DevOps-Project-Kubernetes-Multicluster-Multicloud

By Ritesh Kumar Singh

Email Address: - riteshkumarsingh9559@gmail.com

LinkedIn: - https://www.linkedin.com/in/ritesh-kumar-singh-41113128b/

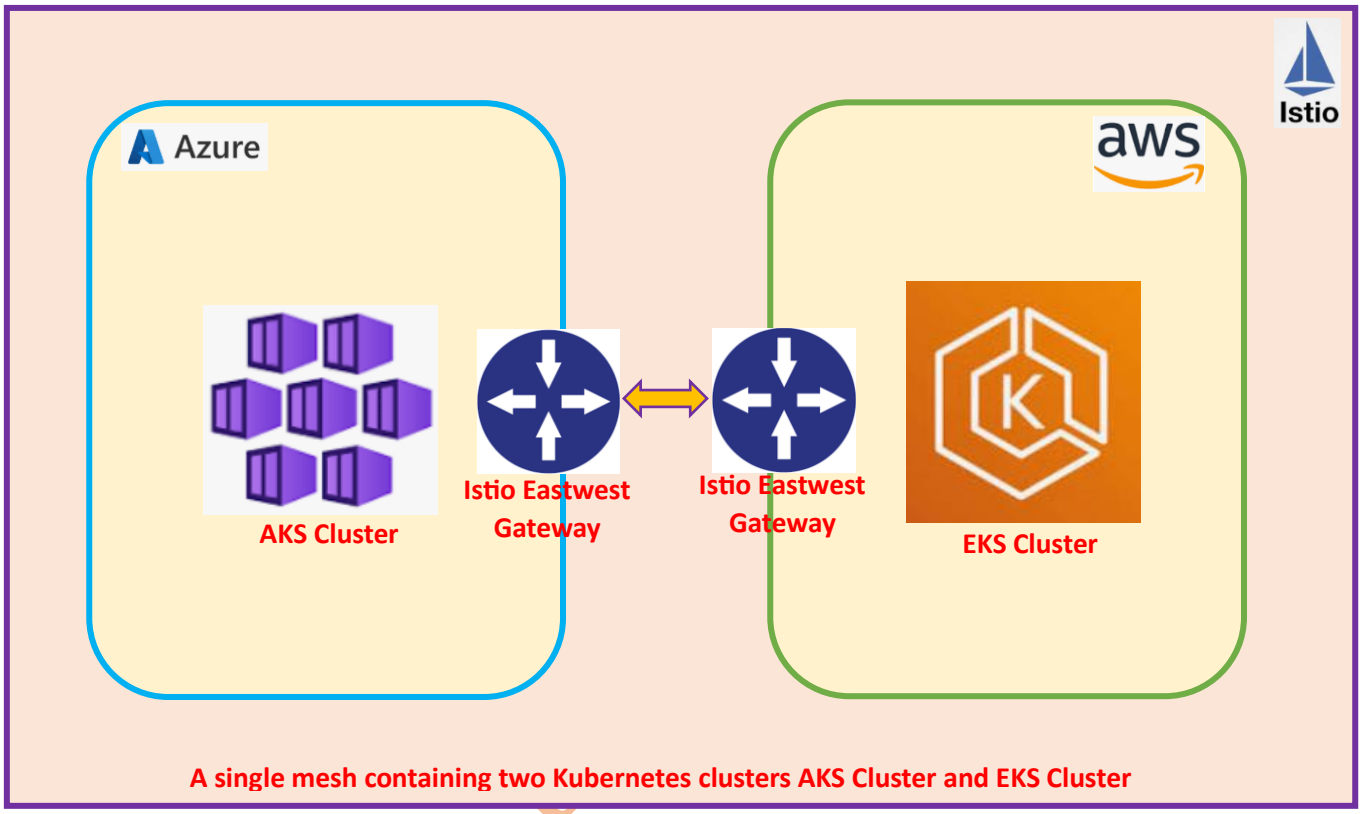
GitHub: - https://github.com/singhritesh85

या कुन्देन्दुतुषारहारधवला या शुभ्रवस्त्रावृता
या वीणावरदण्डमण्डितकरा या श्वेतपद्मासना।
या ब्रह्माच्युत शंकरप्रभृतिभिर्देवैः सदा वन्दिता
सा मां पातु सरस्वती भगवती निःशेषजाड्यापहा ॥

DevOps-Project-Kubernetes-Multicluster-Multicloud

Module 1: DevOps-Project-Kubernetes-Multicluster-using-Istio-and-DisasterRecovery-Multicloud

Kubernetes Multicluster refers to managing and deploying multiple kubernetes Clusters as a single and unified platform.



A single mesh containing two Kubernetes clusters AKS Cluster and EKS Cluster

I don not want to make this concept difficult by drawing mind numbing diagrams. As shown in the self-explanatory diagram drawn above, I had created a single mesh using Istio and two Kubernetes Clusters (AKS Cluster and EKS Cluster) with the help of **Istio Eastwest Gateway** which is **Gateway in Istio for internal cross-cluster communication**. I had created AKS (Azure Kubernetes Services) Cluster and EKS (Elastic Kubernetes Services) Cluster using the Terraform Script present in the GitHub Repo https://github.com/singhritesh85/DevOps-Project-Kubernetes-Multicluster-Multicloud.git. After creation of the two Kubernetes Cluster (AKS and EKS Cluster) I installed Istio in each of the Kubernetes Cluster using helm. Before installing the Istio I create the Root CA Certificate and generated intermediate certificate for each cluster. A common root CA certificate is used to establish a shared trust and enable secure communication across clusters. I installed Istio using **multi-primary on different networks**.

I installed terraform on Alma Linux2 Azure VM and State file was kept in the S3 Bucket and state lock had been achieved using the AWS DynamoDB. Before running the terraform script I ran the shell script present in the directory **terraform-multi-kubernetes-cluster-multicloud** as shown in the screenshot attached below. This shell script will install the aws cli, kubectl and helm.

```
[root@                terraform-multi-kubernetes-cluster-multicloud]# ./initial-setup.sh
```

Then I logged-out and login again ran the command aws configure as shown in the screenshot

attached below. As it is an important step to Authenticate and Authorize the user before creating resources using terraform in the AWS Account.

```
[root@                 terraform-multi-kubernetes-cluster-multicloud]# cd main/
[root@                 main]# aws configure
AWS Access Key ID [****************    ]:
AWS Secret Access Key [****************    ]:
Default region name [          ]:
Default output format [    ]:
[root@                 main]#
```

Then I installed Azure CLI and authenticated and Authorize the user as shown in the screenshot attached below.

```
[root@             main]# yum install -y https://packages.microsoft.com/config/rhel/8/packages-microsoft-prod.rpm
```

```
[root@                 main]# yum install azure-cli -y
```

```
[root@             main]# az login
To sign in, use a web browser to open the page                  and enter the code       to authenticate.
```

Then you can run the below commands

terraform init -------> initializes a working directory containing configuration files and installs plugins for required providers.

terraform validate --------> verify that terraform configuration file is correct or not

terraform plan ---------> Check which resources are going to be created.

Then you can run the command **terraform apply -auto-approve** -----> Finally, Create the resources.

After creation of the resources a kubeconfig file was generated, I did below change in the kubeconfig file as shown in the screenshot attached below.

```
    server: https://aks-cluster-dns-          .eastus.[redacted].io:443
  name: aks-cluster
- cluster:
    certificate-authority-data:

    server: https://                  .us-east-2.eks.amazonaws.com
  name: eks-demo-cluster-dev
contexts:
- context:
    cluster: aks-cluster
    user: clusterUser_aks-rg_aks-cluster
  name: aks-cluster
- context:
    cluster: eks-demo-cluster-dev
    user: arn:aws:eks:us-east-2:02        5:cluster/eks-demo-cluster-dev
  name: eks-demo-cluster-dev
current-context: aks-cluster
kind: Config
preferences: {}
users:
- name: clusterUser_aks-rg_aks-cluster
  user:
    client-certificate-data:

@@@
"~/.kube/config" 42L, 11795C                                    20,28         19%
```

It is my suggestion before any change in the kubeconfig file take a backup of the original kubeconfig file.

Install Istioctl as shown in the screenshot attached below.

```
[root@                ~]# curl -L https://istio.io/downloadIstio | sh -
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   101  100   101    0     0    371      0 --:--:-- --:--:-- --:--:--   369
100  5124  100  5124    0     0  17488      0 --:--:-- --:--:-- --:--:-- 17488

Downloading istio-1.25.2 from https://github.com/istio/istio/releases/download/1.25.2/istio-1.25.2-linux-amd64.tar.gz ...

Istio 1.25.2 download complete!

The Istio release archive has been downloaded to the istio-1.25.2 directory.

To configure the istioctl client tool for your workstation,
add the /root/istio-1.25.2/bin directory to your environment path variable with:
        export PATH="$PATH:/root/istio-1.25.2/bin"

Begin the Istio pre-installation check by running:
         istioctl x precheck

Try Istio in ambient mode
       https://istio.io/latest/docs/ambient/getting-started/
Try Istio in sidecar mode
       https://istio.io/latest/docs/setup/getting-started/
Install guides for ambient mode
       https://istio.io/latest/docs/ambient/install/
Install guides for sidecar mode
       https://istio.io/latest/docs/setup/install/

Need more information? Visit https://istio.io/latest/docs/
```

```
[root@                ~]# cat ~/.bashrc
# .bashrc

# User specific aliases and functions

alias rm='rm -i'
alias cp='cp -i'
alias mv='mv -i'

# Source global definitions
if [ -f /etc/bashrc ]; then
        . /etc/bashrc
fi

PATH=/usr/local/sbin:/sbin:/bin:/usr/sbin:/usr/bin:/root/bin:/usr/local/bin
PATH="$PATH:/root/istio-1.25.2/bin"
```

I created first the **istio-system** namespace in both the clusters (AKS and EKS Cluster) as shown in the screenshot attached below.

kubectl create ns istio-system --context=aks-cluster

kubectl create ns istio-system --context=eks-demo-cluster-dev

```
[root@                ~]# kubectl create ns istio-system --context=aks-cluster
namespace/istio-system created
[root@                ~]# kubectl create ns istio-system --context=eks-demo-cluster-dev
namespace/istio-system created
```

To create the Root CA certificate and certificate for each of the clusters I used make command and I install the same as shown in the screenshot attached below.

yum groupinstall 'Development Tools' -y

```
[root@                    ~]# yum groupinstall 'Development Tools' -y
```

It will install Java-1.8 which you can uninstall later after creating the Root CA certificate and certificate for each of the clusters.

```
[root@                    ~]# java -version
openjdk version "1.8.0_452"
OpenJDK Runtime Environment (build 1.8.0_452-b09)
OpenJDK 64-Bit Server VM (build 25.452-b09, mixed mode)
```

I used below command to create the Root CA certificate and certificate for each of the clusters.

mkdir certs

cd certs/

make -f ../istio-1.25.2/tools/certs/Makefile.selfsigned.mk root-ca

make -f ../istio-1.25.2/tools/certs/Makefile.selfsigned.mk aks-cluster-cacerts

make -f ../istio-1.25.2/tools/certs/Makefile.selfsigned.mk eks-demo-cluster-dev-cacerts

```
[root@                    ~]# mkdir certs
[root@                    ~]# cd certs/
[root@                    certs]# make -f ../istio-1.25.2/tools/certs/Makefile.selfsigned.mk root-ca
generating root-key.pem
Generating RSA private key, 4096 bit long modulus (2 primes)
...................
.......++++
....++++
e is 65537 (0x010001)
generating root-cert.csr
generating root-cert.pem
Signature ok
subject=O = Istio, CN = Root CA
Getting Private key
```

```
[root@                    certs]# make -f ../istio-1.25.2/tools/certs/Makefile.selfsigned.mk aks-cluster-cacerts
generating aks-cluster/ca-key.pem
Generating RSA private key, 4096 bit long modulus (2 primes)
............
..............................
........................
...++++
.................................
.............++++
e is 65537 (0x010001)
generating aks-cluster/cluster-ca.csr
generating aks-cluster/ca-cert.pem
Signature ok
subject=O = Istio, CN = Intermediate CA, L = aks-cluster
Getting CA Private Key
generating aks-cluster/cert-chain.pem
Intermediate inputs stored in aks-cluster/
done
rm aks-cluster/cluster-ca.csr aks-cluster/intermediate.conf
```

```
[root@                    certs]# make -f ../istio-1.25.2/tools/certs/Makefile.selfsigned.mk eks-demo-cluster-dev-cacerts
generating eks-demo-cluster-dev/ca-key.pem
Generating RSA private key, 4096 bit long modulus (2 primes)
.....................................................++++
.....................................................................................................................................................................
.....................................................................................................................................................................
.....................................................++++
e is 65537 (0x010001)
generating eks-demo-cluster-dev/cluster-ca.csr
generating eks-demo-cluster-dev/ca-cert.pem
Signature ok
subject=O = Istio, CN = Intermediate CA, L = eks-demo-cluster-dev
Getting CA Private Key
generating eks-demo-cluster-dev/cert-chain.pem
Intermediate inputs stored in eks-demo-cluster-dev/
done
rm eks-demo-cluster-dev/intermediate.conf eks-demo-cluster-dev/cluster-ca.csr
```

Then created the kubernetes secrets **cacerts** in the namespace **istio-system** in each of the kubernetes cluster (AKS Cluster and EKS Cluster) as shown in the screenshot attached below.

kubectl create secret generic cacerts -n istio-system --context=aks-cluster --from-file=certs/aks-cluster/ca-cert.pem --from-file=certs/aks-cluster/ca-key.pem --from-file=certs/aks-cluster/root-cert.pem --from-file=certs/aks-cluster/cert-chain.pem

kubectl create secret generic cacerts -n istio-system --context=eks-demo-cluster-dev --from-file=certs/eks-demo-cluster-dev/ca-cert.pem --from-file=certs/eks-demo-cluster-dev/ca-key.pem --from-file=certs/eks-demo-cluster-dev/root-cert.pem --from-file=certs/eks-demo-cluster-dev/cert-chain.pem

```
[root@                    ~]# kubectl create secret generic cacerts -n istio-system --context=aks-cluster --from-file=certs/aks-cluster/ca-cert.pem --from-file=c
erts/aks-cluster/ca-key.pem --from-file=certs/aks-cluster/root-cert.pem --from-file=certs/aks-cluster/cert-chain.pem
secret/cacerts created
[root@                    ~]# kubectl create secret generic cacerts -n istio-system --context=eks-demo-cluster-dev --from-file=certs/eks-demo-cluster-dev/ca-cert
.pem --from-file=certs/eks-demo-cluster-dev/ca-key.pem --from-file=certs/eks-demo-cluster-dev/root-cert.pem --from-file=certs/eks-demo-cluster-dev/cert-chain.
pem
secret/cacerts created
```

For Istio installation I installed istio-base, istiod and then istio Eastwest gateway on AKS Cluster as shown in the screenshot attached below.

kubectl --context=aks-cluster get namespace istio-system && kubectl --context=aks-cluster label namespace istio-system topology.istio.io/network=network1

helm repo add istio https://istio-release.storage.googleapis.com/charts --kube-context=aks-cluster

helm repo update --kube-context=aks-cluster

helm install istio-base istio/base -n istio-system --kube-context aks-cluster

helm install istiod istio/istiod -n istio-system --kube-context aks-cluster --set global.meshID=mesh1 --set global.multiCluster.clusterName=aks-cluster --set global.network=network1

helm install istio-eastwestgateway istio/gateway -n istio-system --kube-context aks-cluster --set name=istio-eastwestgateway --set networkGateway=network1

kubectl --context=aks-cluster get svc istio-eastwestgateway -n istio-system

I applied the label **istio-system topology.istio.io/network=network1** to the namespace **istio-system** in AKS Cluster as shown in the screenshot attached below.

```
[root@ ~]# kubectl --context=aks-cluster get namespace istio-system && kubectl --context=aks-cluster label namespace istio-system topology.ist
io.io/network=network1
NAME           STATUS   AGE
istio-system   Active   32m
namespace/istio-system labeled
```

```
[root@ ~]# helm repo add istio https://istio-release.storage.googleapis.com/charts --kube-context=aks-cluster
"istio" already exists with the same configuration, skipping
[root@ ~]# helm repo update --kube-context=aks-cluster
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "ingress-nginx" chart repository
...Successfully got an update from the "istio" chart repository
Update Complete. ⎈Happy Helming!⎈
[root@ ~]# helm install istio-base istio/base -n istio-system --kube-context aks-cluster
```

```
[root@ ~]# helm install istiod istio/istiod -n istio-system --kube-context aks-cluster --set global.meshID=mesh1 --set global.multiCluster.clu
sterName=aks-cluster --set global.network=network1
```

```
[root@ ~]# helm install istio-eastwestgateway istio/gateway -n istio-system --kube-context aks-cluster --set name=istio-eastwestgateway --set
networkGateway=network1
```

```
[root@ ~]# kubectl --context=aks-cluster get svc istio-eastwestgateway -n istio-system
NAME                    TYPE           CLUSTER-IP    EXTERNAL-IP   PORT(S)                                                      AGE
istio-eastwestgateway   LoadBalancer   10.  .33   <pending>     15021:30554/TCP,15443:31089/TCP,15012:30874/TCP,15017:30646/TCP   5s
[root@ ~]# kubectl --context=aks-cluster get svc istio-eastwestgateway -n istio-system --watch
NAME                    TYPE           CLUSTER-IP    EXTERNAL-IP   PORT(S)                                                      AGE
istio-eastwestgateway   LoadBalancer   10.  .33   4.  .59   15021:30554/TCP,15443:31089/TCP,15012:30874/TCP,15017:30646/TCP   12s
```

```
[root@ ~]# cat expose-services.yaml
apiVersion: networking.istio.io/v1
kind: Gateway
metadata:
  name: cross-network-gateway
spec:
  selector:
    istio: eastwestgateway
  servers:
    - port:
        number: 15443
        name: tls
        protocol: TLS
      tls:
        mode: AUTO_PASSTHROUGH
      hosts:
        - "*.local"
```

kubectl --context=aks-cluster apply -n istio-system -f expose-services.yaml

```
[root@ ~]# kubectl --context=aks-cluster apply -n istio-system -f expose-services.yaml
gateway.networking.istio.io/cross-network-gateway created
```

For Istio installation on EKS Cluster I installed istio-base, istiod and then istio Eastwest gateway on as shown in the screenshot attached below.

kubectl --context=eks-demo-cluster-dev get namespace istio-system && kubectl --context=eks-demo-cluster-dev label namespace istio-system topology.istio.io/network=network2

helm repo add istio https://istio-release.storage.googleapis.com/charts --kube-context=eks-demo-cluster-dev

helm repo update --kube-context=eks-demo-cluster-dev

helm install istio-base istio/base -n istio-system --kube-context eks-demo-cluster-dev

helm install istiod istio/istiod -n istio-system --kube-context eks-demo-cluster-dev --set global.meshID=mesh1 --set global.multiCluster.clusterName=eks-demo-cluster-dev --set global.network=network2

helm install istio-eastwestgateway istio/gateway -n istio-system --kube-context eks-demo-cluster-dev --set name=istio-eastwestgateway --set networkGateway=network2

kubectl --context=eks-demo-cluster-dev get svc istio-eastwestgateway -n istio-system

I applied the label **istio-system topology.istio.io/network=network2** to the namespace **istio-system** in EKS Cluster as shown in the screenshot attached below.

cat expose-services.yaml

```
apiVersion: networking.istio.io/v1
kind: Gateway
metadata:
  name: cross-network-gateway
spec:
  selector:
    istio: eastwestgateway
  servers:
    - port:
        number: 15443
        name: tls
        protocol: TLS
      tls:
        mode: AUTO_PASSTHROUGH
      hosts:
        - "*.local"
```



```
[root@                    ~]# cat expose-services.yaml
apiVersion: networking.istio.io/v1
kind: Gateway
metadata:
  name: cross-network-gateway
spec:
  selector:
    istio: eastwestgateway
  servers:
    - port:
        number: 15443
        name: tls
        protocol: TLS
      tls:
        mode: AUTO_PASSTHROUGH
      hosts:
        - "*.local"
```

kubectl --context=eks-demo-cluster-dev apply -n istio-system -f expose-services.yaml

```
[root@                ~]# kubectl --context=eks-demo-cluster-dev apply -n istio-system -f expose-services.yaml
gateway.networking.istio.io/cross-network-gateway created
```

==========

==========

Enable endpoint discovery in AKS and EKS Cluster

==========

==========

istioctl create-remote-secret --context=aks-cluster --name=aks-cluster | kubectl apply -f - --context=eks-demo-cluster-dev

istioctl create-remote-secret --context=eks-demo-cluster-dev --name=eks-demo-cluster-dev | kubectl apply -f - --context=aks-cluster

```
[root@                ~]# istioctl create-remote-secret --context=aks-cluster --name=aks-cluster | kubectl apply -f - --context=eks-demo-cluster-dev
secret/istio-remote-secret-aks-cluster created
[root@                ~]# istioctl create-remote-secret --context=eks-demo-cluster-dev --name=eks-demo-cluster-dev | kubectl apply -f - --context=aks-cluster
secret/istio-remote-secret-eks-demo-cluster-dev created
```

Now verify that istiod can communicate with the remote cluster control plan

istioctl remote-clusters --context=aks-cluster

istioctl remote-clusters --context=eks-demo-cluster-dev

```
[root@                ~]# istioctl remote-clusters --context=aks-cluster
NAME                  SECRET                                                  STATUS     ISTIOD
aks-cluster                                                                   synced     istiod-b                b
eks-demo-cluster-dev  istio-system/istio-remote-secret-eks-demo-cluster-dev   synced     istiod-b                b
[root@                ~]# istioctl remote-clusters --context=eks-demo-cluster-dev
NAME                  SECRET                                                  STATUS     ISTIOD
eks-demo-cluster-dev                                                          synced     istiod-6                q
aks-cluster           istio-system/istio-remote-secret-aks-cluster            synced     istiod-6                q
```

Finally, Multicluster Kubernetes Cluster (with AKS and EKS Clusters) had been created using Istio with **multi-primary on different networks**.

Now I will demonstrate the **Disaster Recovery using multi-cluster**. To do so I will deploy the microservice in both the kubernetes cluster AKS and EKS cluster using the kubernetes manifests file present in GitHub Repo https://github.com/singhritesh85/microservices-demo.git at the path **microservices-demo/release/kubernetes-manifests.yaml**.

git clone https://github.com/singhritesh85/microservices-demo.git

cd microservices-demo/release/

kubectl create ns microservice --context=aks-cluster

kubectl create ns microservice --context=eks-demo-cluster-dev

kubectl label --context=aks-cluster namespace microservice istio-injection=enabled

kubectl label --context=eks-demo-cluster-dev namespace microservice istio-injection=enabled

kubectl apply -f kubernetes-manifests.yaml -n microservice --context=aks-cluster

kubectl apply -f kubernetes-manifests.yaml -n microservice --context=eks-demo-cluster-dev

```
[root@                    ~]# git clone https://github.com/singhritesh85/microservices-demo.git
Cloning into 'microservices-demo'...
remote: Enumerating objects: 9164, done.
remote: Total 9164 (delta 0), reused 0 (delta 0), pack-reused 9164 (from 1)
Receiving objects: 100% (9164/9164), 30.86 MiB | 69.31 MiB/s, done.
Resolving deltas: 100% (6763/6763), done.
[root@                    ~]# cd microservices-demo/release/
[root@                    release]# kubectl create ns microservice --context=aks-cluster
namespace/microservice created
[root@                    release]# kubectl create ns microservice --context=eks-demo-cluster-dev
namespace/microservice created
[root@                    release]# kubectl label --context=aks-cluster namespace microservice istio-injection=enabled
namespace/microservice labeled
[root@                    release]# kubectl label --context=eks-demo-cluster-dev namespace microservice istio-injection=enabled
namespace/microservice labeled
[root@                    release]# kubectl apply -f kubernetes-manifests.yaml -n microservice --context=aks-cluster
```

```
[root@                    release]# kubectl apply -f kubernetes-manifests.yaml -n microservice --context=eks-demo-cluster-dev
```

```
[root@                    release]# kubectl get pods -n microservice --context=aks-cluster
NAME                                     READY   STATUS    RESTARTS   AGE
adservice-54c7b78694-jxvwp               0/2     Pending   0          4m27s
cartservice-58bb7599c9-6b88v             2/2     Running   0          4m27s
checkoutservice-844b55f77b-pxdjb         2/2     Running   0          4m27s
currencyservice-85964959db-hqbnh         2/2     Running   0          4m27s
emailservice-79795c57fb-q8wbk            2/2     Running   0          4m27s
frontend-dc845d658-4rb8s                 2/2     Running   0          4m27s
loadgenerator-67977fd76-2z69r            2/2     Running   0          4m27s
paymentservice-68f64fcf9b-d9hbb          2/2     Running   0          4m27s
productcatalogservice-56977877f4-zkl69   2/2     Running   0          4m27s
recommendationservice-d77bb8b48-l5qms    2/2     Running   0          4m27s
redis-cart-675444f959-qdxs5              2/2     Running   0          4m27s
shippingservice-68c77f56f8-8cn59         2/2     Running   0          4m27s
[root@                    release]# kubectl get pods -n microservice --context=eks-demo-cluster-dev
NAME                                     READY   STATUS    RESTARTS   AGE
adservice-54c7b78694-g4hdd               2/2     Running   0          4m25s
cartservice-58bb7599c9-57fv4             2/2     Running   0          4m25s
checkoutservice-844b55f77b-slsfp         2/2     Running   0          4m26s
currencyservice-85964959db-b24xk         2/2     Running   0          4m25s
emailservice-79795c57fb-n4bw9            2/2     Running   0          4m26s
frontend-dc845d658-7psq9                 2/2     Running   0          4m26s
loadgenerator-67977fd76-pp48r            2/2     Running   0          4m25s
paymentservice-68f64fcf9b-bgrzf          2/2     Running   0          4m26s
productcatalogservice-56977877f4-lqggb   2/2     Running   0          4m26s
recommendationservice-d77bb8b48-m26bt    2/2     Running   0          4m26s
redis-cart-675444f959-dnv74              2/2     Running   0          4m25s
shippingservice-68c77f56f8-6xw94         2/2     Running   0          4m25s
```

I applied the label **istio-injection=enabled** in the **namespace microservice** on AKS and EKS Cluster due to which the Envoy Proxy containers was created to all the pods in this namespace.

```
[root@Terraform-Server release]# kubectl get svc -n microservice --context=aks-cluster
NAME                  TYPE           CLUSTER-IP     EXTERNAL-IP    PORT(S)        AGE
adservice             ClusterIP      10.[redacted].162   <none>         9555/TCP       5m45s
cartservice           ClusterIP      10.[redacted].35    <none>         7070/TCP       5m45s
checkoutservice       ClusterIP      10.[redacted].165   <none>         5050/TCP       5m46s
currencyservice       ClusterIP      10.[redacted].18    <none>         7000/TCP       5m45s
emailservice          ClusterIP      10.[redacted].110   <none>         5000/TCP       5m46s
frontend              ClusterIP      10.[redacted].89    <none>         80/TCP         5m45s
frontend-external     LoadBalancer   10.[redacted].213   4.[redacted].200   80:30840/TCP   5m45s
paymentservice        ClusterIP      10.[redacted].107   <none>         50051/TCP      5m45s
productcatalogservice ClusterIP      10.[redacted].96    <none>         3550/TCP       5m45s
recommendationservice ClusterIP      10.[redacted].248   <none>         8080/TCP       5m45s
redis-cart            ClusterIP      10.[redacted].131   <none>         6379/TCP       5m45s
shippingservice       ClusterIP      10.[redacted].255   <none>         50051/TCP      5m45s
[root@Terraform-Server release]# kubectl get svc -n microservice --context=eks-demo-cluster-dev
NAME                  TYPE           CLUSTER-IP       EXTERNAL-IP                                                                PORT(S)        AGE
adservice             ClusterIP      192.[redacted].107   <none>                                                                     9555/TCP       5m42s
cartservice           ClusterIP      192.[redacted].96    <none>                                                                     7070/TCP       5m42s
checkoutservice       ClusterIP      192.[redacted].115   <none>                                                                     5050/TCP       5m43s
currencyservice       ClusterIP      192.[redacted].120   <none>                                                                     7000/TCP       5m42s
emailservice          ClusterIP      192.[redacted].220   <none>                                                                     5000/TCP       5m43s
frontend              ClusterIP      192.[redacted].70    <none>                                                                     80/TCP         5m43s
frontend-external     LoadBalancer   192.[redacted].41    a[redacted]5.us-east-2.elb.amazonaws.com   80:32484/TCP   5m43s
paymentservice        ClusterIP      192.[redacted].220   <none>                                                                     50051/TCP      5m43s
productcatalogservice ClusterIP      192.[redacted].10    <none>                                                                     3550/TCP       5m43s
recommendationservice ClusterIP      192.[redacted].102   <none>                                                                     8080/TCP       5m43s
redis-cart            ClusterIP      192.[redacted].91    <none>                                                                     6379/TCP       5m42s
shippingservice       ClusterIP      192.[redacted].149   <none>                                                                     50051/TCP      5m42s
```

