**Perquisites:**

* AWSCLI
* Install Git
* IAM User
* IAM Policy
* Route53
* docker
* EKS Cluster (with 4 nodes, instance type: t2.xlarge)
* Install Helm

Create Amazon linux-2 Instance with:

Instance type: t2.xlarge

**Install git:**

yum install git –y

**Install docker:**

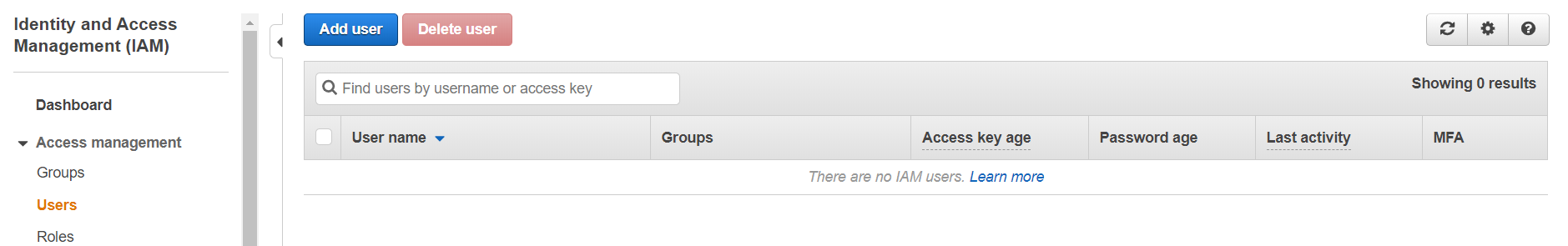
sudo yum install -y docker

sudo service docker start

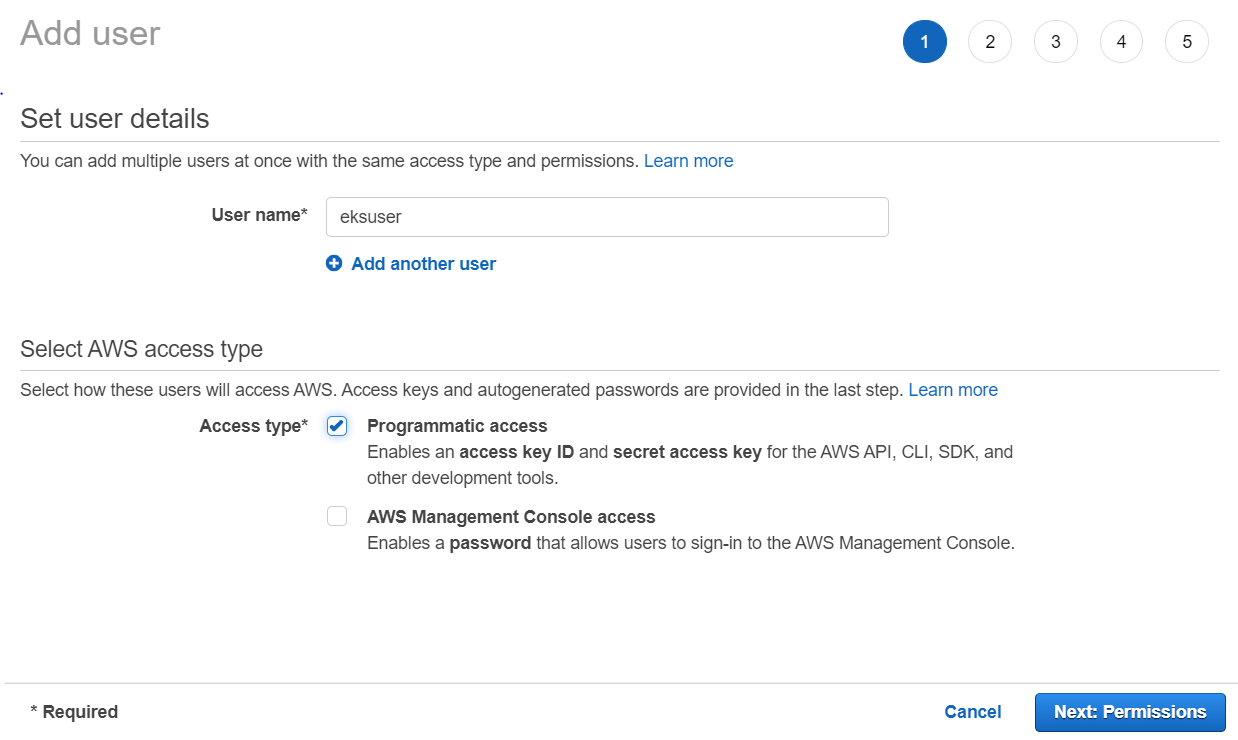
sudo usermod -a -G docker ec2-user

**Create user:**

Goto IAM and click on user

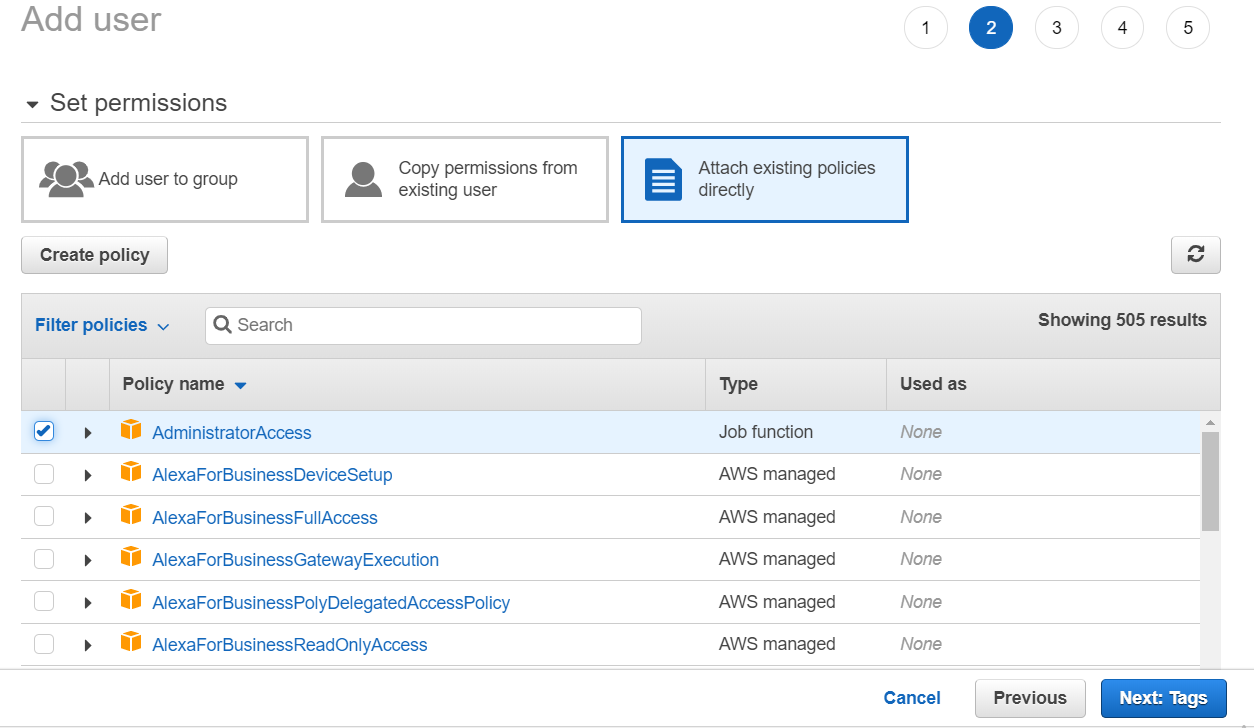


Click on Add user



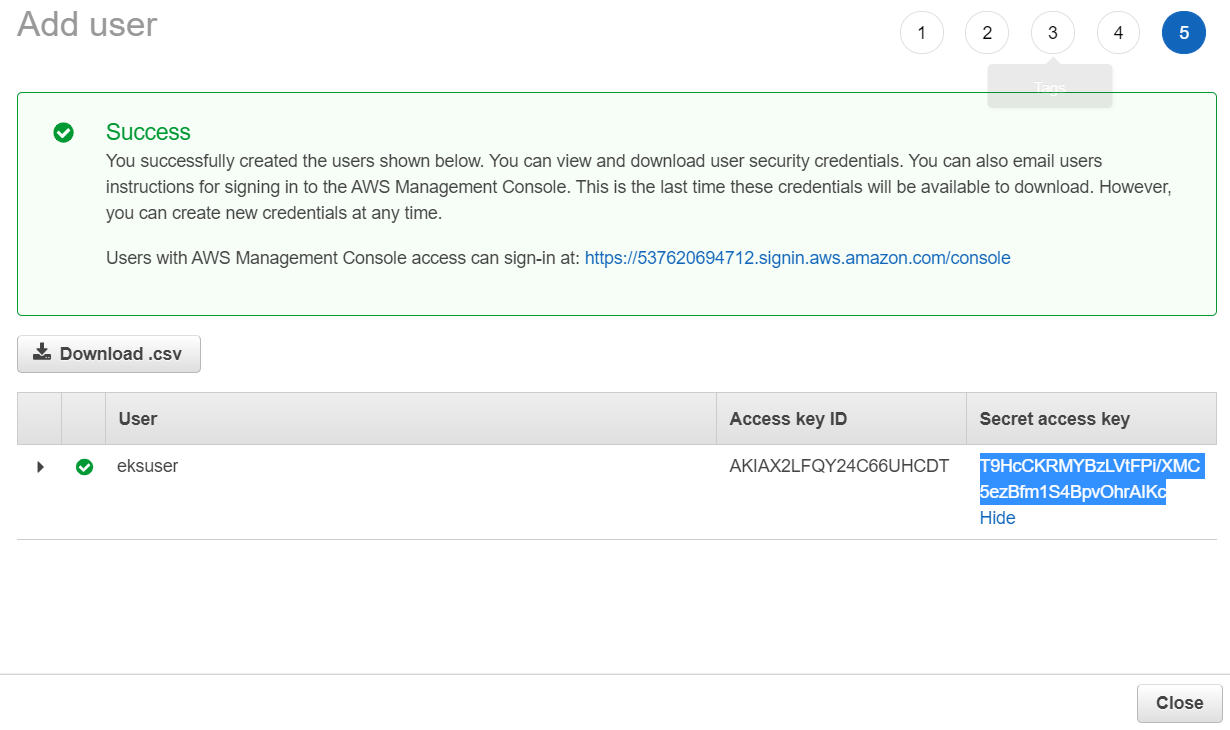
Give any name for user and check Accesstype as Programmatic access

