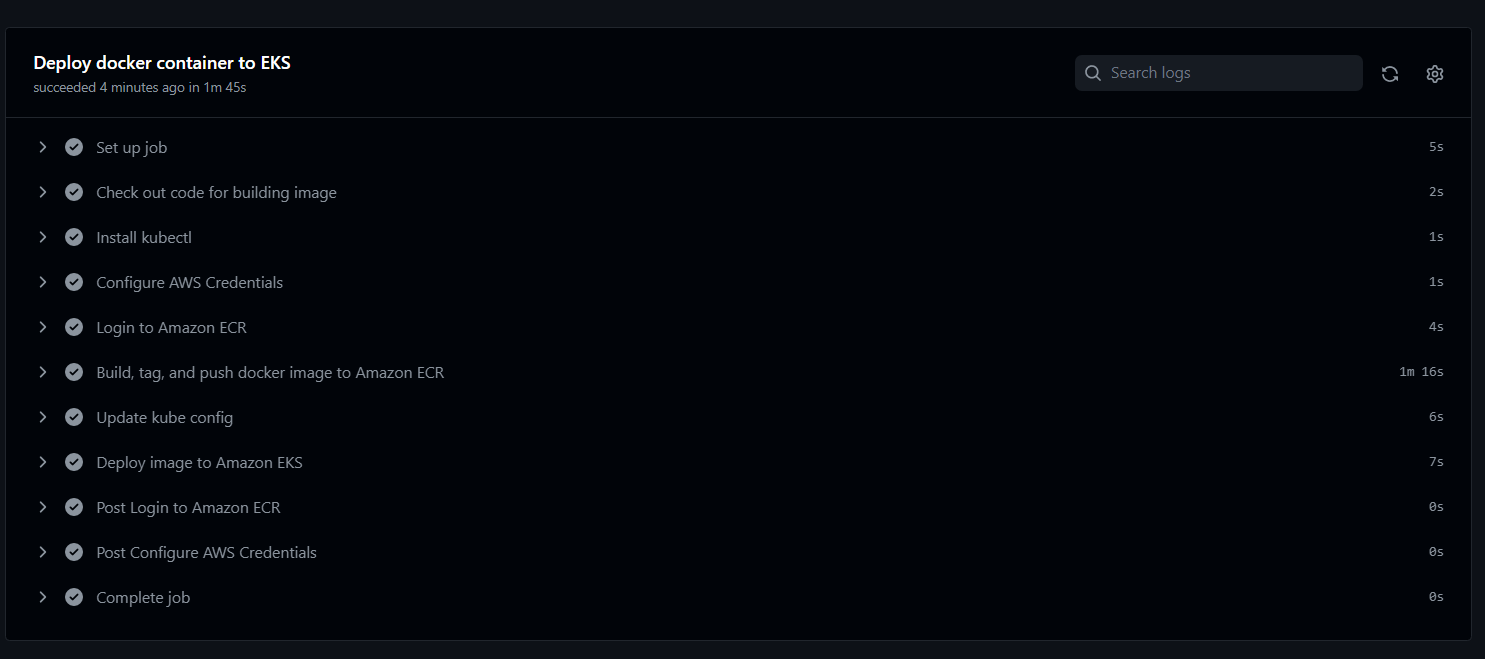
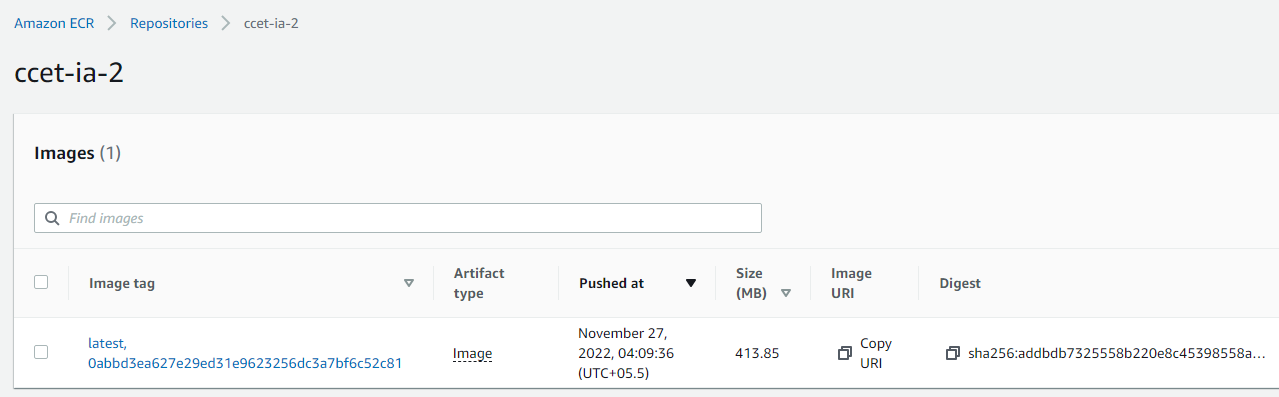
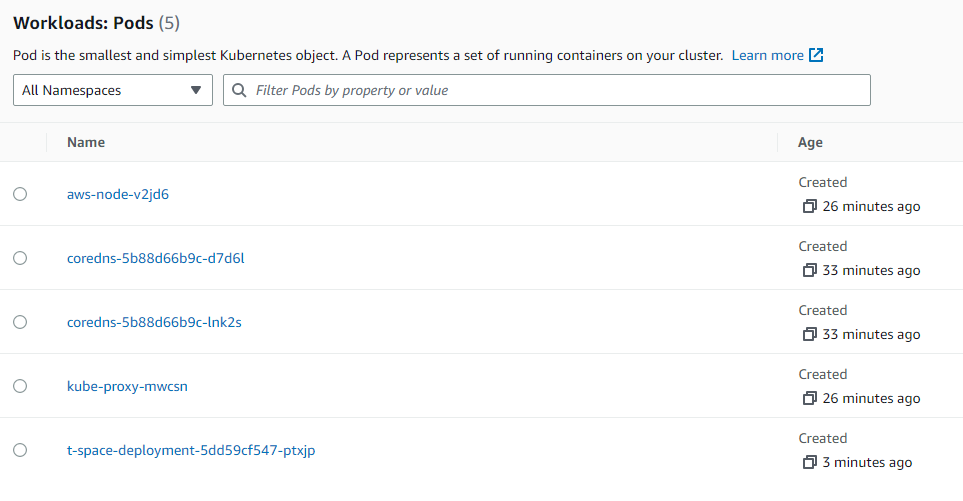