Then I installed ingress nginx controller in EKS Cluster which I considered as the primary cluster as shown in the screenshot attached below.

kubectl create ns ingress-nginx

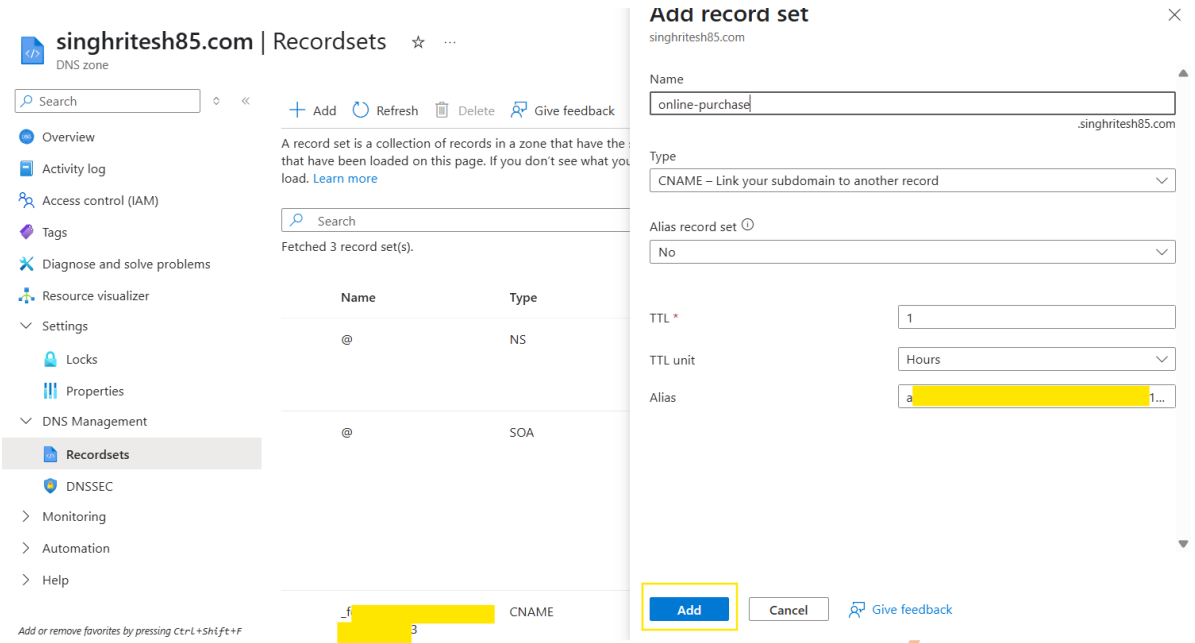
helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx

helm repo update

helm install ingress-nginx ingress-nginx/ingress-nginx -n ingress-nginx --set controller.service.annotations."service\.beta\.kubernetes\.io/aws-load-balancer-ssl-cert"=arn:aws:acm:us-east-2:02XXXXXXXXX6:certificate/XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX --set controller.service.annotations."service\.beta\.kubernetes\.io/aws-load-balancer-connection-idle-timeout"="60" --set controller.service.annotations."service\.beta\.kubernetes\.io/aws-load-balancer-cross-zone-load-balancing-enabled"="true" --set controller.service.annotations."service\.beta\.kubernetes\.io/aws-load-balancer-type"="elb" --set controller.service.annotations."service\.beta\.kubernetes\.io/aws-load-balancer-backend-protocol"=http --set controller.service.annotations."service\.beta\.kubernetes\.io/aws-load-balancer-ssl-ports"=https --set controller.service.targetPorts.https=http --set-string controller.config.use-forwarded-headers="true"

```
[root@[redacted] release]# kubectl create ns ingress-nginx

namespace/ingress-nginx created
[root@[redacted] release]# helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx
"ingress-nginx" already exists with the same configuration, skipping
[root@[redacted] release]# helm repo update
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "ingress-nginx" chart repository
...Successfully got an update from the "istio" chart repository
Update Complete. ⎈Happy Helming!⎈
[root@[redacted] release]#
[root@[redacted] release]# helm install ingress-nginx ingress-nginx/ingress-nginx -n ingress-nginx --set controller.service.annotations."service\.beta\.
kubernetes\.io/aws-load-balancer-ssl-cert"=arn:aws:acm:us-east-2:0[redacted]6:certificate/8[redacted]1 --set controller.service.annota
tions."service\.beta\.kubernetes\.io/aws-load-balancer-connection-idle-timeout"="60" --set controller.service.annotations."service\.beta\.kubernetes\.io/aws-l
oad-balancer-cross-zone-load-balancing-enabled"="true" --set controller.service.annotations."service\.beta\.kubernetes\.io/aws-load-balancer-type"="elb" --set
 controller.service.annotations."service\.beta\.kubernetes\.io/aws-load-balancer-backend-protocol"=http --set controller.service.annotations."service\.beta\.k
ubernetes\.io/aws-load-balancer-ssl-ports"=https --set controller.service.targetPorts.https=http --set-string controller.config.use-forwarded-headers="true"
```

Created the ingress rule and did the entry in Azure DNS Zone to create the Record Set for HOST with DNS Name of the LoadBalancer Service as shown in the screenshot attached below.

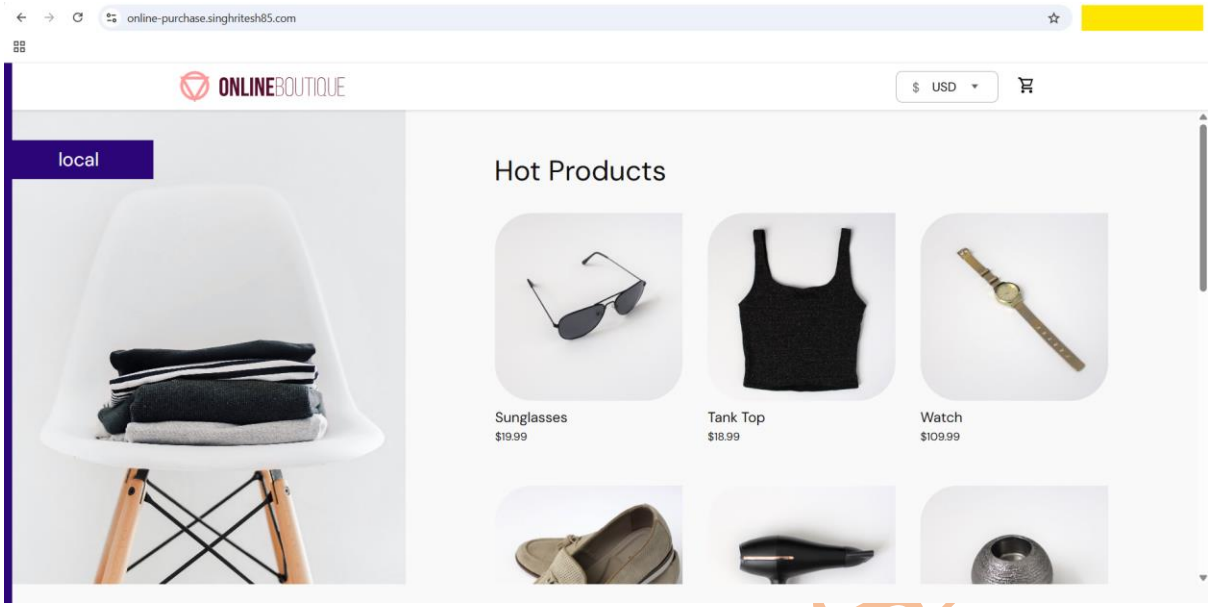



```
[root@                ~]# cat ingress-rule.yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: microservice-ingress
  namespace: microservice
  annotations:
    kubernetes.io/ingress.class: nginx
spec:
  ingressClassName: nginx
  rules:
  - host: online-purchase.singhritesh85.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: frontend-external
            port:
              number: 80
[root@                ~]# kubectl apply -f ingress-rule.yaml --context=eks-demo-cluster-dev
ingress.networking.k8s.io/microservice-ingress created
[root@                ~]# kubectl get ing -n microservice --context=eks-demo-cluster-dev
NAME                  CLASS   HOSTS                                ADDRESS   PORTS   AGE
microservice-ingress  nginx   online-purchase.singhritesh85.com              80      16s
[root@                ~]# kubectl get ing -n microservice --context=eks-demo-cluster-dev --watch
NAME                  CLASS   HOSTS                                ADDRESS   PORTS   AGE
microservice-ingress  nginx   online-purchase.singhritesh85.com              80      21s
microservice-ingress  nginx   online-purchase.singhritesh85.com   a                                    0.us-east-2.elb.amazonaws.com   80      29s
```

cat ingress-rule.yaml

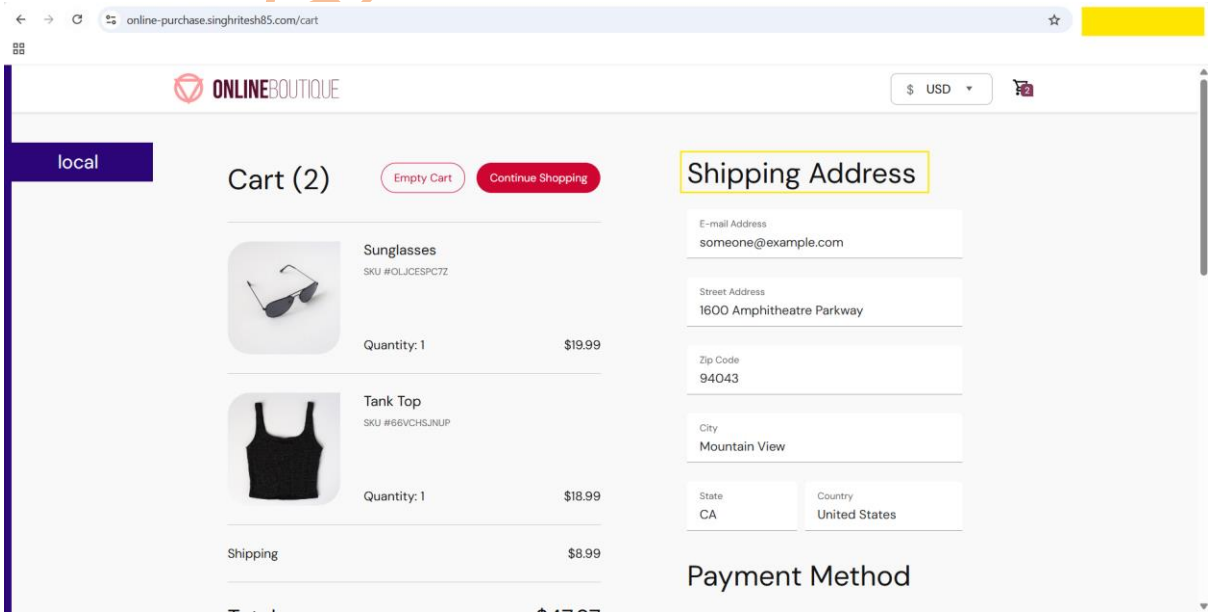
```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: microservice-ingress
  namespace: microservice
  annotations:
    kubernetes.io/ingress.class: nginx
spec:
  ingressClassName: nginx
  rules:
  - host: online-purchase.singhritesh85.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: frontend-external
            port:
              number: 80
```

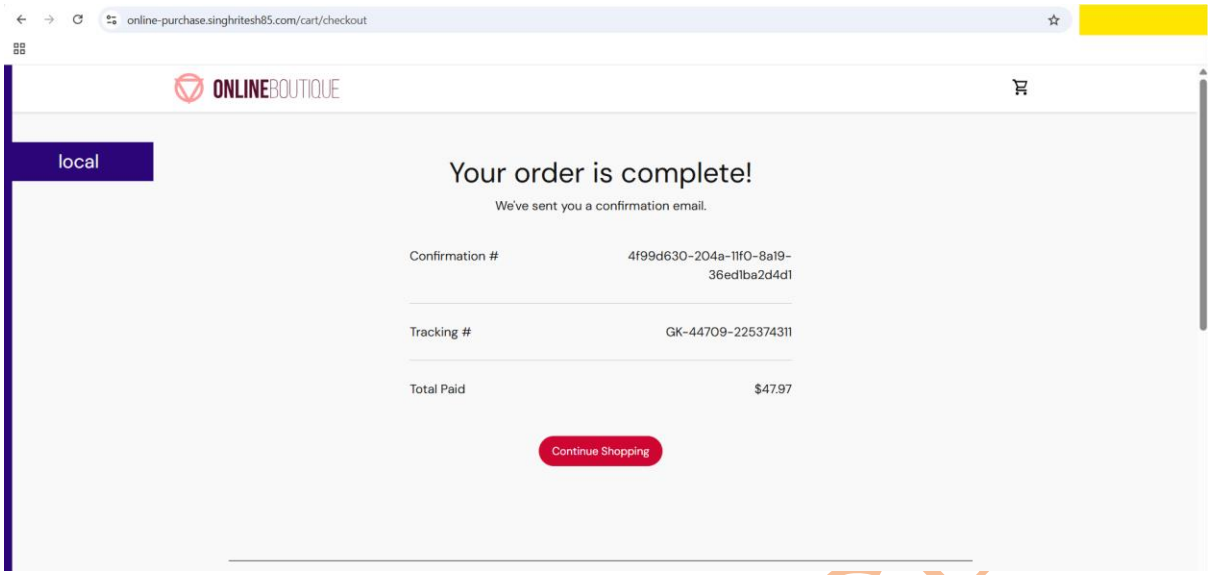
Finally, I was able to access the Application as shown in the screenshot attached below.



Now, I deleted the deployment for shipping service as shown in the screenshot attached below and tested that I was able to access the online shopping Application and the shipping service as shown in the screenshot attached below.

```
[root@                    ~]# kubectl delete deploy shippingservice -n microservice --context=eks-demo-cluster-dev
deployment.apps "shippingservice" deleted
[root@                    ~]# kubectl get pods -n microservice --context=eks-demo-cluster-dev
NAME                                     READY   STATUS    RESTARTS   AGE
adservice-5                  d           2/2     Running   0          34m
cartservice-5                /4          2/2     Running   0          34m
checkoutservice-                         2/2     Running   0          34m
currencyservice-                         2/2     Running   0          34m
emailservice-                            2/2     Running   0          34m
frontend-                                2/2     Running   0          34m
loadgenerator-                           2/2     Running   0          34m
paymentservice-                          2/2     Running   0          34m
productcatalogservice-                   2/2     Running   0          34m
recommendationservice-                   2/2     Running   0          34m
redis-cart-                              2/2     Running   0          34m
```

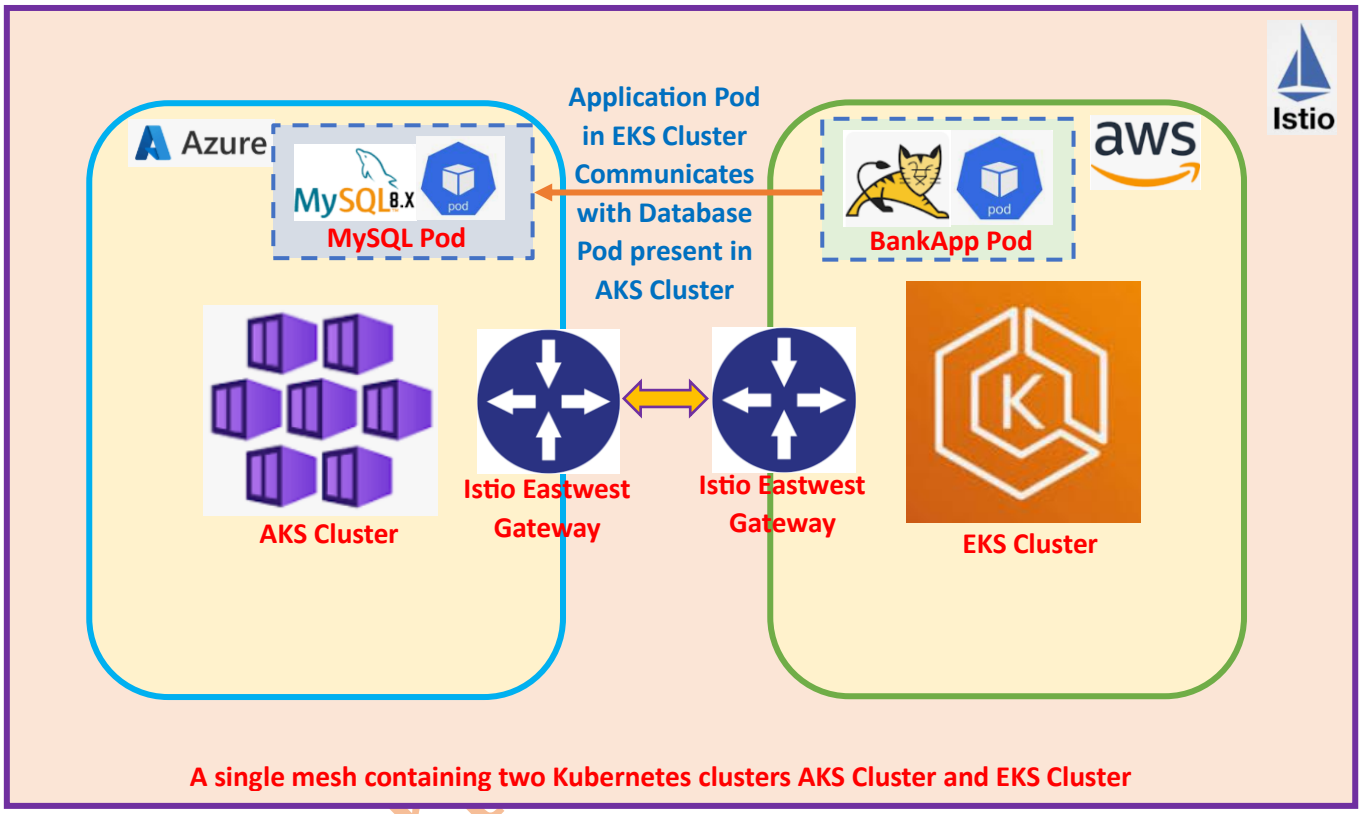




I had deleted the shipping service but I was able to access the Application and the shipping service successfully as shown in the screenshot attached above. The reason behind this is I had also deployed this Application in the AKS Cluster and Multicluster had been created between the AKS and EKS cluster and hence both will be considered as a single unit and when I deleted the shippingservice deployment from EKS Cluster but the same was present on the AKS Cluster so this was accessed through the AKS Cluster.

Module 2: DevOps-Project-BankApp-using-Kubernetes-Multicluster-and-Multicloud

As explained in **Module 1** I had created an AKS and EKS Cluster then established the Multicluster with the created AKS and EKS cluster using Istio. In the second module of the project, I created mysql pods (using the MySQL statefulset) and service in the namespace mysql in AKS Cluster and only the mysql service in the namespace mysql in the EKS Cluster. I provided the kubernetes label **istio-injection=enabled** to the namespace mysql in the AKS and EKS Cluster. Then I created the namespace bankapp in the EKS Cluster and provided the kubernetes label **istio-injection=enabled**.



In current module (Module 2) of the project the Application Pod present in the EKS Cluster communicates with the Database pod present in AKS Cluster. I had created the AKS and EKS Cluster using the Terraform script present in the GitHub Repo https://github.com/singhritesh85/DevOps-Project-Kubernetes-Multicluster-Multicloud.git at the path **module-1/terraform-multi-kubernetes-cluster-multicloud**. Then I created the Multicluster using Istio as explained in **Module 1**.

First, I created the namespace mysql in the AKS and EKS Cluster then provided the kubernetes label **istio-injection=enabled** as shown in the screenshot attached below.

```
[jenkins@                ~]$ kubectl create ns bankapp --context=eks-demo-cluster-dev
namespace/bankapp created
[jenkins@                ~]$ kubectl create ns mysql --context=aks-cluster
namespace/mysql created
[jenkins@                ~]$ kubectl label --context=aks-cluster namespace mysql istio-injection=enabled
namespace/mysql labeled
[jenkins@                ~]$ kubectl label --context=eks-demo-cluster-dev namespace bankapp istio-injection=enabled
```

```
[root@Terraform-Server ~]# kubectl create ns mysql --context=eks-demo-cluster-dev
```

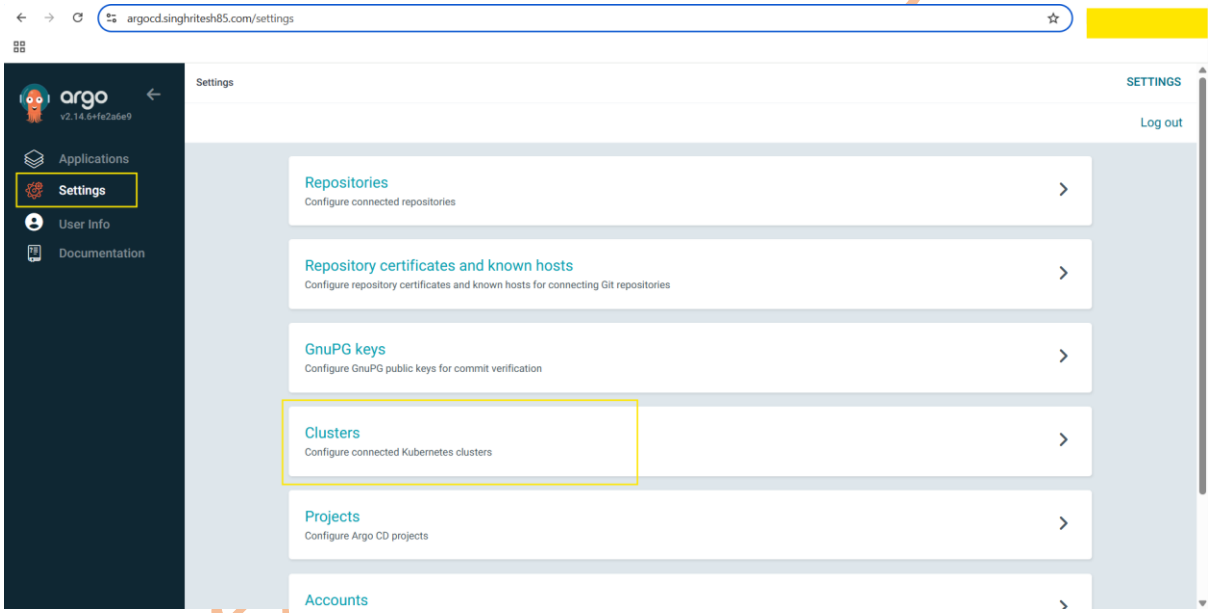
```
[jenkins@                ~]$ kubectl label --context=eks-demo-cluster-dev namespace mysql istio-injection=enabled
namespace/mysql labeled
```

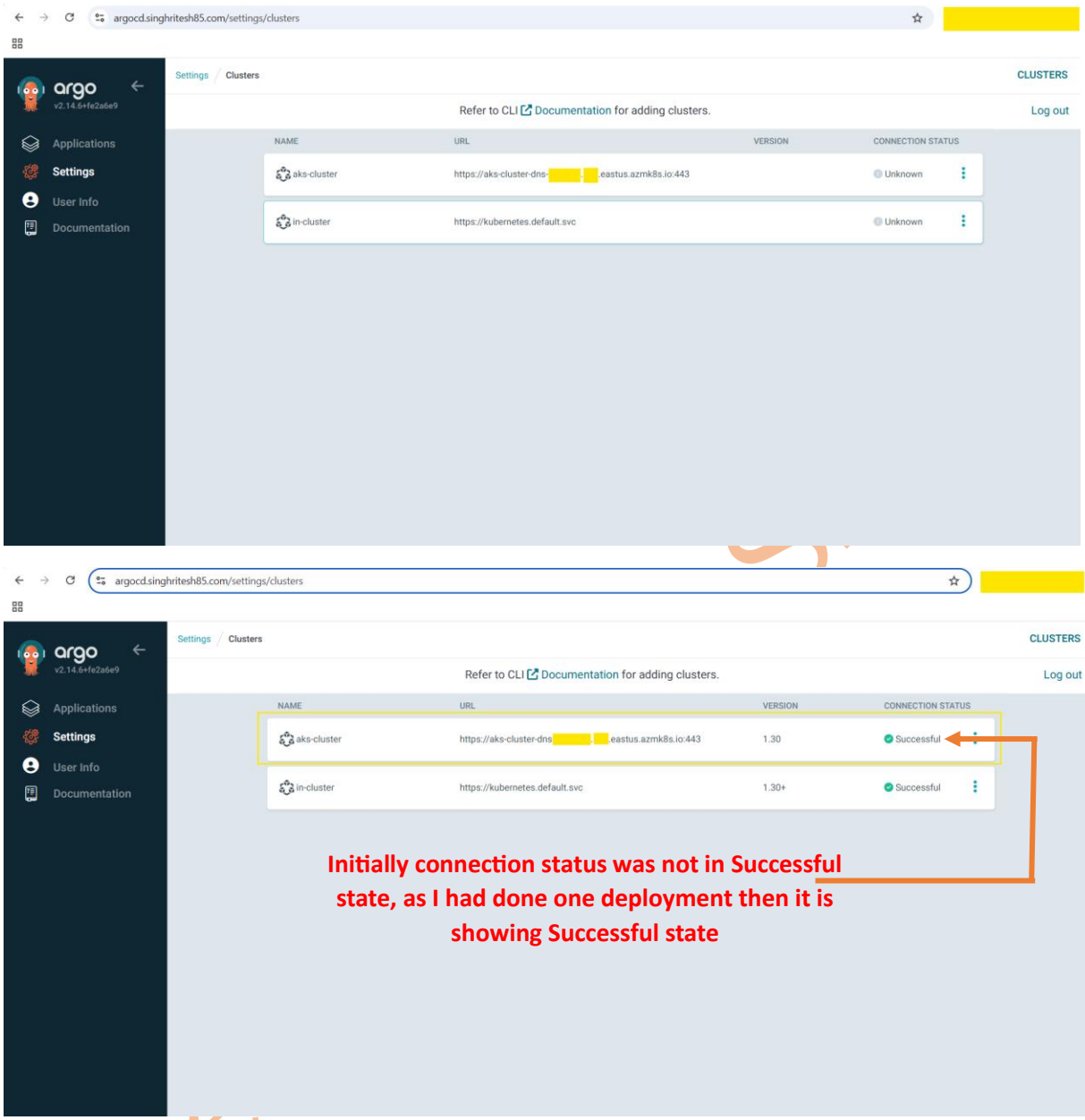
```
[root@                    ~]# kubectl get ns mysql --show-labels --context=eks-demo-cluster-dev
NAME     STATUS   AGE   LABELS
mysql    Active   83m   istio-injection=enabled,kubernetes.io/metadata.name=mysql
[root@                    ~]# kubectl get ns bankapp --show-labels --context=eks-demo-cluster-dev
NAME       STATUS   AGE    LABELS
bankapp    Active   95m    istio-injection=enabled,kubernetes.io/metadata.name=bankapp
[root@                    ~]# kubectl get ns mysql --show-labels --context=aks-cluster
NAME     STATUS   AGE   LABELS
mysql    Active   74m   istio-injection=enabled,kubernetes.io/metadata.name=mysql
```

I had installed the ArgoCD in the EKS Cluster which steps I already discussed in Module 1. I had added the AKS Cluster in the created ArgoCD using the command **argocd cluster add aks-cluster** as shown in the screenshot attached below.

```
[root@                ~]# argocd cluster add aks-cluster
WARNING: This will create a service account `argocd-manager` on the cluster referenced by context `aks-cluster` with full cluster level privileges. Do you wan
t to continue [y/N]? y
INFO[0001] ServiceAccount "argocd-manager" already exists in namespace "kube-system"
INFO[0001] ClusterRole "argocd-manager-role" updated
INFO[0001] ClusterRoleBinding "argocd-manager-role-binding" updated
Cluster 'https://aks-cluster-dns-        .    .eastus.azmk8s.io:443' added
```

Then I went to the ArgoCD User Interface and go to **Settings > Clusters** as shown in the screenshot attached below.





