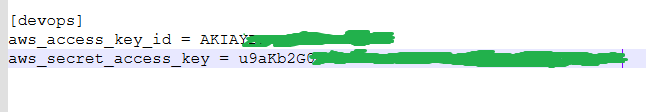
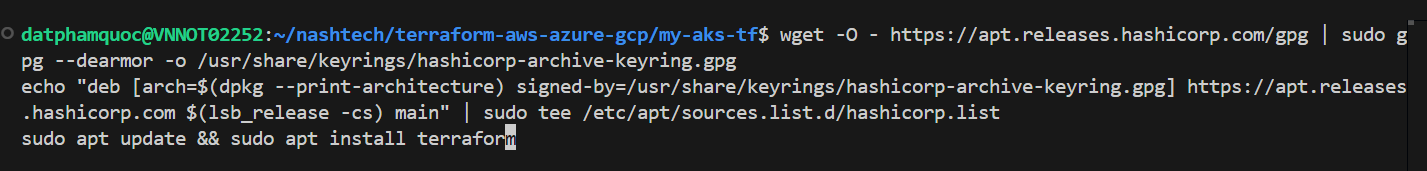
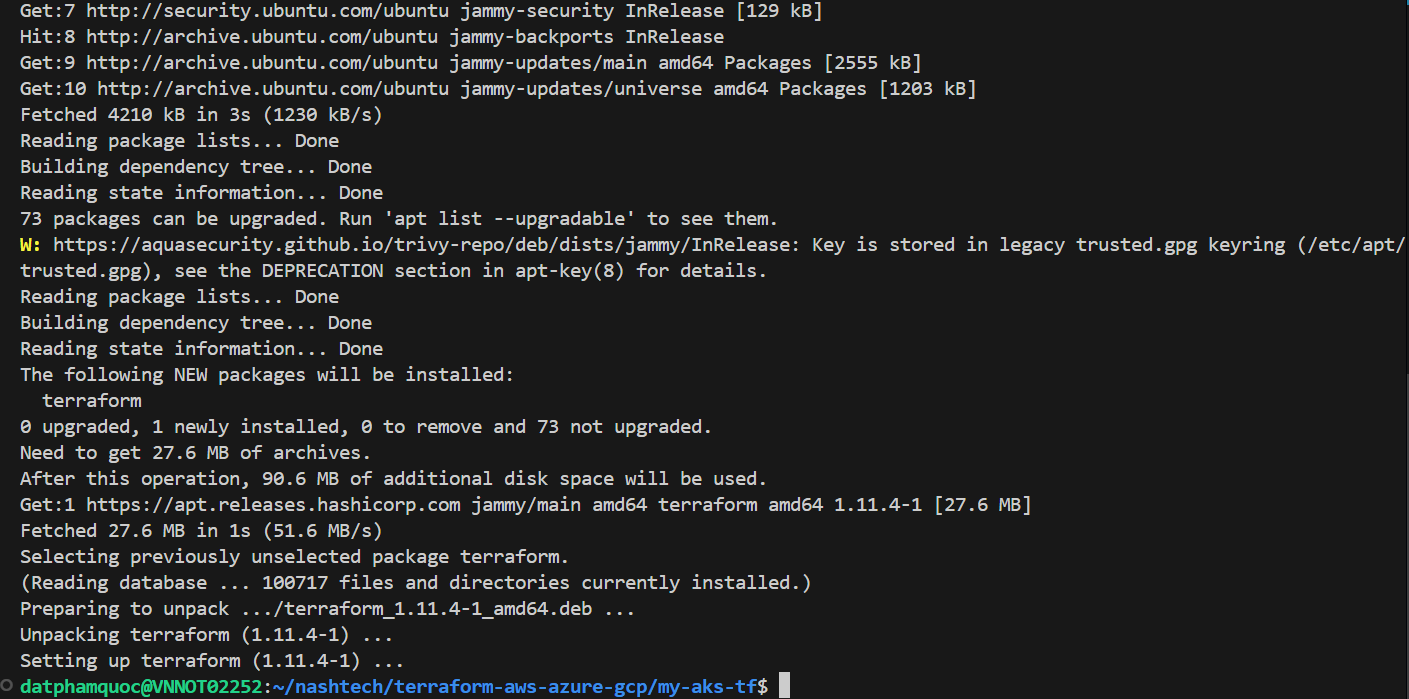
# Setup credentials

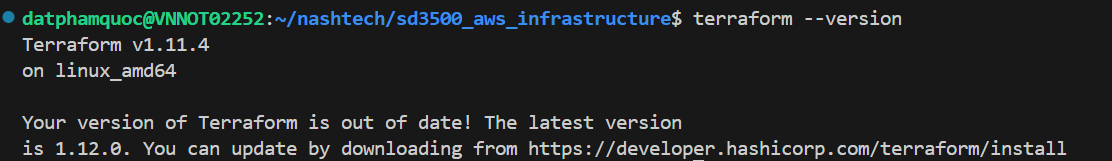


# Provision EKS using Terraform

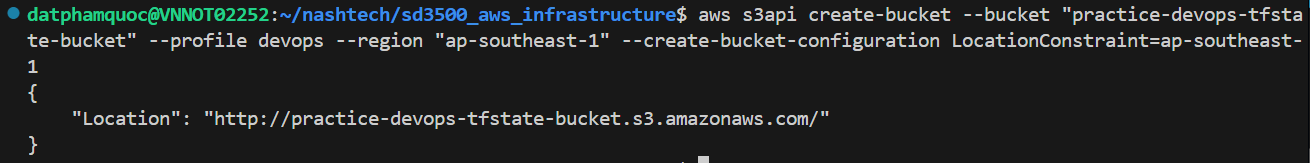
# install terraform

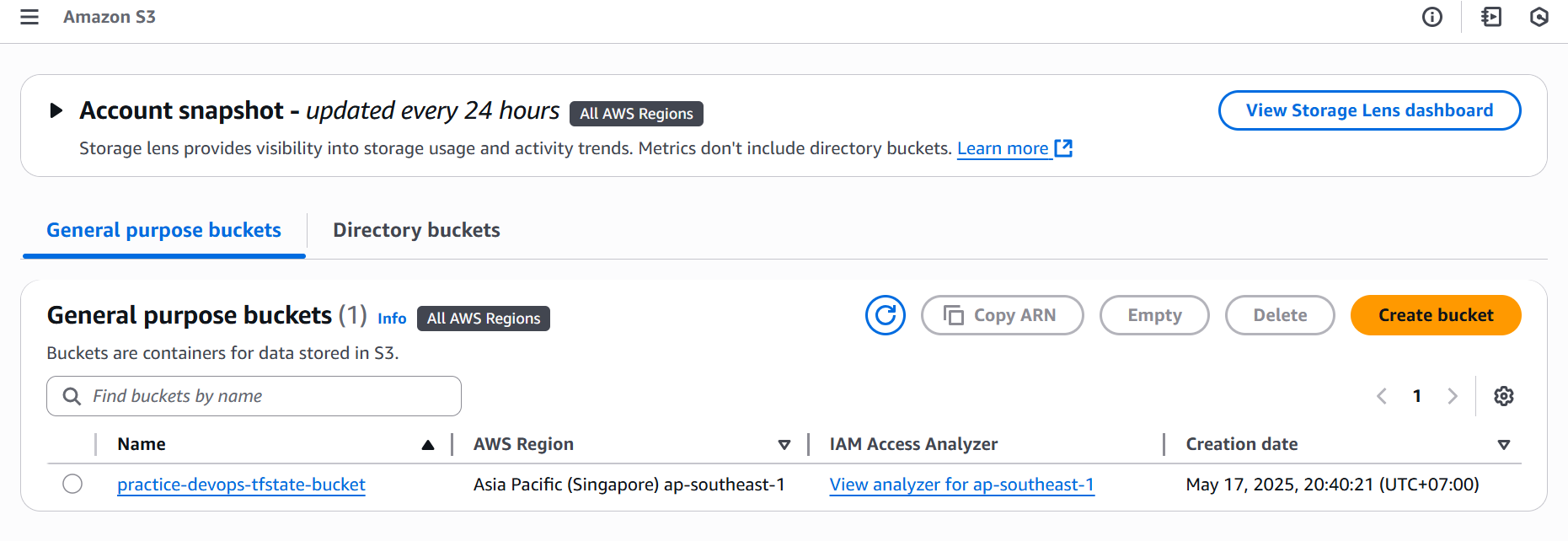




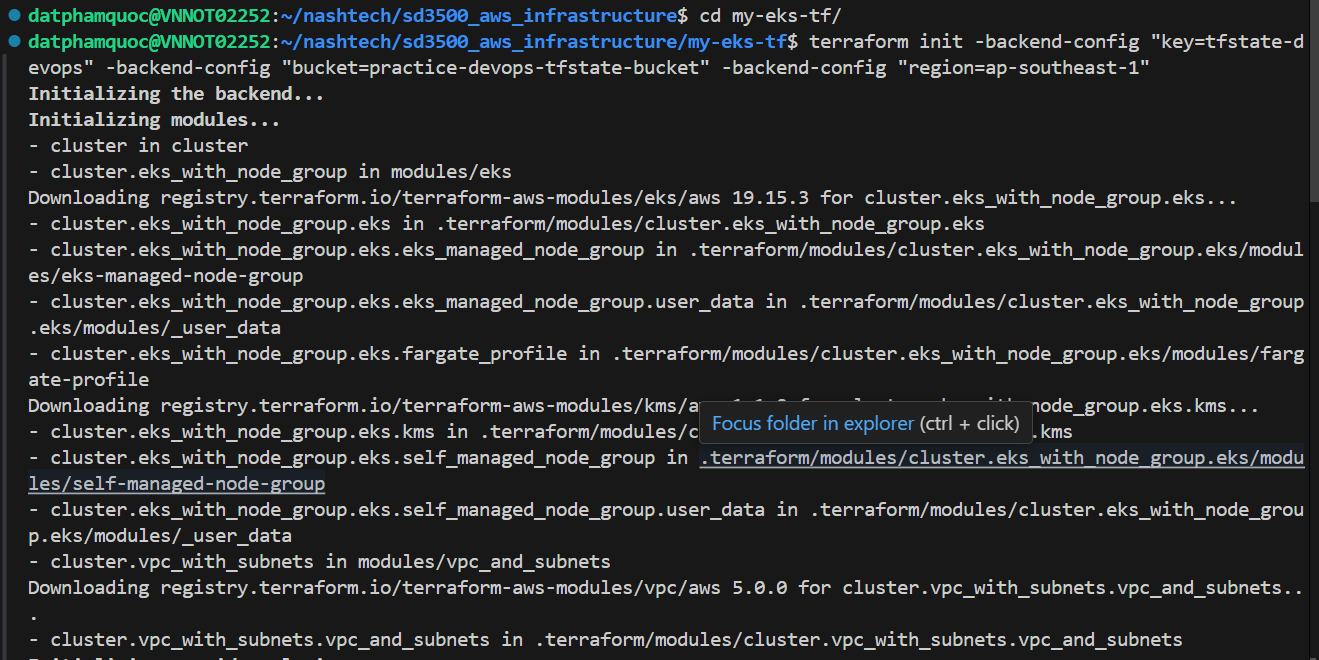
# verify terraform version  


# # create s3 bucket (practice-devops-tfstate-bucket ) to store the tfstate file

aws s3api create-bucket --bucket "practice-devops-tfstate-bucket" --profile devops --region "ap-southeast-1" --create-bucket-configuration LocationConstraint=ap-southeast-1  