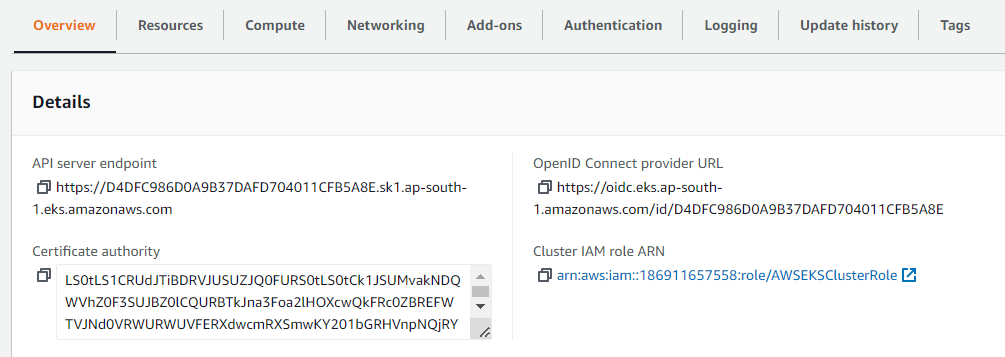
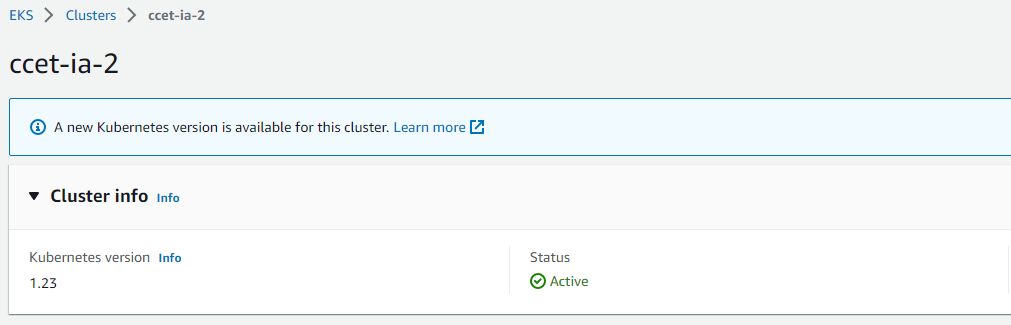
Image pushed to ECR