After first deployment the **connection status** of Newly Added cluster will come into the Successful state. As I had done the deployment and then I am taking this screenshot So it is showing Successful state.

For CI/CD Tool I used Jenkins and below is the Jenkinsfile I used in the Project.

Jenkinsfile for MYSQL

```
pipeline{
  agent{
    node{
      label "Slave-1"
      customWorkspace "/home/jenkins/mysql"
    }
  }
  environment{
    JAVA_HOME="/usr/lib/jvm/java-17-amazon-corretto.x86_64"
    PATH="$PATH:$JAVA_HOME/bin:/opt/apache-maven/bin:/usr/local/bin"
  }
  stages{
    stage("MySQL-Deployment"){
      steps{
        //MySQL
        sh 'argocd login argocd.singhritesh85.com --username admin --password Admin@123 --skip-test-tls  --grpc-web'
        sh 'argocd app create mysql --project default --repo https://github.com/singhritesh85/helm-repo-for-bitnami.git --path ./bitnami/mysql --dest-namespace mysql --sync-option CreateNamespace=true --dest-server https://aks-cluster-dns-6ir8js3n.hcp.eastus.azmk8s.io:443 --helm-set secondary.replicaCount=1 --helm-set primary.persistence.enabled=true --helm-set primary.persistence.size=1Gi --helm-set architecture=replication --helm-set secondary.persistence.enabled=true --helm-set secondary.persistence.size=1Gi --helm-set primary.service.type=ClusterIP --helm-set auth.rootPassword=Dexter@123 --helm-set auth.database=bankappdb --helm-set global.storageClass=managed-csi --upsert'
        sh 'argocd app sync mysql'
        sh 'argocd app create mysql --project default --repo https://github.com/singhritesh85/helm-repo-for-bitnami-mysql-onlyservice.git --path ./bitnami/mysql --dest-namespace mysql --sync-option CreateNamespace=true --dest-server https://kubernetes.default.svc --helm-set secondary.replicaCount=1 --helm-set primary.persistence.enabled=false --helm-set architecture=replication --helm-set secondary.persistence.enabled=false --helm-set primary.service.type=ClusterIP --helm-set auth.rootPassword=Dexter@123 --helm-set auth.database=bankappdb --upsert'
        sh 'argocd app sync mysql'
      }
    }
  }
}
```

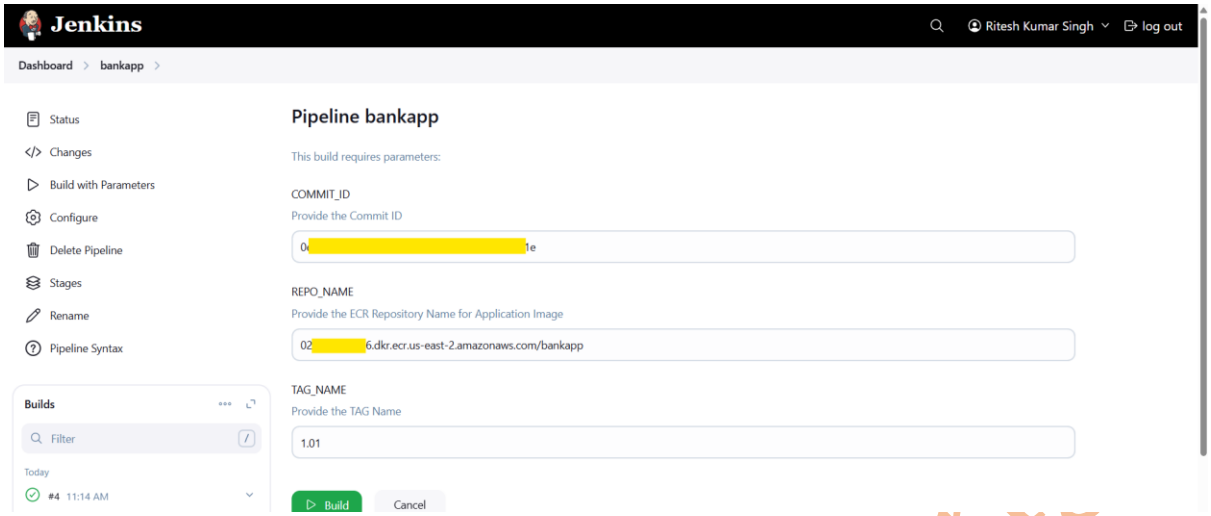
Jenkinsfile for BankApp

```
pipeline{
  agent{
    node{
      label "Slave-1"
      customWorkspace "/home/jenkins/bankapp"
    }
  }
  environment{
    JAVA_HOME="/usr/lib/jvm/java-17-amazon-corretto.x86_64"
    PATH="$PATH:$JAVA_HOME/bin:/opt/apache-maven/bin:/opt/node-v16.0.0/bin:/usr/local/bin"
  }
  parameters {
    string(name: 'COMMIT_ID', defaultValue: '', description: 'Provide the Commit ID')
    string(name: 'REPO_NAME', defaultValue: '', description: 'Provide the ECR Repository Name for Application Image')
    string(name: 'TAG_NAME', defaultValue: '', description: 'Provide the TAG Name')
  }
  stages{
    stage("Clone-Code"){
      steps{
        cleanWs()
        checkout scmGit(branches: [[name: '${COMMIT_ID}']], extensions: [], userRemoteConfigs: [[credentialsId: 'github-cred', url: 'https://github.com/singhritesh85/Bank-App.git']])
      }
    }
    stage("Maven-Build"){
      steps{
        sh 'mvn clean install'
      }
    }
    stage("Docker Build"){
      steps{
```

```
            sh 'docker system prune -f --all'

            sh 'docker build -t ${REPO_NAME}:${TAG_NAME} -f Dockerfile-Project-1 .'

            sh 'aws ecr get-login-password --region us-east-2 | docker login --username AWS --password-stdin 027330342406.dkr.ecr.us-east-2.amazonaws.com'

            sh 'docker push ${REPO_NAME}:${TAG_NAME}'

        }

    }

    stage("Deployment"){

        steps{

            sh 'argocd login argocd.singhritesh85.com --username admin --password Admin@123 --skip-test-tls  --grpc-web'

            sh 'argocd app create bankapp --project default --repo https://github.com/singhritesh85/helm-repo-for-Blue-Green-Deployment.git --path ./folo --dest-namespace bankapp --sync-option CreateNamespace=true --dest-server https://kubernetes.default.svc --helm-set service.port=80 --helm-set image.repository=${REPO_NAME} --helm-set image.tag=${TAG_NAME} --helm-set replicaCount=1 --upsert'

            sh 'argocd app sync bankapp'

        }

    }

  }

}
```

Both the Jenkinsfile for MySQL and BankApp are present in the GitHub Repo https://github.com/singhritesh85/DevOps-Project-Kubernetes-Multicluster-Multicloud.git at the path **module-2**.

First, I ran the Jenkins Job mysql then bankapp and for Running Jenkins Job bankapp I had to provide three string parameters Commit ID, Repo Name and Image Tag as shown in the screenshot attached below.



For BankApp Deployment I did not use the SonarQube, Nexus and Trivy Image Scan. In this Project I did not use the Prometheus, Grafana, Loki and triggering Email for notification If you want to use these then you can refer the project present in the GitHub Repo https://github.com/singhritesh85/DevOps-Project-BankApplication-BlueGreen-Deployment-MultiCloud.git.

After successful deployment the screenshot of Jenkins Job is as shown below.

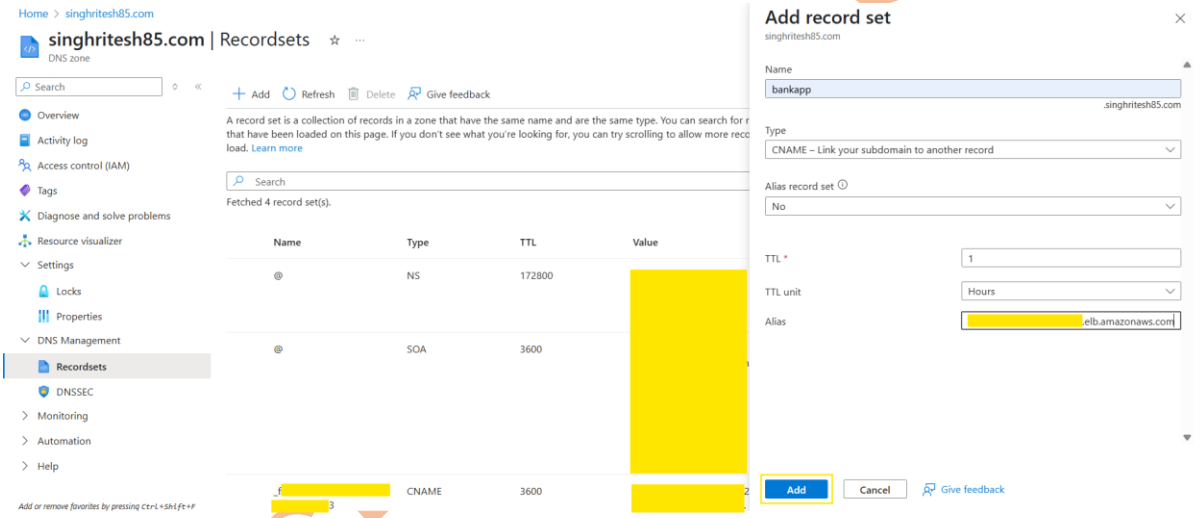
S	W	Name ↓	Last Success	Last Failure	Last Duration
✓		bankapp	5 min 19 sec #5	2 hr 28 min #3	33 sec
✓		mysql	8 min 4 sec #7	2 hr 37 min #5	11 sec

```
[root@                ~]# kubectl get pods -n bankapp --context=eks-demo-cluster-dev
NAME                              READY   STATUS    RESTARTS   AGE
bankapp-folo-5                p   2/2     Running   0          28m
[root@                ~]# kubectl get svc -n bankapp --context=eks-demo-cluster-dev
NAME                   TYPE        CLUSTER-IP        EXTERNAL-IP   PORT(S)   AGE
bankapp-folo-active    ClusterIP   192.      .173    <none>        80/TCP    28m
bankapp-folo-preview   ClusterIP   192.      .45     <none>        80/TCP    28m
[root@                ~]# kubectl get all -n mysql --context=eks-demo-cluster-dev
NAME                                 TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
service/mysql-primary                ClusterIP   192.      .143   <none>        3306/TCP   32m
service/mysql-primary-headless       ClusterIP   None             <none>        3306/TCP   32m
service/mysql-secondary              ClusterIP   192.      .194   <none>        3306/TCP   32m
service/mysql-secondary-headless     ClusterIP   None             <none>        3306/TCP   32m
[root@                ~]# kubectl get pods -n mysql --context=aks-cluster
NAME                 READY   STATUS    RESTARTS      AGE
mysql-primary-0      2/2     Running   0             32m
mysql-secondary-0    2/2     Running   1 (30m ago)   32m
[root@                ~]# kubectl get pvc -n mysql --context=aks-cluster
NAME                     STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS   VOLUMEATTRIBUTESCLASS   AGE
data-mysql-primary-0     Bound    pvc-5                                   a  1Gi        RWO            managed-csi    <unset>                 32m
data-mysql-secondary-0   Bound    pvc-b                                   9  1Gi        RWO            managed-csi    <unset>                 32m
[root@                ~]# kubectl get pvc -n mysql --context=eks-demo-cluster-dev
No resources found in mysql namespace.
```

To access the Application, I created the ingress rule for bankapp as shown in the screenshot attached below.

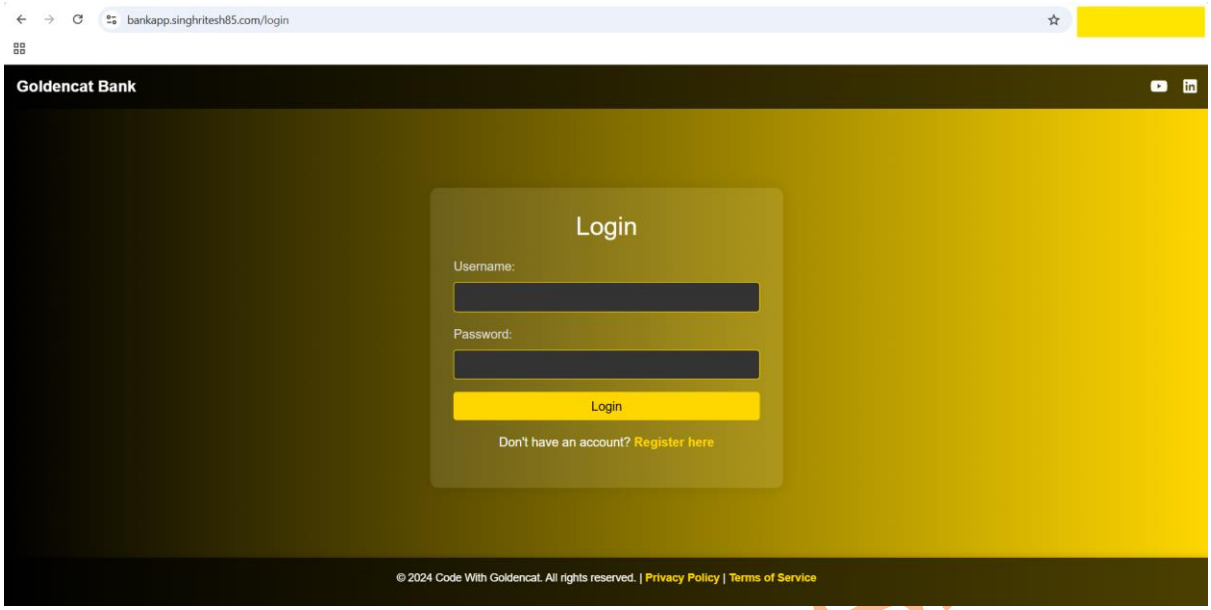
```
[jenkins@                ~]$ cat ingress-rule.yaml
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: bankapp-ingress
  namespace: bankapp
  annotations:
    kubernetes.io/ingress.class: nginx
spec:
  ingressClassName: nginx
  rules:
  - host: bankapp.singhritesh85.com
    http:
      paths:
      - path: /
        backend:
          service:
            name: bankapp-folo-active
            port:
              number: 80
        pathType: Prefix
[jenkins@                ~]$ kubectl apply -f ingress-rule.yaml --context=eks-demo-cluster-dev
ingress.networking.k8s.io/bankapp-ingress created
[jenkins@                ~]$ kubectl get ing -A --context=eks-demo-cluster-dev
NAMESPACE   NAME              CLASS   HOSTS                      ADDRESS                                                                  PORTS   AGE
argocd      minimal-ingress   nginx   argocd.singhritesh85.com   a                                                3.us-east-2.elb.amazonaws.com   80      7h23m
bankapp     bankapp-ingress   nginx   bankapp.singhritesh85.com                                                                           80      18s
[jenkins@                ~]$ kubectl get ing -A --context=eks-demo-cluster-dev --watch
NAMESPACE   NAME              CLASS   HOSTS                      ADDRESS                                                                  PORTS   AGE
argocd      minimal-ingress   nginx   argocd.singhritesh85.com   a                                                3.us-east-2.elb.amazonaws.com   80      7h23m
bankapp     bankapp-ingress   nginx   bankapp.singhritesh85.com                                                                           80      23s
bankapp     bankapp-ingress   nginx   bankapp.singhritesh85.com  a                                                 3.us-east-2.elb.amazonaws.com   80      38s
```

I did the entry in Azure DNS Zone to create the Record Set of type CNAME corresponding to HOSTS and the DNS Name as shown in the screenshot attached below.

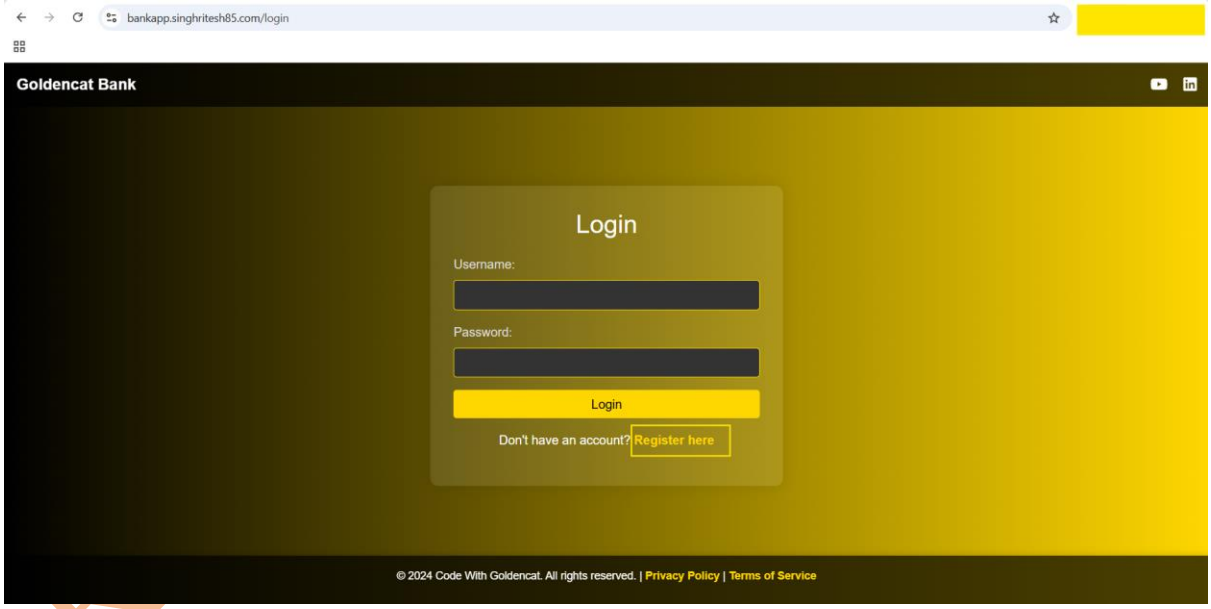


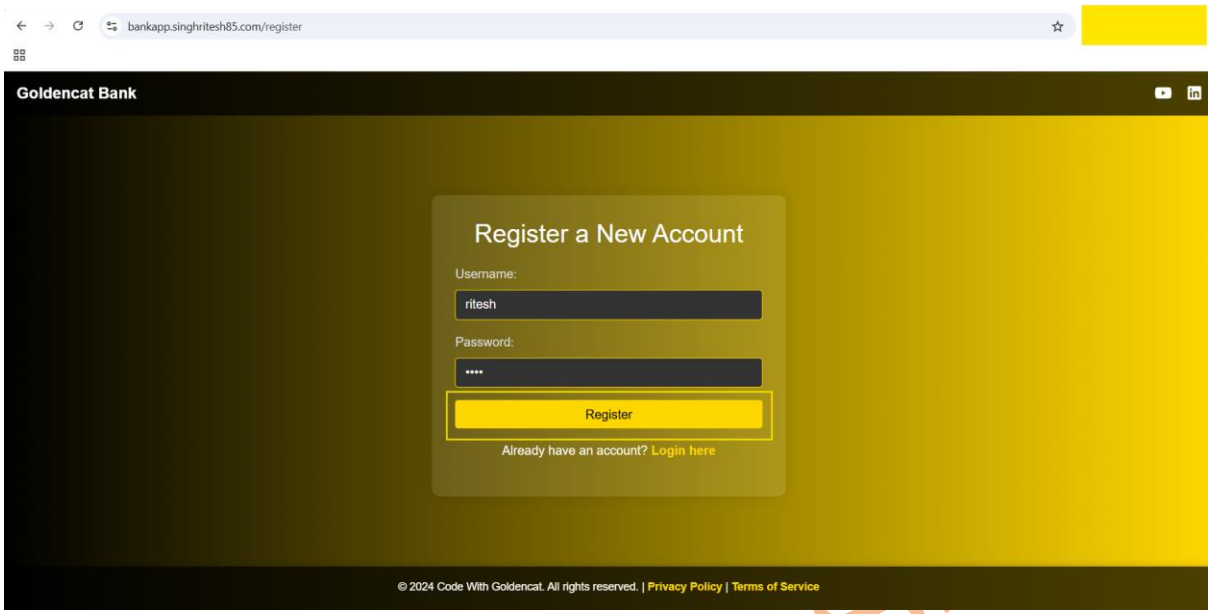
I want to mention here that **Argocd CLI** should be installed on Jenkins Slave node. Install **Argo Rollout Controller** and **Argo Rollouts Kubectl plugin** on both **aks-cluster** and **eks-demo-cluster-dev**.

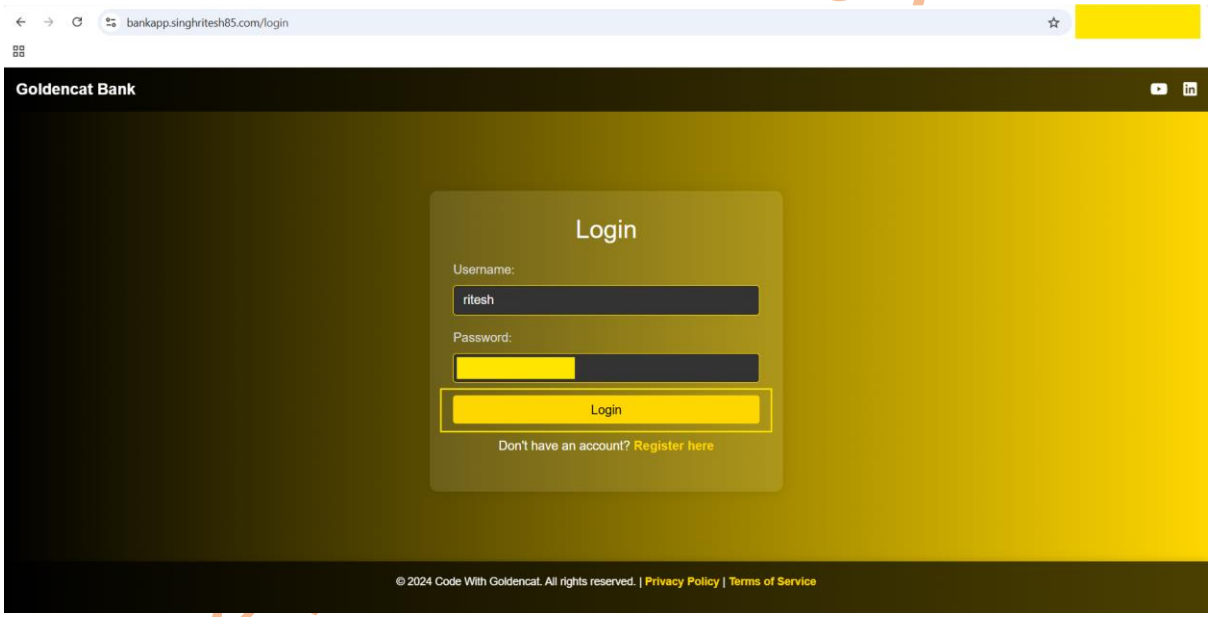
Finally, I accessed the Application using the URL as shown in the screenshot attached below.

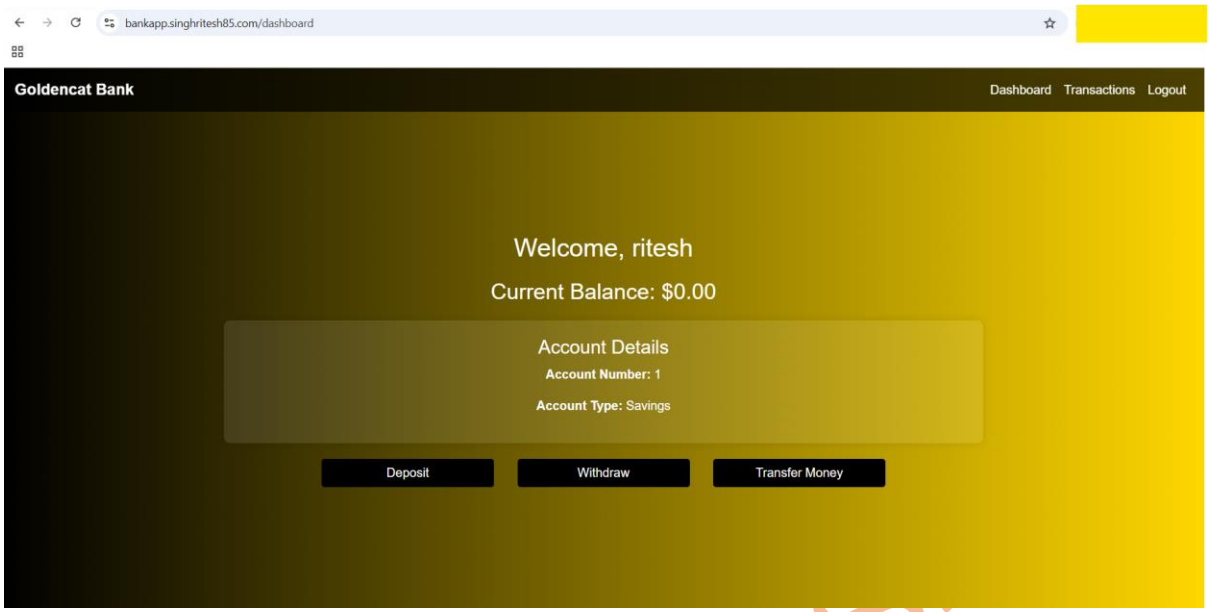


I registered with a user in BankApp and checked the user in the MySQL Pod and found its entry was present there.




bankapp.singhritesh85.com/register
Goldencat Bank
Register a New Account
Username:
ritesh
Password:
Register
Already have an account? Login here
© 2024 Code With Goldencat. All rights reserved. | Privacy Policy | Terms of Service


bankapp.singhritesh85.com/login
Goldencat Bank
Login
Username:
ritesh
Password:
Login
Don't have an account? Register here
© 2024 Code With Goldencat. All rights reserved. | Privacy Policy | Terms of Service



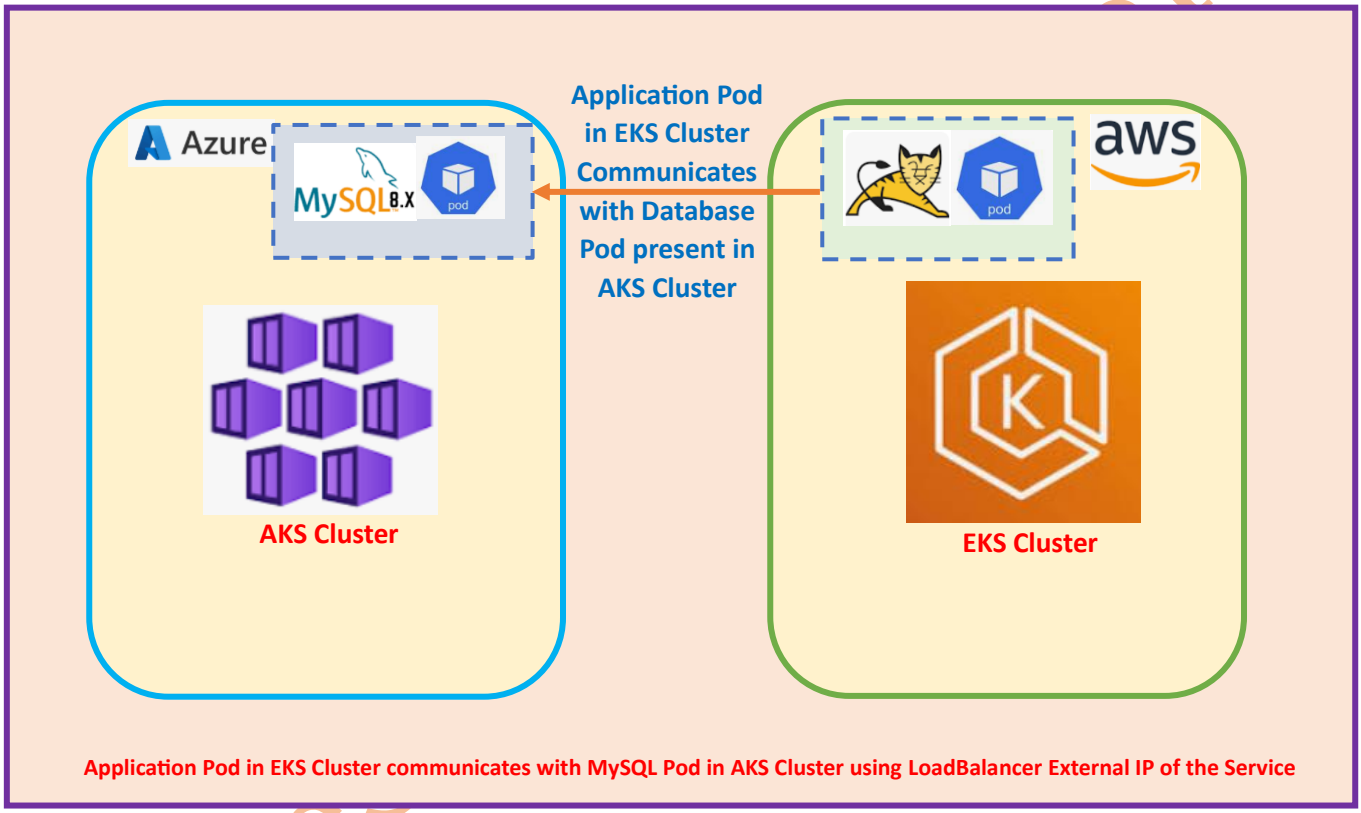

```
[root@                ~]# kubectl exec -it mysql-primary-0 bash -n mysql --context=aks-cluster
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [POD] -- [COMMAND] instead.
I have no name!@mysql-primary-0:/$ mysql -h mysql-primary-0 -u root --password
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 812
Server version: 8.4.0 Source distribution

Copyright (c) 2000, 2024, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| bankappdb          |
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
5 rows in set (0.01 sec)

mysql> use bankappdb;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+---------------------+
| Tables_in_bankappdb |
+---------------------+
| account             |
| transaction         |
```

```
mysql> select * from account;
+----+---------+--------------------------------------------------------------+----------+
| id | balance | password                                                     | username |
+----+---------+--------------------------------------------------------------+----------+
|  1 |    0.00 |                                                              | ritesh   |
+----+---------+--------------------------------------------------------------+----------+
1 row in set (0.00 sec)

mysql> \q
Bye
I have no name!@mysql-primary-0:/$
exit
[root@                ~]#
```

Module 3: DevOps-Project-Kubernetes-BankApp-and-MySQL-Pods-on-Diferent-clusters-Multicloud

I had created an AKS Cluster and an EKS Cluster using the terraform script present in the GitHub Repo https://github.com/singhritesh85/DevOps-Project-Kubernetes-Multicluster-Multicloud.git at the path **module-3/terraform-different-kubernetes-cluster-multicloud**.