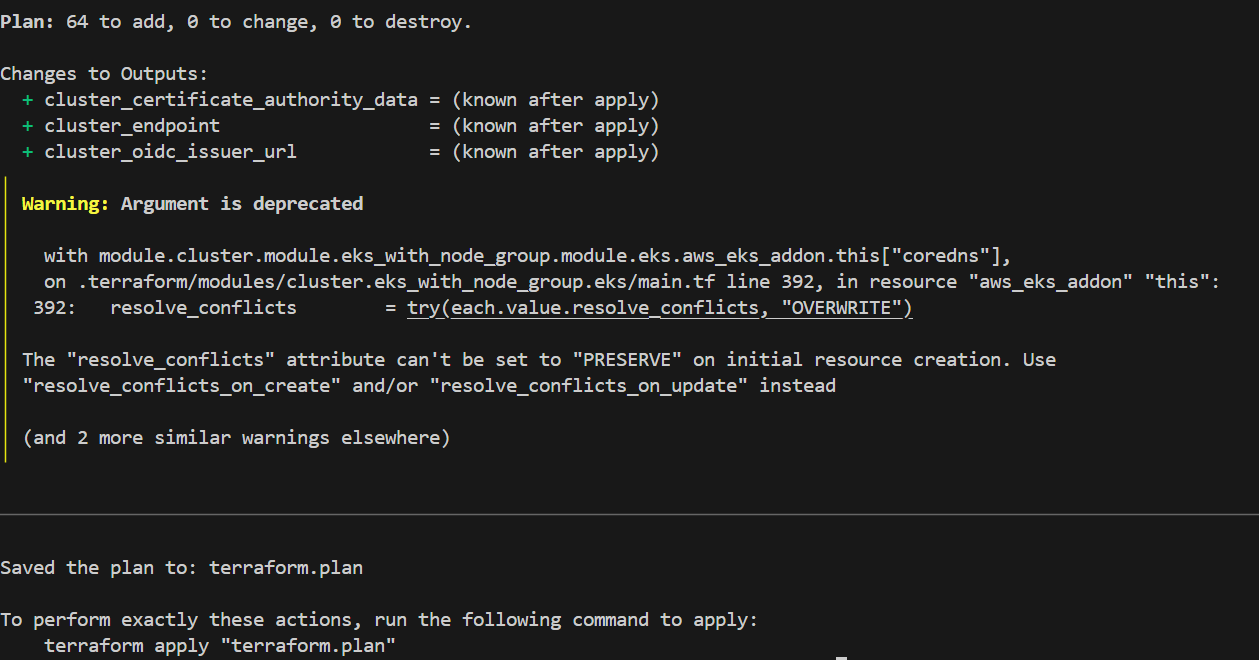


# # Initialize the terraform module with configure S3 bucket (this just configures Terraform to use that bucket as the backend for storing the state. But it does not upload anything unless there's actual state to manage.)

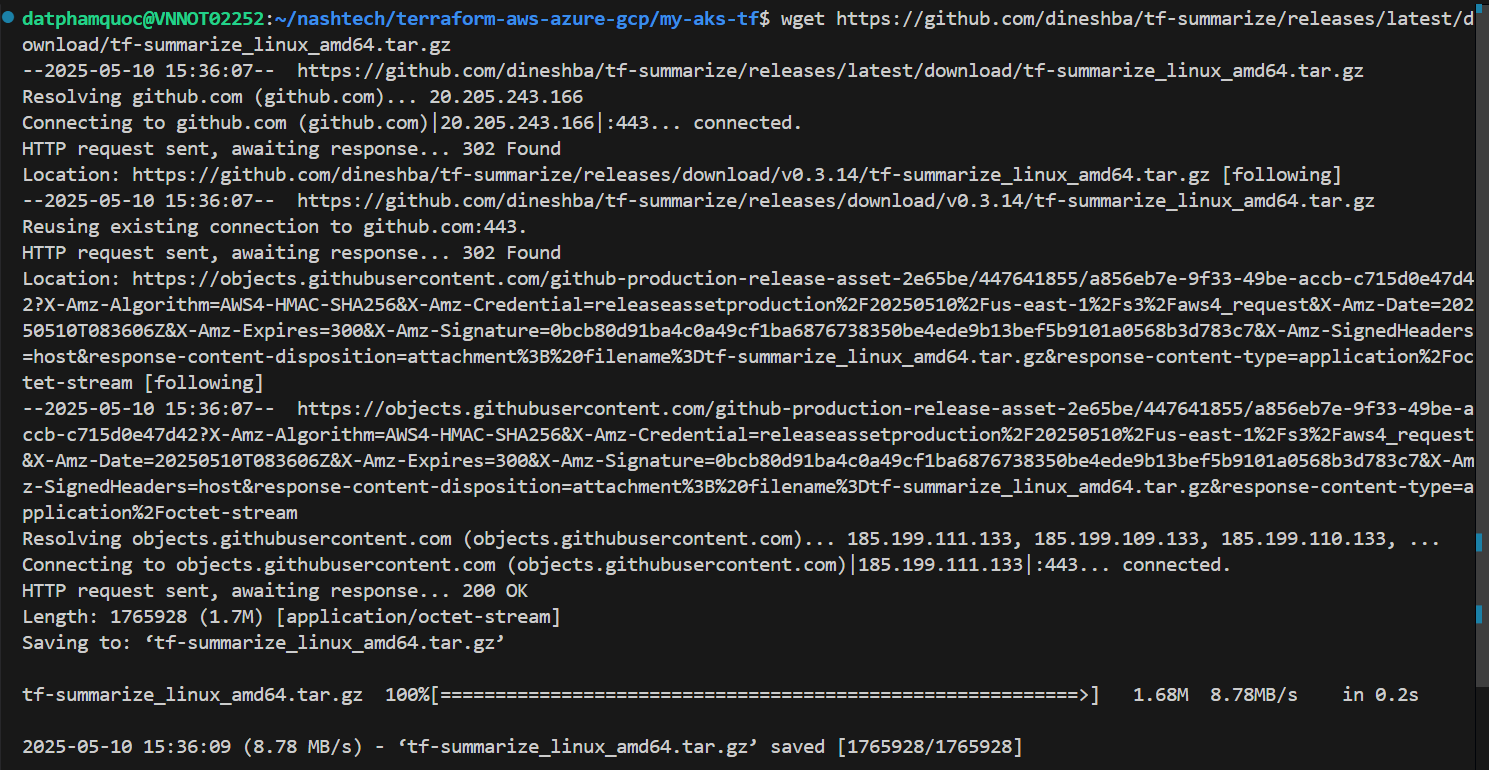
terraform init -backend-config "key=tfstate-devops" -backend-config "bucket=practice-devops-tfstate-bucket" -backend-config "region=ap-southeast-1"  


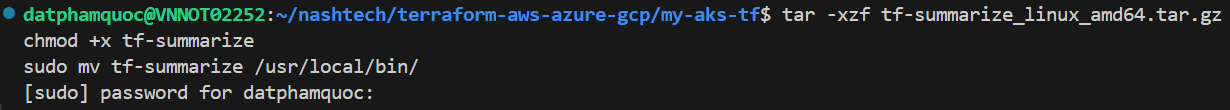
# # view terraform plan

AWS\_PROFILE=devops terraform plan -var-file="sample.tfvars" --out terraform.plan



# Install tf-summarize

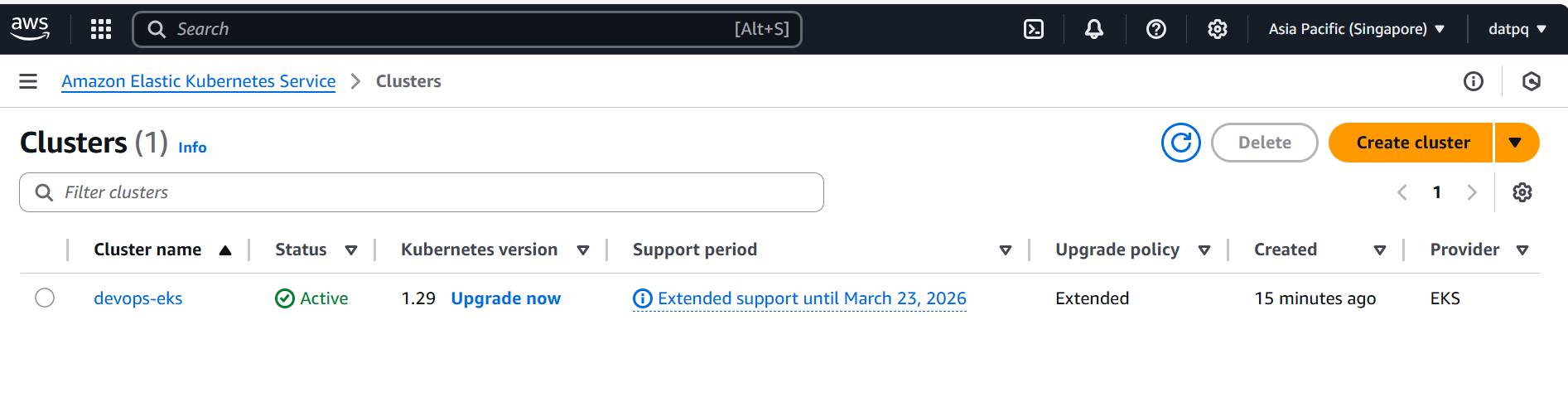
wget <https://github.com/dineshba/tf-summarize/releases/latest/download/tf-summarize_linux_amd64.tar.gz>  