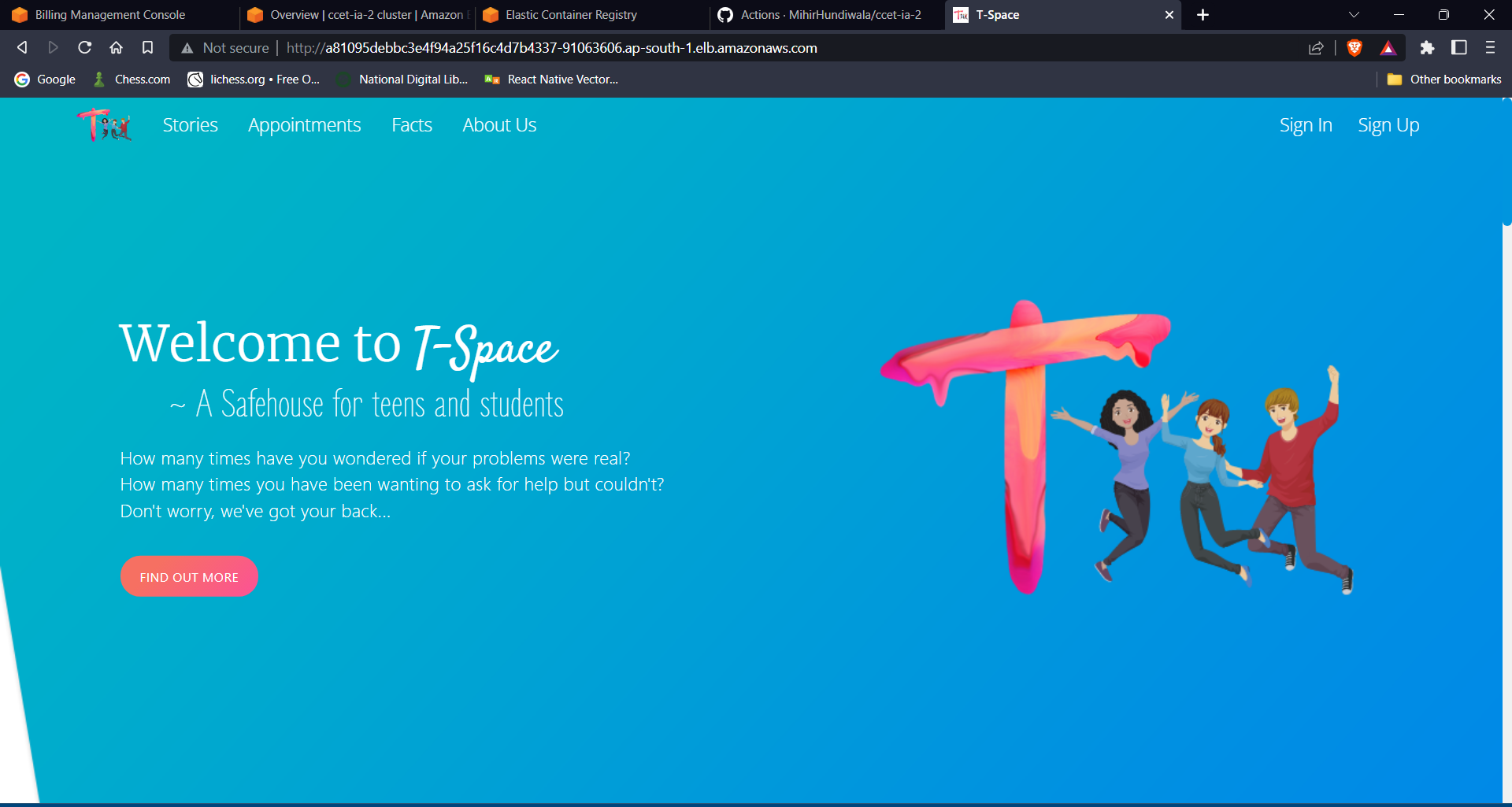
Pods created



1. Let’s go to the external Ip to see if our website loads or not.



1. Paste the Endpoint in browser.



**Congratulations the app is deployed!**