Click on Next



Add tag as Administrator and click on next

Click on create

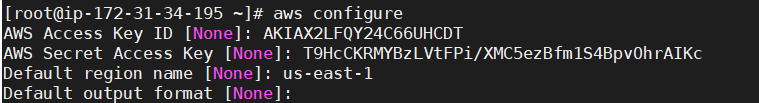


Here user created and copy Access key ID and Secret access key

These Access Key ID and Secret access key gives to configure aws

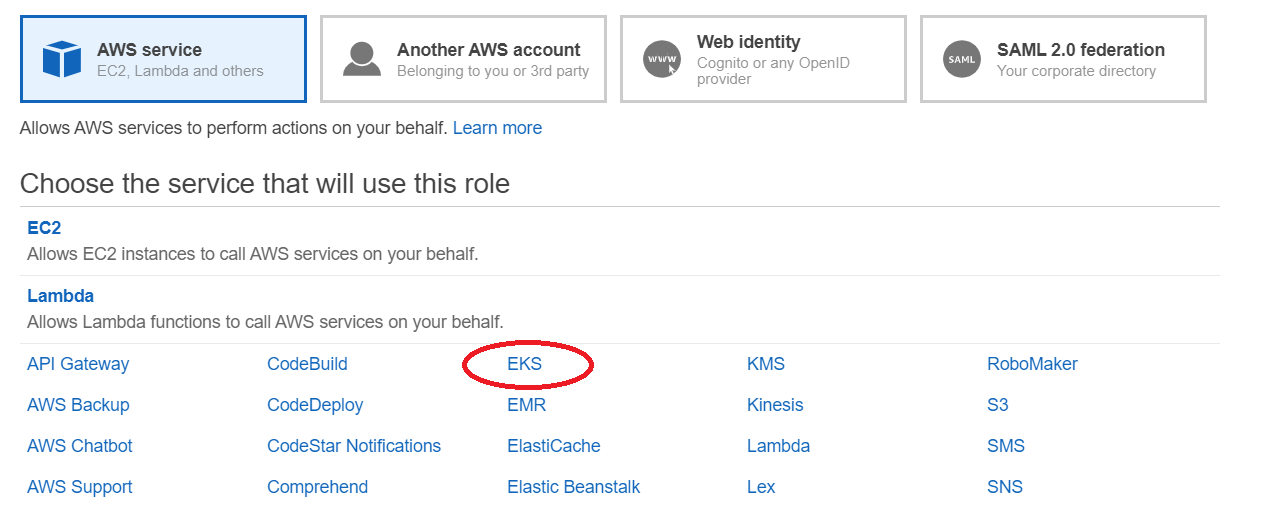
**AWS-CLI Installation:**

aws configure

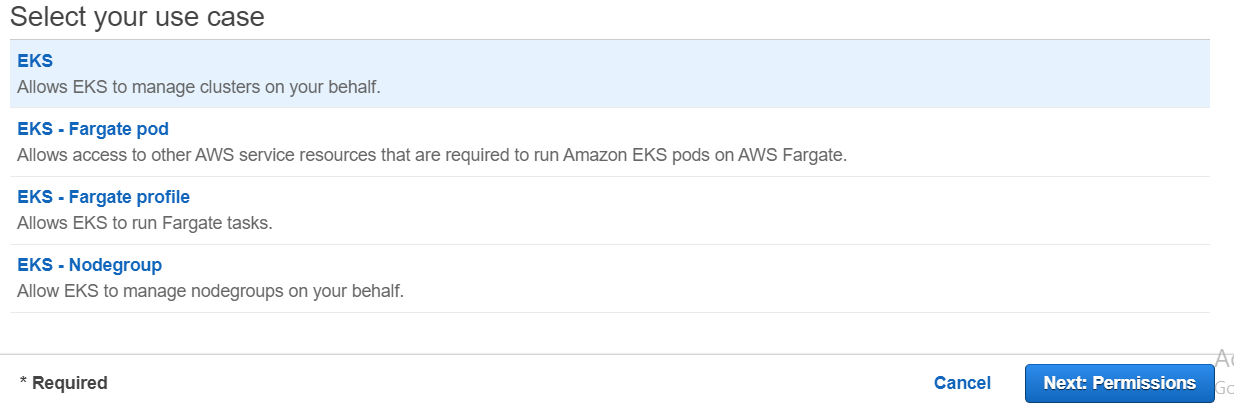


**To create Role For EKS:**

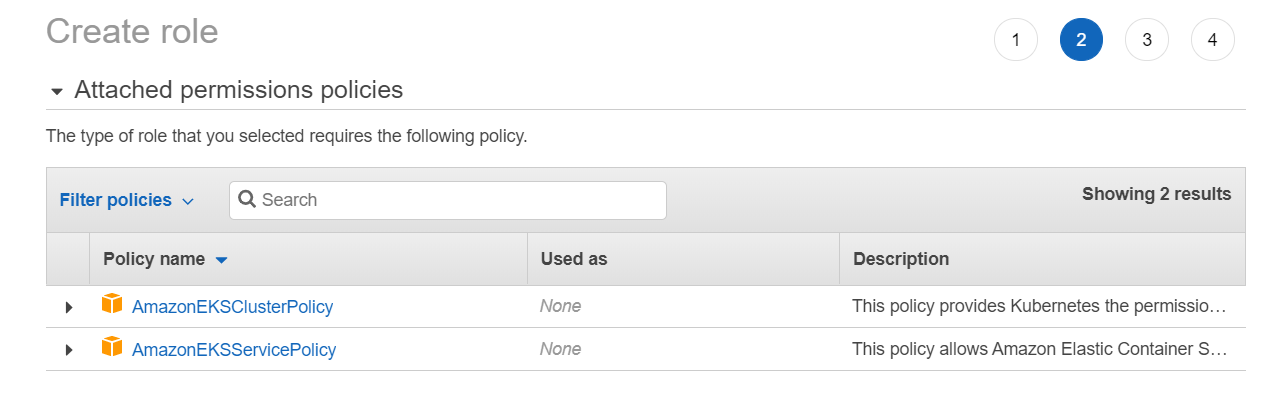
* Click on Create Role
* Click on EKS



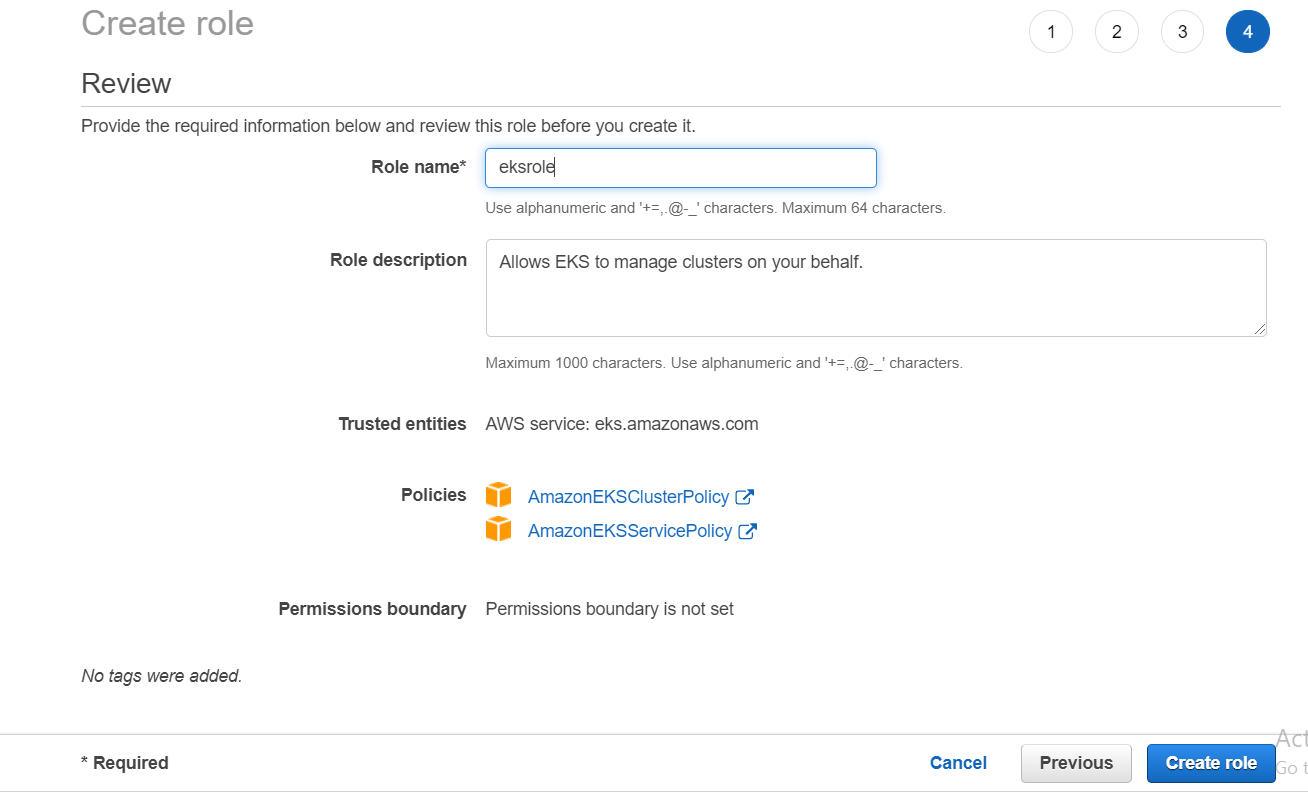
* Click on EKS again and then click on Next



* We will get window like this



* Click on Next
* Give Role Name as “eksrole” and click create role



**EKS Cluster Creation:**

Kubectl: Used for communicating with the cluster API server

curl -o kubectl https://amazon-eks.s3-us-west-2.amazonaws.com/1.14.6/2019-08-22/bin/linux/amd64/kubectl

chmod +x ./kubectl

mkdir -p $HOME/bin

cp ./kubectl $HOME/bin/kubectl

export PATH=$HOME/bin:$PATH

echo 'export PATH=$HOME/bin:$PATH' >> ~/.bashrc

source $HOME/.bashrc

kubectl version --short --client

**AWS-IAM-Authenticator:** To allow IAM authentication with the Kubernetes cluster

curl -o aws-iam-authenticator <https://amazon-eks.s3-us-west-2.amazonaws.com/1.14.6/2019-08->22/bin/linux/amd64/aws-iam-authenticator

chmod +x ./aws-iam-authenticator

cp ./aws-iam-authenticator $HOME/bin/aws-iam-authenticator

export PATH=$HOME/bin:$PATH

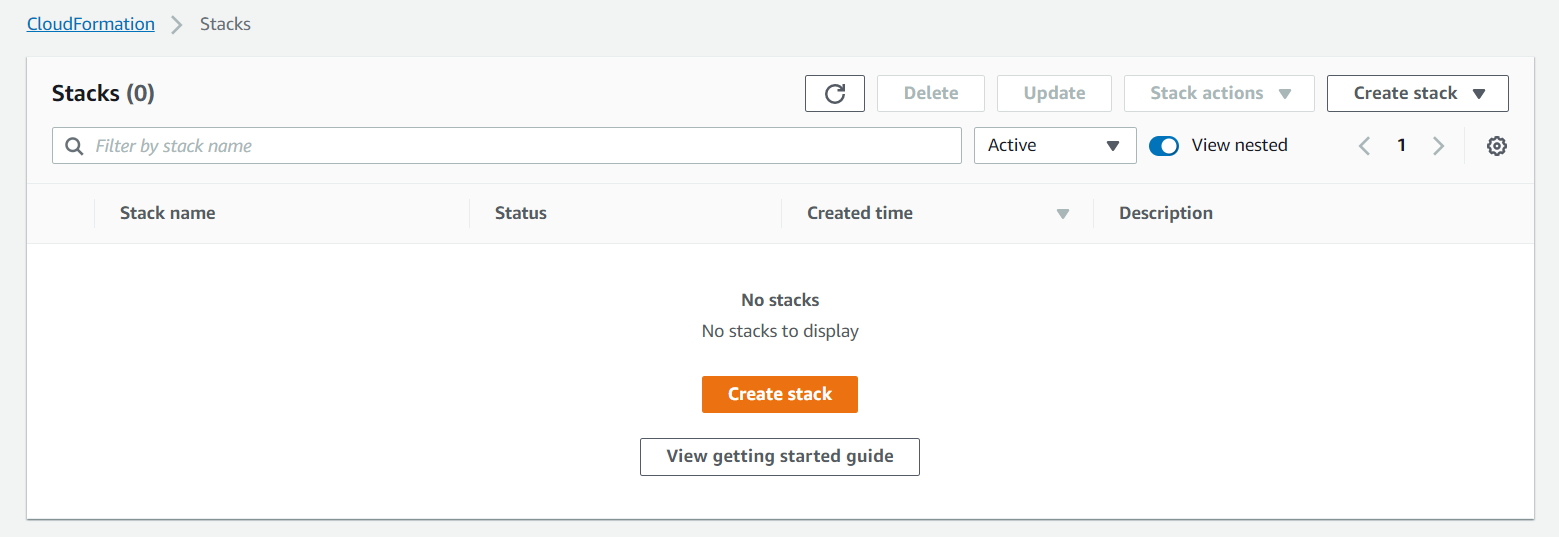
echo 'export PATH=$HOME/bin:$PATH' >> ~/.bashrc