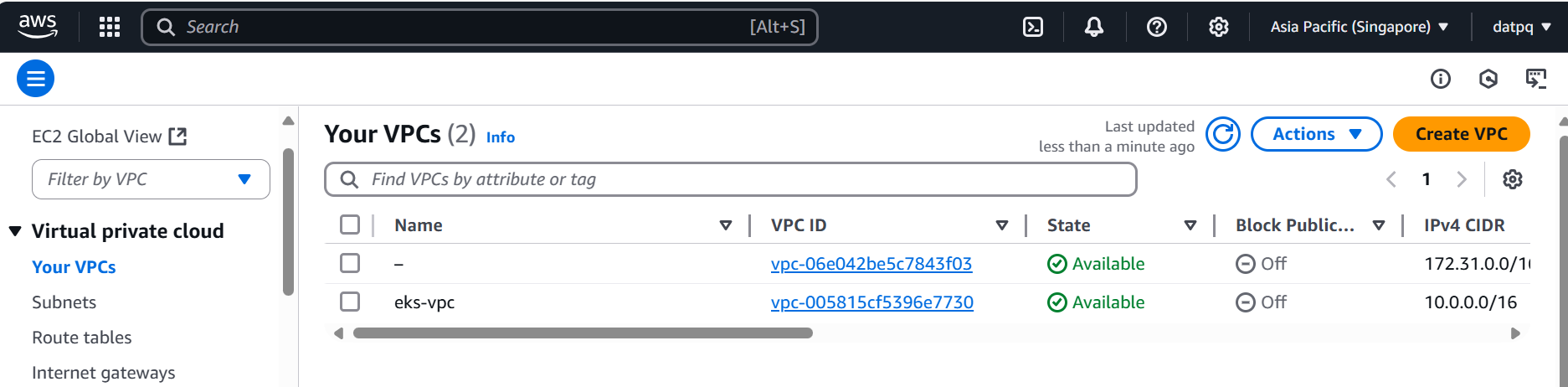
# unpack and install  


# Verify tf-summarize  


# view summary of terraform plan  

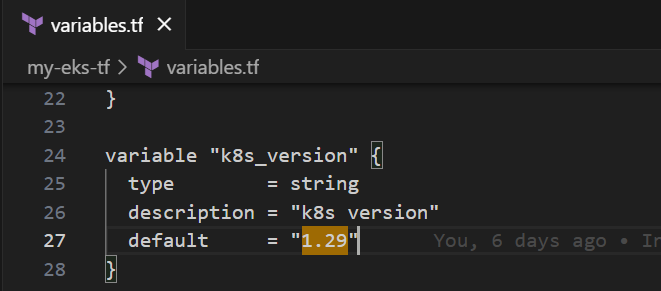

# apply terraform plan to EKS  
AWS\_PROFILE=devops terraform apply -var-file="sample.tfvars"





# Install kubectl

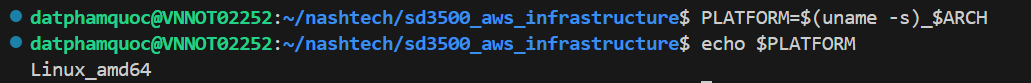
# download the kubectl binary for your cluster’s Kubernetes version (EKS is configured to use version 1.29)



ARCH=amd64



PLATFORM=$(uname -s)\_$ARCH



curl -sLO "https://github.com/eksctl-io/eksctl/releases/latest/download/eksctl\_$PLATFORM.tar.gz"

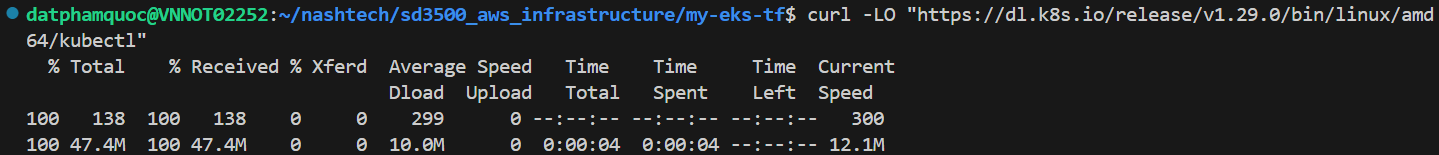


tar -xzf eksctl\_$PLATFORM.tar.gz -C /tmp && rm eksctl\_$PLATFORM.tar.gz



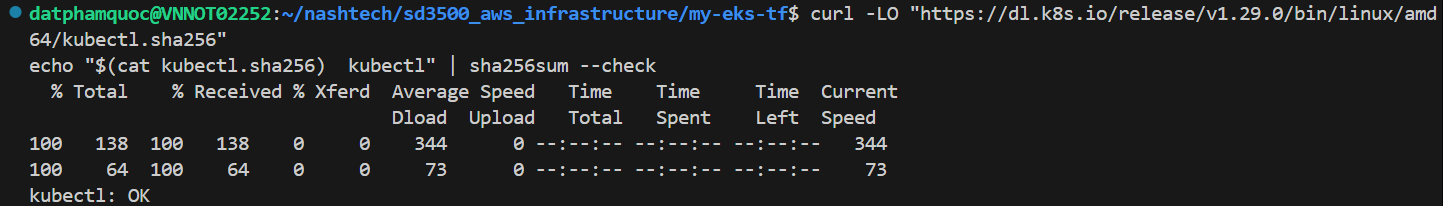
sudo install -m 0755 /tmp/eksctl /usr/local/bin && rm /tmp/eksctl



curl -LO "https://dl.k8s.io/release/v1.29.0/bin/linux/amd64/kubectl"  


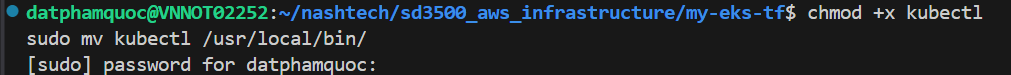
# verify the binary (optional but recommended):

curl -LO <https://dl.k8s.io/release/v1.29.0/bin/linux/amd64/kubectl.sha256>  
echo "$(cat kubectl.sha256) kubectl" | sha256sum –check



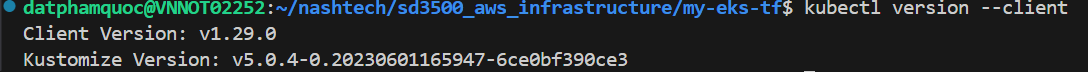
# install the binary:

chmod +x kubectl  
sudo mv kubectl /usr/local/bin/



# verify the version:

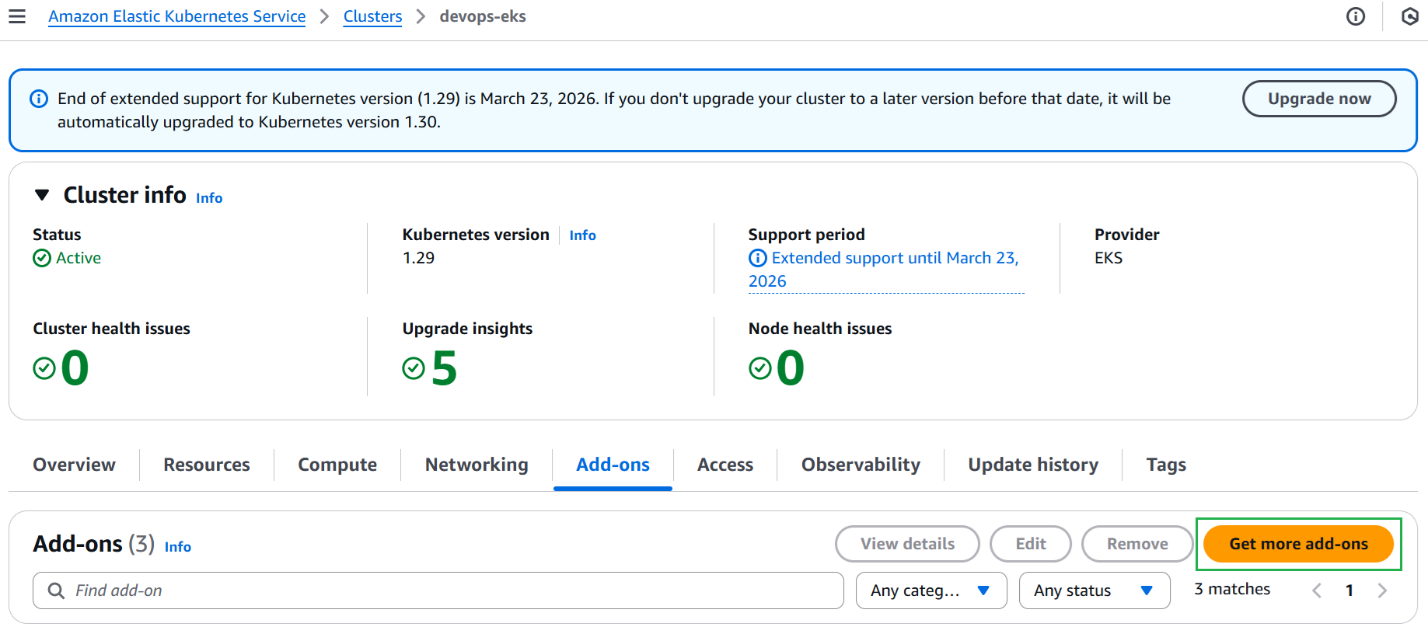
kubectl version --client



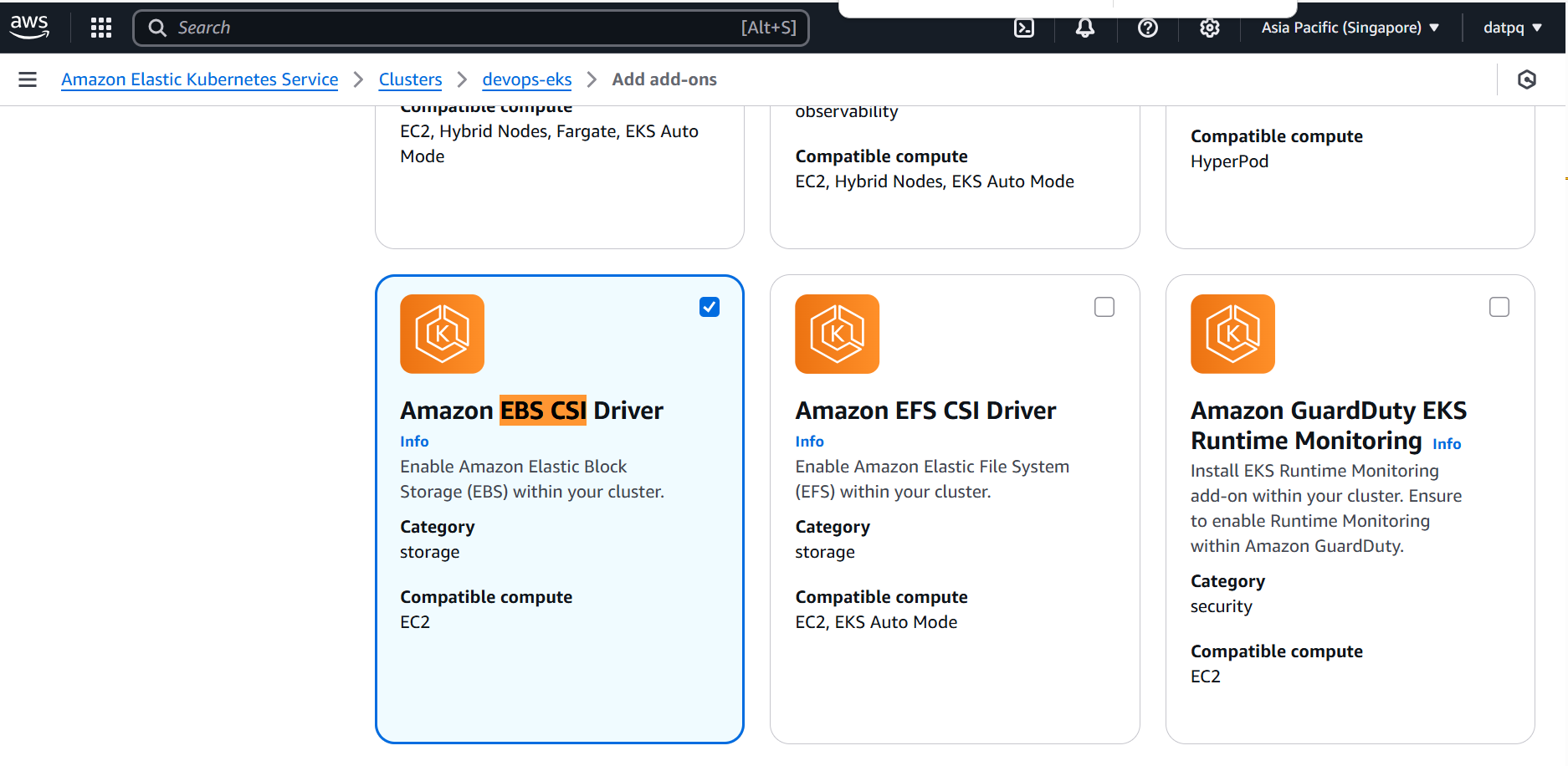
# Create application in created cluster by command line

Note: This is practice sample, in production, we will use CD for deploying

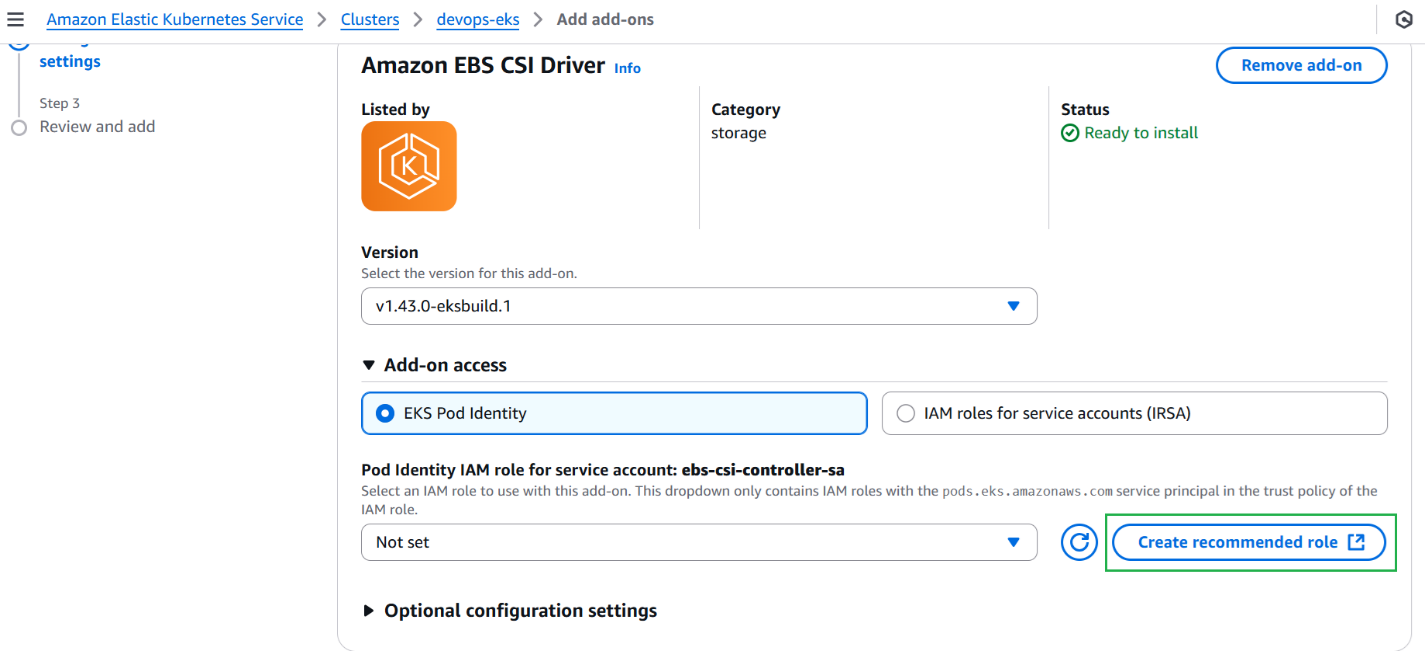
Firstly, we need setup EBS CSI driver in created cluster   
[Store Kubernetes volumes with Amazon EBS - Amazon EKS](https://docs.aws.amazon.com/eks/latest/userguide/ebs-csi.html#managing-ebs-csi)  
[Create an Amazon EKS add-on - Amazon EKS](https://docs.aws.amazon.com/eks/latest/userguide/creating-an-add-on.html)  
For simple, I will install EBS CSI via AWS console  
Navigate to created cluster => Get more add-on