I had installed the ArgoCD in the EKS Cluster which steps I already discussed in Module 1. I had added the AKS Cluster in the created ArgoCD using the command **argocd cluster add aks-cluster** as discussed in Module 2. **Argocd CLI** should be installed on Jenkins Slave node. Install **Argo Rollout Controller** and **Argo Rollouts Kubectl plugin** on both **aks-cluster** and **eks-demo-cluster-dev** as discussed in **Module 2**.



In **Module-3** I did not use the concept of Multicluster and the Application Pod present on EKS Cluster will communicate with the MySQL Pod present on the AKS Cluster using LoadBalancer Service External IP. I ran the Jenkins Job for MySQL first and the LoadBalancer Service External IP had been provided to the file **application.properties** present at the path **src/main/resources/application.properties** GitHub Repo https://github.com/singhritesh85/Bank-App.git as shown in the screenshot attached below.

```
[root@                    ~]# kubectl get pods -n mysql --context=aks-cluster --watch
NAME                READY   STATUS    RESTARTS   AGE
mysql-primary-0     1/1     Running   0          110s
mysql-secondary-0   1/1     Running   0          110s
^C[root@                    ~]#
[root@                    ~]# kubectl get svc -n mysql --context=aks-cluster --watch
NAME                      TYPE           CLUSTER-IP     EXTERNAL-IP     PORT(S)          AGE
mysql-primary             LoadBalancer   10.      .82   128.      .9    3306:32246/TCP   115s
mysql-primary-headless    ClusterIP      None           <none>          3306/TCP         115s
mysql-secondary           ClusterIP      10.      .21   <none>          3306/TCP         115s
mysql-secondary-headless  ClusterIP      None           <none>          3306/TCP         115s
```

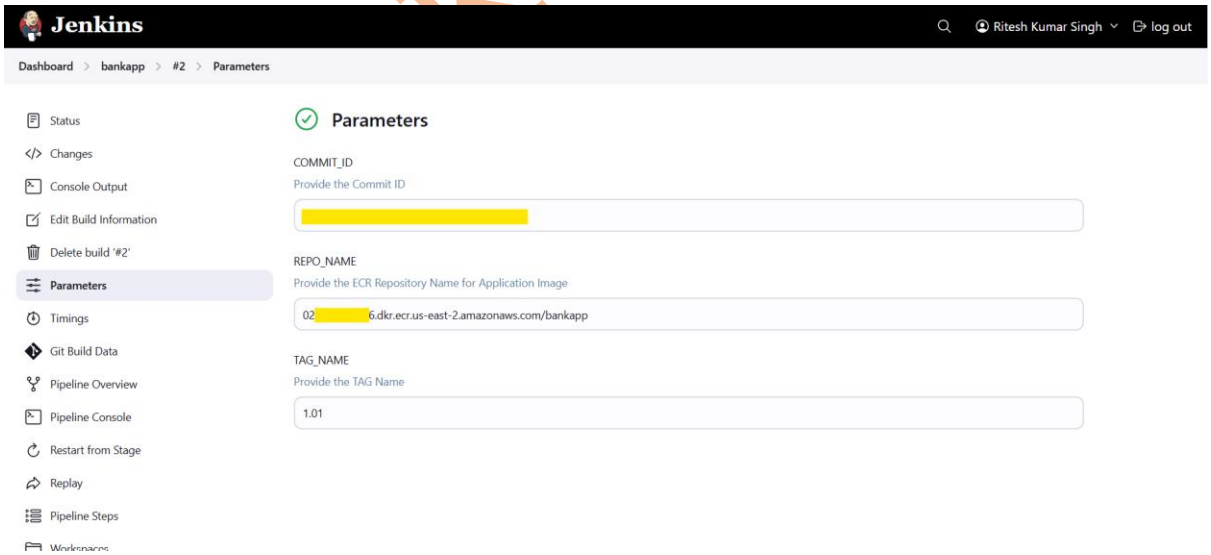
Bank-App / src / main / resources / **application.properties**

singhritesh85 Update application.properties

Code | Blame | 11 lines (10 loc) · 461 Bytes

```
1   spring.application.name=bankapp
2   # MySQL Database configuration
3   spring.datasource.url=jdbc:mysql://128.      .9:3306/bankappdb?useSSL=false&serverTimezone=UTC
4   spring.datasource.username=root
5   spring.datasource.password=
6   spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
7
8   # JPA & Hibernate configuration
9   spring.jpa.hibernate.ddl-auto=update
10  spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect
11  spring.jpa.show-sql=true
```

String Parameters which I provided to ran the Jenkins Job Bankapp is as shown in the screenshot attached below.



The Jenkinsfile for MySQL and Bankapp is as shown in the screenshot attached below.

Jenkinsfile for MySQL

```
pipeline{
   agent{
       node{
          label "Slave-1"
          customWorkspace "/home/jenkins/mysql"
       }
   }
   environment{
      JAVA_HOME="/usr/lib/jvm/java-17-amazon-corretto.x86_64"
      PATH="$PATH:$JAVA_HOME/bin:/opt/apache-maven/bin:/usr/local/bin"
   }
   stages{
      stage("MySQL-Deployment"){
         steps{
            //MySQL
            sh 'argocd login argocd.singhritesh85.com --username admin --password Admin@123 --skip-test-tls  --grpc-web'
            sh 'argocd app create mysql --project default --repo https://github.com/singhritesh85/helm-repo-for-bitnami.git --path ./bitnami/mysql --dest-namespace mysql --sync-option CreateNamespace=true --dest-server https://aks-cluster-dns-b190ise8.hcp.eastus.azmk8s.io:443 --helm-set secondary.replicaCount=1 --helm-set primary.persistence.enabled=true --helm-set primary.persistence.size=1Gi --helm-set architecture=replication --helm-set secondary.persistence.enabled=true --helm-set secondary.persistence.size=1Gi --helm-set primary.service.type=ClusterIP --helm-set auth.rootPassword=Dexter@123 --helm-set auth.database=bankappdb --helm-set primary.service.type=LoadBalancer --helm-set global.storageClass=managed-csi --upsert'
            sh 'argocd app sync mysql'
         }
      }
   }
}
```

Jenkinsfile for Bankapp

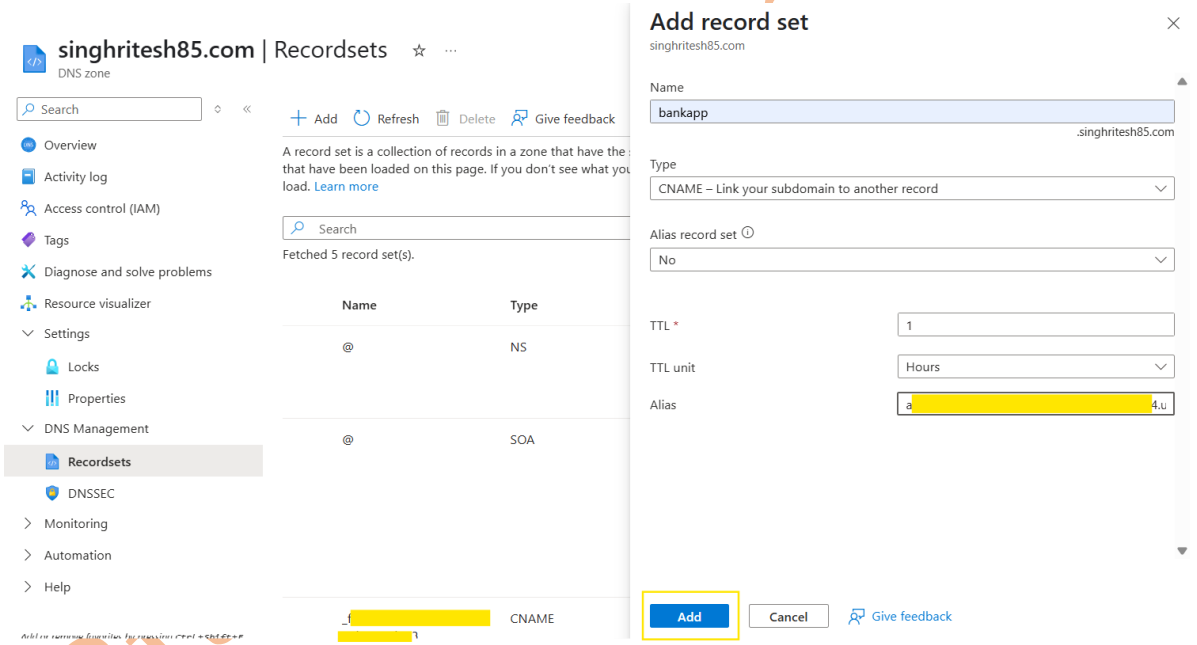
```
pipeline{
   agent{
       node{
          label "Slave-1"
          customWorkspace "/home/jenkins/bankapp"
       }
   }
   environment{
      JAVA_HOME="/usr/lib/jvm/java-17-amazon-corretto.x86_64"
      PATH="$PATH:$JAVA_HOME/bin:/opt/apache-maven/bin:/opt/node-v16.0.0/bin:/usr/local/bin"
   }
   parameters {
      string(name: 'COMMIT_ID', defaultValue: '', description: 'Provide the Commit ID')
      string(name: 'REPO_NAME', defaultValue: '', description: 'Provide the ECR Repository Name for Application Image')
      string(name: 'TAG_NAME', defaultValue: '', description: 'Provide the TAG Name')
   }
   stages{
      stage("Clone-Code"){
         steps{
            cleanWs()
            checkout scmGit(branches: [[name: '${COMMIT_ID}']], extensions: [], userRemoteConfigs: [[credentialsId: 'github-cred', url: 'https://github.com/singhritesh85/Bank-App.git']])
         }
      }
      stage("Maven-Build"){
         steps{
            sh 'mvn clean install'
         }
      }
      stage("Docker Build"){
```

```
        steps{
            sh 'docker system prune -f --all'
            sh 'docker build -t ${REPO_NAME}:${TAG_NAME} -f Dockerfile-Project-1 .'
            sh 'aws ecr get-login-password --region us-east-2 | docker login --username AWS --password-stdin 027330342406.dkr.ecr.us-east-2.amazonaws.com'
            sh 'docker push ${REPO_NAME}:${TAG_NAME}'
        }
    }
    stage("Deployment"){
        steps{
            sh 'argocd login argocd.singhritesh85.com --username admin --password Admin@123 --skip-test-tls  --grpc-web'
            sh 'argocd app create bankapp --project default --repo https://github.com/singhritesh85/helm-repo-for-Blue-Green-Deployment.git --path ./folo --dest-namespace bankapp --sync-option CreateNamespace=true --dest-server https://kubernetes.default.svc --helm-set service.port=80 --helm-set image.repository=${REPO_NAME} --helm-set image.tag=${TAG_NAME} --helm-set replicaCount=1 --upsert'
            sh 'argocd app sync bankapp'
        }
    }
  }
}
```

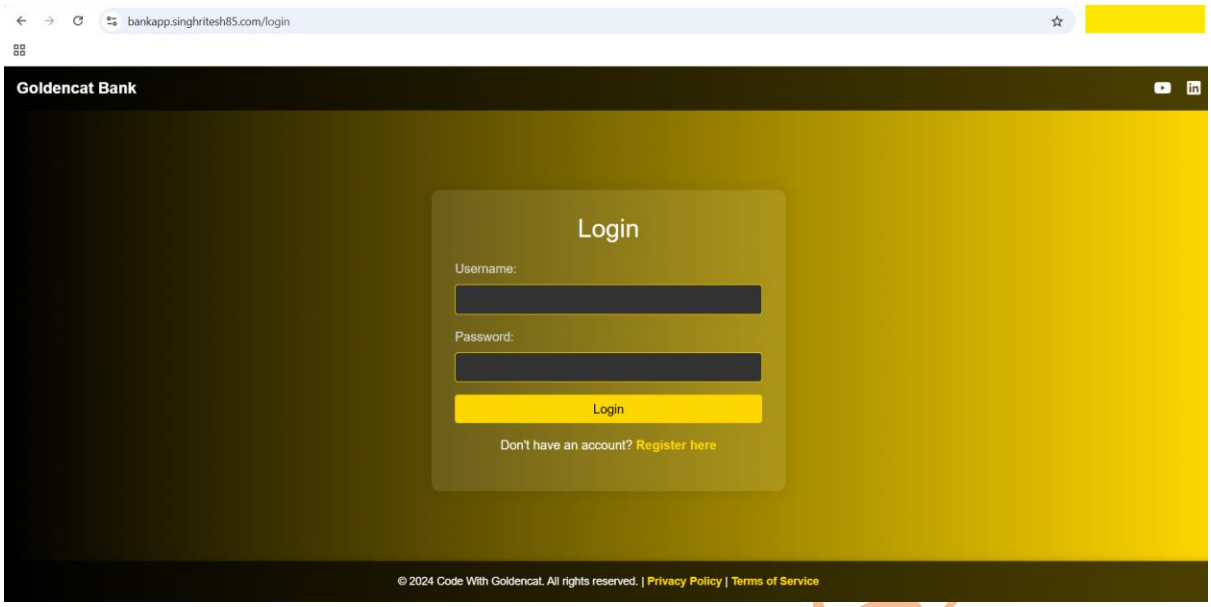
Ingress Rule for Bankapp is as shown in the screenshot attached below.

```
[root@                ~]# cat ingress-rule.yaml
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: bankapp-ingress
  namespace: bankapp
  annotations:
    kubernetes.io/ingress.class: nginx
spec:
  ingressClassName: nginx
  rules:
  - host: bankapp.singhritesh85.com
    http:
      paths:
      - path: /
        backend:
          service:
            name: bankapp-folo-active
            port:
              number: 80
        pathType: Prefix

[root@                ~]# kubectl apply -f ingress-rule.yaml --context=eks-demo-cluster-dev
ingress.networking.k8s.io/bankapp-ingress created
[root@                ~]# kubectl get ing -A --context=eks-demo-cluster-dev --watch
NAMESPACE   NAME              CLASS   HOSTS                      ADDRESS                                                                  PORTS   AGE
argocd      minimal-ingress   nginx   argocd.singhritesh85.com   a                                  4.us-east-2.elb.amazonaws.com   80      21m
bankapp     bankapp-ingress   nginx   bankapp.singhritesh85.com                                                                           80      13s
bankapp     bankapp-ingress   nginx   bankapp.singhritesh85.com  a                                  4.us-east-2.elb.amazonaws.com   80      27s
```

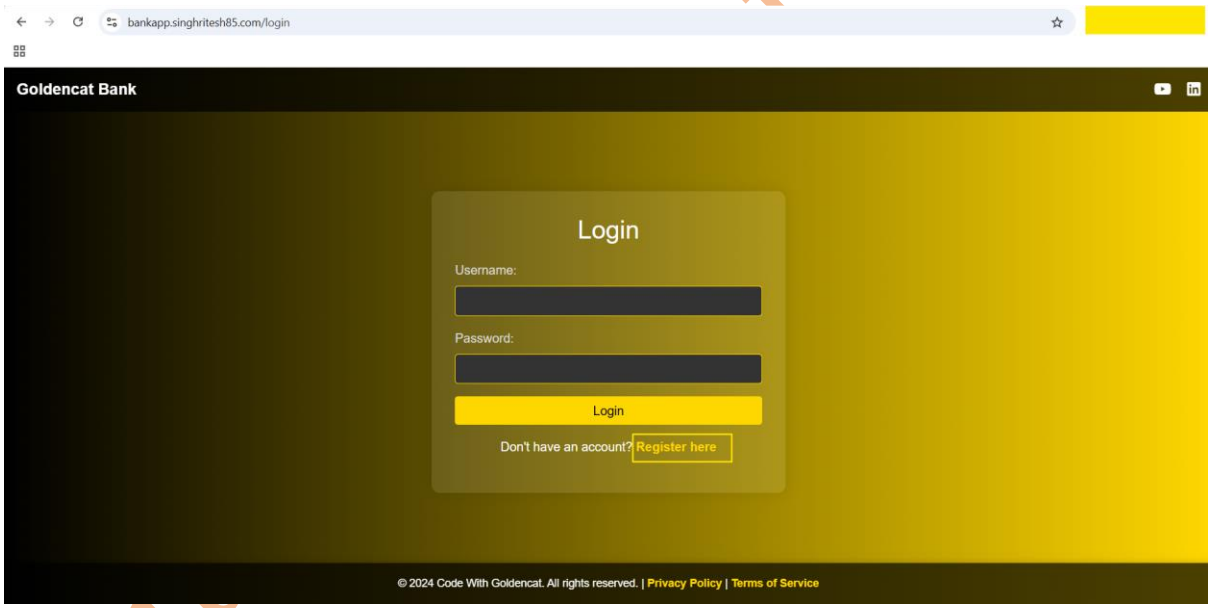
I had done the entry for HOSTS with DNS Name in Azure DNS Zone to create the record set of type **CNAME** as shown in the screenshot attached below.

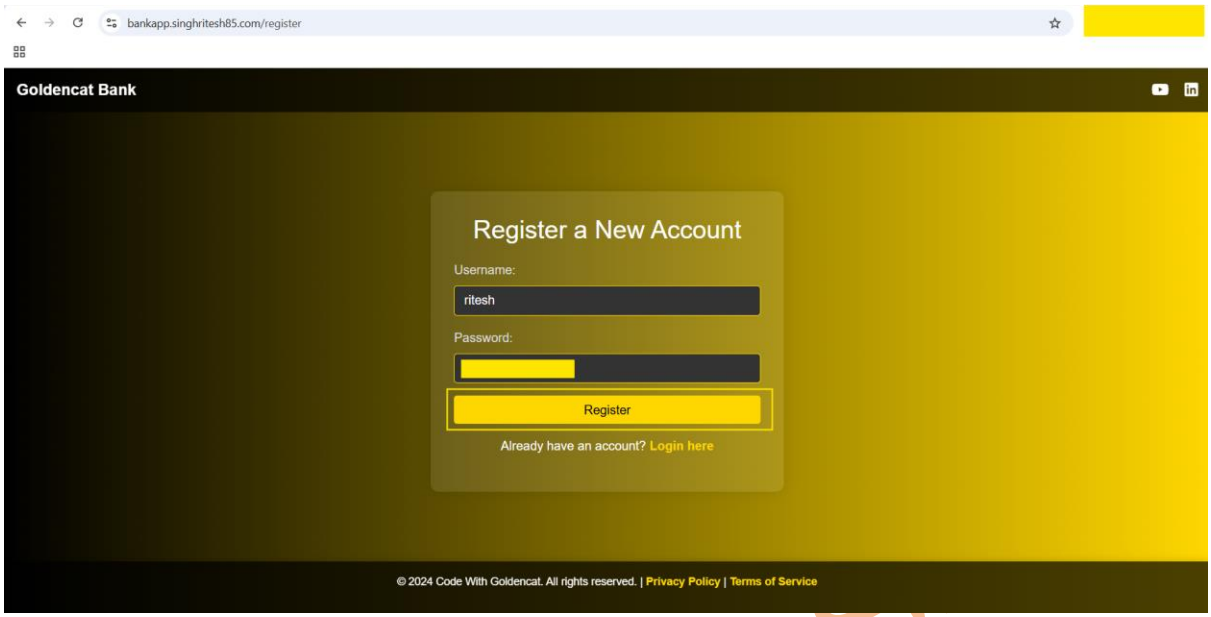


Finally, I was able to access the Application as shown in the screenshot attached below.

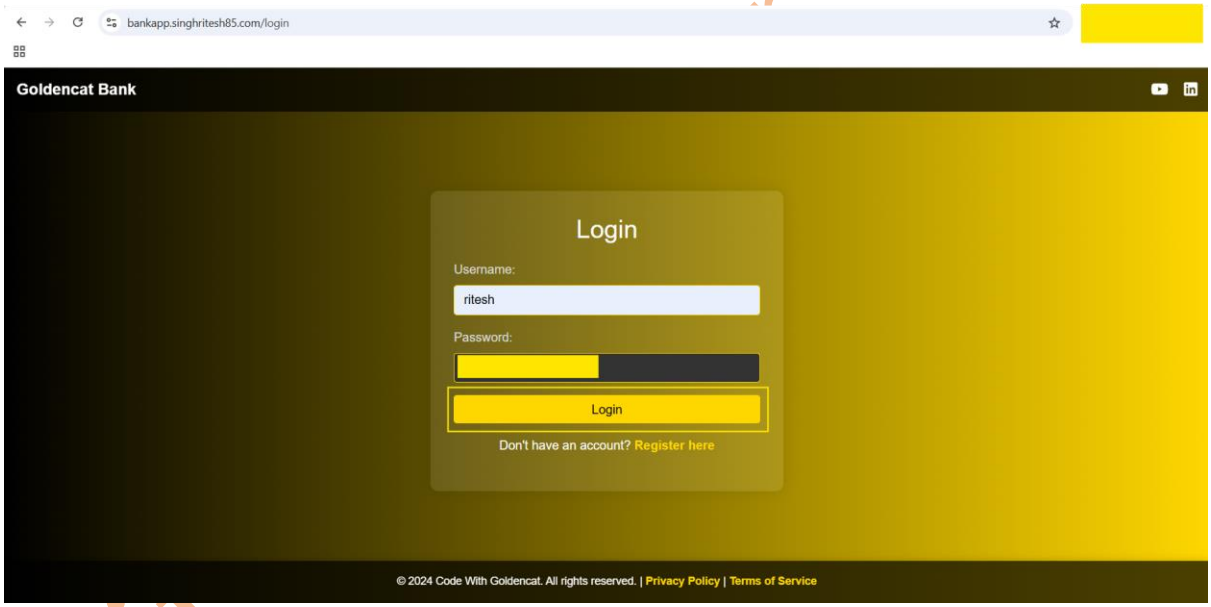


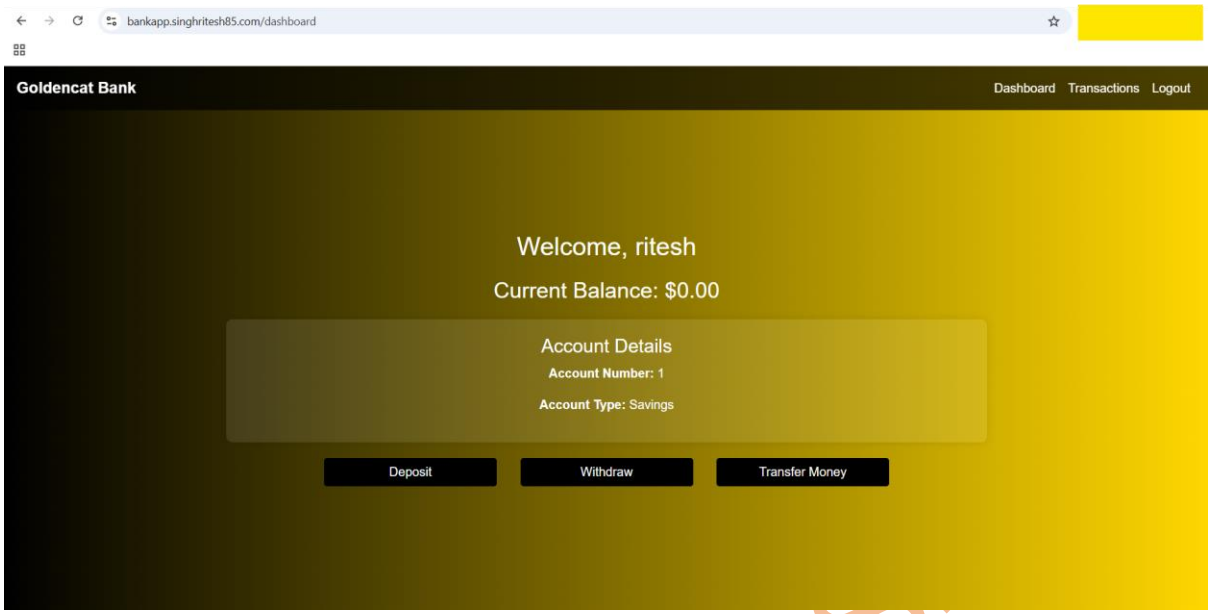
I registered with a user in this Bankapp and checked the entry for MySQL database and found the same user which I registered in the Bankapp as shown in the screenshot attached below.





I logged-in with the registered user and found I was able to login successfully.