source ~/.bashrc

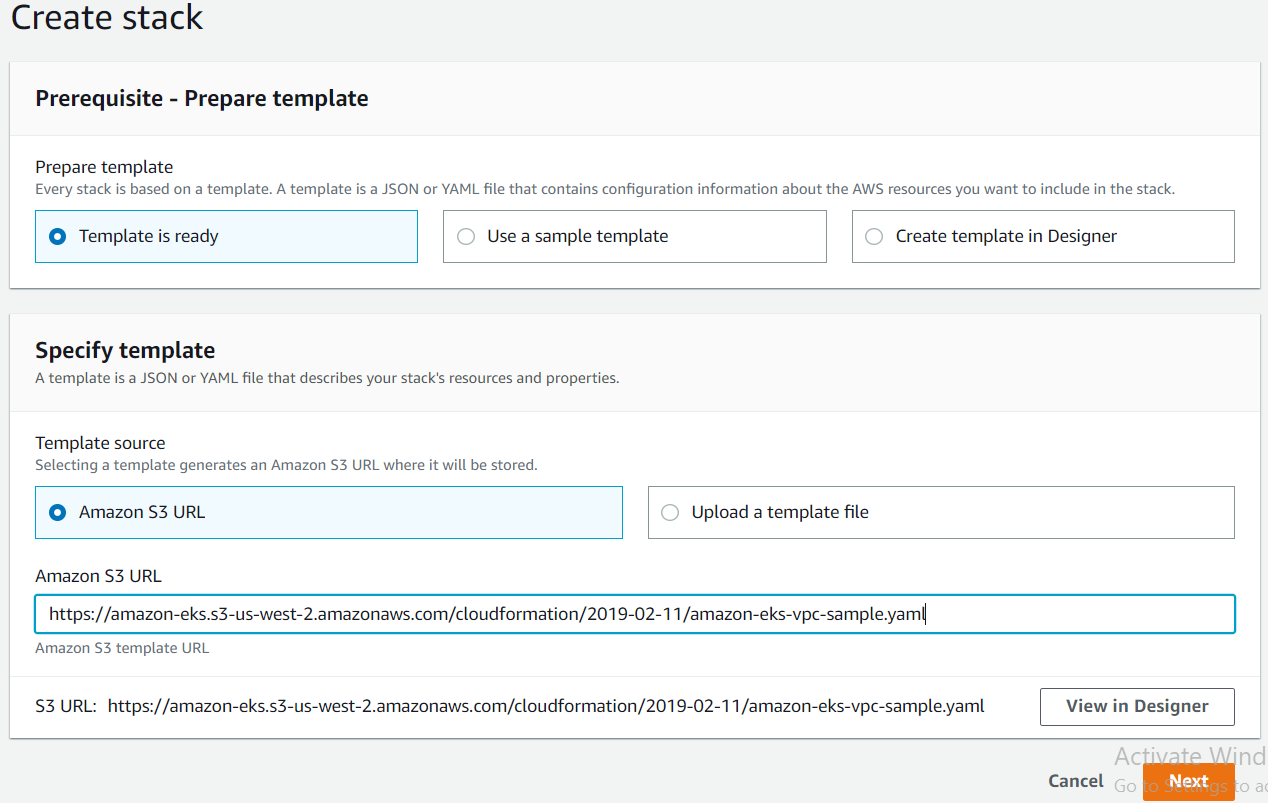
aws-iam-authenticator –help

**Create VPC for EKS:**

Open Cloud Formation



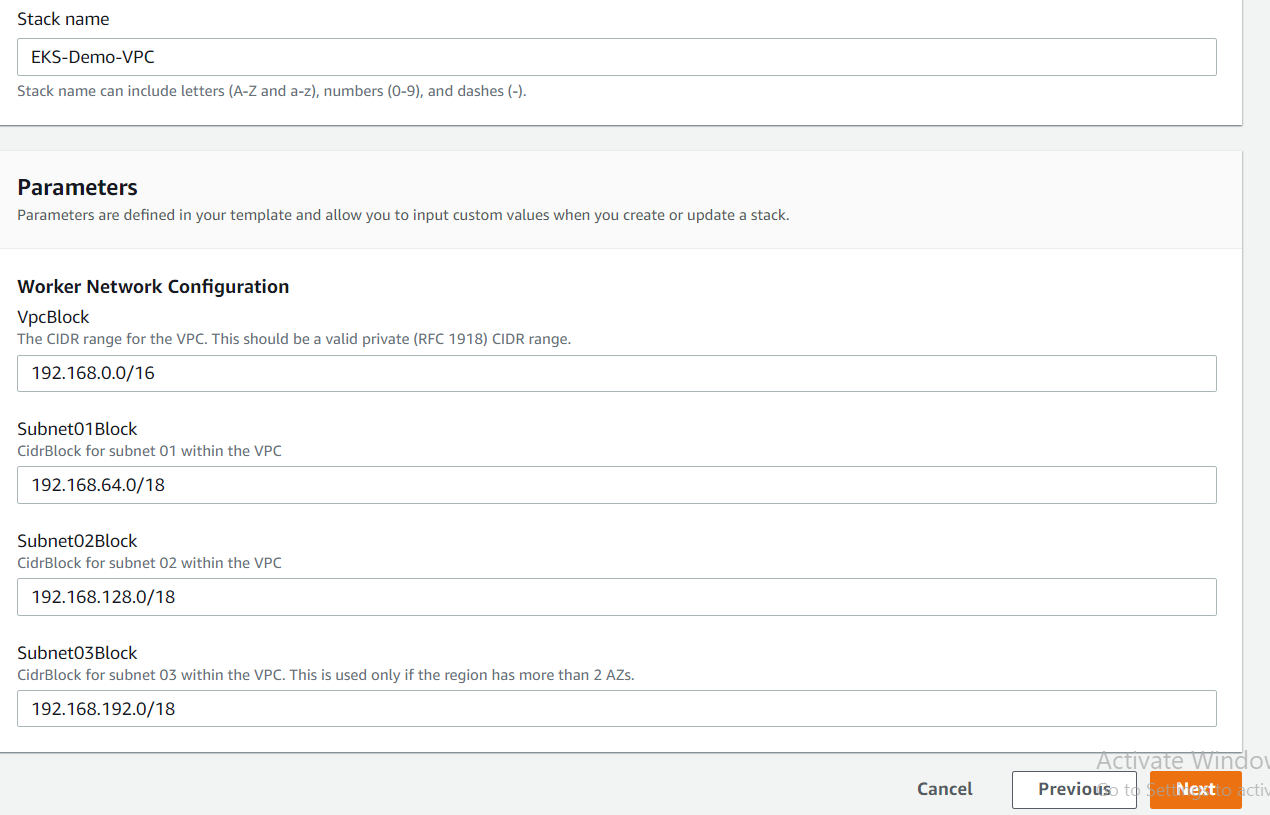
* Click on Stack



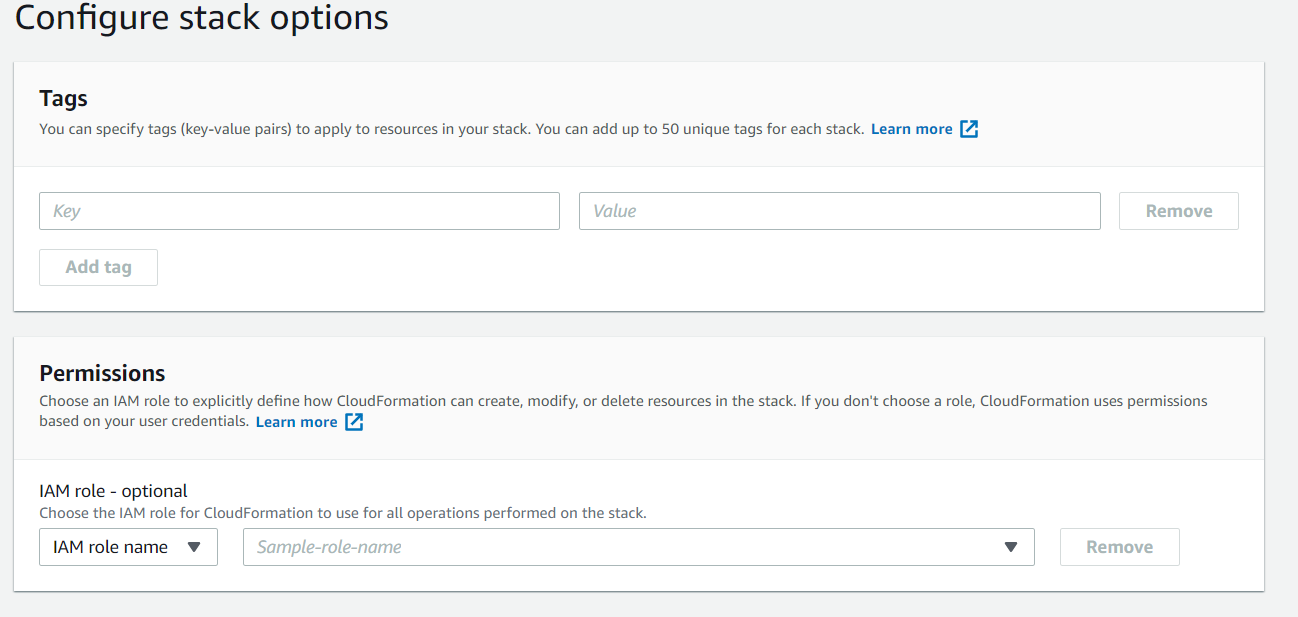
* Give Amazon S3 URL-

https://amazon-eks.s3-us-west-2.amazonaws.com/cloudformation/2019-02-11/amazon-eks-vpc-sample.yaml

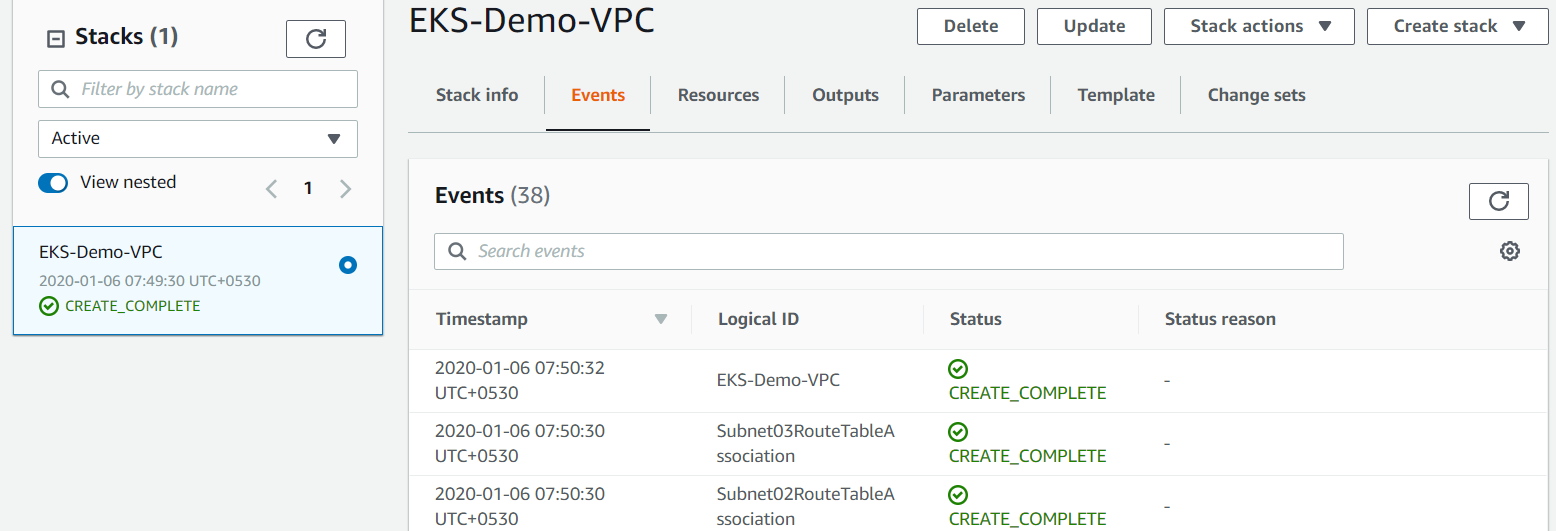
* Click on Next



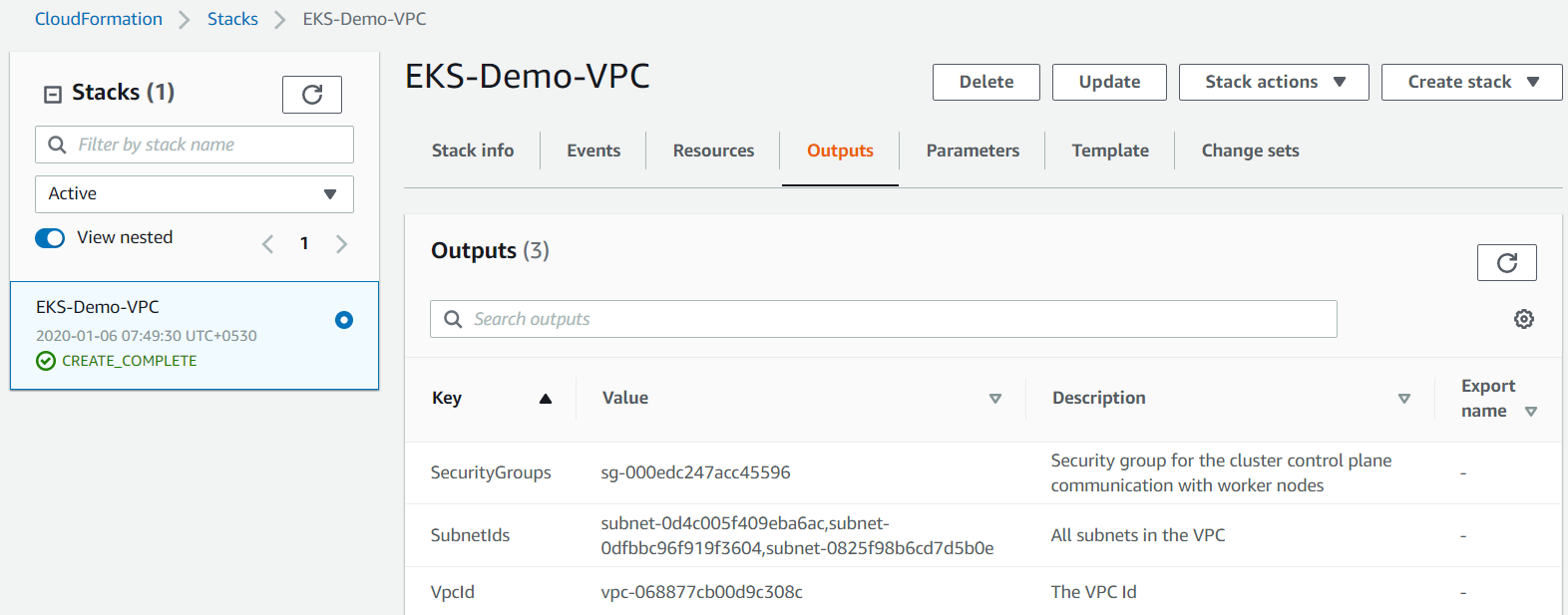
Give Stack name and click on next



Click on Next and Click on Create stack



Once complete stack creation click on outputs and check details



* The **SecurityGroups** value for the security group that was created. We need this when we create our EKS cluster; this security group is applied to the cross-account elastic network interfaces that are created in our subnets that allow the Amazon EKS control plane to communicate with our worker nodes.
* The **VpcId** for the subnets that were created. We need this when we launch our worker node group template.
* The **SubnetIds** for the subnets that were created. We need this when we create our EKS cluster; these are the subnets that our worker nodes are launched into.

**Create EKS Cluster:**

To create EKS instance we need to give bellow command by changing region, Role-arn, resource- vpc-config details.