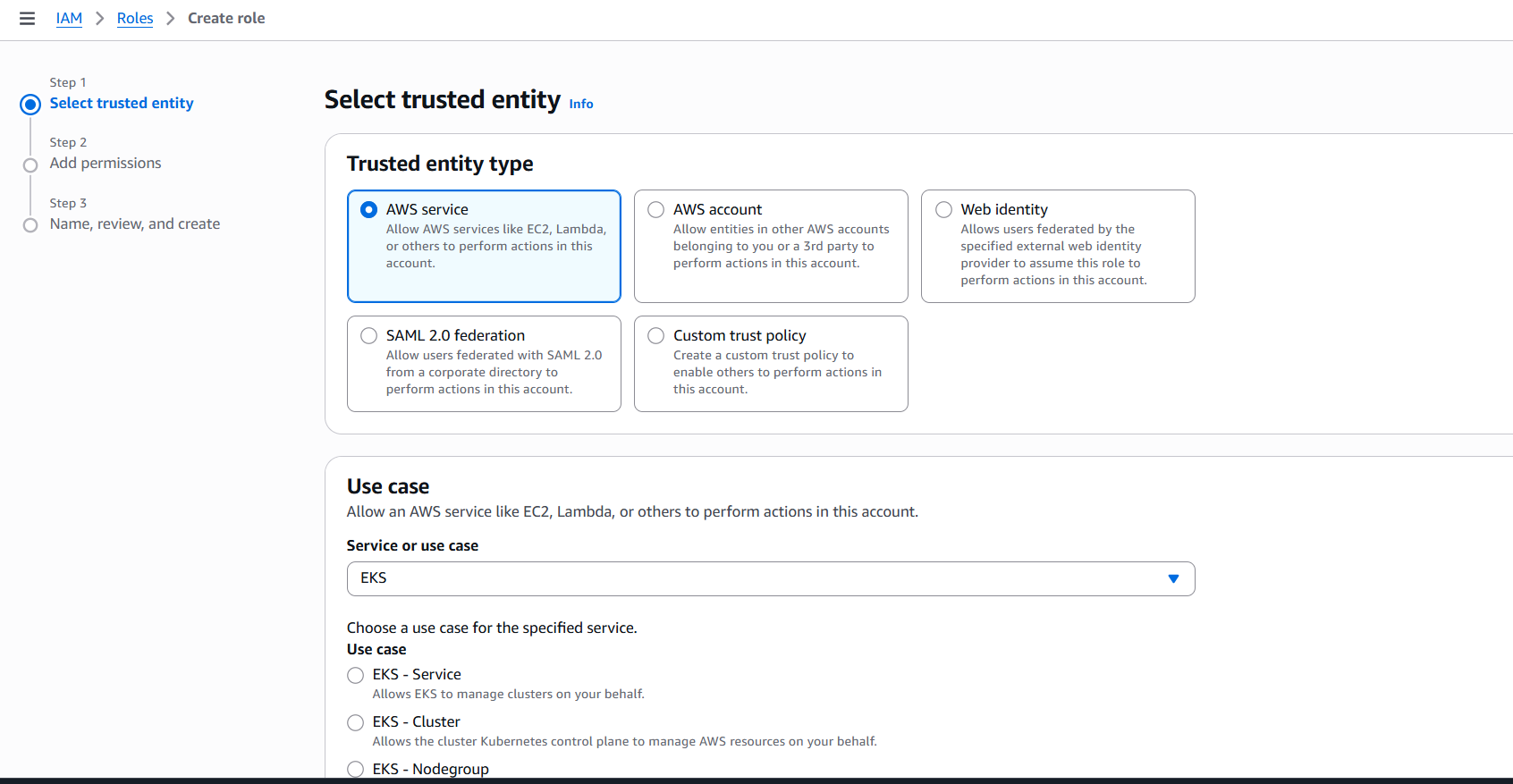
# choose Amazon EBS CSI Driver => Next

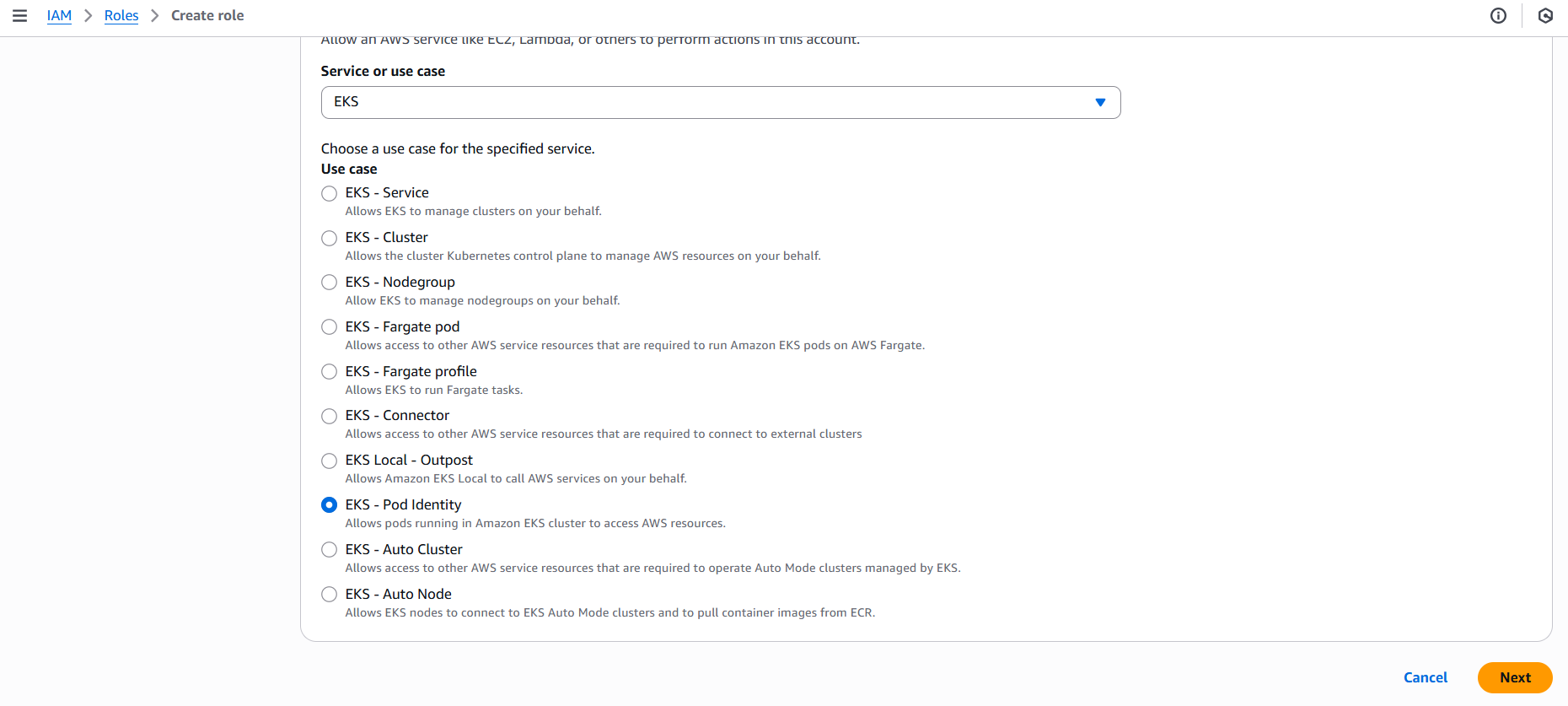


# choose Create recommended role



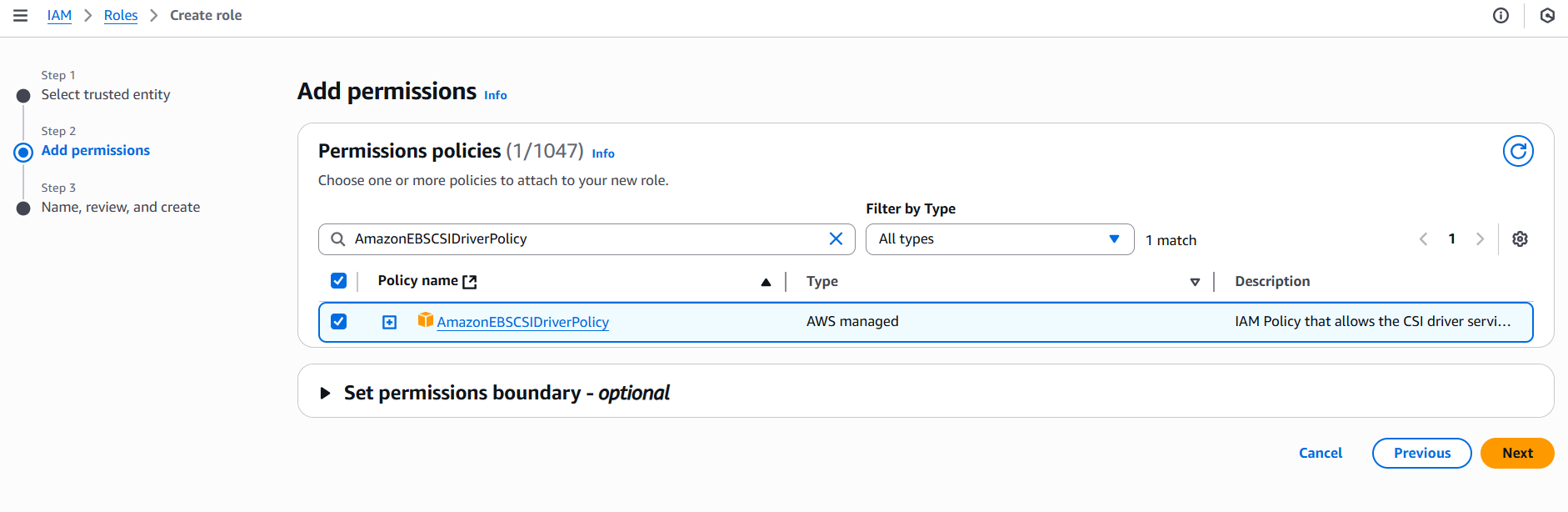
# keep as default => Next



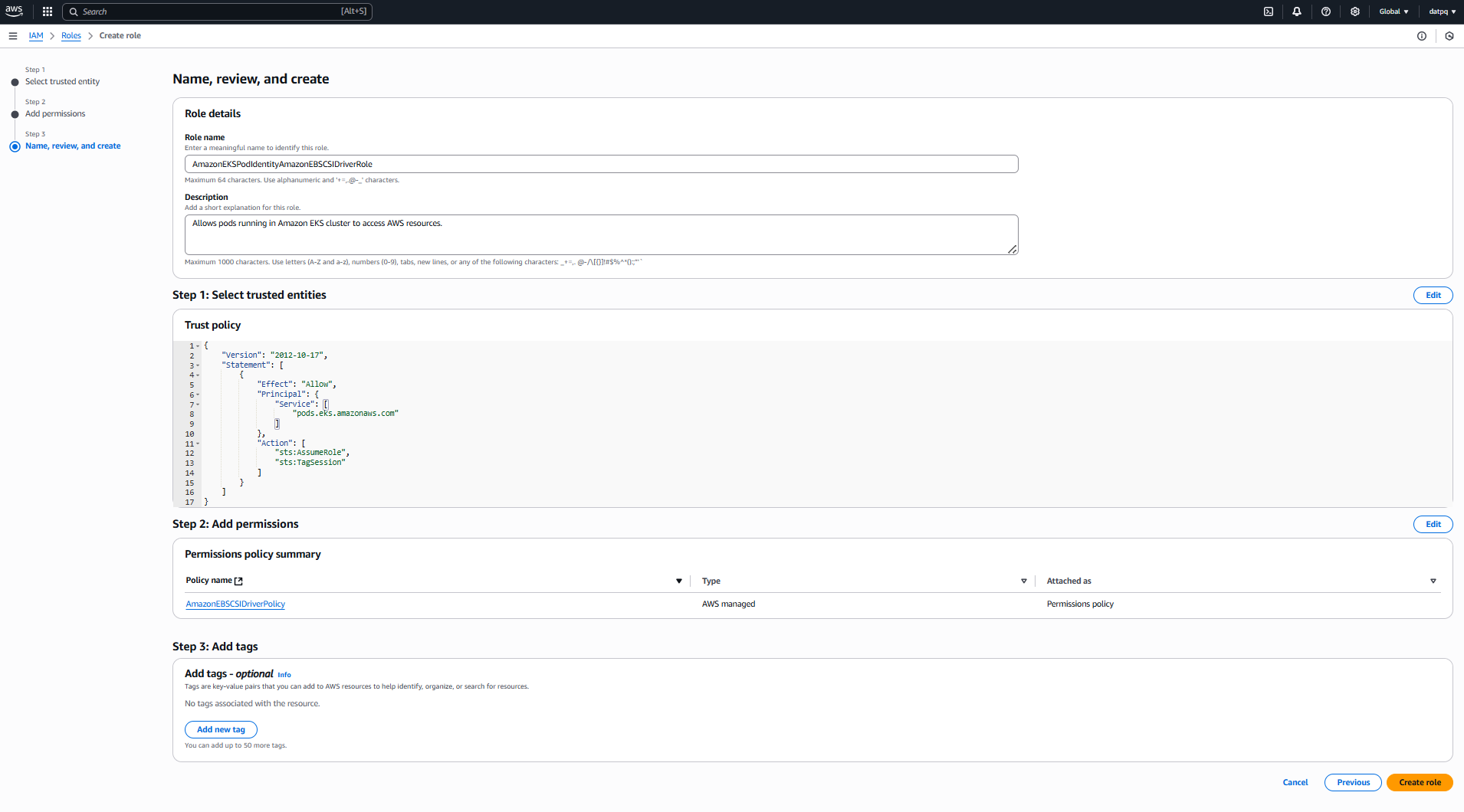


# on the **Add permissions** step, the appropriate managed policy for the role policy is preselected for the add-on

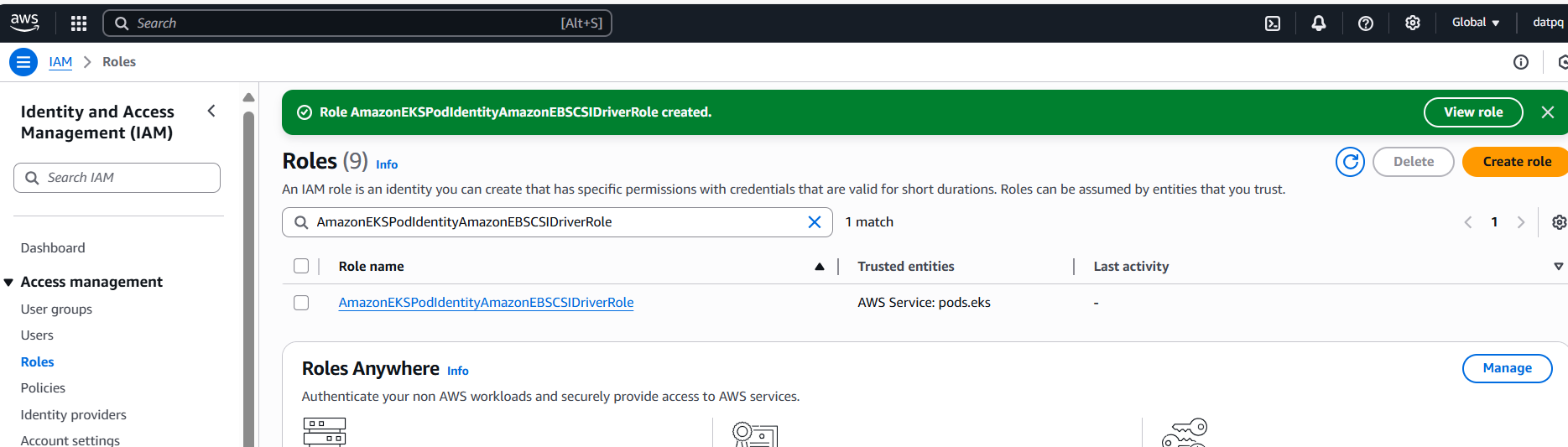
AmazonEBSCSIDriverPolicy is selected as default