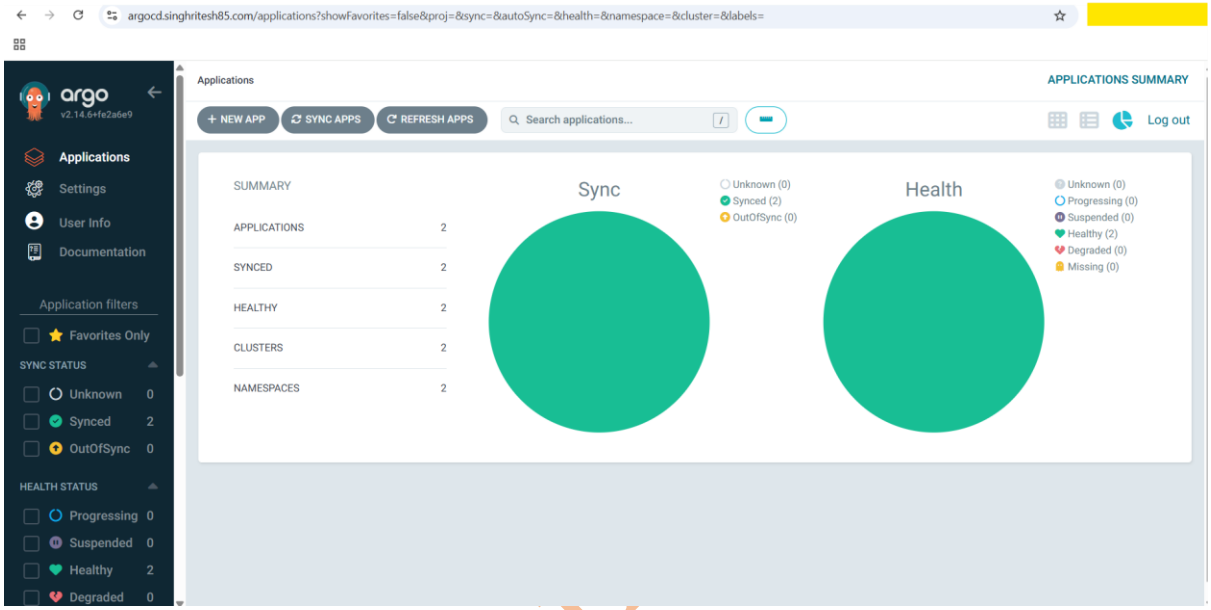



```
[root@                ~]# kubectl exec -it mysql-primary-0 bash -n mysql --context=aks-cluster
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [POD] -- [COMMAND] instead.
Defaulted container "mysql" out of: mysql, preserve-logs-symlinks (init)
I have no name!@mysql-primary-0:/$ mysql -h localhost -u root --password
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 4048
Server version: 8.4.0 Source distribution

Copyright (c) 2000, 2024, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| bankappdb          |
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
5 rows in set (0.00 sec)
```

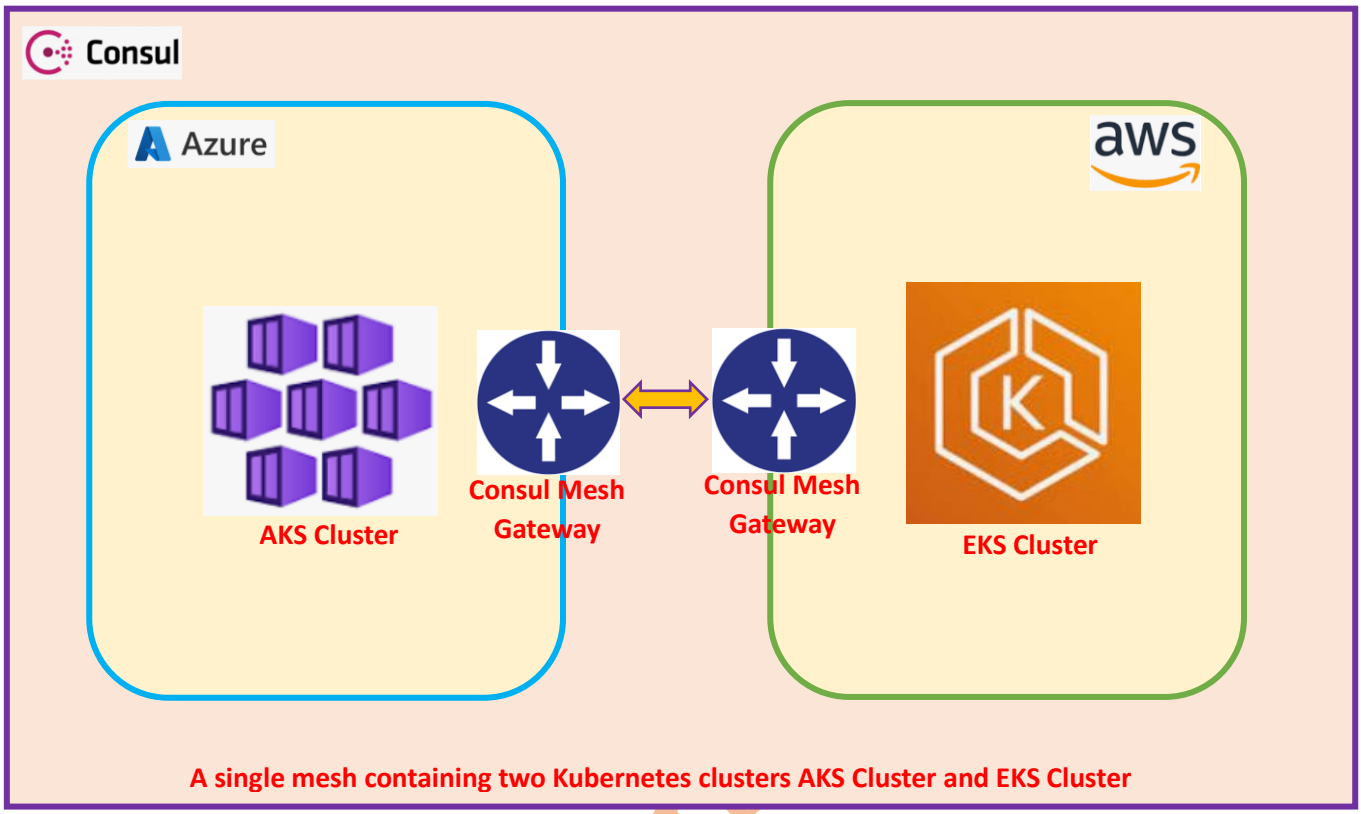
```
mysql> use bankappdb;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+----------------------+
| Tables_in_bankappdb |
+----------------------+
| account              |
| transaction          |
+----------------------+
2 rows in set (0.00 sec)

mysql> select * from account;
+----+---------+------------------------------------------------------------------+----------+
| id | balance | password                                                         | username |
+----+---------+------------------------------------------------------------------+----------+
|  1 |    0.00 |                                                                  | ritesh   |
+----+---------+------------------------------------------------------------------+----------+
1 row in set (0.00 sec)

mysql> ^DBye
I have no name!@mysql-primary-0:/$
exit
```

For BankApp Deployment I did not use the SonarQube, Nexus and Trivy Image Scan. In this Project I did not use the Prometheus, Grafana, Loki and triggering Email for notification If you want to use these then you can refer the project present in the GitHub Repo https://github.com/singhritesh85/DevOps-Project-BankApplication-BlueGreen-Deployment-MultiCloud.git.

The screenshot for ArgoCD after successfully running both the Jenkins Job bankapp and mysql is as shown in the screenshot attached below.



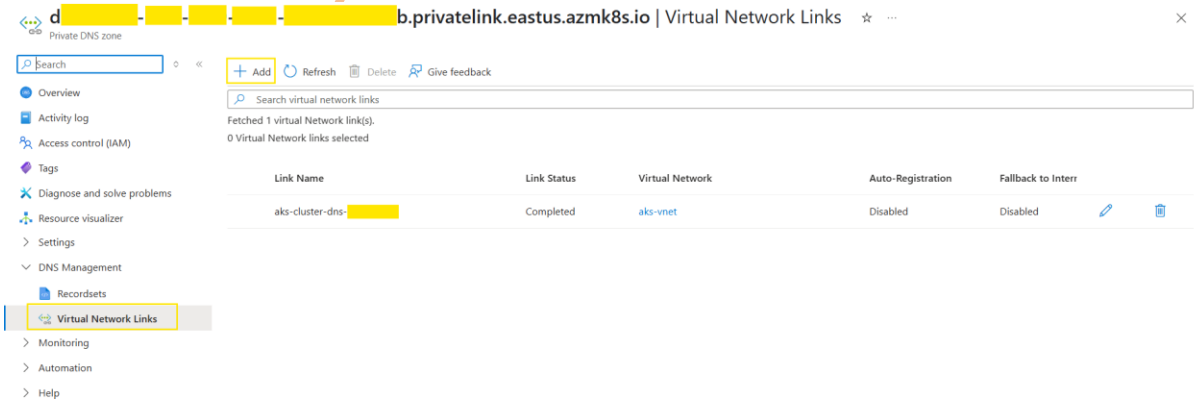
Module 4: DevOps-Project-Kubernetes-Multicluster-using-Consul-and-DisasterRecovery-Multicloud



A single mesh containing two Kubernetes clusters AKS Cluster and EKS Cluster

I had created a private AKS Cluster and an EKS Cluster using the terraform script present in the GitHub Repo https://github.com/singhritesh85/DevOps-Project-Kubernetes-Multicluster-Multicloud.git at the path **module-4/terraform-multi-kubernetes-cluster-multicloud**.

To access private AKS Cluster using kubectl from Terraform-Server/Jenkins Slave Node you need to create the virtual network link in private DNS Zone for AKS Cluster and create the VNet Peering between VNet of AKS Cluster and VNet of Terraform-Server/Jenkins Slave Node as shown in the screenshot attached below.



Add Virtual Network Link

[redacted]b.privatelink.eastus.azmk8s.io

Link name *

dexter

Virtual network details

Only virtual networks with Resource Manager deployment model are supported for linking with Private DNS zones. Virtual networks with Classic deployment model are not supported.

I know the resource ID of virtual network

Subscription *

Pay-As-You-Go

Virtual Network *

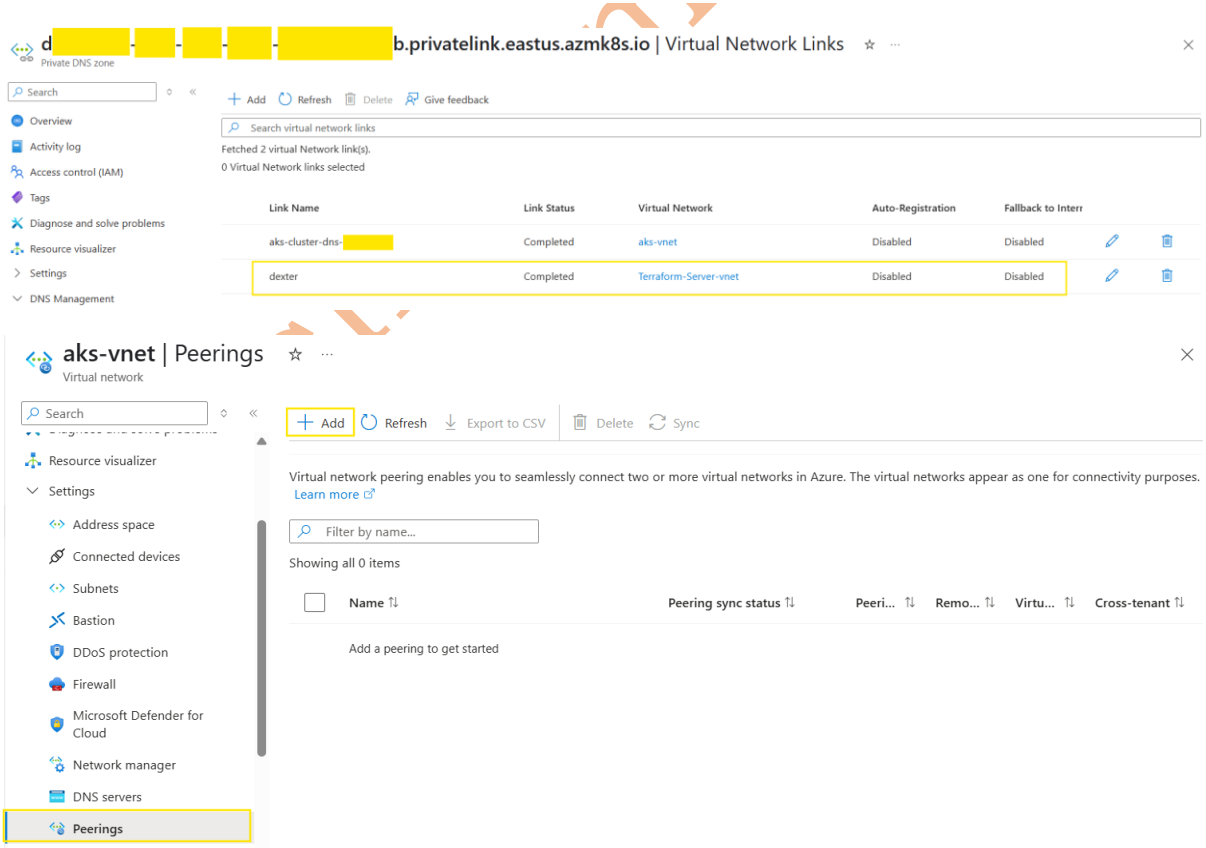
Terraform-Server-vnet (ritesh)

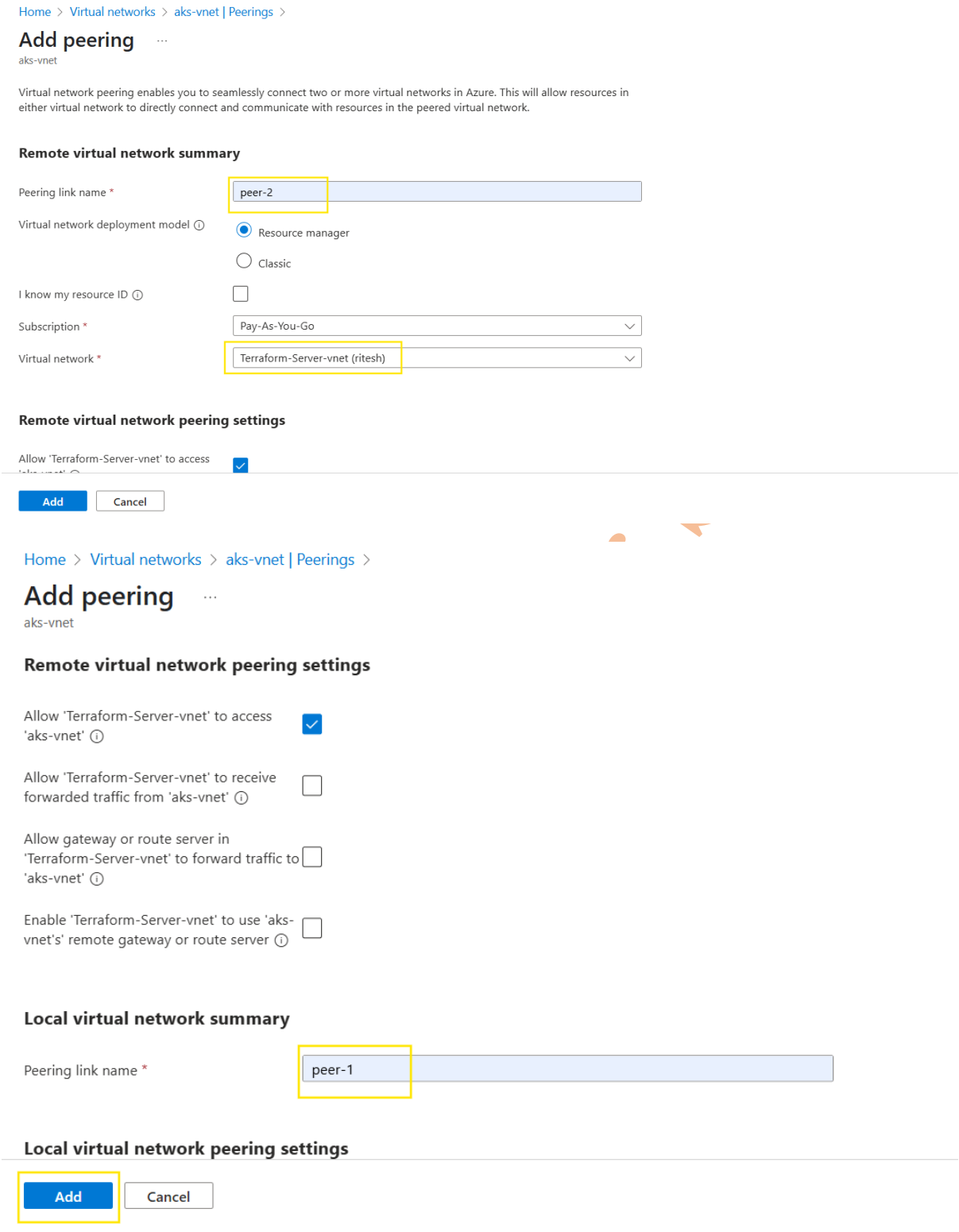
Configuration

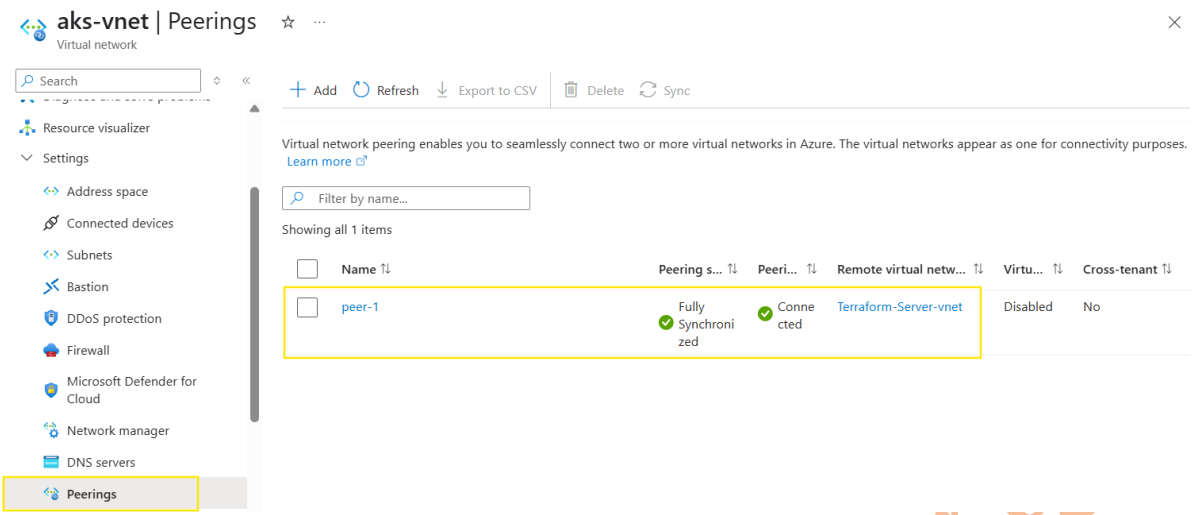
Enable auto registration

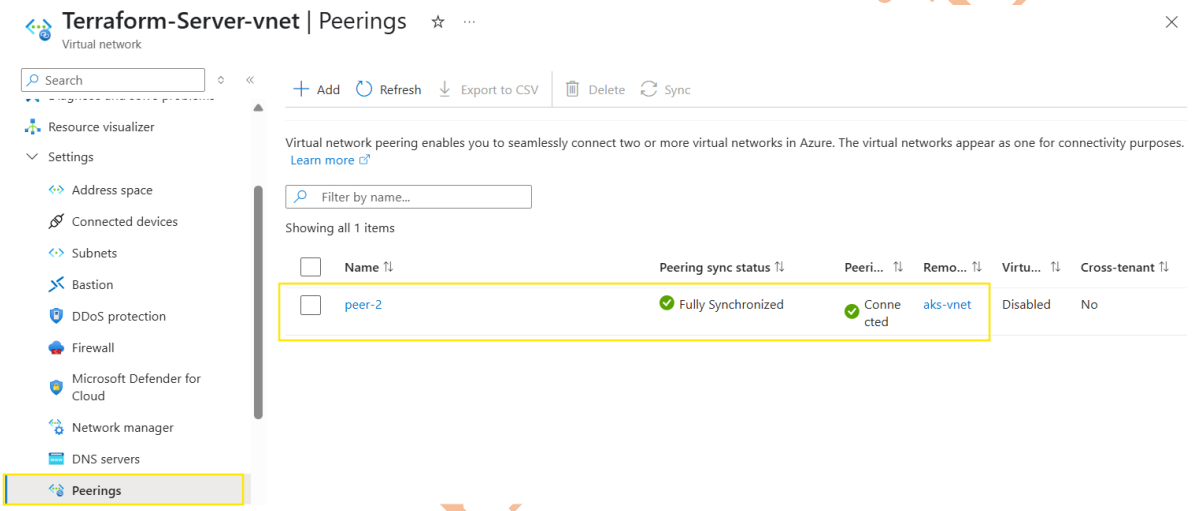
Enable fallback to internet

Create | Cancel | Give feedback




Home > Virtual networks > aks-vnet | Peerings >
Add peering
aks-vnet
Virtual network peering enables you to seamlessly connect two or more virtual networks in Azure. This will allow resources in either virtual network to directly connect and communicate with resources in the peered virtual network.
Remote virtual network summary
Peering link name *
peer-2
Virtual network deployment model
Resource manager
Classic
I know my resource ID
Subscription *
Pay-As-You-Go
Virtual network *
Terraform-Server-vnet (ritesh)
Remote virtual network peering settings
Allow 'Terraform-Server-vnet' to access
Add
Cancel
Home > Virtual networks > aks-vnet | Peerings >
Add peering
aks-vnet
Remote virtual network peering settings
Allow 'Terraform-Server-vnet' to access 'aks-vnet'
Allow 'Terraform-Server-vnet' to receive forwarded traffic from 'aks-vnet'
Allow gateway or route server in 'Terraform-Server-vnet' to forward traffic to 'aks-vnet'
Enable 'Terraform-Server-vnet' to use 'aks-vnet's' remote gateway or route server
Local virtual network summary
Peering link name *
peer-1
Local virtual network peering settings
Add
Cancel





Now, I can access the EKS Cluster and Private AKS Cluster using kubectl from Terraform-Server/Jenkins Slave Node as shown in the screenshot attached below.

```
[root@                    ~]# kubectl get nodes --context=eks-demo-cluster-dev
NAME                                          STATUS   ROLES    AGE   VERSION
ip-10-      -220.us-east-2.compute.internal   Ready    <none>   97m   v1.30.4-eks-a      9
ip-10-      -139.us-east-2.compute.internal   Ready    <none>   97m   v1.30.4-eks-a      9
ip-10-      -13.us-east-2.compute.internal    Ready    <none>   97m   v1.30.4-eks-a      9
[root@                    ~]# kubectl get nodes --context=aks-cluster
NAME                                STATUS   ROLES    AGE   VERSION
aks-agentpool-5      2-vmss000000   Ready    <none>   99m   v1.30.0
aks-agentpool-5      2-vmss000001   Ready    <none>   99m   v1.30.0
aks-userpool-2       0-vmss000000   Ready    <none>   89m   v1.30.0
aks-userpool-2       0-vmss000001   Ready    <none>   89m   v1.30.0
```

I had installed nginx ingress controller in AKS and EKS Cluster as shown in the screenshot attached below.

Installation of Nginx Ingress Controller in AKS Cluster

kubectl create ns ingress-nginx

helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx

helm repo update

helm install ingress-nginx ingress-nginx/ingress-nginx -n ingress-nginx

helm upgrade ingress-nginx ingress-nginx/ingress-nginx --namespace ingress-nginx --set controller.service.annotations."service\.beta\.kubernetes\.io/azure-load-balancer-health-probe-request-path"=/healthz --set controller.service.externalTrafficPolicy=Local

```
[root@                    ~]# kubectl config get-contexts
CURRENT   NAME                  CLUSTER               AUTHINFO                                                NAMESPACE
*         aks-cluster           aks-cluster           clusterUser_aks-rg_aks-cluster
          eks-demo-cluster-dev  eks-demo-cluster-dev  arn:aws:eks:us-east-2:02          6:cluster/eks-demo-cluster-dev
[root@                    ~]# kubectl create ns ingress-nginx
namespace/ingress-nginx created
[root@                    ~]# helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx
"ingress-nginx" has been added to your repositories
[root@                    ~]# helm repo update
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "ingress-nginx" chart repository
Update Complete. ⎈Happy Helming!⎈
[root@                    ~]# helm install ingress-nginx ingress-nginx/ingress-nginx -n ingress-nginx

[root@                    ~]# helm upgrade ingress-nginx ingress-nginx/ingress-nginx --namespace ingress-nginx --set controller.service.annotations."service\.bet
a\.kubernetes\.io/azure-load-balancer-health-probe-request-path"=/healthz --set controller.service.externalTrafficPolicy=Local
```

Installation of Nginx Ingress Controller in EKS Cluster

```
[root@                    ~]# kubectl config use-context eks-demo-cluster-dev
Switched to context "eks-demo-cluster-dev".
[root@Terraform-Server ~]# kubectl create ns ingress-nginx
namespace/ingress-nginx created
[root@                    ~]# helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx
"ingress-nginx" already exists with the same configuration, skipping
[root@                    ~]# helm repo update
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "ingress-nginx" chart repository
Update Complete. ⎈Happy Helming!⎈
[root@                    ~]# helm install ingress-nginx ingress-nginx/ingress-nginx -n ingress-nginx --set controller.service.annotations."service\.beta\.kubern
etes\.io/aws-load-balancer-ssl-cert"=arn:aws:acm:us-east-2:02          6:certificate/          -     -     -            --set controller.service.annotations.
"service\.beta\.kubernetes\.io/aws-load-balancer-connection-idle-timeout"="60" --set controller.service.annotations."service\.beta\.kubernetes\.io/aws-load-ba
lancer-cross-zone-load-balancing-enabled"="true" --set controller.service.annotations."service\.beta\.kubernetes\.io/aws-load-balancer-type"="elb" --set contr
oller.service.annotations."service\.beta\.kubernetes\.io/aws-load-balancer-backend-protocol"=http --set controller.service.annotations."service\.beta\.kuberne
tes\.io/aws-load-balancer-ssl-ports"=https --set controller.service.targetPorts.https=http --set-string controller.config.use-forwarded-headers="true"
```

kubectl create ns ingress-nginx

helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx

helm repo update

helm install ingress-nginx ingress-nginx/ingress-nginx -n ingress-nginx --set controller.service.annotations."service\.beta\.kubernetes\.io/aws-load-balancer-ssl-cert"=arn:aws:acm:us-east-2:02XXXXXXXXX6:certificate/XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX --set controller.service.annotations."service\.beta\.kubernetes\.io/aws-load-balancer-connection-idle-timeout"="60" --set controller.service.annotations."service\.beta\.kubernetes\.io/aws-load-balancer-cross-zone-load-balancing-enabled"="true" --set controller.service.annotations."service\.beta\.kubernetes\.io/aws-load-balancer-type"="elb" --set controller.service.annotations."service\.beta\.kubernetes\.io/aws-load-balancer-backend-protocol"=http --set controller.service.annotations."service\.beta\.kubernetes\.io/aws-load-balancer-ssl-ports"=https --set controller.service.targetPorts.https=http --set-string controller.config.use-forwarded-headers="true"

```
[root@                ~]# kubectl get all -n ingress-nginx --context=aks-cluster
NAME                                        READY   STATUS    RESTARTS   AGE
pod/ingress-nginx-controller-6              1/1     Running   0          10m

NAME                                 TYPE           CLUSTER-IP    EXTERNAL-IP     PORT(S)                      AGE
service/ingress-nginx-controller             LoadBalancer   10.     5     4.      .192   80:32127/TCP,443:31056/TCP   10m
service/ingress-nginx-controller-admission   ClusterIP      10.     .96   <none>          443/TCP                      10m

NAME                                    READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/ingress-nginx-controller   1/1     1            1           10m

NAME                                              DESIRED   CURRENT   READY   AGE
replicaset.apps/ingress-nginx-controller-6        4         1         1       1       10m
[root@Terraform-Server ~]# kubectl get all -n ingress-nginx --context=eks-demo-cluster-dev
NAME                                        READY   STATUS    RESTARTS   AGE
pod/ingress-nginx-controller-6              1/1     Running   0          2m18s

NAME                                 TYPE           CLUSTER-IP    EXTERNAL-IP                                                                  PORT(S)
                AGE
service/ingress-nginx-controller             LoadBalancer   10.     .93   a                                                1.us-east-2.elb.amazonaws.com   80:31301/TC
P,443:32714/TCP   2m19s
service/ingress-nginx-controller-admission   ClusterIP      10.     250   <none>                                                                       443/TCP
                2m19s

NAME                                    READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/ingress-nginx-controller   1/1     1            1           2m19s

NAME                                              DESIRED   CURRENT   READY   AGE
replicaset.apps/ingress-nginx-controller-6        4         1         1       2m19s
```

```
[root@[redacted] ~]# cat consul-values.yaml
global:
  image: "hashicorp/consul:1.20.5"
  peering:
    enabled: true
  tls:
    enabled: true

server:
  replicas: 1
  bootstrapExpect: 1
  extraConfig: |
    {
      "log_level": "TRACE"
    }

connectInject:
  enabled: true
  default: false

meshGateway:
  enabled: true
  replicas: 1

controller:
  enabled: true

ui:
  enabled: true
  service:
    enabled: true
    type: ClusterIP
```

```
cat consul-values.yaml
global:
  image: "hashicorp/consul:1.20.5"
  peering:
    enabled: true
  tls:
    enabled: true

server:
  replicas: 1
  bootstrapExpect: 1
  extraConfig: |
    {
      "log_level": "TRACE"
    }

connectInject:
  enabled: true
  default: false

meshGateway:
  enabled: true
  replicas: 1

controller:
  enabled: true

ui:
  enabled: true
  service:
    enabled: true
    type: ClusterIP
```

Then I installed Consul (Service Mesh) in AKS and EKS Cluster as shown in the screenshot attached blow.