Region: Where you are working

Role-arn: Get arn from role which we create before (Here create Role is “eksrole”)

resources-vpc-config: Get from stack details output

aws eks create-cluster \

--name eks-cluster \

--region ap-south-1 \

--role-arn arn:aws:iam::704505967285:role/eksrole \

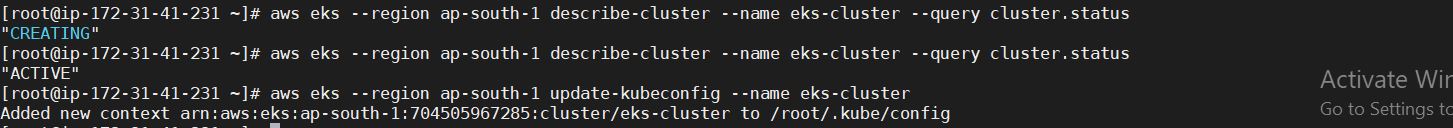
--resources-vpc-config subnetIds=subnet-00b6f96312e896bef,subnet-0772bb028da998529,securityGroupIds=sg-0d2863f4357209a86

To check status of the cluster creation:

aws eks --region ap-south-1 describe-cluster --name eks-cluster --query cluster.status

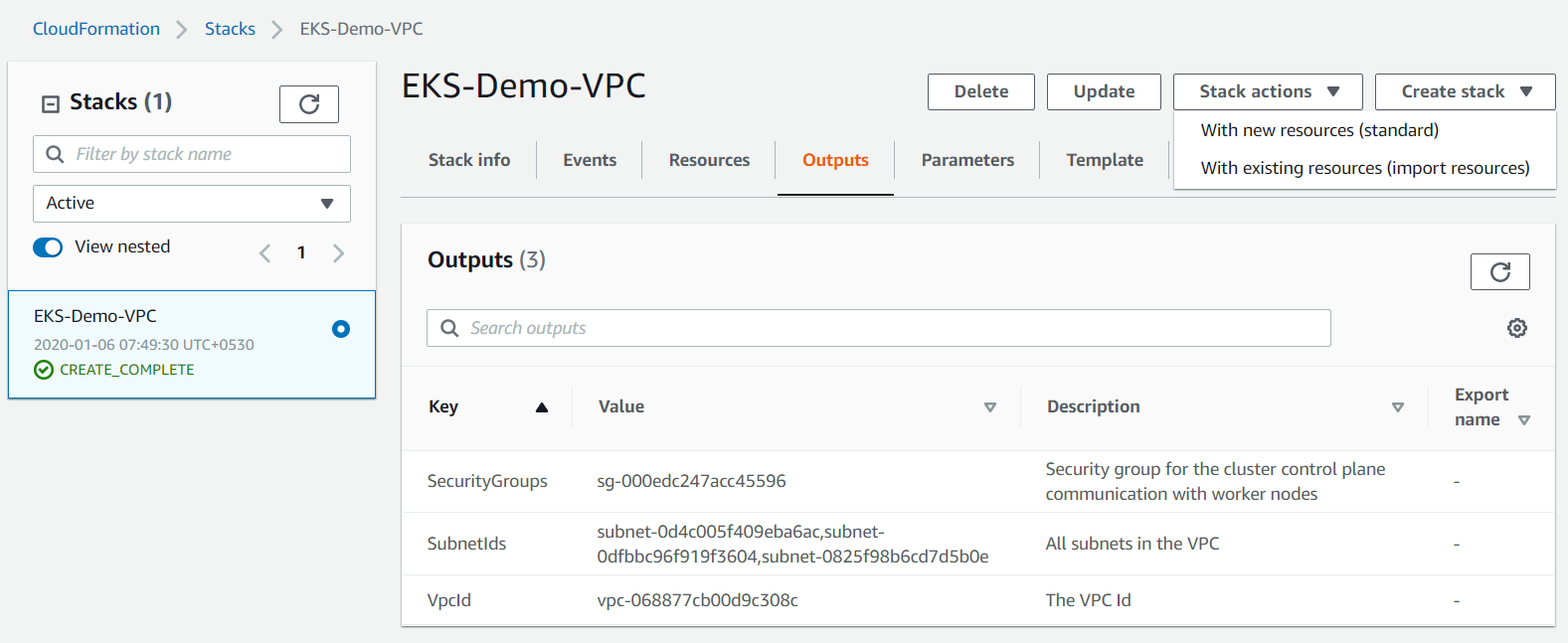
Update cluster:

aws eks --region ap-south-1 update-kubeconfig --name eks-cluster

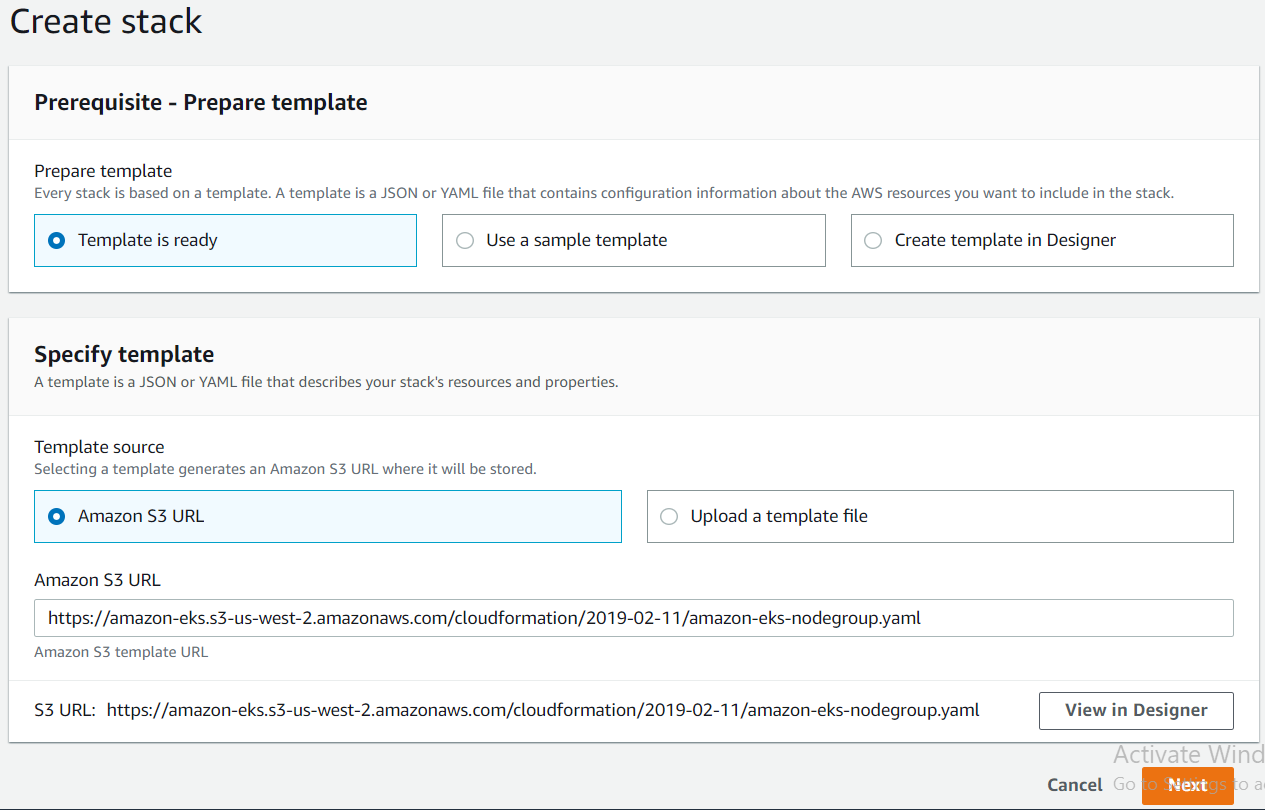


**Launching Kubernetes worker nodes:**

Click on Create Stack



Click on with new resources



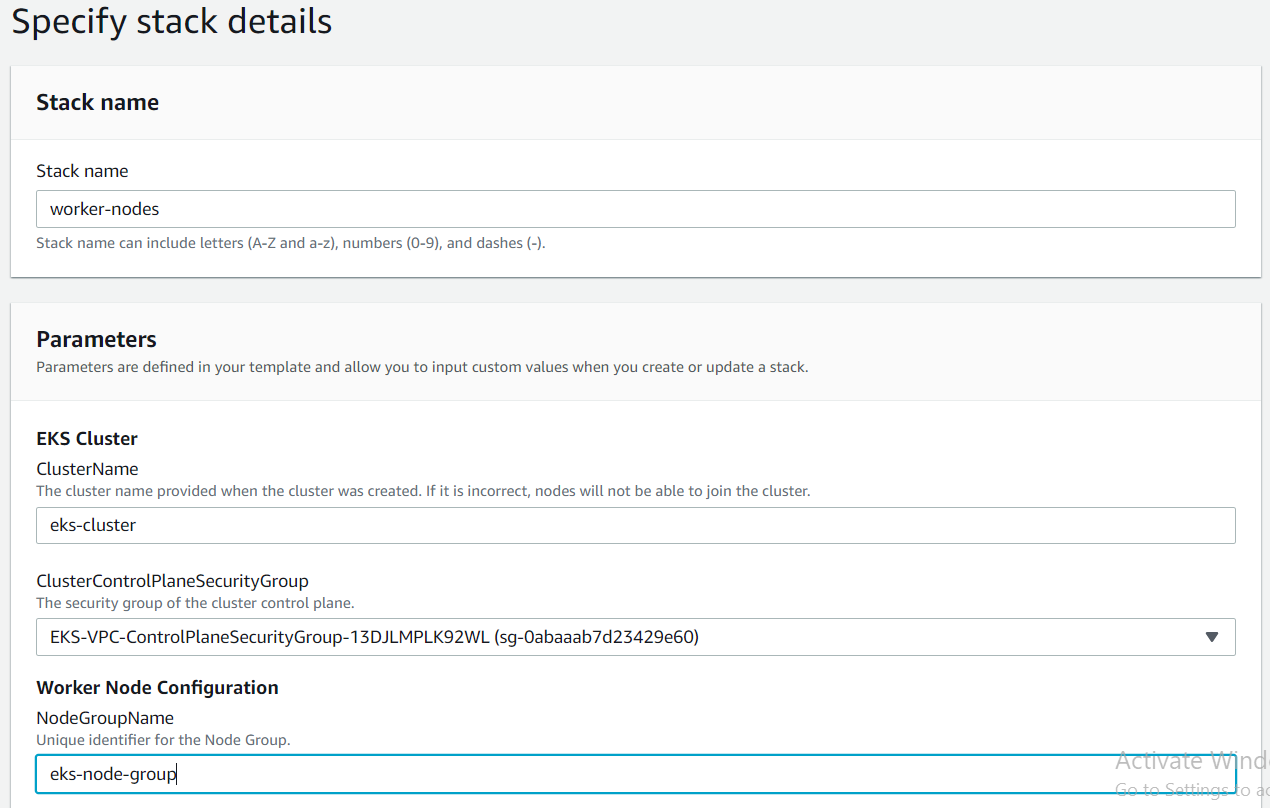
**Give below link at Amazon S3 URL to create nodes:**

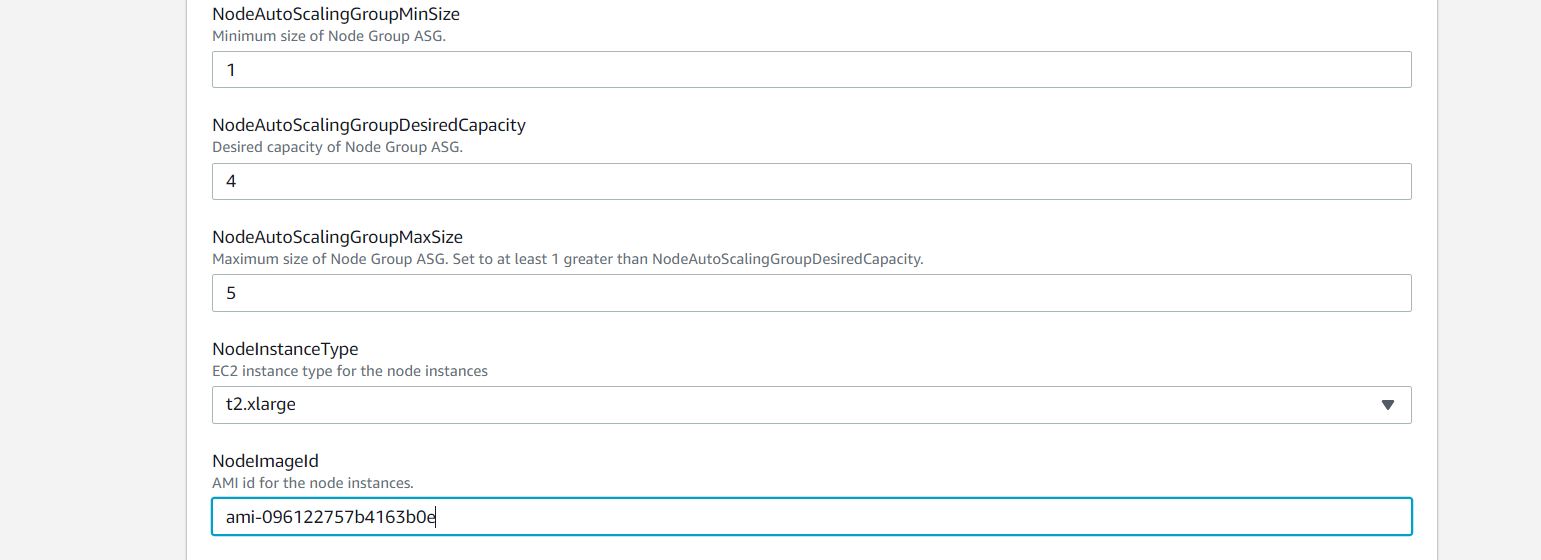
<https://amazon-eks.s3-us-west-2.amazonaws.com/cloudformation/2019-02-11/amazon-eks-nodegroup.yaml>