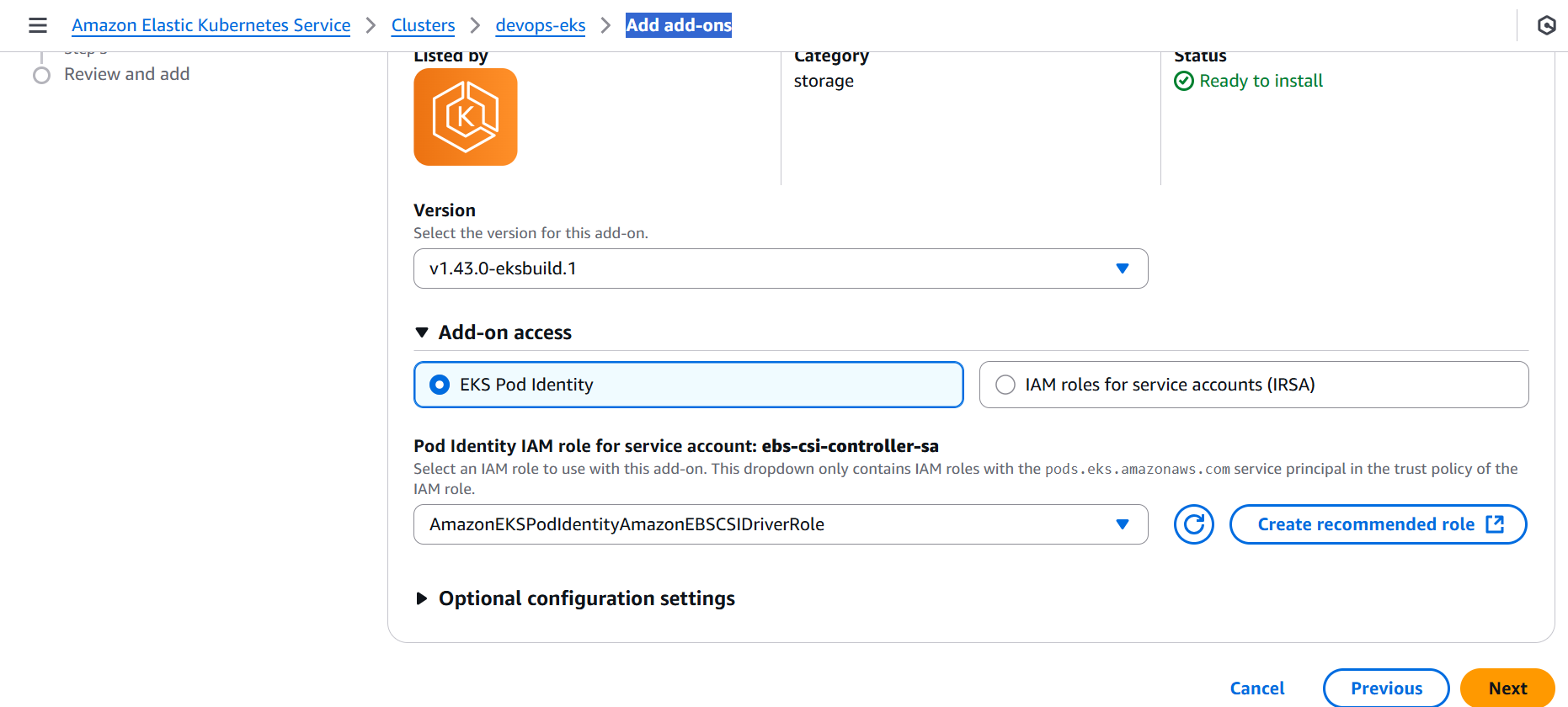
# choose Next



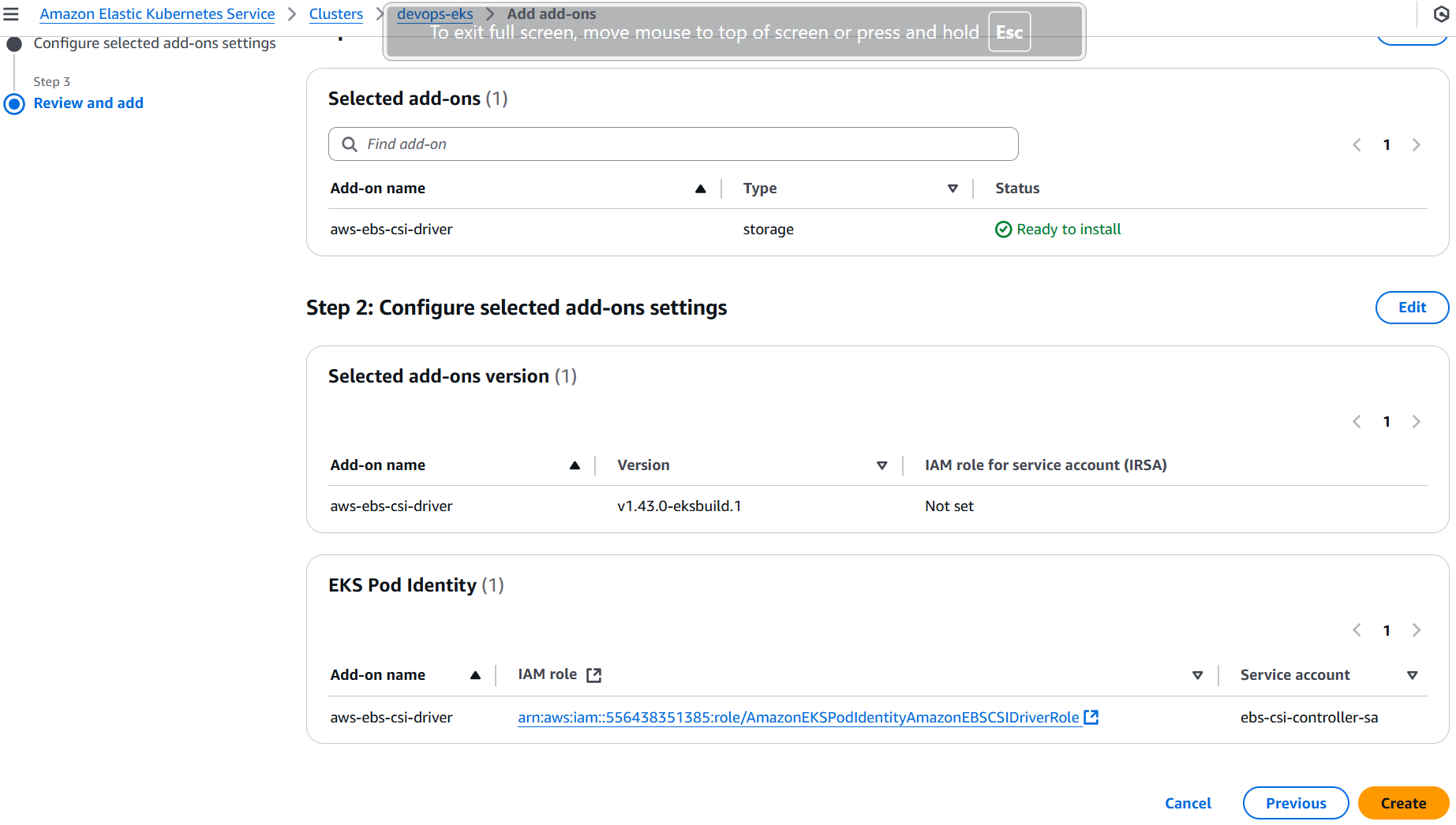
# choose Create role



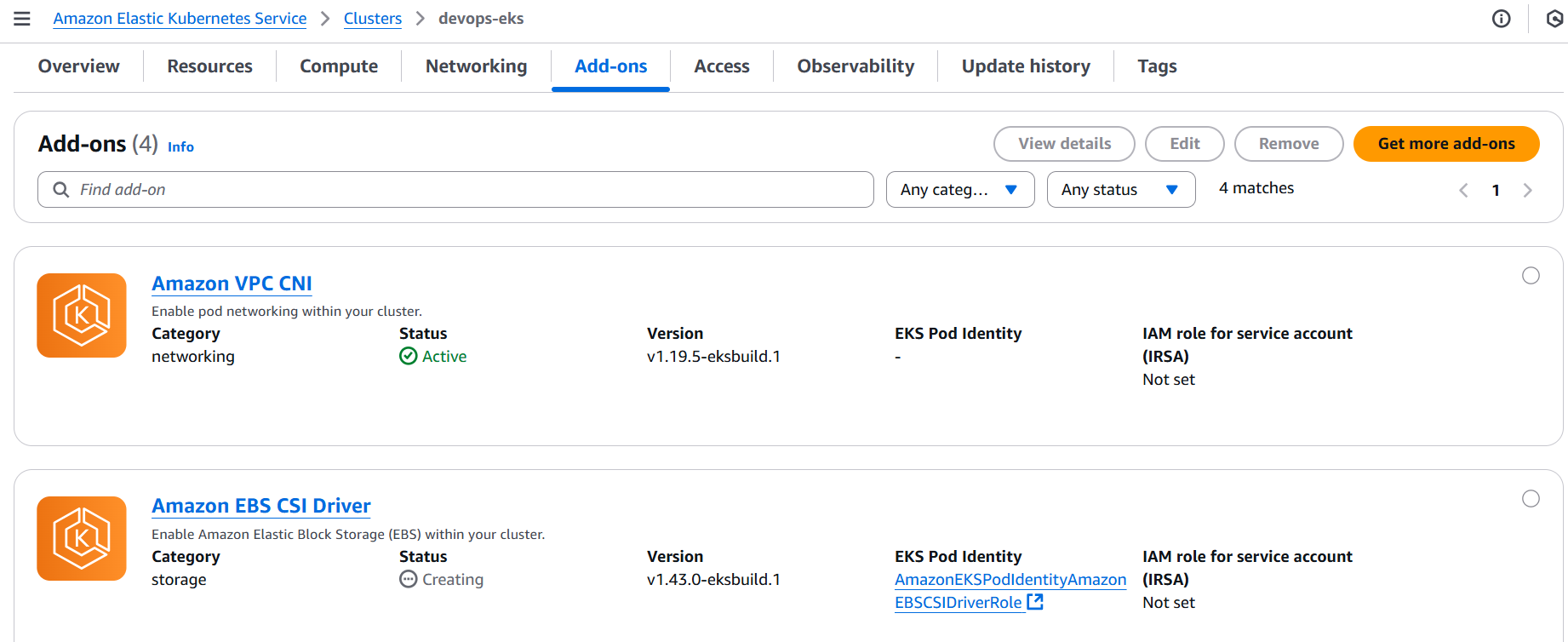
# back again Add add-ons, choose Next



# choose Create



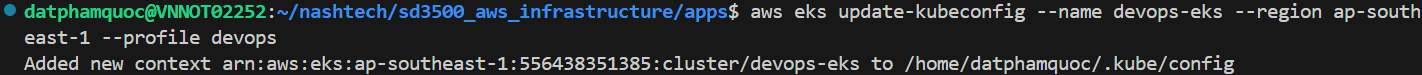
# add-on is added to devops-cluster



Back to terminal to deploy application in cluster

# configure kubeconfig (Note: Need update AWS cli to latest version)

aws eks update-kubeconfig --name devops-eks --region ap-southeast-1 --profile devops

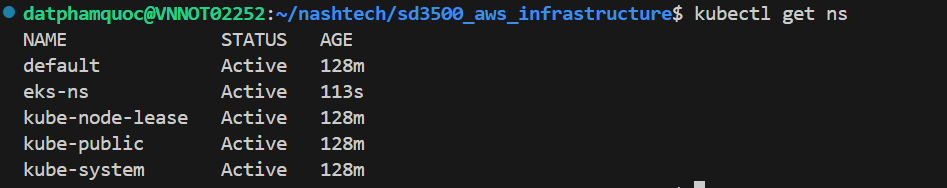


# create the a new namspace called “eks-ns” if it is not yet present

kubectl create ns eks-ns



# verify created namespace



# change default working namespace to that one

kubectl config set-context --current --namespace eks-ns



# verify if you’ve set to this namespace:

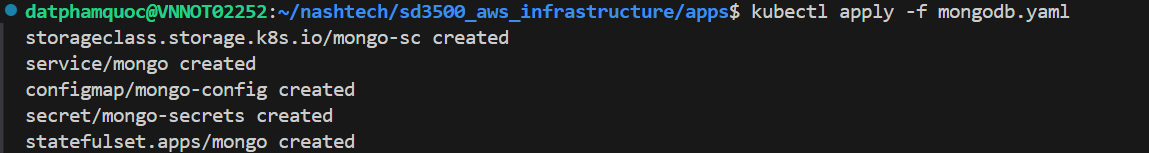