Installation of Consul in AKS Cluster

helm repo add hashicorp https://helm.releases.hashicorp.com --kube-context=aks-cluster

helm search repo hashicorp/consul --kube-context=aks-cluster

helm upgrade --install consul-aks hashicorp/consul --version=1.6.3 --values=consul-values.yaml --set global.datacenter=aks --set server.storage=10Gi --set server.storageClass=managed-csi --namespace=consul --create-namespace --kube-context=aks-cluster

```
[root@                ~]# kubectl config use-context aks-cluster
Switched to context "aks-cluster".
[root@                ~]# helm repo add hashicorp https://helm.releases.hashicorp.com --kube-context=aks-cluster
"hashicorp" has been added to your repositories
[root@                ~]# helm search repo hashicorp/consul --kube-context=aks-cluster
NAME                    CHART VERSION   APP VERSION     DESCRIPTION
hashicorp/consul        1.6.4           1.20.6          Official HashiCorp Consul Chart
[root@                ~]# helm upgrade --install consul-aks hashicorp/consul --version=1.6.3 --values=consul-values.yaml --set global.datacenter=aks --set se
rver.storage=10Gi --set server.storageClass=managed-csi  --namespace=consul --create-namespace --kube-context=aks-cluster
```

```
[root@                ~]# kubectl get all -n consul --context=aks-cluster
NAME                                                READY   STATUS    RESTARTS   AGE
pod/consul-aks-consul-connect-injector-5            1/1     Running   0          3m10s
pod/consul-aks-consul-mesh-gateway-5                1/1     Running   0          3m10s
pod/consul-aks-consul-server-0                      1/1     Running   0          3m10s
pod/consul-aks-consul-webhook-cert-manager-6        1/1     Running   0          3m10s

NAME                                    TYPE           CLUSTER-IP     EXTERNAL-IP     PORT(S)
              AGE
service/consul-aks-consul-connect-injector   ClusterIP      10.      .76    <none>          443/TCP
              3m10s
service/consul-aks-consul-dns                ClusterIP      10.      165    <none>          53/TCP,53/UDP
              3m10s
service/consul-aks-consul-mesh-gateway       LoadBalancer   10.      124    4.      .162    443:30408/TCP
              3m10s
service/consul-aks-consul-server             ClusterIP      None           <none>          8501/TCP,8502/TCP,8301/TCP,8301/UDP,8302/TCP,8302/UDP,8300/TCP,8600
/TCP,8600/UDP   3m10s
service/consul-aks-consul-ui                 ClusterIP      10.      .103   <none>          443/TCP
              3m10s

NAME                                           READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/consul-aks-consul-connect-injector   1/1     1            1           3m11s
deployment.apps/consul-aks-consul-mesh-gateway       1/1     1            1           3m11s
deployment.apps/consul-aks-consul-webhook-cert-manager   1/1     1            1           3m11s

NAME                                                      DESIRED   CURRENT   READY   AGE
replicaset.apps/consul-aks-consul-connect-injector-5      1         1         1       3m11s
replicaset.apps/consul-aks-consul-mesh-gateway-5          1         1         1       3m11s
replicaset.apps/consul-aks-consul-webhook-cert-manager-6  1         1         1       3m11s

NAME                                       READY   AGE
statefulset.apps/consul-aks-consul-server  1/1     3m11s
```

Created a HOST with External IP using ingress rule as shown in the screenshot attached below. Its entry I did in Azure DNS Zone to create the Record Set of A Type.

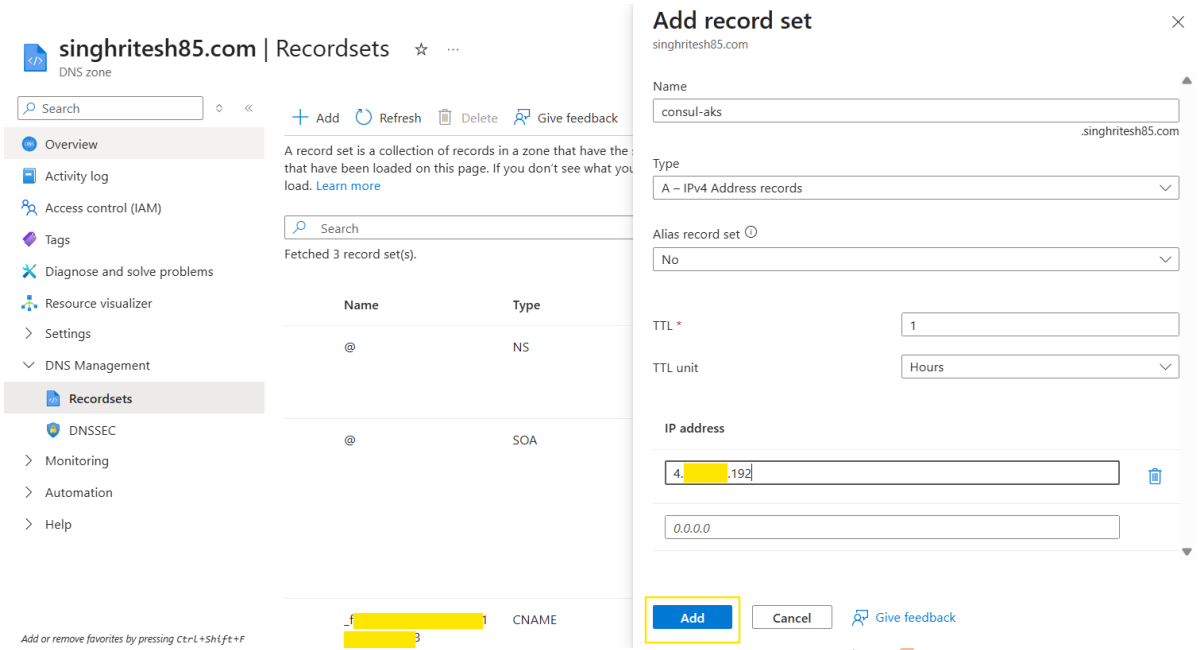
```
[root@                ~]# cat consul-aks-ingress-rule.yaml
# kubectl create secret tls consul-secret --cert=STAR_singhritesh85_com.crt --key=mykey.key -n consul --context=aks-cluster
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: consul-ingress
  namespace: consul
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/backend-protocol: HTTPS
    nginx.ingress.kubernetes.io/force-ssl-redirect: "true"
spec:
  ingressClassName: nginx
  tls:
  - hosts:
    - consul-aks.singhritesh85.com
    secretName: consul-secret
  rules:
  - host: consul-aks.singhritesh85.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: consul-aks-consul-ui
            port:
              number: 443

[root@                ~]# kubectl create secret tls consul-secret --cert=STAR_singhritesh85_com.crt --key=mykey.key -n consul --context=aks-cluster
secret/consul-secret created
[root@                ~]# kubectl apply -f consul-aks-ingress-rule.yaml --context=aks-cluster
ingress.networking.k8s.io/consul-ingress created
[root@                ~]# kubectl get ing -A --watch --context=aks-cluster
NAMESPACE   NAME             CLASS   HOSTS                        ADDRESS         PORTS     AGE
consul      consul-ingress   nginx   consul-aks.singhritesh85.com   4.      .192   80, 443   15s
```

cat consul-aks-ingress-rule.yaml

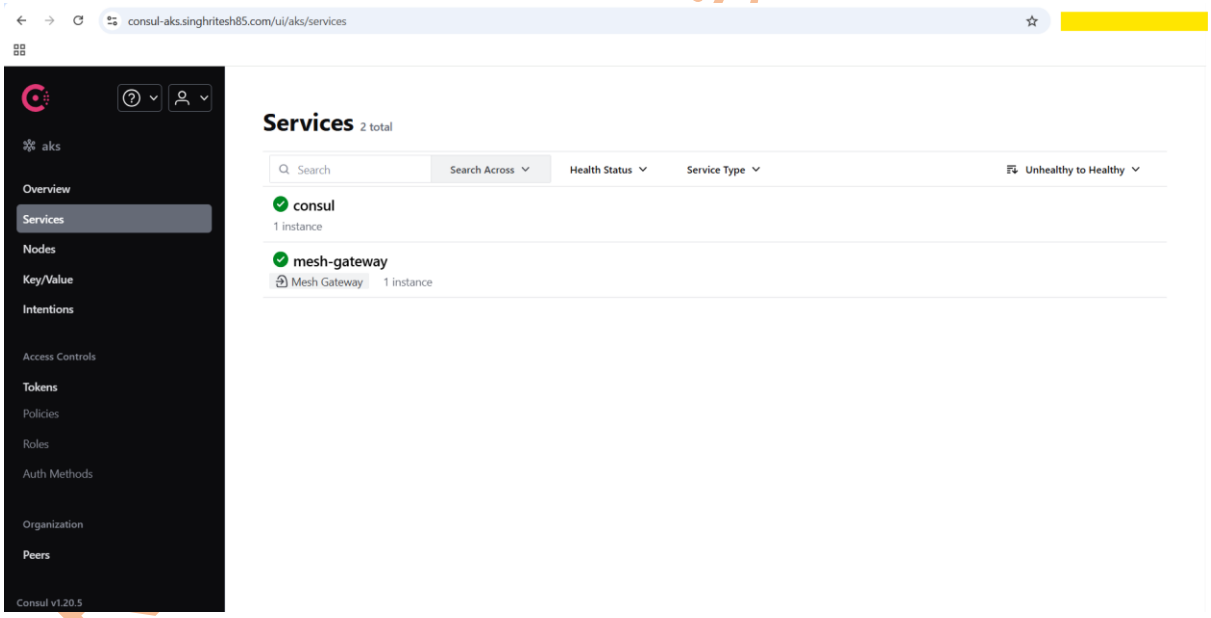
kubectl create secret tls consul-secret --cert=STAR_singhritesh85_com.crt --key=mykey.key -n consul --context=aks-cluster

---

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: consul-ingress
  namespace: consul
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/backend-protocol: HTTPS
    nginx.ingress.kubernetes.io/force-ssl-redirect: "true"
spec:
  ingressClassName: nginx
  tls:
  - hosts:
    - consul-aks.singhritesh85.com
    secretName: consul-secret
  rules:
  - host: consul-aks.singhritesh85.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: consul-aks-consul-ui
            port:
              number: 443
```



Finally, I was able to access Consul as shown in the screenshot attached below.



Installation of Consul in AKS Cluster

helm repo add hashicorp https://helm.releases.hashicorp.com --kube-context=eks-demo-cluster-dev

helm search repo hashicorp/consul --kube-context=eks-demo-cluster-dev

helm upgrade --install consul-eks hashicorp/consul --version=1.6.3 --values=consul-values.yaml --set global.datacenter=eks --set server.storage=10Gi --set server.storageClass=gp2 --namespace=consul --create-namespace --kube-context=eks-demo-cluster-dev

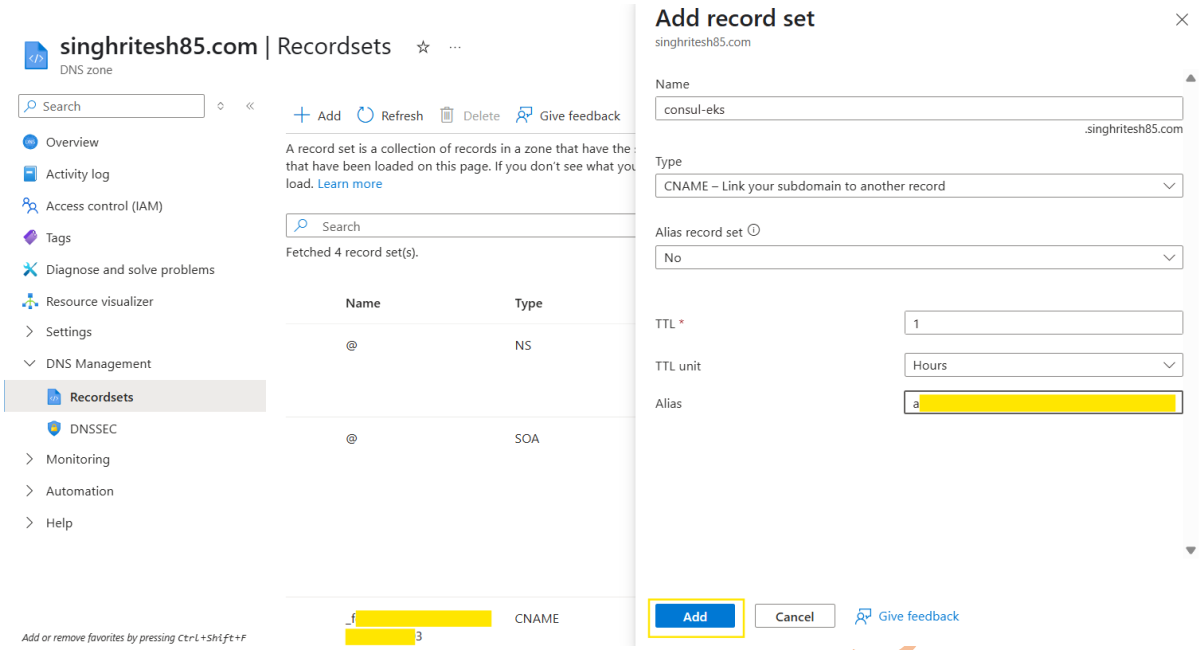
```
[root@ ~]# kubectl config use-context eks-demo-cluster-dev
Switched to context "eks-demo-cluster-dev".
[root@ ~]# helm repo add hashicorp https://helm.releases.hashicorp.com --kube-context=eks-demo-cluster-dev
"hashicorp" already exists with the same configuration, skipping
[root@ ~]# helm search repo hashicorp/consul --kube-context=eks-demo-cluster-dev
NAME                    CHART VERSION   APP VERSION     DESCRIPTION
hashicorp/consul        1.6.4           1.20.6          Official HashiCorp Consul Chart
[root@ ~]# helm upgrade --install consul-eks hashicorp/consul --version=1.6.3 --values=consul-values.yaml --set global.datacenter=eks --set se
rver.storage=10Gi --set server.storageClass=gp2  --namespace=consul --create-namespace --kube-context=eks-demo-cluster-dev
```

```
[root@ ~]# kubectl get all -n consul --context=eks-demo-cluster-dev
NAME                                                  READY   STATUS    RESTARTS   AGE
pod/consul-eks-consul-connect-injector-6              1/1     Running   0          87s
pod/consul-eks-consul-mesh-gateway-5                  1/1     Running   0          87s
pod/consul-eks-consul-server-0                        1/1     Running   0          87s
pod/consul-eks-consul-webhook-cert-manager-5          1/1     Running   0          87s

NAME                                        TYPE           CLUSTER-IP     EXTERNAL-IP                                                              PORT(S)
                                                                          AGE
service/consul-eks-consul-connect-injector  ClusterIP      10.    .205    <none>                                                                   443/TCP
                                                                          89s
service/consul-eks-consul-dns               ClusterIP      10.    .43     <none>                                                                   53/TCP,53
/UDP                                                                      89s
service/consul-eks-consul-mesh-gateway      LoadBalancer   10.    .27     a                                    0.us-east-2.elb.amazonaws.com       443:32144
/TCP                                                                      89s
service/consul-eks-consul-server            ClusterIP      None           <none>                                                                   8501/TCP,
8502/TCP,8301/TCP,8301/UDP,8302/TCP,8302/UDP,8300/TCP,8600/TCP,8600/UDP   89s
service/consul-eks-consul-ui                ClusterIP      10.    .9      <none>                                                                   443/TCP
                                                                          89s

NAME                                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/consul-eks-consul-connect-injector  1/1     1            1           89s
deployment.apps/consul-eks-consul-mesh-gateway      1/1     1            1           89s
deployment.apps/consul-eks-consul-webhook-cert-manager   1/1     1            1           89s

NAME                                                        DESIRED   CURRENT   READY   AGE
replicaset.apps/consul-eks-consul-connect-injector-6        1         1         1       89s
replicaset.apps/consul-eks-consul-mesh-gateway-5            1         1         1       89s
replicaset.apps/consul-eks-consul-webhook-cert-manager-5    1         1         1       89s

NAME                                       READY   AGE
statefulset.apps/consul-eks-consul-server  1/1     88s
```

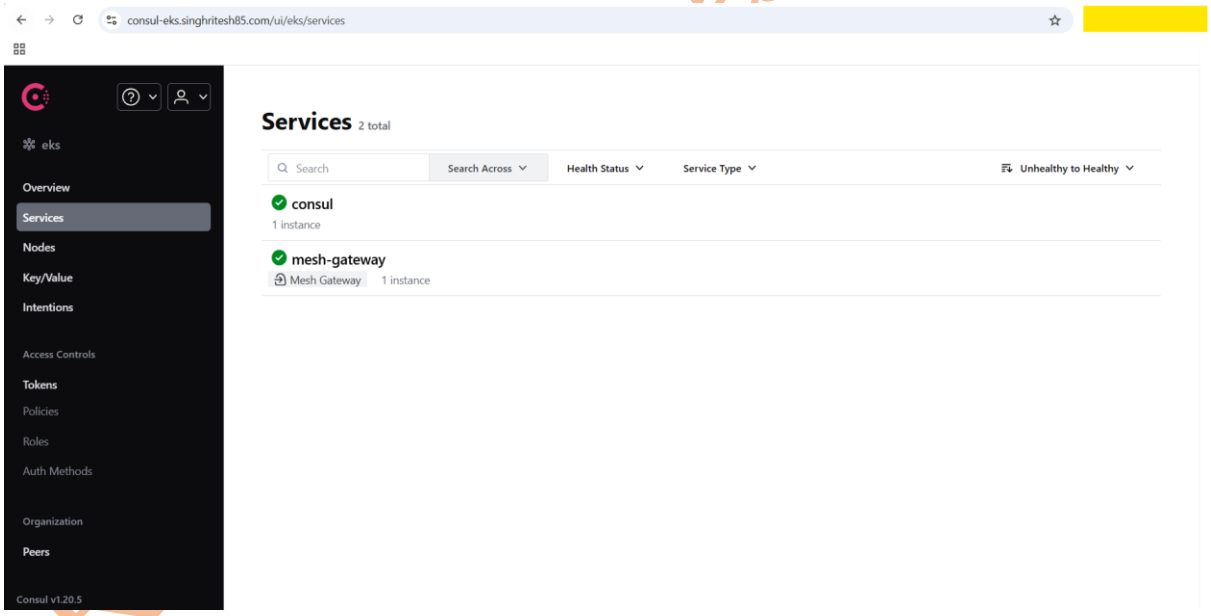
Created a HOST with DNS Name of LoadBalancer using ingress rule as shown in the screenshot attached below. Its entry I did in Azure DNS Zone to create the Record Set of CNAME Type.

```
[root@ ~]# cat consul-eks-ingress-rule.yaml
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: consul-ingress
  namespace: consul
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/backend-protocol: HTTPS
    nginx.ingress.kubernetes.io/force-ssl-redirect: "true"
spec:
  ingressClassName: nginx
  rules:
  - host: consul-eks.singhritesh85.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: consul-eks-consul-ui
            port:
              number: 443

[root@ ~]# kubectl apply -f consul-eks-ingress-rule.yaml --context=eks-demo-cluster-dev
ingress.networking.k8s.io/consul-ingress created
[root@ ~]# kubectl get ing -A --watch --context=eks-demo-cluster-dev
NAMESPACE   NAME             CLASS   HOSTS                        ADDRESS                                                  PORTS   AGE
consul      consul-ingress   nginx   consul-eks.singhritesh85.com                                                         80      14s
consul      consul-ingress   nginx   consul-eks.singhritesh85.com   a                                    1.us-east-2.elb.amazonaws.com   80      24s
```

```
cat consul-eks-ingress-rule.yaml
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: consul-ingress
  namespace: consul
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/backend-protocol: HTTPS
    nginx.ingress.kubernetes.io/force-ssl-redirect: "true"
spec:
  ingressClassName: nginx
  rules:
  - host: consul-eks.singhritesh85.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: consul-eks-consul-ui
            port:
              number: 443
```



Finally, I was able to access Consul as shown in the screenshot attached below.



Now my aim is to deploy the online boutique microservice based application in EKS and AKS Cluster to achieve the Disaster Recovery (DR). I am elaborating more on the aim if any of the service (consider as shippingservice) gets deleted in EKS Cluster then the service can be accessible through the microservice deployed in AKS Cluster.

To achieve this aim, I had created a namespace **microservice** in EKS and AKS Cluster. Then I had applied the label **connect-inject=enabled** to the namespace **microservice** to create the envoy proxy sidecar containers in the EKS and AKS Cluster. Although I had applied the label to the namespace but I should also apply the annotations **consul.hashicorp.com/connect-inject: 'true'** with each of the deployment in that namespace.

kubectl create ns microservice --context=eks-demo-cluster-dev

kubectl create ns microservice --context=aks-cluster

kubectl label --context=aks-cluster namespace microservice connect-inject=enabled

kubectl label --context=eks-demo-cluster-dev namespace microservice connect-inject=enabled

```
[root@                ~]# kubectl create ns microservice --context=eks-demo-cluster-dev
namespace/microservice created
[root@                ~]# kubectl create ns microservice --context=aks-cluster
namespace/microservice created
[root@                ~]# kubectl label --context=aks-cluster namespace microservice connect-inject=enabled
namespace/microservice labeled
[root@                ~]# kubectl label --context=eks-demo-cluster-dev namespace microservice connect-inject=enabled
namespace/microservice labeled
```

```
[root@                release]# kubectl get ns microservice --show-labels --context=eks-demo-cluster-dev
NAME           STATUS   AGE   LABELS
microservice   Active   16m   connect-inject=enabled,kubernetes.io/metadata.name=microservice
[root@                release]# kubectl get ns microservice --show-labels --context=aks-cluster
NAME           STATUS   AGE   LABELS
microservice   Active   16m   connect-inject=enabled,kubernetes.io/metadata.name=microservice
```

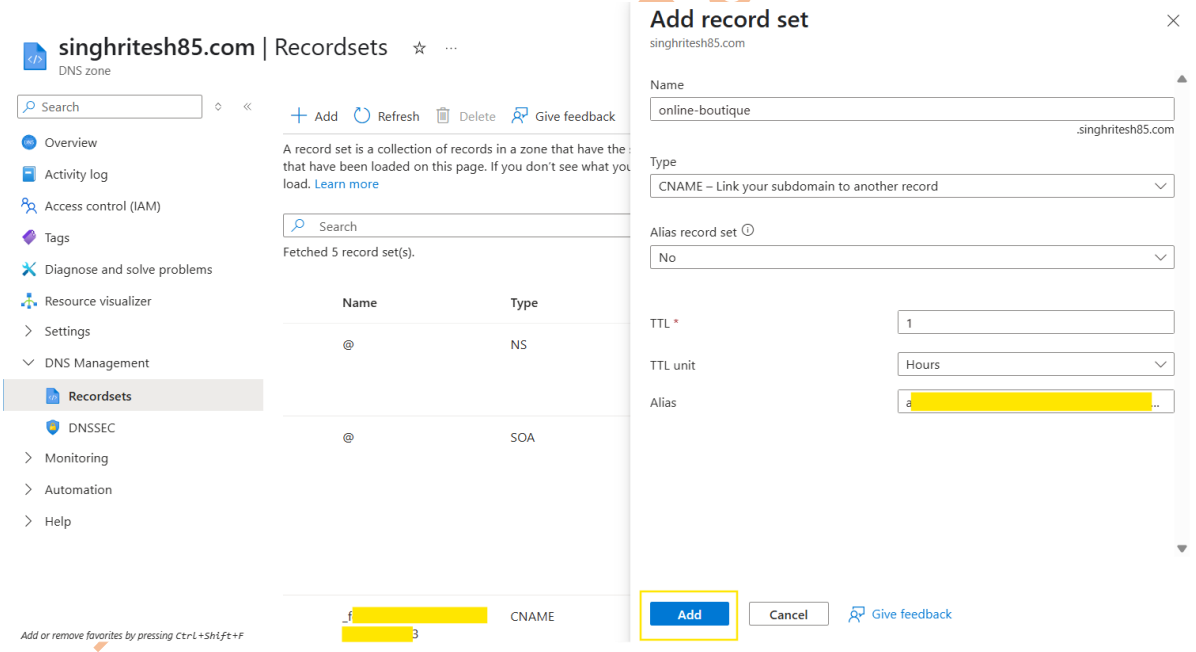
Then I had deployed the online boutique microservice based application in EKS and AKS Cluster using the manifests file which is present is the GitHub Repo https://github.com/singhritesh85/consul-crash-course.git at the path **kubernetes** and the file name is **config-consul.yaml**.

```
[root@                ~]# git clone https://github.com/singhritesh85/consul-crash-course.git
Cloning into 'consul-crash-course'...
remote: Enumerating objects: 15, done.
remote: Counting objects: 100% (15/15), done.
remote: Compressing objects: 100% (14/14), done.
remote: Total 15 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (15/15), 5.72 KiB | 5.72 MiB/s, done.
Resolving deltas: 100% (1/1), done.
[root@                ~]# cd consul-crash-course/kubernetes/
[root@                kubernetes]# kubectl apply -f config-consul.yaml -n microservice --context=aks-cluster
```

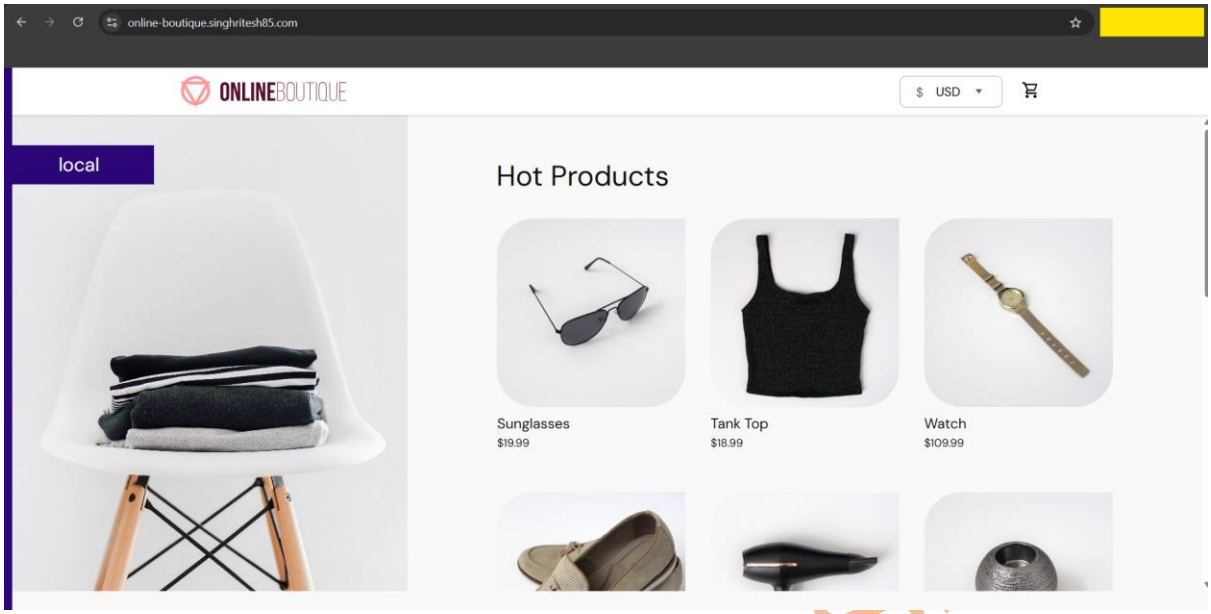
```
[root@                kubernetes]# kubectl apply -f config-consul.yaml -n microservice --context=eks-demo-cluster-dev
```

```
[root@                    kubernetes]# kubectl get pods -n microservice --context=eks-demo-cluster-dev
NAME                                     READY   STATUS             RESTARTS      AGE
adservice-857bf87485-lc6jr               2/2     Running            0             38s
cartservice-644f9d65cd-ksw7p             2/2     Running            0             38s
checkoutservice-5696787454-hxx28         2/2     Running            0             38s
currencyservice-6d99654d6f-n4dvk         2/2     Running            0             39s
emailservice-74698d99d9-w42bx            2/2     Running            0             39s
frontend-5476787c94-fp669                2/2     Running            0             38s
paymentservice-669944c9f-9mjht           1/2     CrashLoopBackOff   2 (14s ago)   39s
productcatalogservice-98fcdf4b5-krzft    2/2     Running            0             39s
recommendationservice-77d79ff5d5-fksp8   2/2     Running            0             39s
redis-cart-689bb95f4-ccscr               2/2     Running            0             38s
shippingservice-66cd9846c7-q66d8         2/2     Running            0             38s
[root@                    kubernetes]# kubectl get pods -n microservice --context=aks-cluster
NAME                                     READY   STATUS             RESTARTS       AGE
adservice-857bf87485-7mj4m               2/2     Running            0              27m
cartservice-644f9d65cd-b4rnm             2/2     Running            0              27m
checkoutservice-5696787454-zx6vj         2/2     Running            0              27m
currencyservice-6d99654d6f-6mr4p         2/2     Running            0              27m
emailservice-74698d99d9-sdvfw            2/2     Running            0              27m
frontend-5476787c94-h6bt6                2/2     Running            0              27m
paymentservice-669944c9f-7gg7q           1/2     CrashLoopBackOff   10 (37s ago)   27m
productcatalogservice-98fcdf4b5-ft4nv    2/2     Running            0              27m
recommendationservice-77d79ff5d5-hjw4s   2/2     Running            0              27m
redis-cart-689bb95f4-7kqjz               2/2     Running            0              27m
shippingservice-66cd9846c7-zsgg5         2/2     Running            0              27m
```

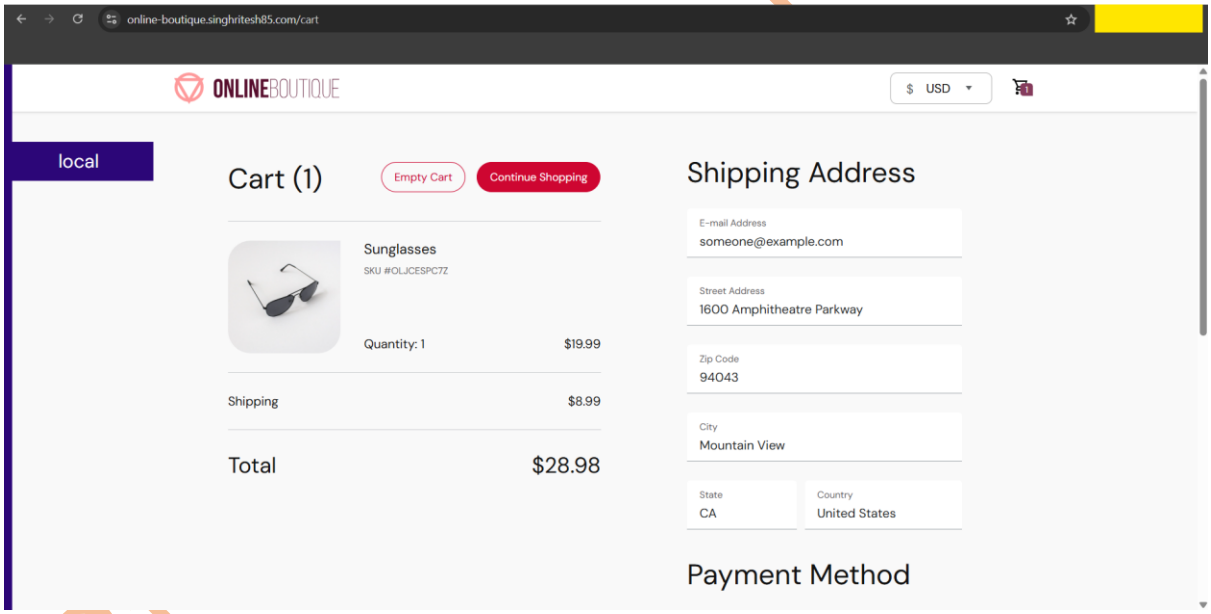
Then I created the ingress rule and hence the HOST and DNS Name of the LoadBalancer which entry I did in the Azure DNS Zone to create the Record Set of type CNAME as shown in the screenshot attached below.



Finally, accessed the online boutique application as shown in the screenshot attached below.



At this stage shippingservice is working properly which can be seen in the screenshot attached below.



To configure Disaster Recovery first I need to connect the two clusters (EKS and AKS Clusters) using peering.

To use mesh gateway to send or accept the peering connection I need to install the CRD Mesh on both the EKS and AKS Clusters as shown in the screenshot attached below.