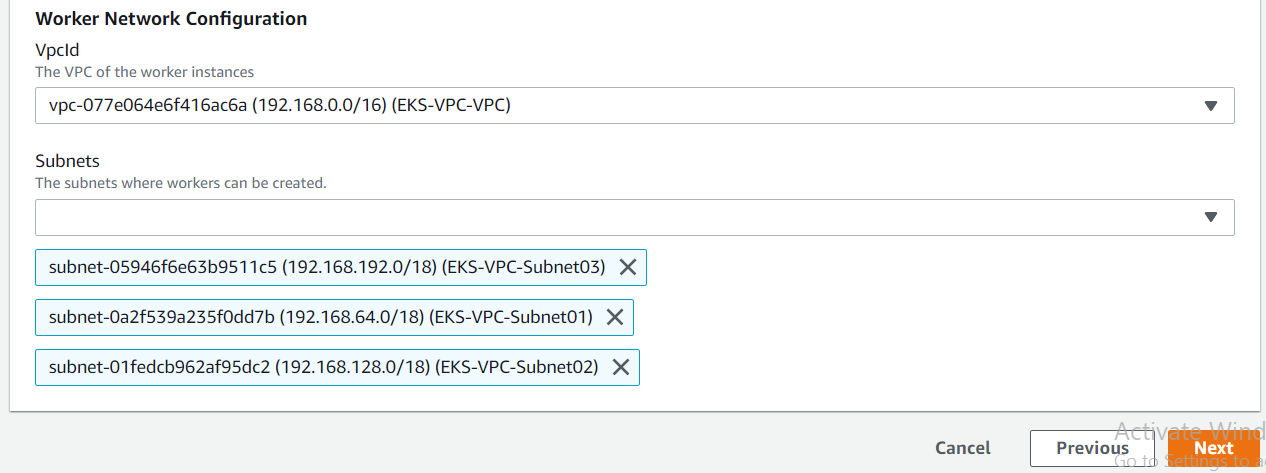
Fill details as shown below

Note: Give details for Security group, VPC and Subnets (These details we get at output of VPC creation for EKS)

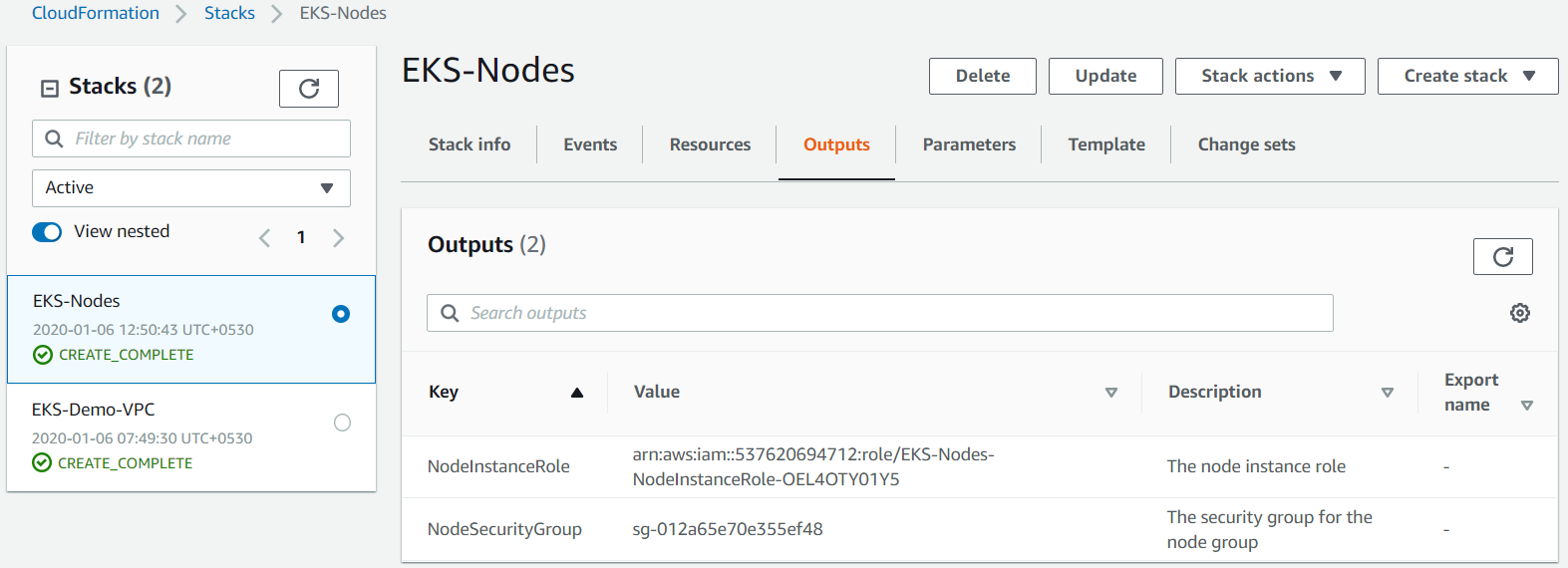
Worker Node AMI - **ami-096122757b4163b0e** (Should give this only for ap-south-1)







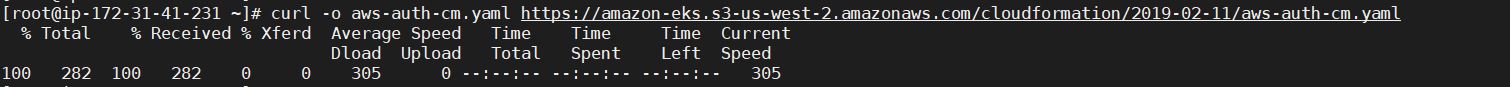
Nodes created for EKS Cluster and Copy Role **arn** (we get at output tab)

****

* **To enable worker nodes to join your cluster:**

Download the AWS authenticator configuration map

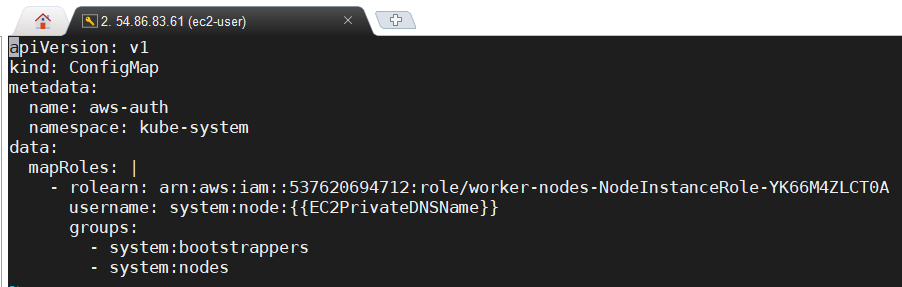
curl -o aws-auth-cm.yaml <https://amazon-eks.s3-us-west-2.amazonaws.com/cloudformation/2019-02-11/aws-auth-cm.yaml>



vi aws-auth-cm.yaml

Change relearn which one is created while creating worker nodes

arn:aws:iam::537620694712:role/worker-nodes-NodeInstanceRole-YK66M4ZLCT0A

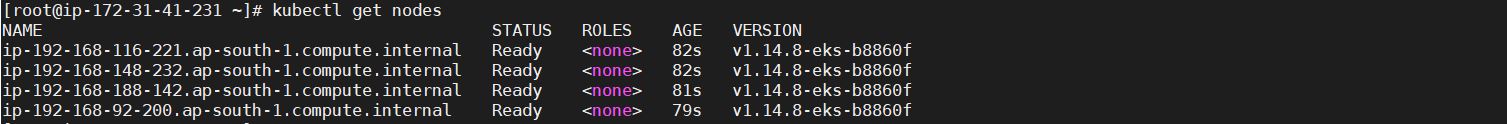
****

kubectl apply -f aws-auth-cm.yaml

D:\Naresh\snipping\Capture.JPG

Nodes added to cluster. Check by using below command.

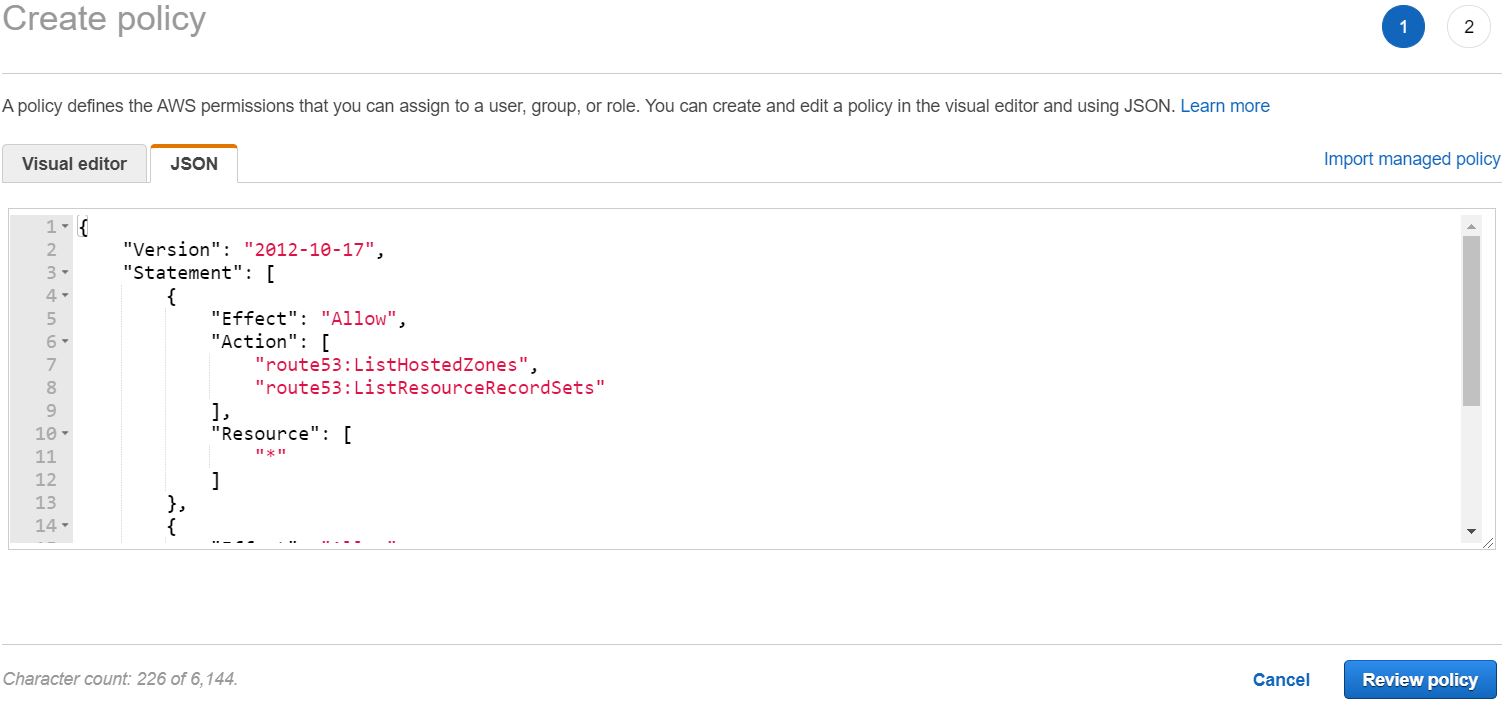
kubectl get nodes –watch



**Create Policy:**

Goto IAM Service in UI and click on **Policies.**

Click on Create Policy



{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Action": [

"route53:ListHostedZones",

"route53:ListResourceRecordSets"

],

"Resource": [

"\*"

]

},

{

"Effect": "Allow",

"Action": [

"route53:ChangeResourceRecordSets"

],

"Resource": [

"\*"

]

}

]