kubectl config view --minify | grep namespace



**Create a database**

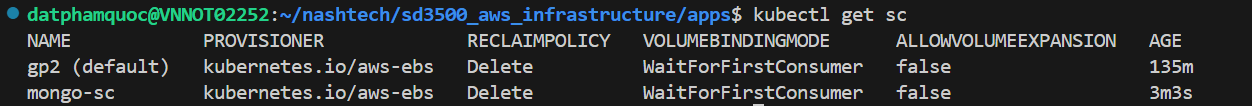
# create storage class, pvc and database

kubectl apply -f mongodb.yaml



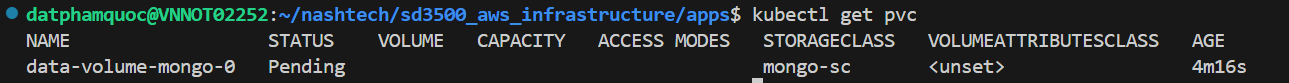
# verify storage class

kubectl get sc



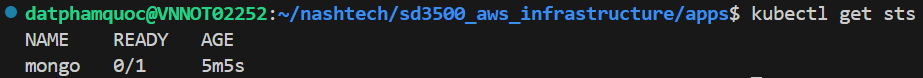
# verify pvc

kubectl get pvc

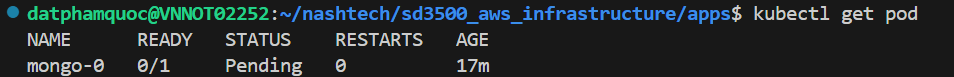


# verify StatefulSet

kubectl get sts

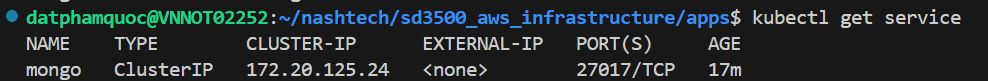


# verify database pods



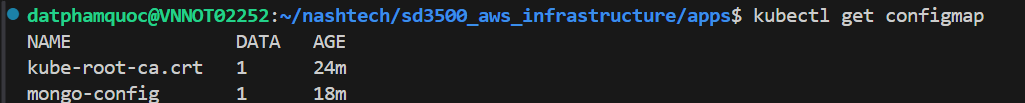
# verify database service

kubectl get service



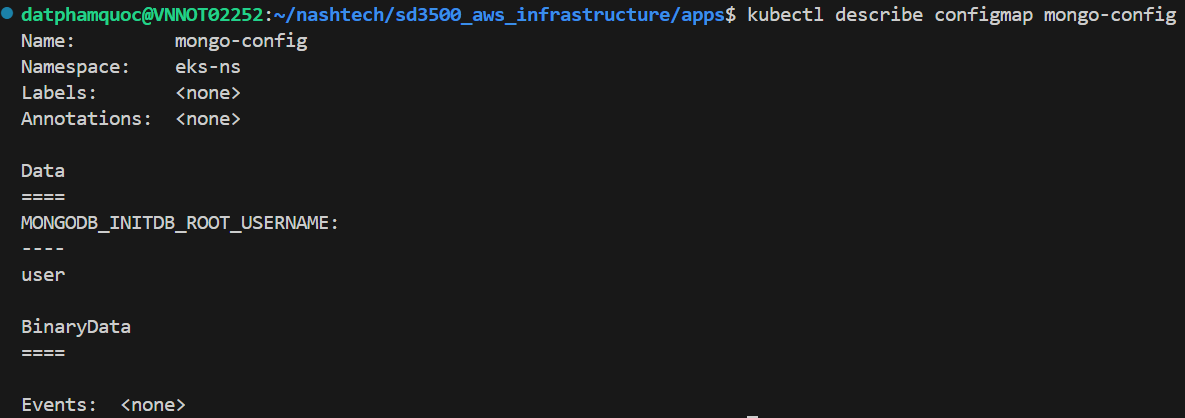
# verify configmap

kubectl get configmap



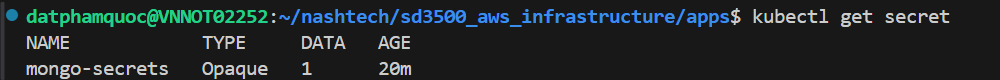
# check configmap details

kubectl describe configmap mongo-config



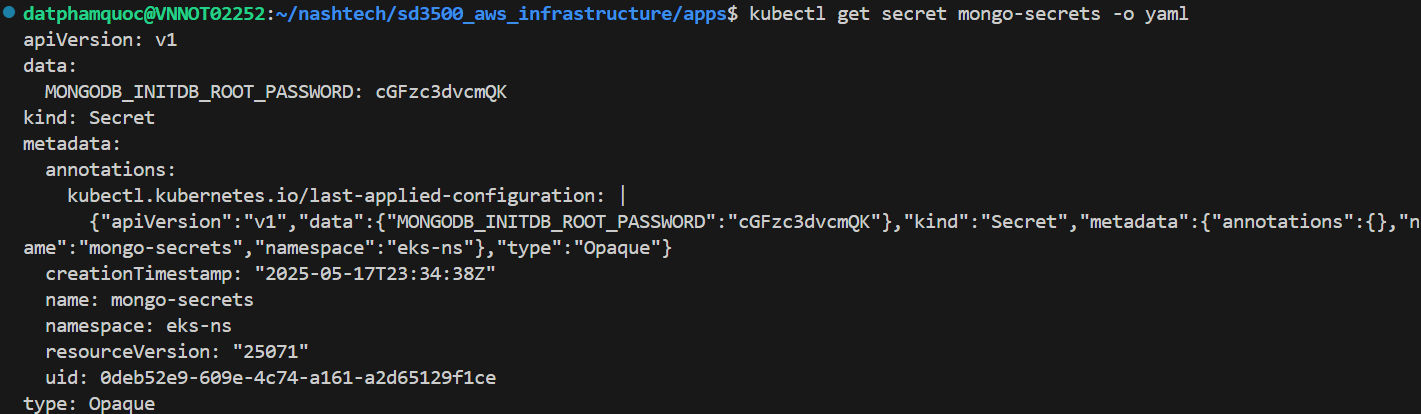
# verify secret:

kubectl get secret



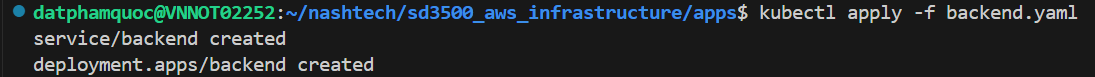
# check secret details

kubectl get secret mongo-secrets -o yaml

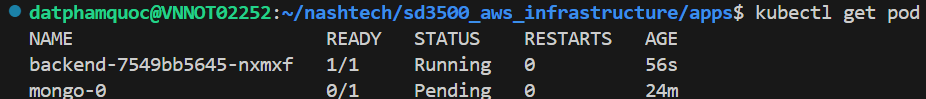


**Create backend**

kubectl apply -f backend.yaml

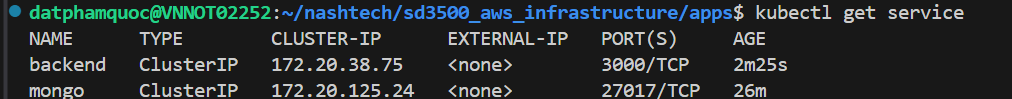
****

# verify backend pods

****

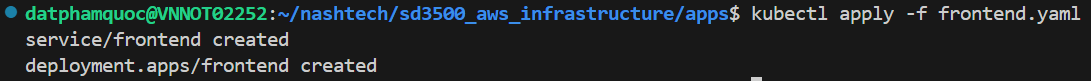
# verify backend service

kubectl get service

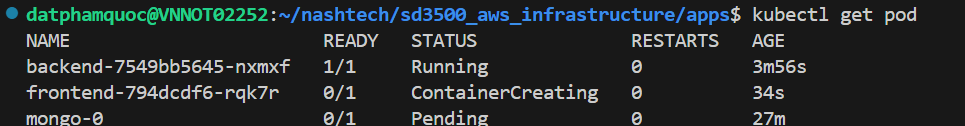
****

**Create frontend**

kubectl apply -f frontend.yaml

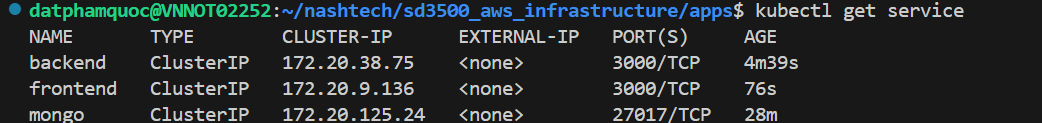
****

# verify frontend pods

****

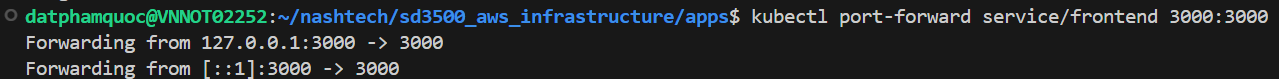
# verify frontend service

kubectl get service

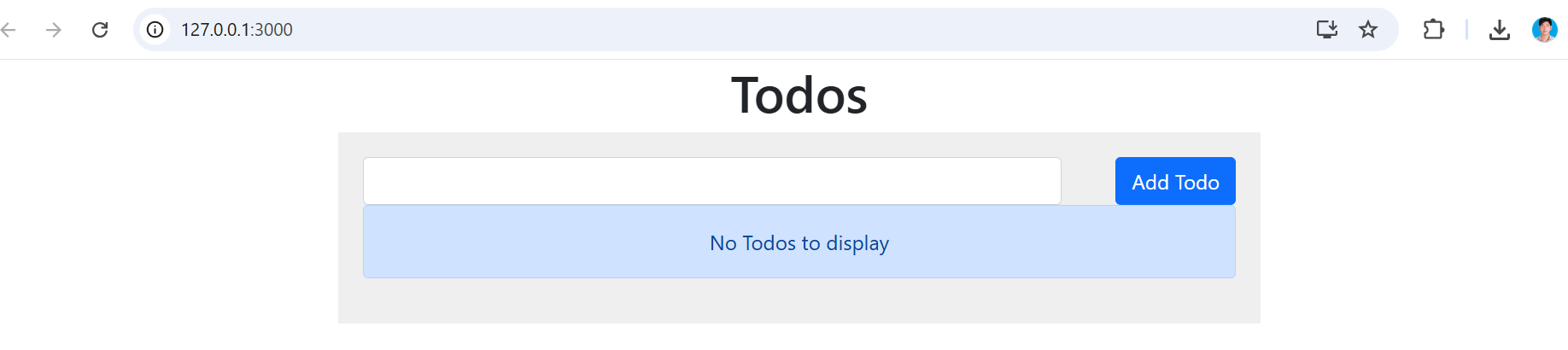
****

# Application verification by exposing frontend service, expose frontend service to access application

kubectl port-forward service/frontend 3000:3000

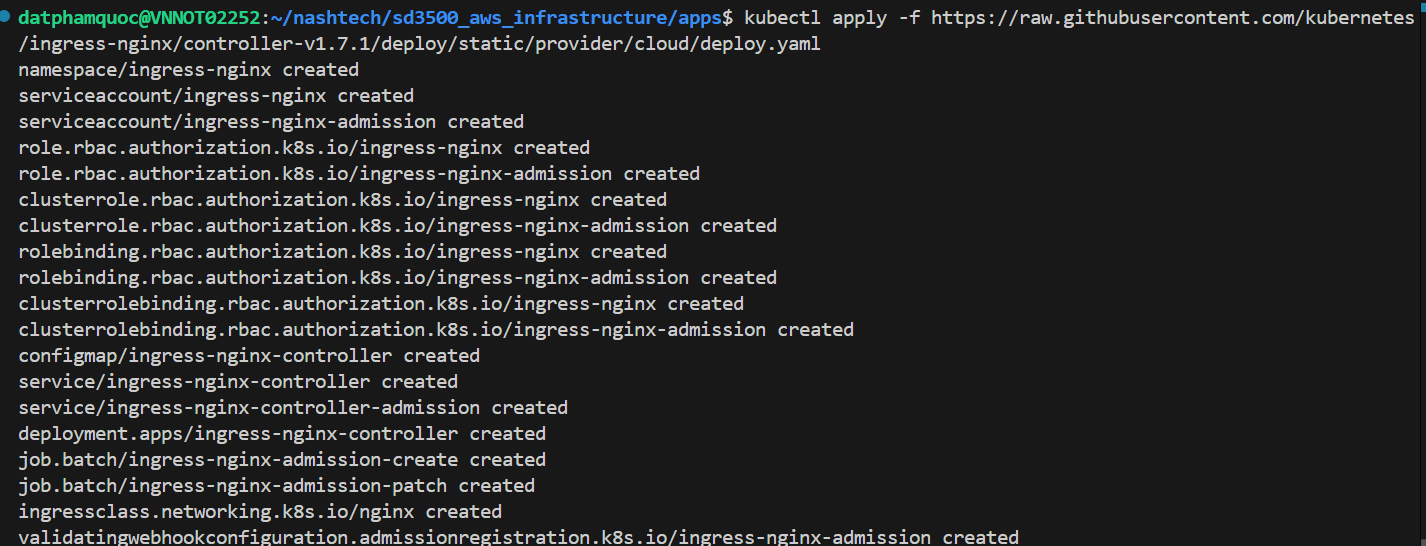
****

# open browser to access application at URL: locahost:3000

****

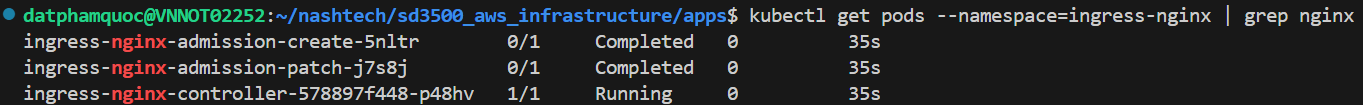
**Application Ingress**  
# install NGINX ingress controller

kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v1.7.1/deploy/static/provider/cloud/deploy.yaml

****

# verify ingress controller installation

kubectl get pods --namespace=ingress-nginx | grep nginx

****

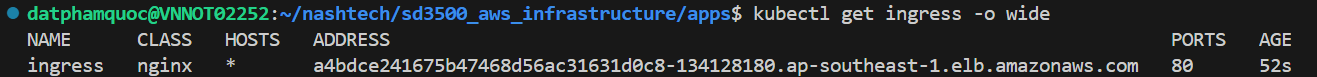
# install application ingress

kubectl apply -f ingress.yml

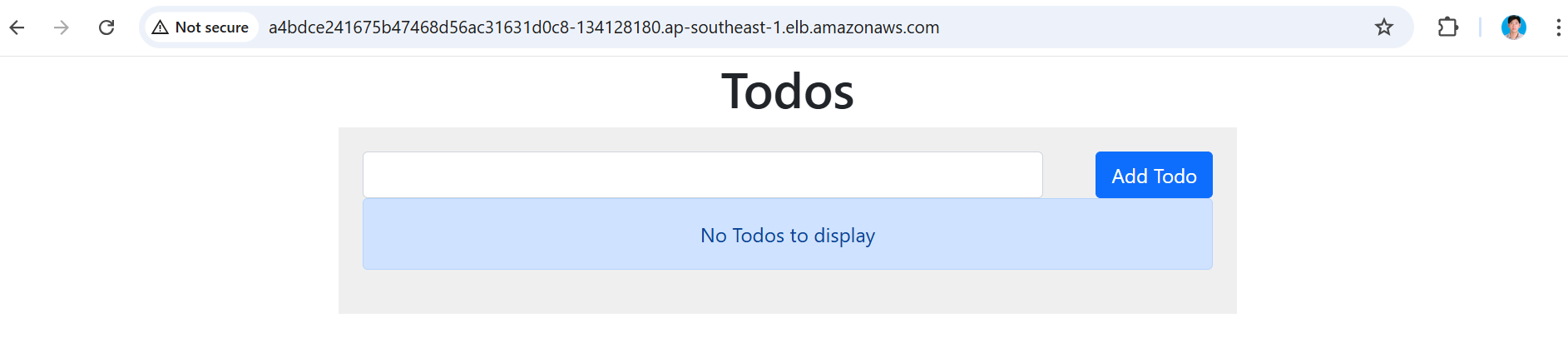
****

# verify application ingress

kubectl get ingress -o wide

****

# Open browser to access application at URL created by ingress controller

****

Clean EKS cluster  
AWS\_PROFILE=devops terraform destroy -var-file="sample.tfvars"