```
[root@                kubernetes]# kubectl apply -f consul-mesh-gateway.yaml -n consul --context=aks-cluster
mesh.consul.hashicorp.com/mesh created
[root@                kubernetes]# kubectl apply -f consul-mesh-gateway.yaml -n consul --context=eks-demo-cluster-dev
mesh.consul.hashicorp.com/mesh created
[root@                kubernetes]# kubectl get mesh -n consul --context=aks-cluster
NAME   SYNCED   LAST SYNCED   AGE
mesh   True     28s           28s
[root@                kubernetes]# kubectl get mesh -n consul --context=eks-demo-cluster-dev
NAME   SYNCED   LAST SYNCED   AGE
mesh   True     23s           23s
```

```
[root@                    kubernetes]# cat consul-mesh-gateway.yaml
apiVersion: consul.hashicorp.com/v1alpha1
kind: Mesh
metadata:
  name: mesh
spec:
  peering:
    peerThroughMeshGateways: true
```

cat consul-mesh-gateway.yaml

apiVersion: consul.hashicorp.com/v1alpha1

kind: Mesh

metadata:

name: mesh

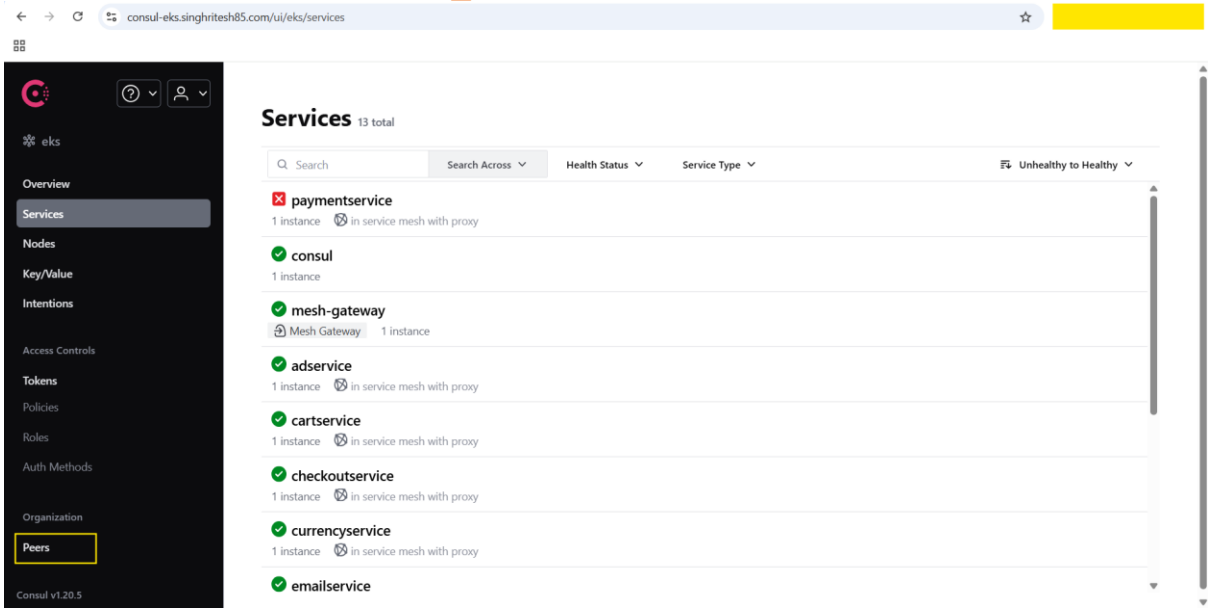
spec:

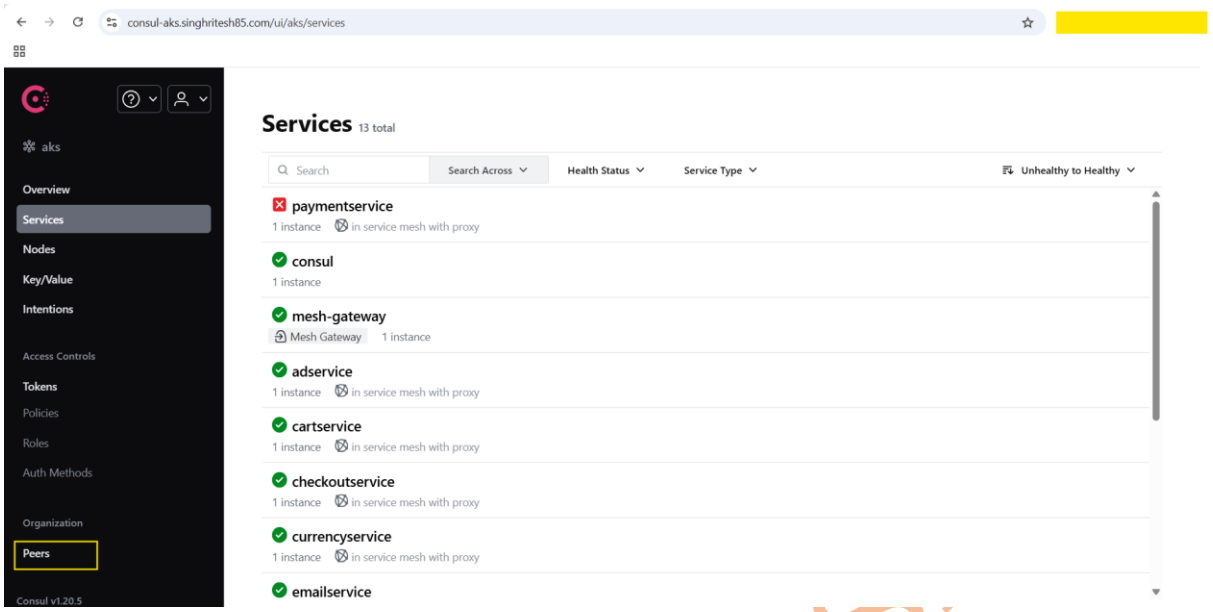
peering:

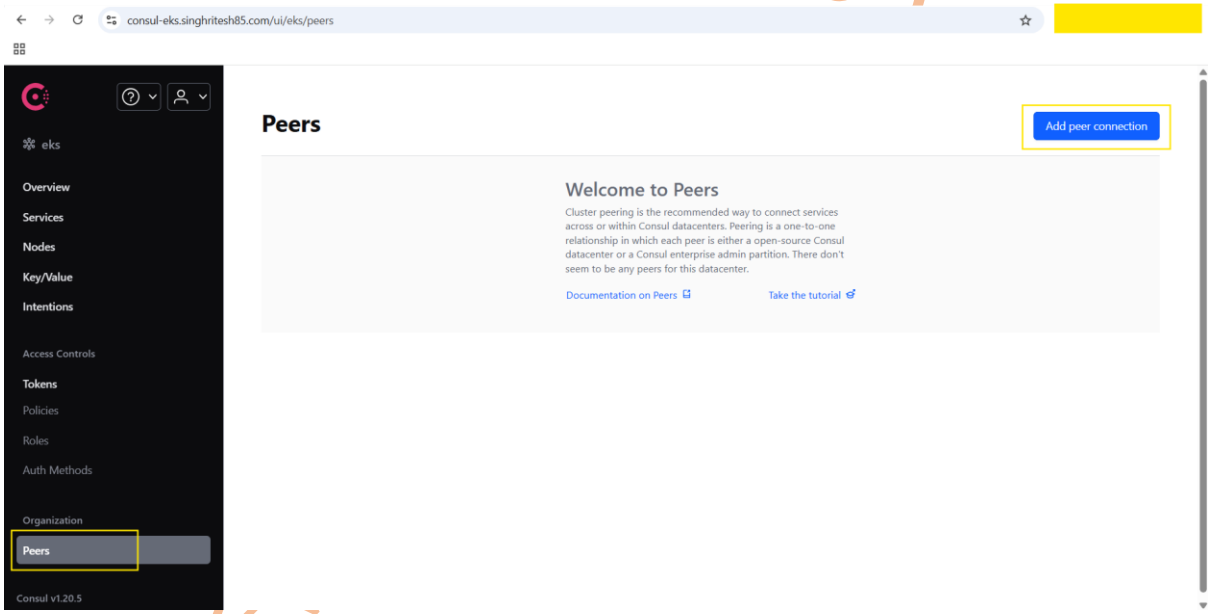
peerThroughMeshGateways: true

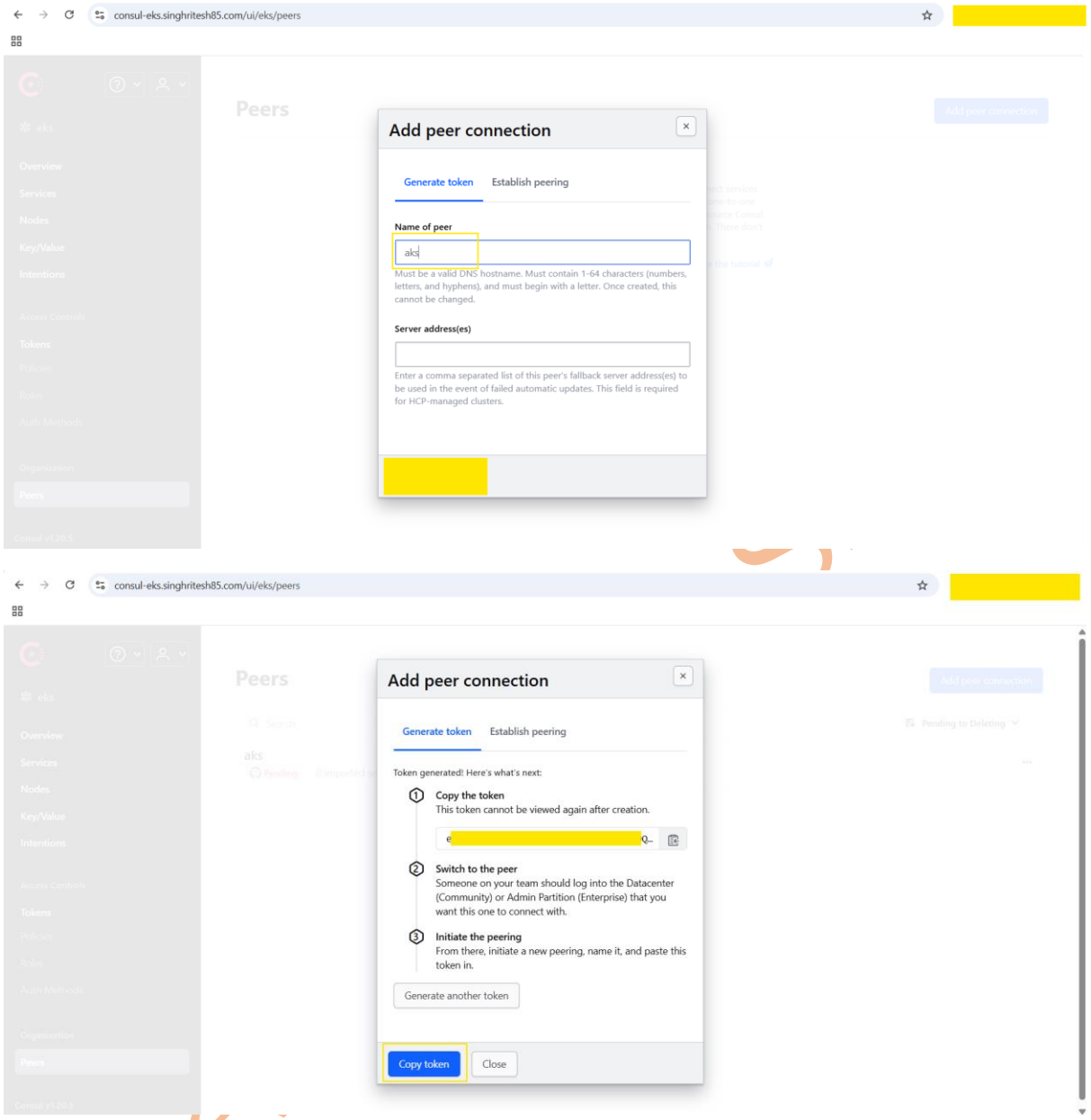
The Mesh CRD I installed on both the EKS and AKS Cluster allows to Route traffic through Mesh Gateway.

Now I went to consul-ui of EKS Cluster and AKS Cluster and configured the peering as shown in the screenshot attached below.

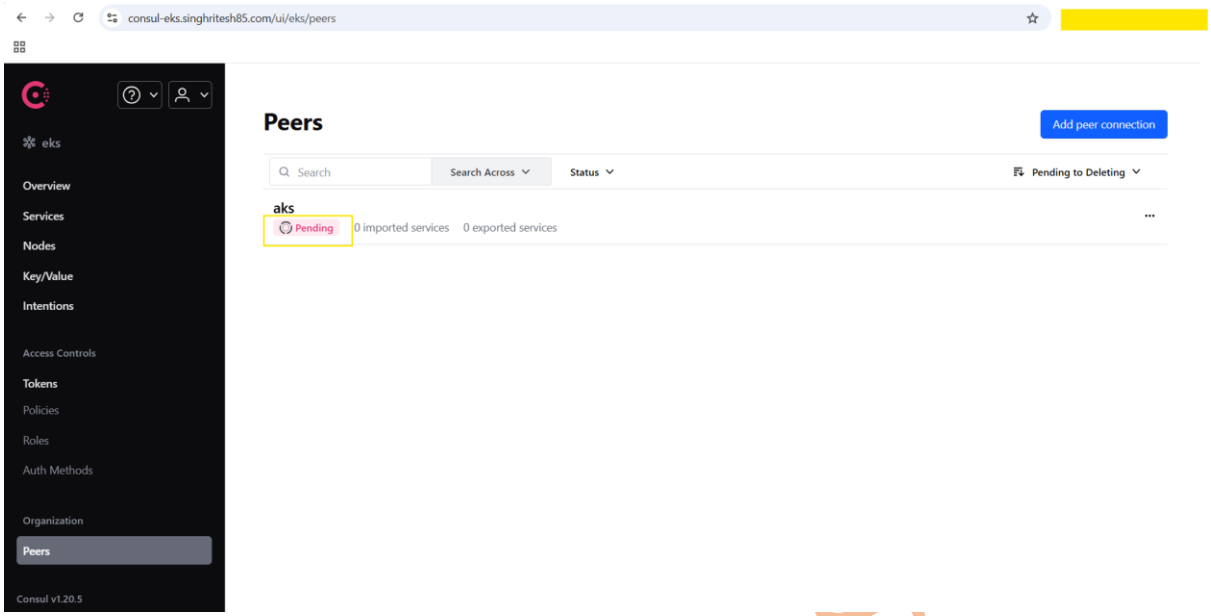


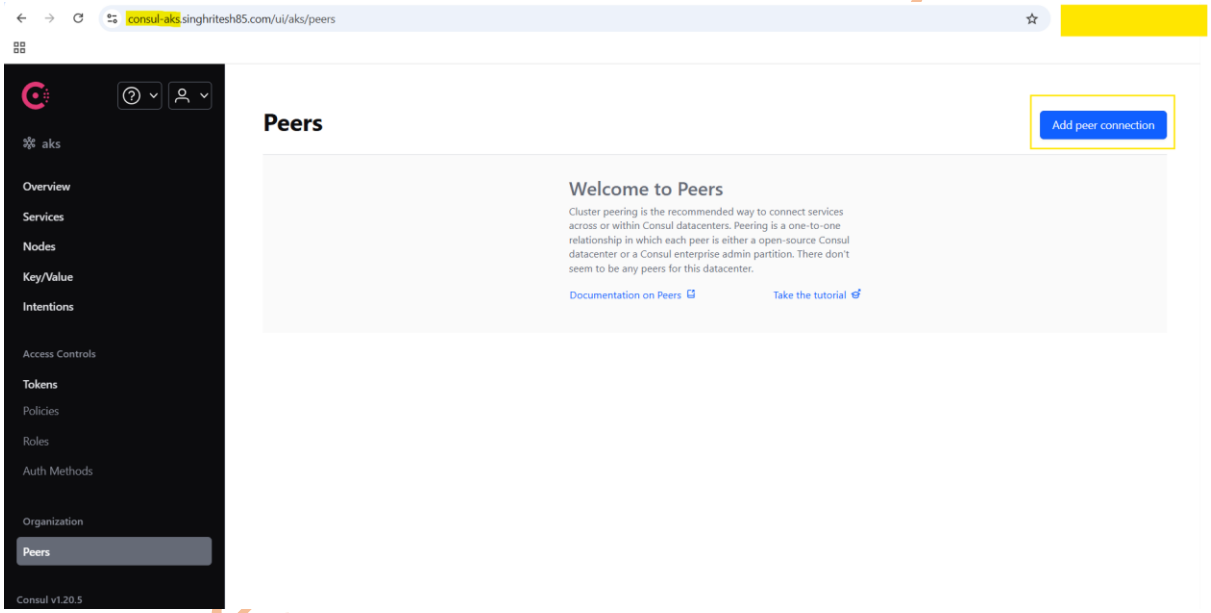

consul-aks.singhritesh85.com/ui/aks/services
aks
Overview
Services
Nodes
Key/Value
Intentions
Access Controls
Tokens
Policies
Roles
Auth Methods
Organization
Peers
Consul v1.20.5
Services 13 total
Search
Search Across
Health Status
Service Type
Unhealthy to Healthy
paymentservice
1 instance
in service mesh with proxy
consul
1 instance
mesh-gateway
Mesh Gateway
1 instance
adservice
1 instance
in service mesh with proxy
cartservice
1 instance
in service mesh with proxy
checkoutservice
1 instance
in service mesh with proxy
currencyservice
1 instance
in service mesh with proxy
emailservice

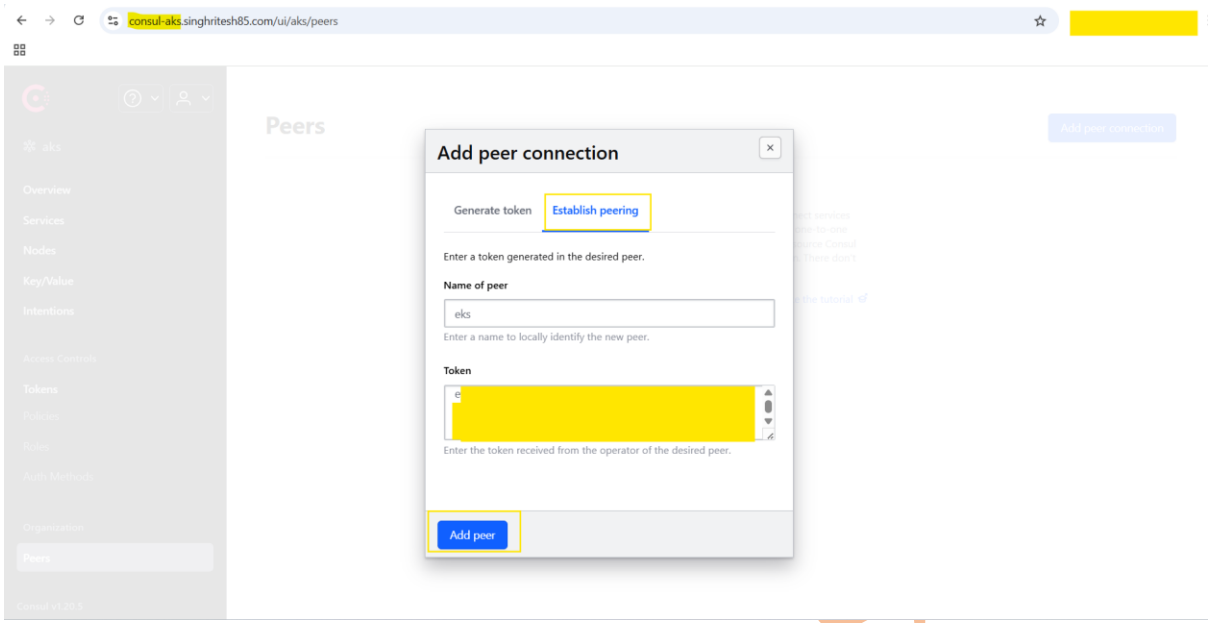

consul-eks.singhritesh85.com/ui/eks/peers
eks
Overview
Services
Nodes
Key/Value
Intentions
Access Controls
Tokens
Policies
Roles
Auth Methods
Organization
Peers
Consul v1.20.5
Peers
Add peer connection
Welcome to Peers
Cluster peering is the recommended way to connect services across or within Consul datacenters. Peering is a one-to-one relationship in which each peer is either a open-source Consul datacenter or a Consul enterprise admin partition. There don't seem to be any peers for this datacenter.
Documentation on Peers
Take the tutorial



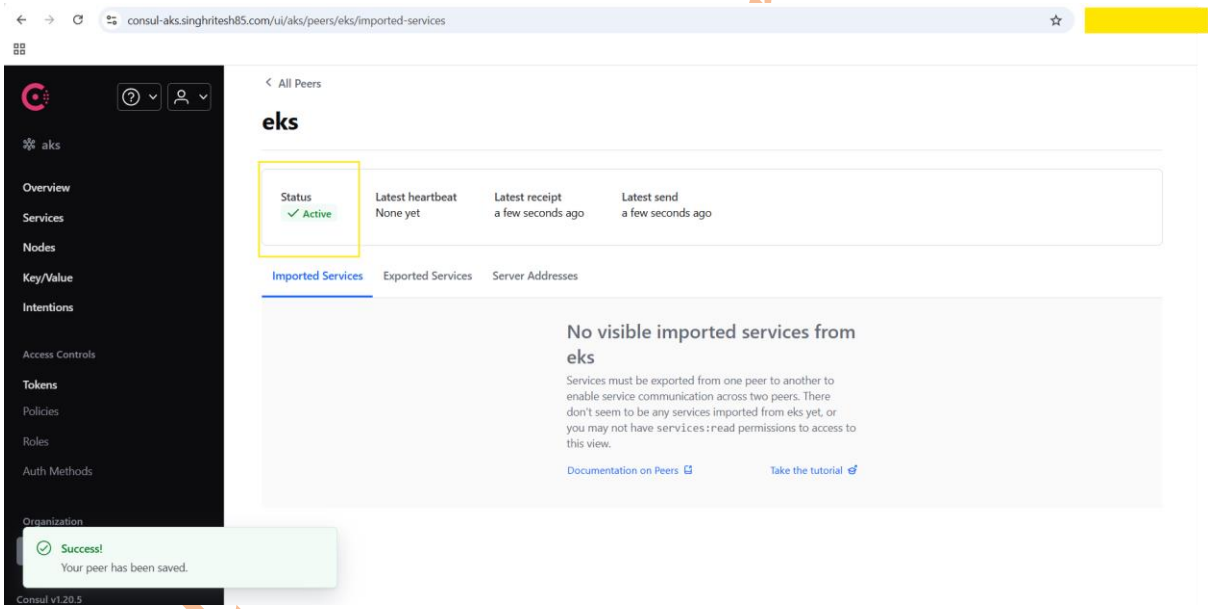
Now it is pending state as can be seen in the screenshot attached beloq.

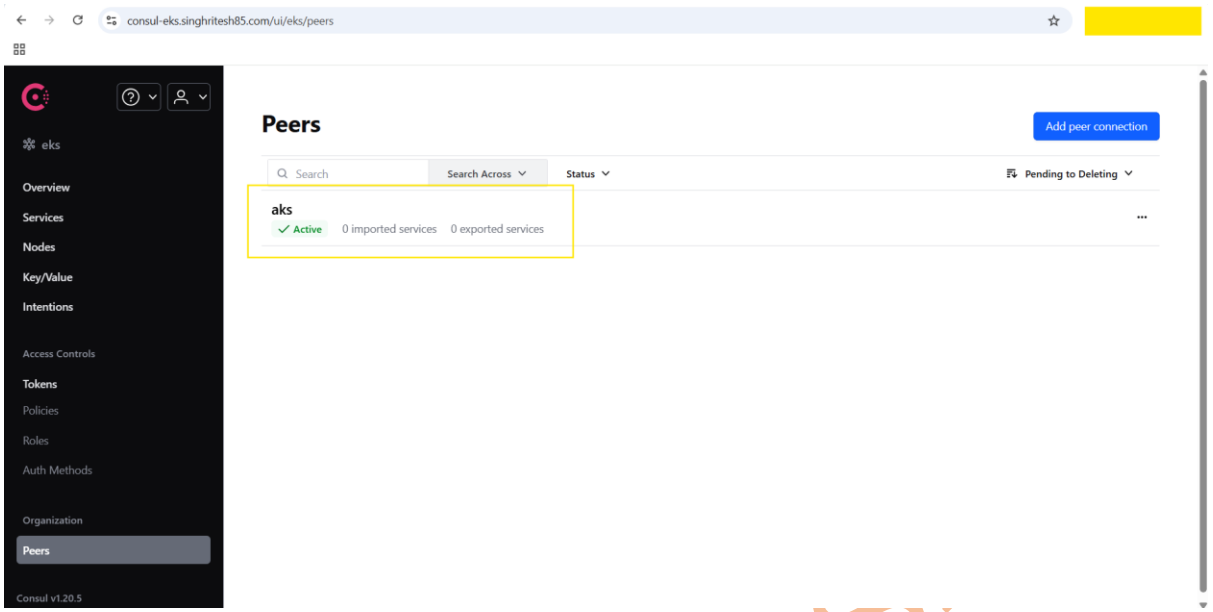

consul-eks.singhritesh85.com/ui/eks/peers
eks
Overview
Services
Nodes
Key/Value
Intentions
Access Controls
Tokens
Policies
Roles
Auth Methods
Organization
Peers
Consul v1.20.5
Peers
Add peer connection
Search
Search Across
Status
Pending to Deleting
aks
Pending
0 imported services
0 exported services


consul-aks.singhritesh85.com/ui/aks/peers
aks
Overview
Services
Nodes
Key/Value
Intentions
Access Controls
Tokens
Policies
Roles
Auth Methods
Organization
Peers
Consul v1.20.5
Peers
Add peer connection
Welcome to Peers
Cluster peering is the recommended way to connect services across or within Consul datacenters. Peering is a one-to-one relationship in which each peer is either a open-source Consul datacenter or a Consul enterprise admin partition. There don't seem to be any peers for this datacenter.
Documentation on Peers
Take the tutorial



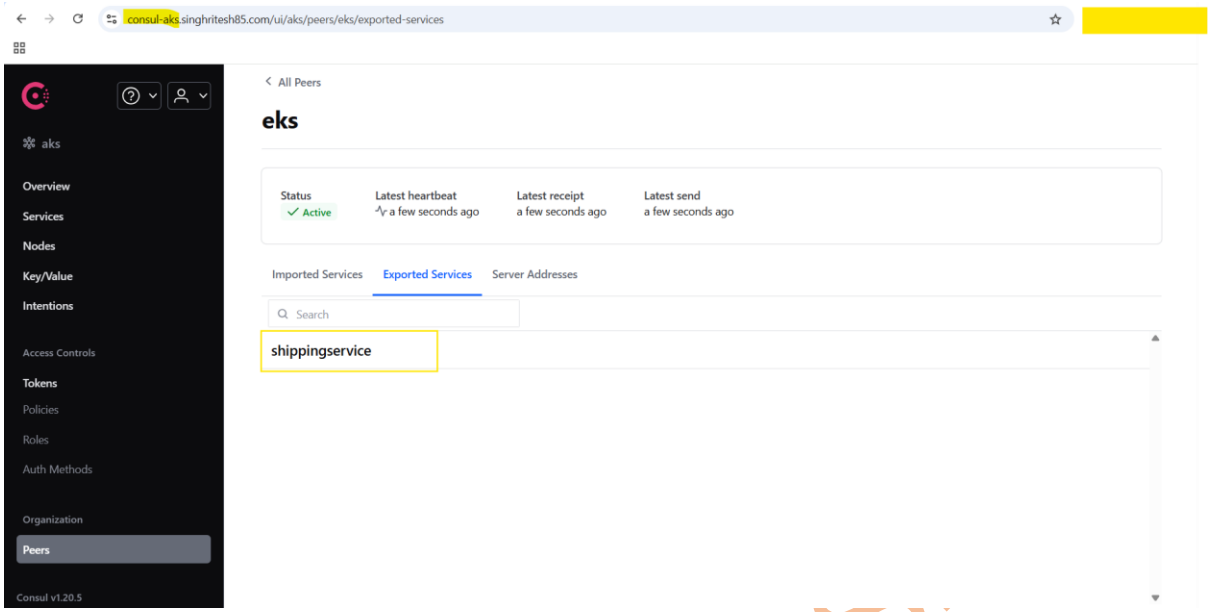
Finally, peering had been established as shown in the screenshot attached below.

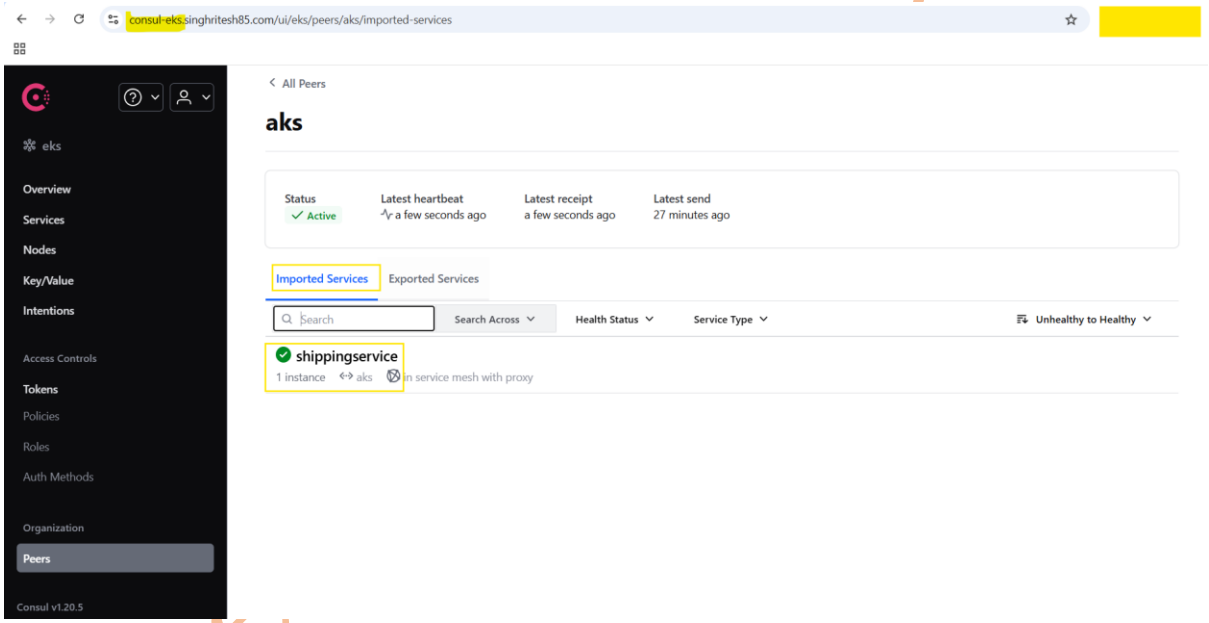


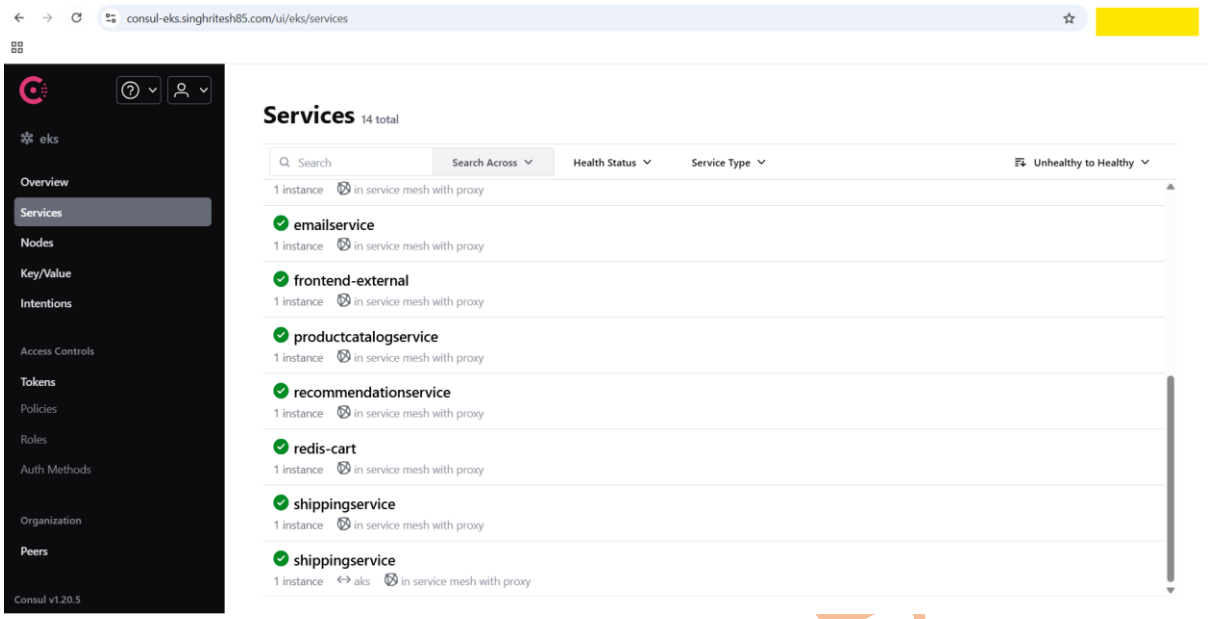