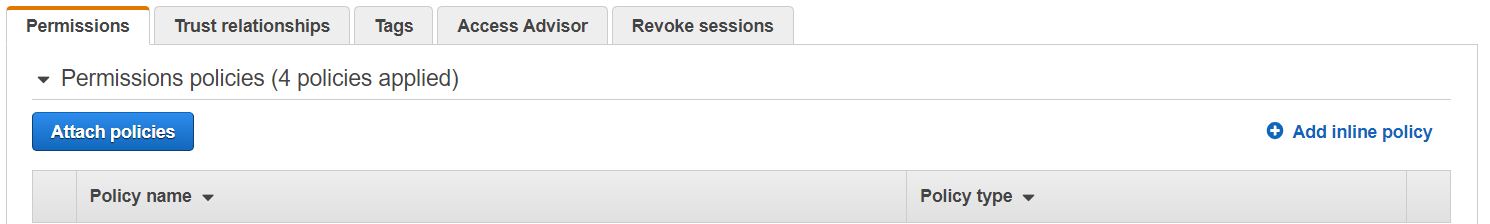
}

Click on Review policy and create

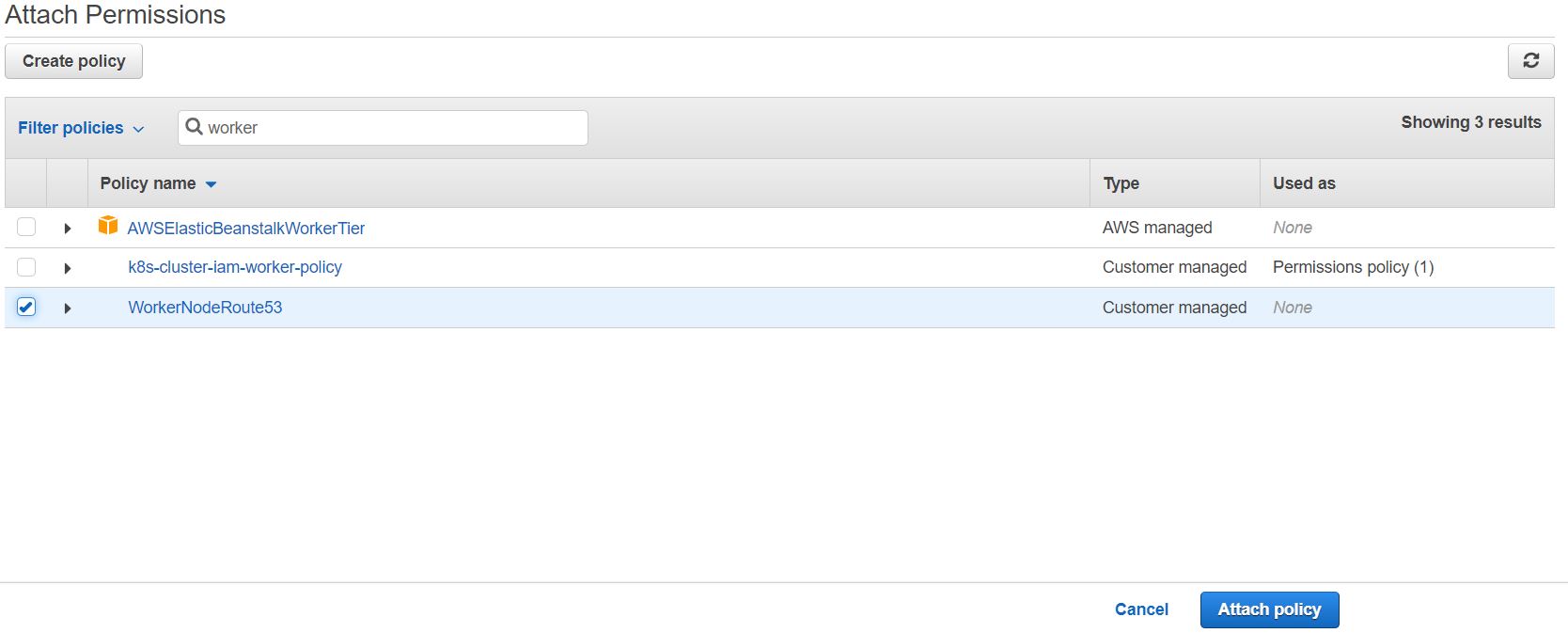
Attach this policy to Role which is create while creation of nodes of EKS cluster

D:\Naresh\snipping\Capture.JPG

Click on worker-Node instance:



Click on Attach policies

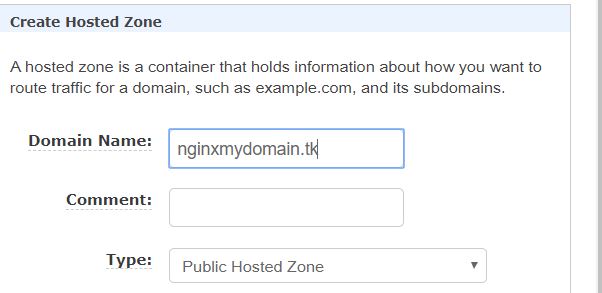


Click on Attach policy.

**Route53:**

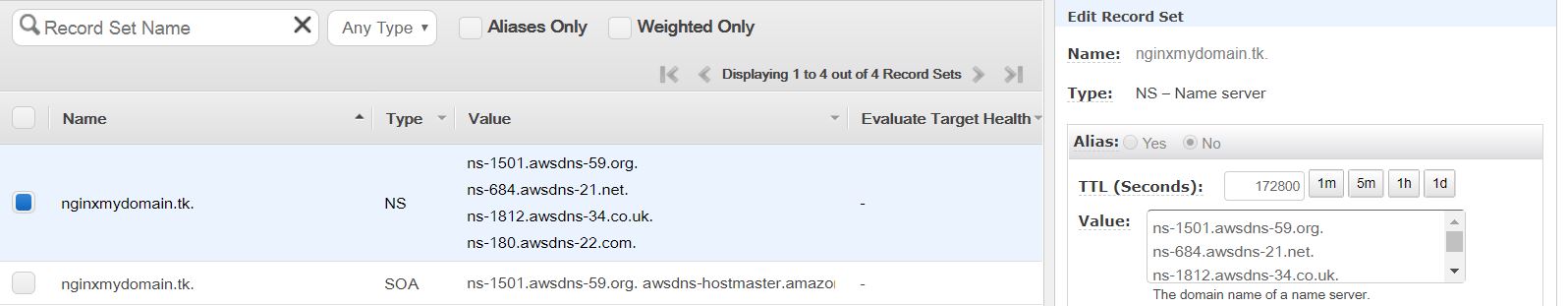
Goto Route53 with in AWS services:

Click on Hosted zone



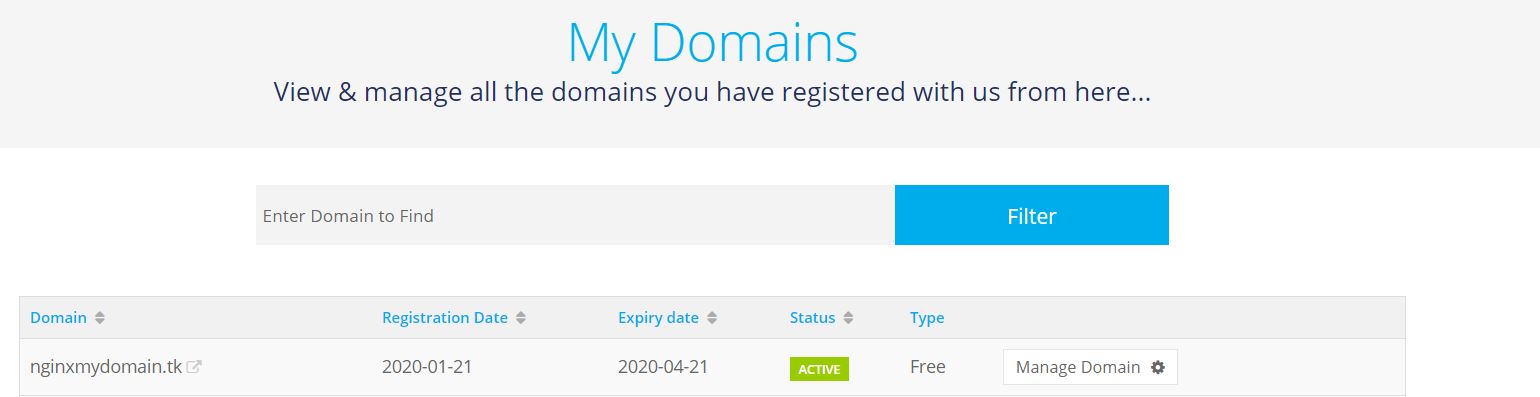
Click on Create

Here we get namespaces for Domain name

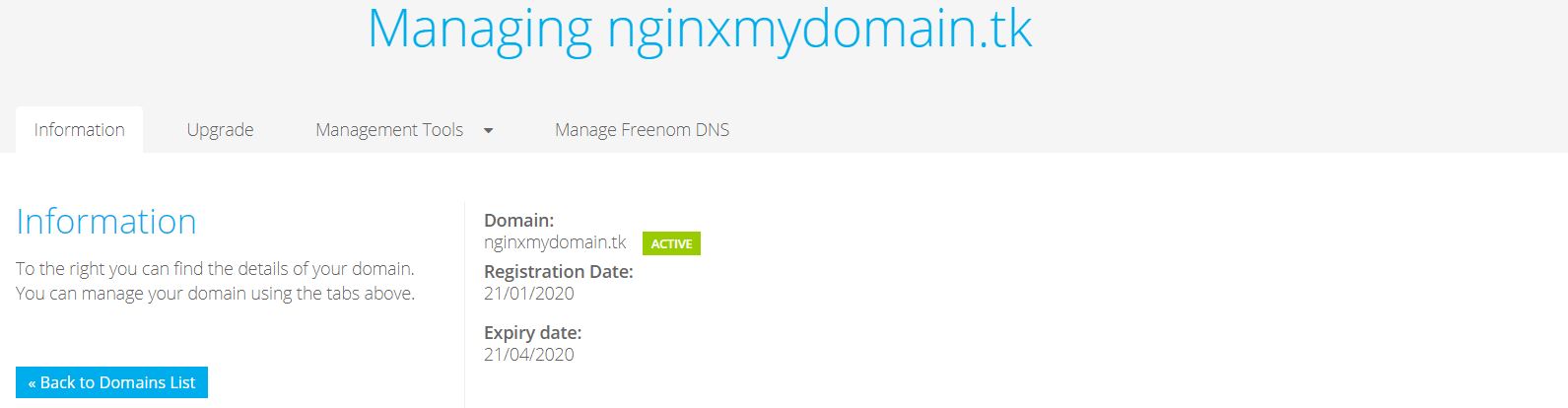


These name spaces need to be update for our domain where we purchased.

Here I taken freenom.com



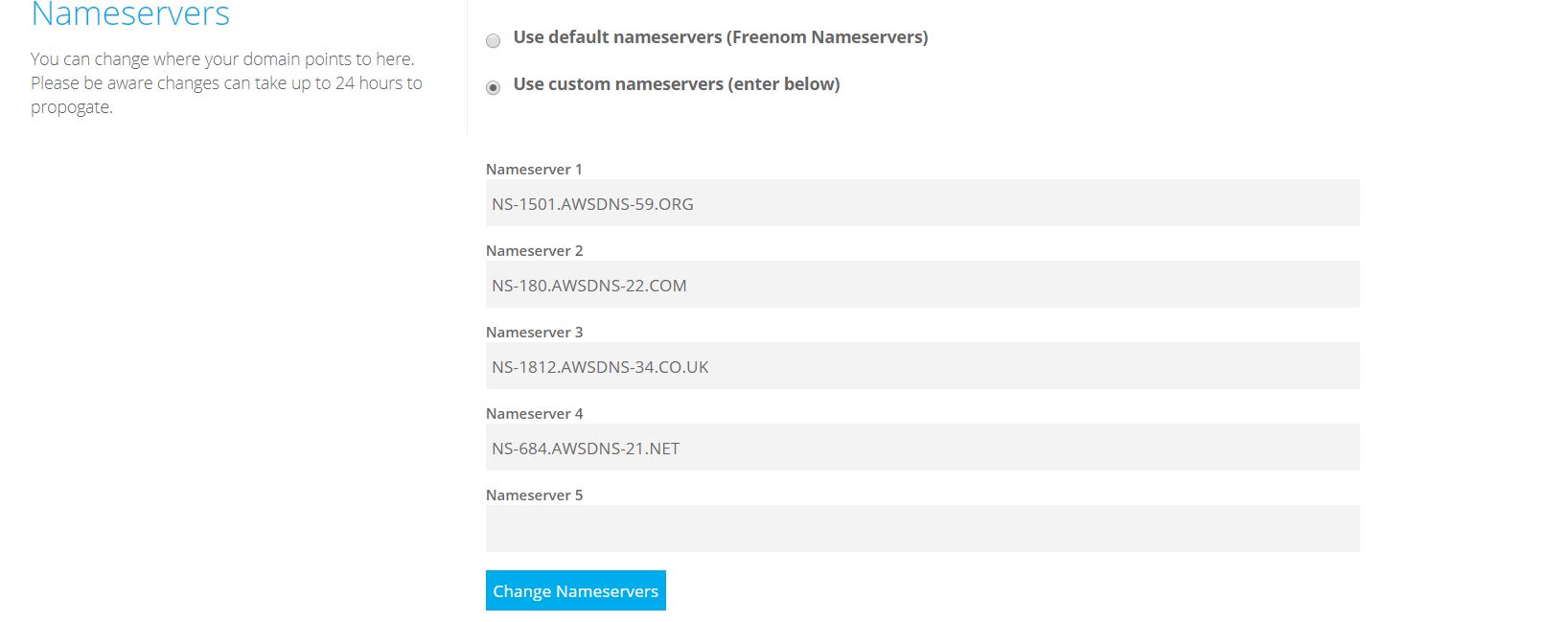
Click on manage Domain



Click on Manage Freenom DNS



Click on Edit Nameservers

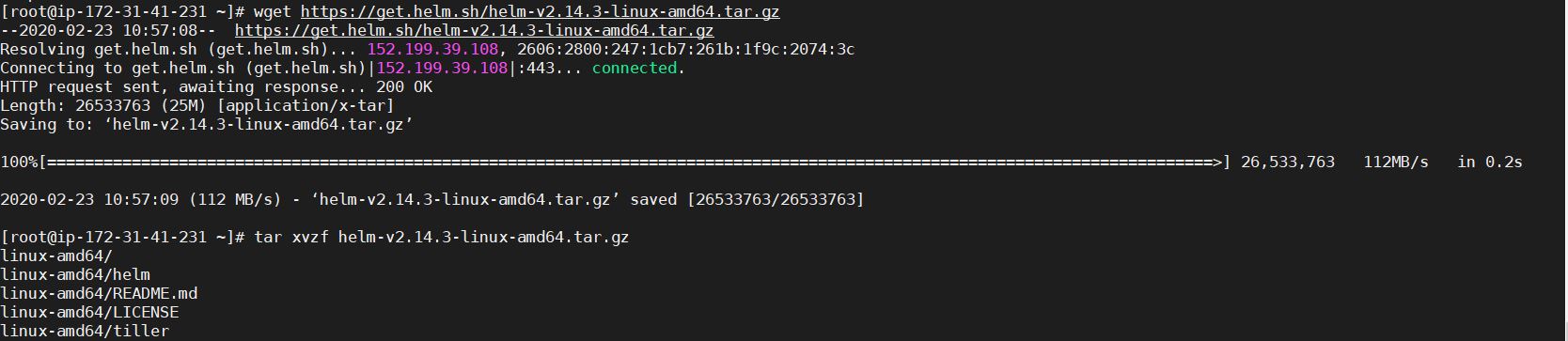


Click on Change Nameservers

**Install Helm:**

wget <https://get.helm.sh/helm-v2.14.3-linux-amd64.tar.gz>

tar xvzf helm-v2.14.3-linux-amd64.tar.gz



cd linux-amd64/

ls

mv helm /usr/bin/

which helm



To check helm version

D:\Naresh\snipping\Capture.JPG

Here tiller is not found. We need to create tiller Service account and Cluster RoleBinding

Create a with the name of rbac-config.yaml and insert below data within the file:

vi rbac-config.yaml

apiVersion: v1

kind: ServiceAccount

metadata:

name: tiller

namespace: kube-system

---

apiVersion: rbac.authorization.k8s.io/v1beta1

kind: ClusterRoleBinding

metadata:

name: tiller

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: ClusterRole

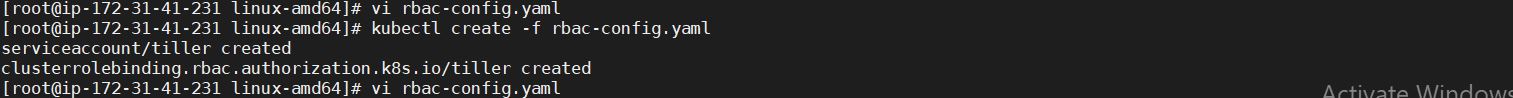
name: cluster-admin

subjects:

- kind: ServiceAccount

name: tiller

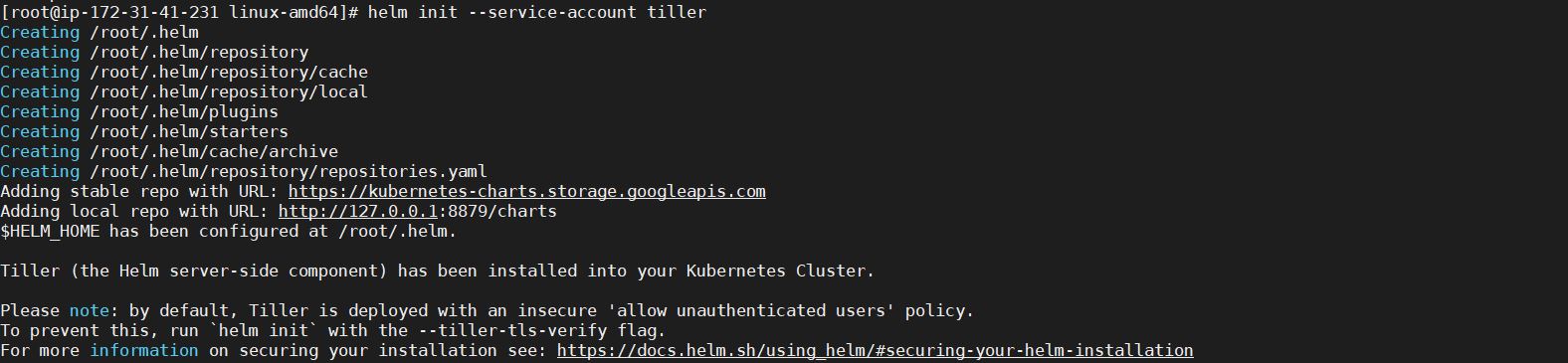
namespace: kube-system



Deploy a tiller Pod:

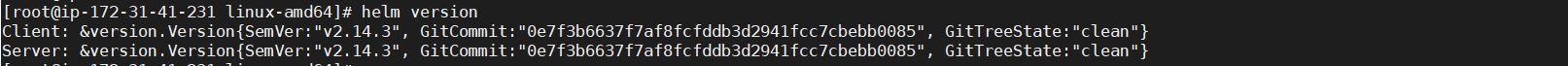
Your Helm client communicates with your kubernetes cluster through a tiller pod. To deploy your tiller, run:

helm init --service-account tiller



Confirm your tiller pod was deployed successfully:

helm version



Create a name space:

kubectl create ns astronomer

D:\Naresh\snipping\Capture.JPG

**Configure Postgres:**

To serve as the backend-db for Airflow and our API, you'll need a running Postgres instance that will be able to talk to your Kubernetes cluster. We recommend using a dedicated Postgres since Airflow will create a new database inside of that Postgres for each Airflow deployment.

If you are using RDS, you'll need the full connection string for a user that has the ability to create, delete, and updated databases and users.

If you just want to get something up and running, you can also use the PostgreSQL helm chart:

helm install --name astro-db stable/postgresql --namespace astronomer

SSL Configuration:

You'll need to obtain a wildcard SSL certificate for your domain (e.g. \*.astro.mydomain.com). This allows for web endpoint protection and encrypted communication between pods. Your options are:

Purchase a wildcard SSL certificate from your preferred vendor.

Obtain a free 90-day wildcard certificate from Let's Encrypt.

Note: You cannot use a self-signed certificate.

Obtain a Free SSL Certificate from Let's Encrypt:

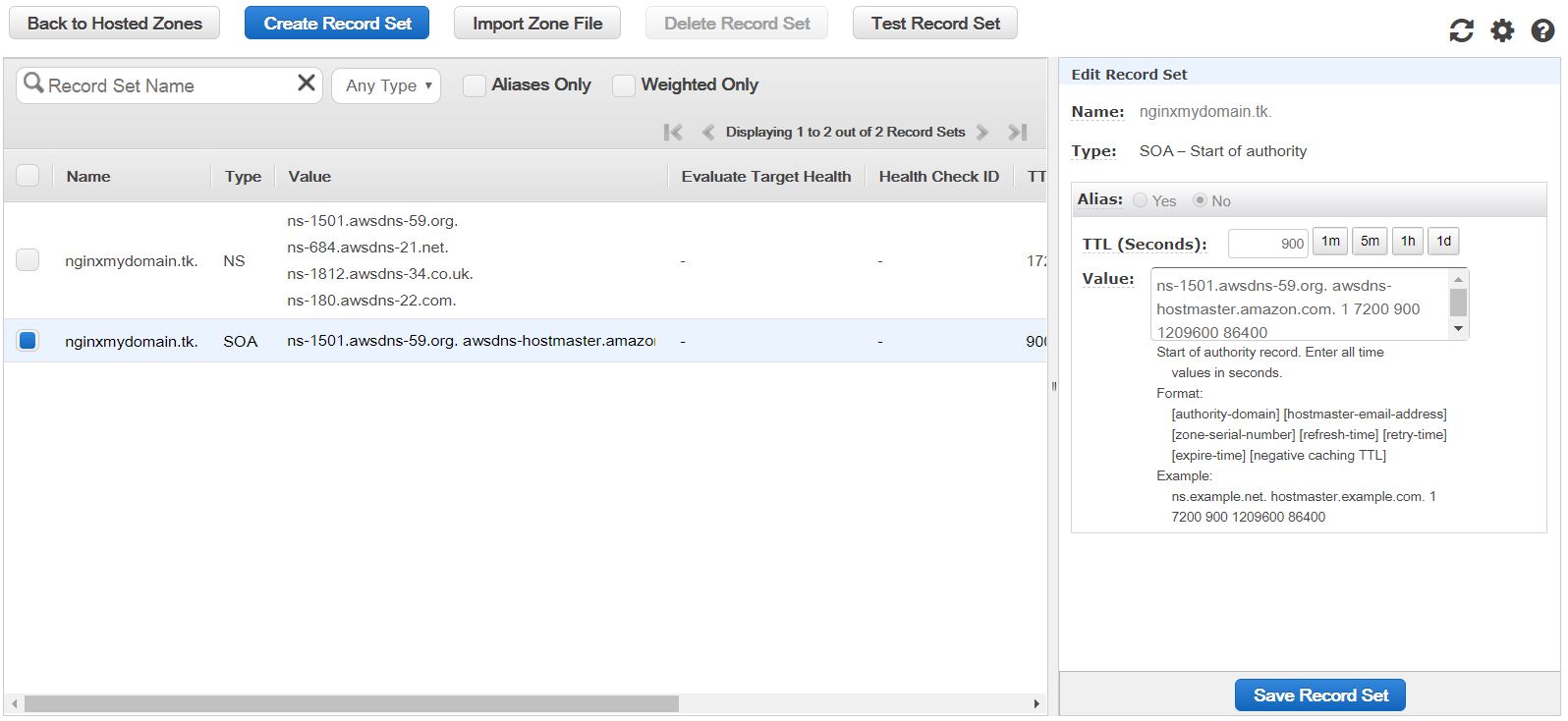
Syntax:

docker run -it --rm --name letsencrypt -v /etc/letsencrypt:/etc/letsencrypt -v /var/lib/letsencrypt:/var/lib/letsencrypt certbot/certbot:latest certonly -d "\*.astro.mydomain.com" --manual --preferred-challenges dns --server <https://acme-v02.api.letsencrypt.org/directory>

Here **mydomain.com** is **nginxmydomain.tk**

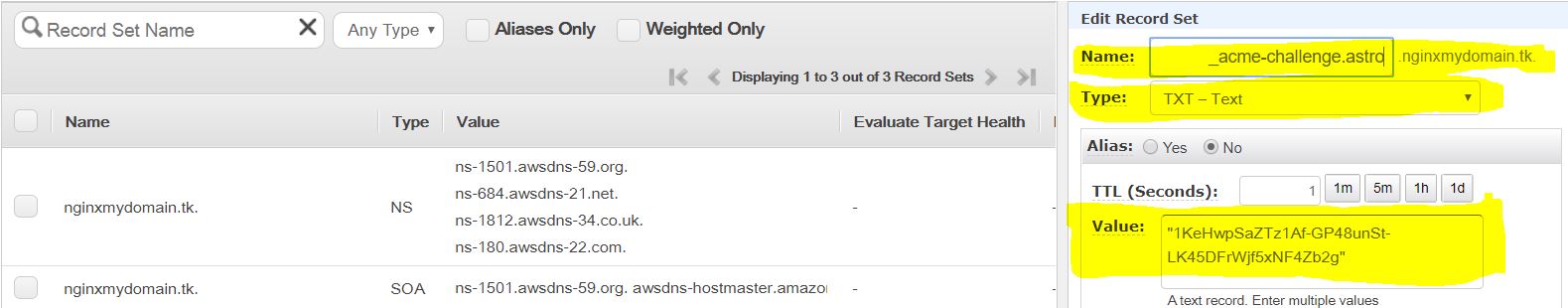
Edir mydomain and run command

docker run -it --rm --name letsencrypt -v /etc/letsencrypt:/etc/letsencrypt -v /var/lib/letsencrypt:/var/lib/letsencrypt certbot/certbot:latest certonly -d "\*.astro.nginxmydomain.tk" --manual --preferred-challenges dns --server https://acme-v02.api.letsencrypt.org/directory



Click on Create Record Set and give details which we get while running the SSLCertificate encrypt comand:





Create Kubernetes Secrets:

You'll need to create two Kubernetes secrets - one for the databases to be created and one for TLS.