```
[root@                    kubernetes]# kubectl apply -f exported-service.yaml --context=aks-cluster
exportedservices.consul.hashicorp.com/default created
[root@                    kubernetes]# kubectl apply -f service-resolver.yaml --context=eks-demo-cluster-dev
serviceresolver.consul.hashicorp.com/shippingservice created
[root@                    kubernetes]# cat exported-service.yaml
apiVersion: consul.hashicorp.com/v1alpha1
kind: ExportedServices
metadata:
  name: default
spec:
  services:
    - name: "shippingservice"
      consumers:
      - peer: eks

[root@                    kubernetes]# cat service-resolver.yaml
apiVersion: consul.hashicorp.com/v1alpha1
kind: ServiceResolver
metadata:
  name: shippingservice
spec:
  connectTimeout: 15s
  failover:
    '*':
      targets:
        - peer: 'aks'
```

Then I checked the ui of consul for AKS and EKS cluster and found that exported and imported service as **shippingservice** respectively.


consul-aks.singhritesh85.com/ui/aks/peers/eks/exported-services
aks
Overview
Services
Nodes
Key/Value
Intentions
Access Controls
Tokens
Policies
Roles
Auth Methods
Organization
Peers
Consul v1.20.5
All Peers
eks
Status
Active
Latest heartbeat
a few seconds ago
Latest receipt
a few seconds ago
Latest send
a few seconds ago
Imported Services
Exported Services
Server Addresses
Search
shippingservice

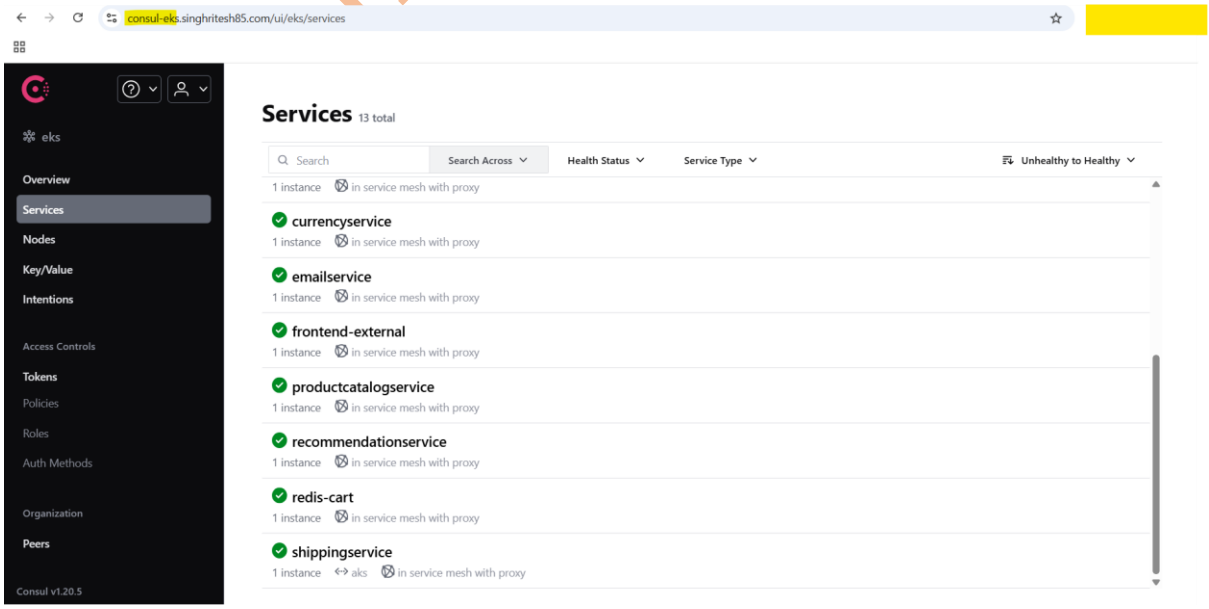

consul-eks.singhritesh85.com/ui/eks/peers/aks/imported-services
eks
Overview
Services
Nodes
Key/Value
Intentions
Access Controls
Tokens
Policies
Roles
Auth Methods
Organization
Peers
Consul v1.20.5
All Peers
aks
Status
Active
Latest heartbeat
a few seconds ago
Latest receipt
a few seconds ago
Latest send
27 minutes ago
Imported Services
Exported Services
Search
Search Across
Health Status
Service Type
Unhealthy to Healthy
shippingservice
1 instance
aks
in service mesh with proxy

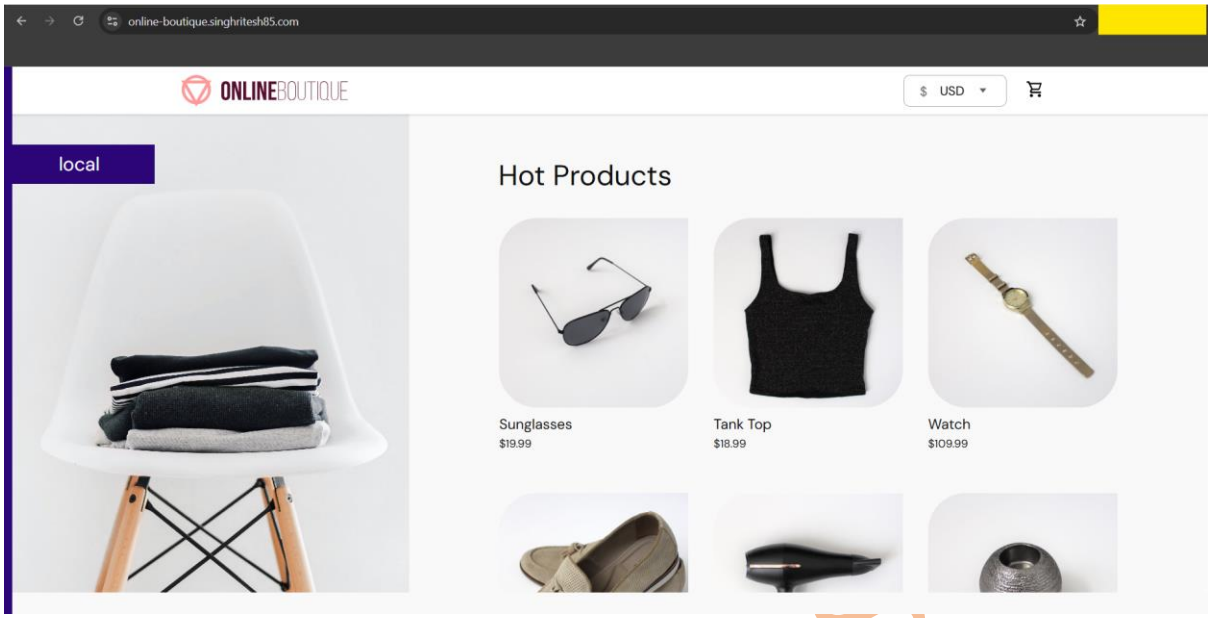


Now, I deleted the shippingservice deployment in EKS Cluster and checked whether it working properly or not through the shipping service in AKS Cluster as shown in the screenshot attached below.

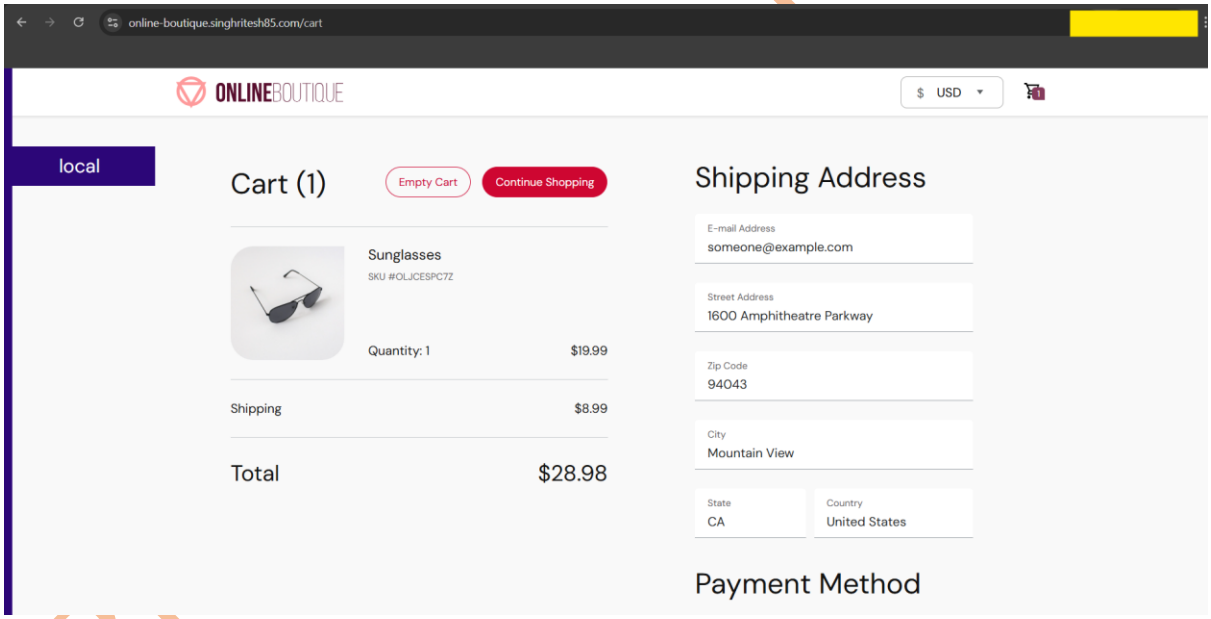
```
[root@                kubernetes]# kubectl delete deploy shippingservice -n microservice --context=eks-demo-cluster-dev
deployment.apps "shippingservice" deleted
```

```
[root@                kubernetes]# kubectl get pods -n microservice --context=eks-demo-cluster-dev --watch
NAME                                 READY   STATUS             RESTARTS       AGE
adservice-857bf87485-lc6jr           2/2     Running            0              88m
cartservice-644f9d65cd-ksw7p         2/2     Running            0              88m
checkoutservice-5696787454-hxx28     2/2     Running            0              88m
currencyservice-6d99654d6f-n4dvk     2/2     Running            0              88m
emailservice-74698d99d9-w42bx        2/2     Running            0              88m
frontend-5476787c94-fp669            2/2     Running            0              88m
paymentservice-669944c9f-9mjht       1/2     CrashLoopBackOff   22 (48s ago)   88m
productcatalogservice-98fcdf4b5-krzft   2/2     Running         0              88m
recommendationservice-77d79ff5d5-fksp8  2/2     Running         0              88m
redis-cart-689bb95f4-ccscr           2/2     Running            0              88m
```



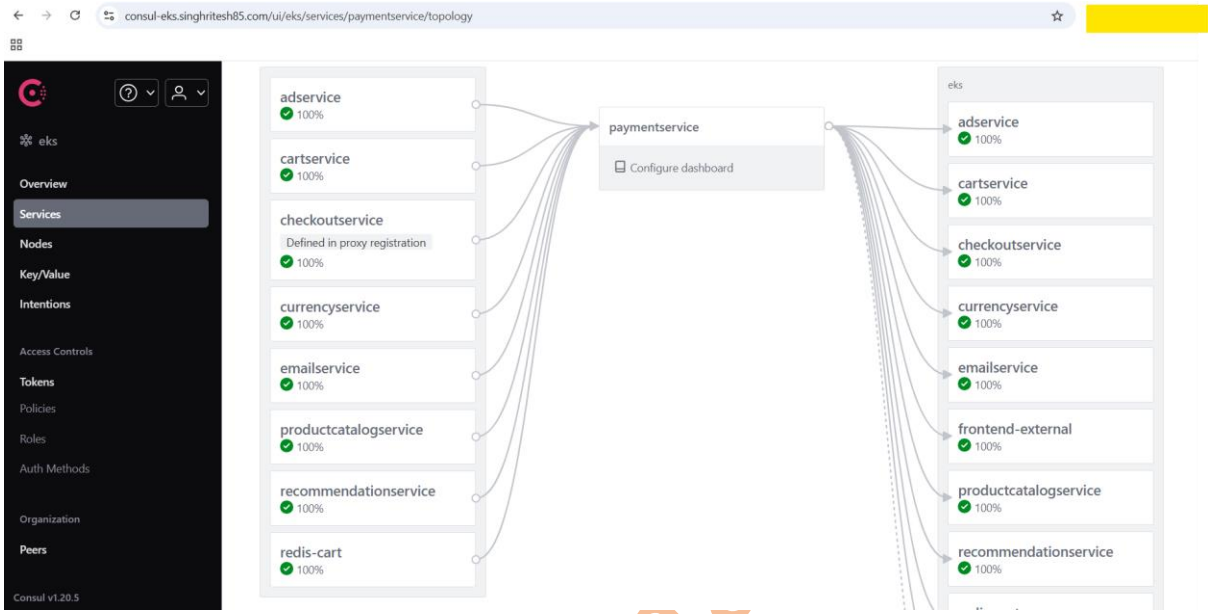


Finally, I found shippingservice failover is working properly as shown in the screenshot attached below.

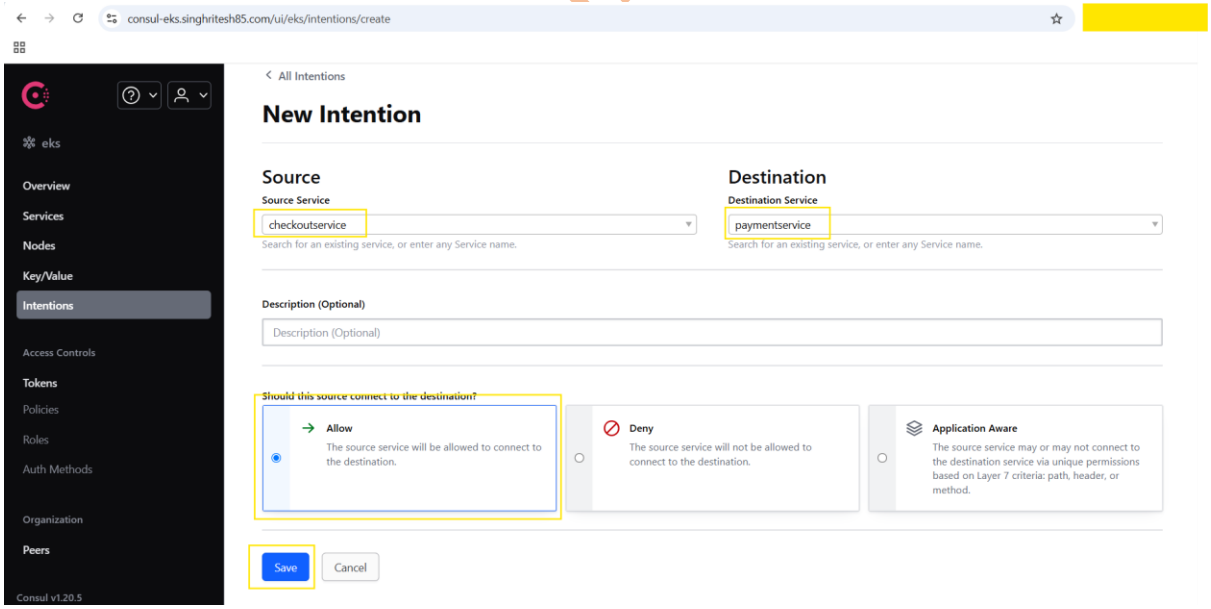


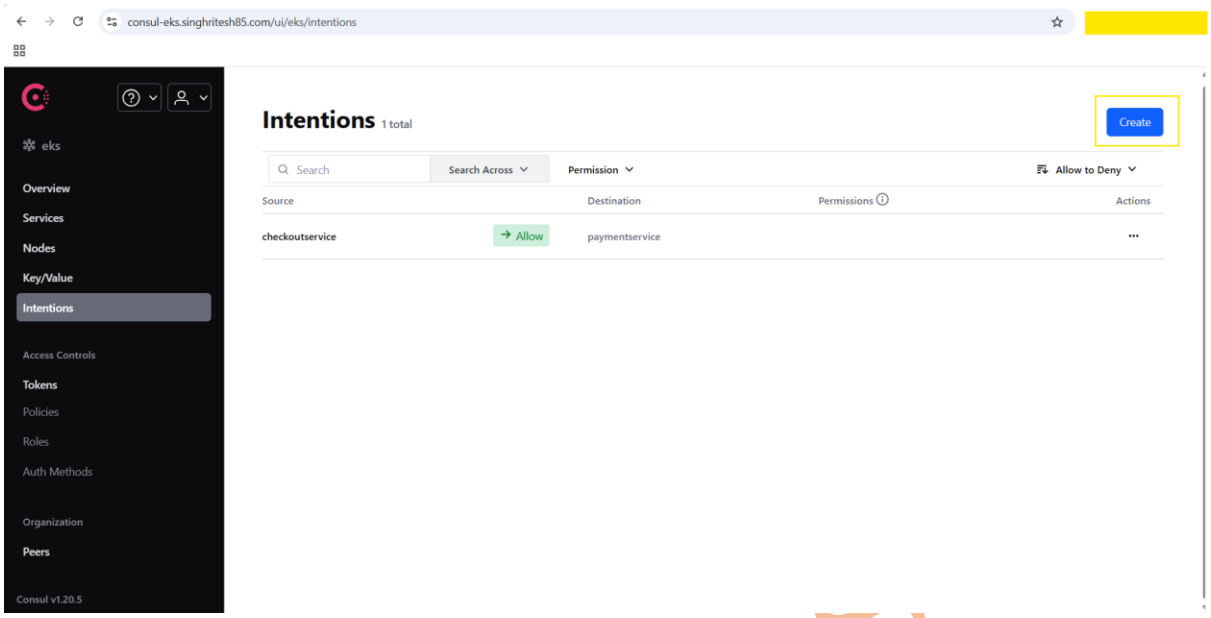
Restrict using Consul UI that which service can communicate to which service

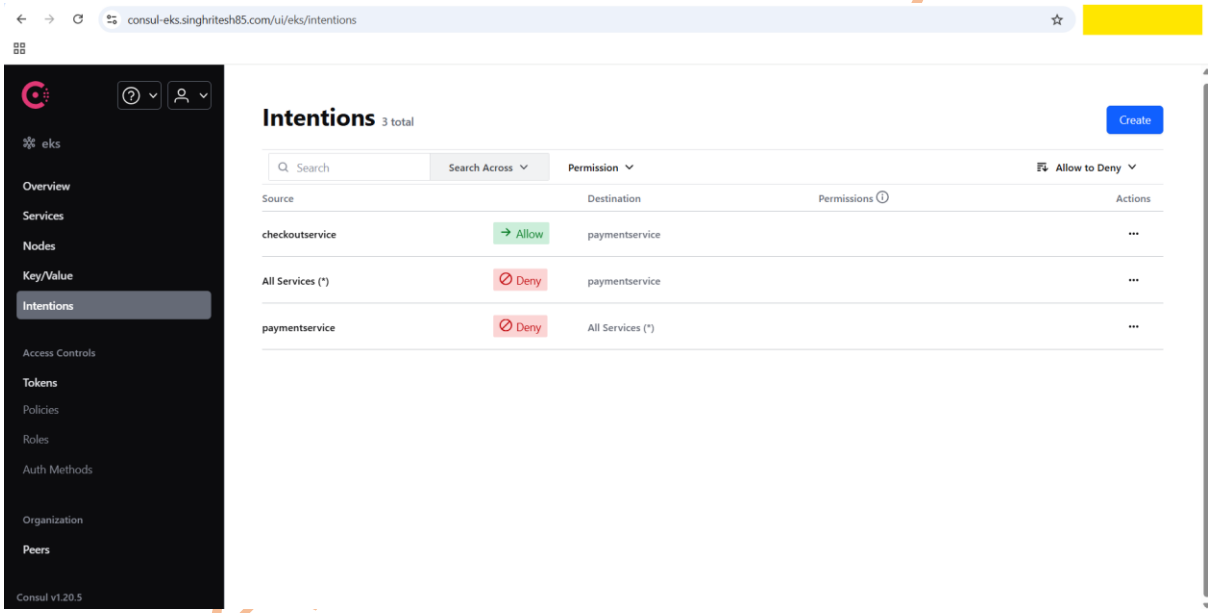
Through the consul ui I am restricting that which service communicates with which service for this I am taking an example of paymentservice. At present payment service can communicates with all the services as shown in the screenshot attached below.



Now go to **Intentions** present in the left side column and create the restriction Rule as shown in the screenshot attached below.







It means now the traffic to paymentservice can only go from checkoutservice and no other service can communicate with paymentservice. Traffic from payment service cannot go outside which can also be verified with the screenshot attached below.