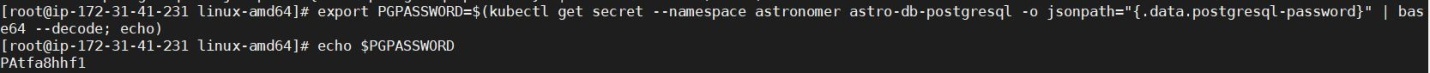
If you are using the PostgreSQL helm chart

Set an environment variable $PGPASSWORD containing your PostgreSQL database password:

export PGPASSWORD=$(kubectl get secret --namespace astronomer astro-db-postgresql -o jsonpath="{.data.postgresql-password}" | base64 --decode; echo)

Confirm your $PGPASSWORD variable is set properly:

echo $PGPASSWORD

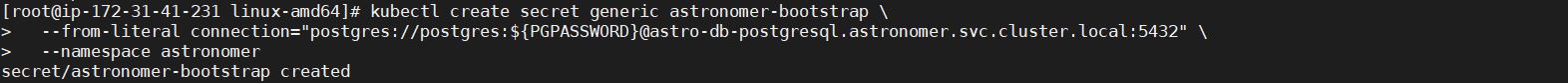


Create a Kubernetes secret named astronomer-bootstrap to hold your database connection string:

kubectl create secret generic astronomer-bootstrap \

--from-literal connection="postgres://postgres:${PGPASSWORD}@astro-db-postgresql.astronomer.svc.cluster.local:5432" \

--namespace astronomer



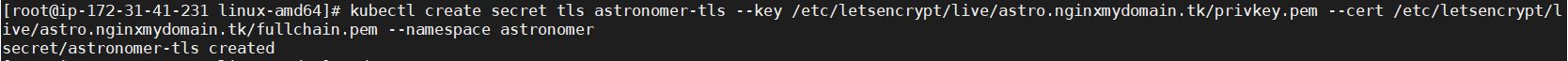
Create TLS Secret:

Create a TLS secret named astronomer-tls using the previously generated SSL certificate files.

Syntax:

kubectl create secret tls astronomer-tls --key /etc/letsencrypt/live/astro.mydomain.com/privkey.pem --cert /etc/letsencrypt/live/astro.mydomain.com/fullchain.pem --namespace <my-namespace>

kubectl create secret tls astronomer-tls --key /etc/letsencrypt/live/astro.nginxmydomain.tk/privkey.pem --cert /etc/letsencrypt/live/astro.nginxmydomain.tk/fullchain.pem --namespace astronomer



Configure your Helm Chart:

Now that your Kubernetes cluster has been configured with all prerequisites, you can deploy Astronomer!

Clone the Astronomer helm charts locally and checkout your desired branch:

git clone <https://github.com/astronomer/astronomer.git>



cd astronomer/

git branch -r

git checkout release-0.11

D:\Naresh\snipping\Capture.JPG

Files with in the release-0.11:

ls

cp ./configs/starter.yaml ./config.yaml

Set the following values in config.yaml:

baseDomain: astro.mydomain.com

tlsSecret: astronomer-tls

SMTP credentails as a houston config

Here is an example of what your config.yaml might look like:

# This is starter yaml configuation file to serve as a template for

# a production installation.

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

## Astronomer global configuration

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

global:

# Base domain for all subdomains exposed through ingress

baseDomain: astro.nginxmydomain.tk

# Name of secret containing TLS certificate

tlsSecret: astronomer-tls

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

## Nginx configuration

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

nginx:

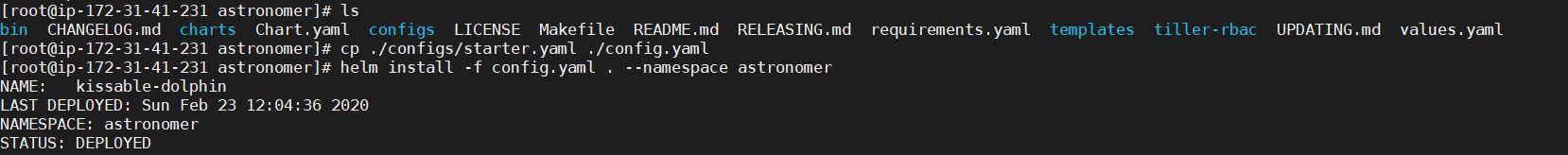
# IP address the nginx ingress should bind to

loadBalancerIP: ~

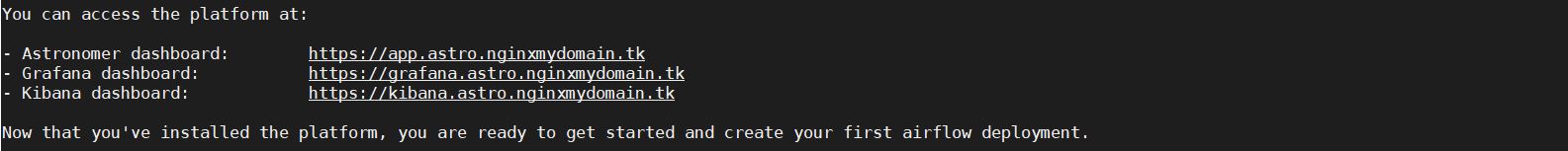
# set to 'true' when deploying to a private EKS cluster

privateLoadBalancer: false

helm install -f config.yaml . --namespace astronomer



With in this installation it gives access platforms:



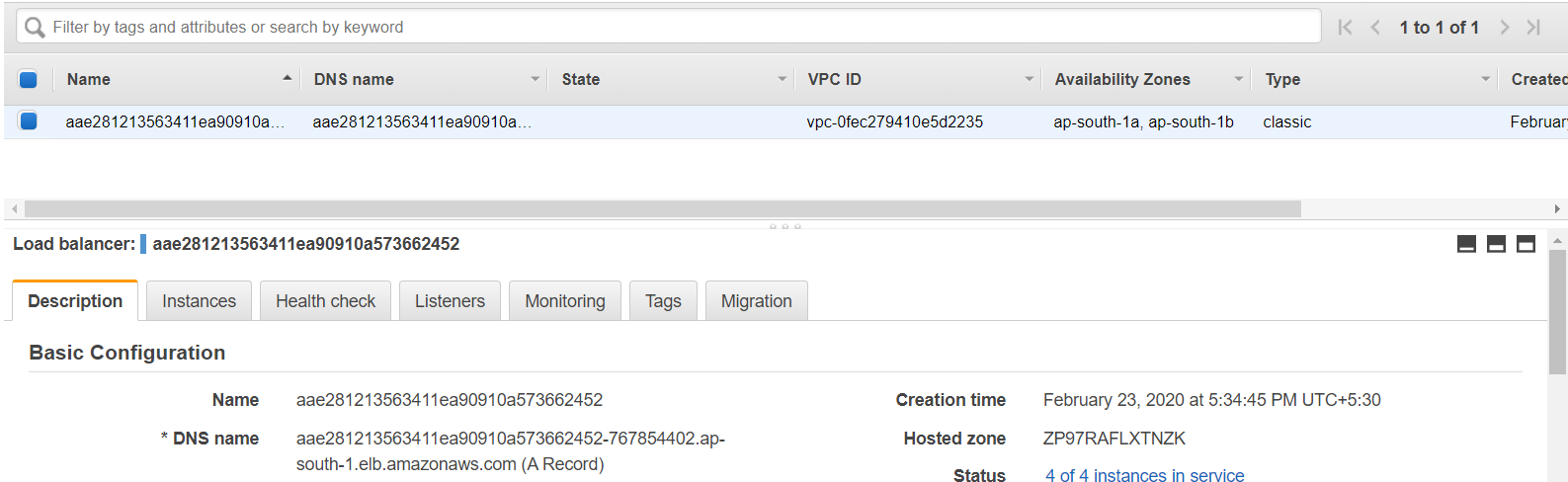
Now check pods which are running under the namespace of **“astronomer”**

kubectl get pods --all-namespaces

Also check services which are running under the namespace of **“astronomer”**

kubectl get services --all-namespaces

Now check whether the LoadBalancer created or not:

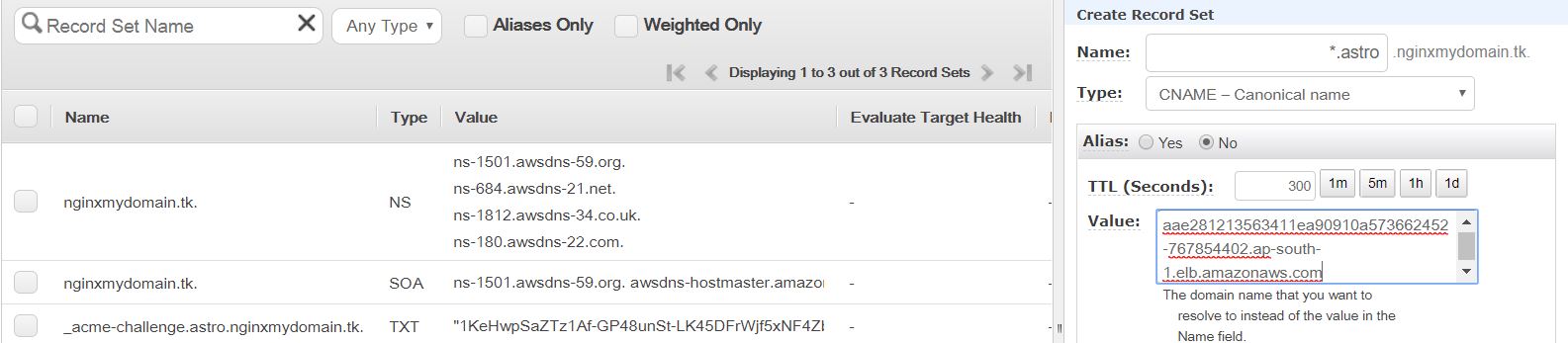


Copy DNS name and Create Record Set with below details:

Name: **\*.astro**

Type: CNAME

Value: DNS name of LoadBalancer



Click on Create

Check helm details of Astronomer:

helm ls

To delete Astronomer:

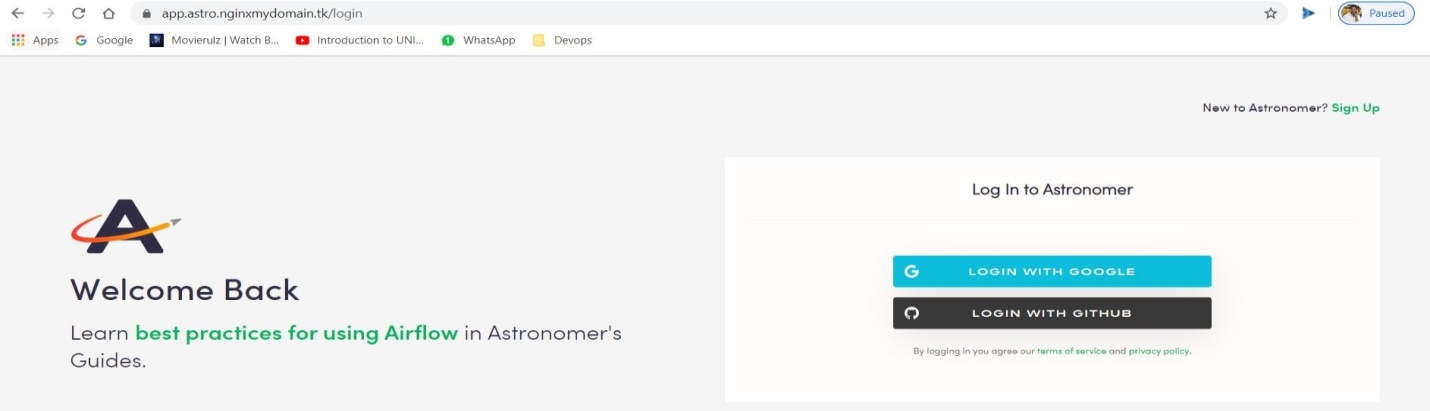
helm delete angry-heron –purge

To check logs of pods with in the namespaces:

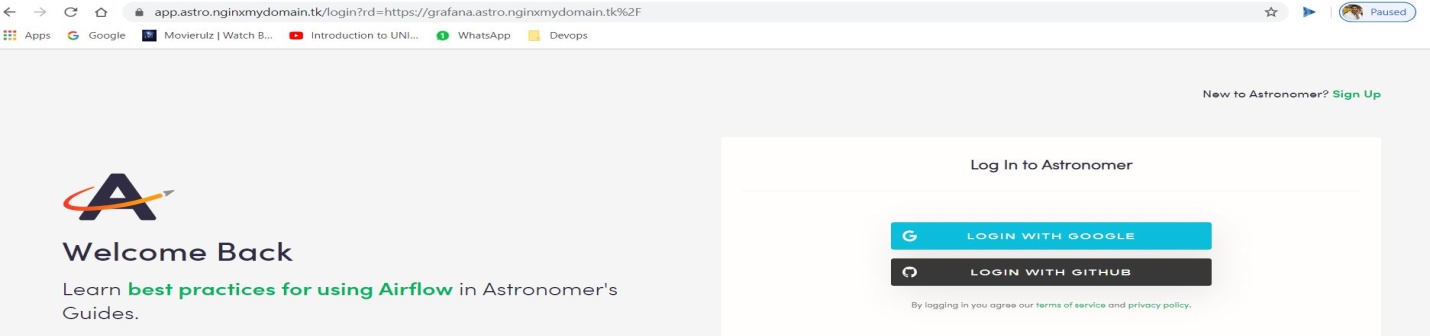
kubectl logs <pod-name> -n astronomer

**Check urls:**

<https://app.astro.nginxmydomain.tk>



<https://grafana.astro.nginxmydomain.tk>



<https://kibana.astro.nginxmydomain.tk>

