**Objective: Enhance microservice connectivity within EKS.**

**Tasks:**

**1.** Deploy microservices in EKS.

**2.** Integrate AWS App Mesh.

**3.** Validate service discovery and load balancing.

**Documentation:**

**-** Service Mesh basics.

**-** AWS App Mesh essentials.

**-** Microservice communication patterns.

**Prerequisites:**

1- An AWS account with administrative access.

2- AWS CLI installed and configured.

3- kubectl and eksctl command line tools installed.

4- Familiarity with Amazon EKS and AWS App Mesh.

5- Basic knowledge of Kubernetes and Microservices.

**Tasks:**

**Task 1: Deploy microservices in EKS:**

**a. Set up an EKS cluster:**

If you haven't set up an EKS cluster already, you can use eksctl to create one:

| **eksctl create cluster \  --name=cluster-name \  --version=1.27 \  --region=us-east-2 \  --spot \  --node-type=t2.medium \  --nodes=2 \  --nodegroup-name=node-grp-muzammil \  --managed** |
| --- |

**b. Deploy your microservices:**

For the sake of this lab, let's assume you have a simple microservice defined in a YAML file:

| apiVersion: apps/v1 kind: Deployment metadata:  name: my-microservice spec:  replicas: 3   # Add or modify this selector to match your template's labels  selector:  matchLabels:  app: my-microservice   template:  metadata:  labels:  app: my-microservice  spec:  containers:  - name: my-microservice-container  image: nginx  ports:  - containerPort: 8080   # Assuming you're also adding the Envoy sidecar as discussed previously  - name: envoy  image: public.ecr.aws/appmesh/aws-appmesh-envoy:v1.18.3.0-prod  env:  - name: "APPMESH\_VIRTUAL\_NODE\_NAME"  value: "mesh/my-mesh/virtualNode/my-service-node"  securityContext:  runAsUser: 1337 |
| --- |

| kubectl apply -f microservice-deployment.yaml |
| --- |

**View the service and deployment.**

| kubectl -n **my**-apps **get** pods |
| --- |

**Task 2:**

**Integrate AWS App Mesh:**

**a.** Install the App Mesh CLI:

Helm Install: [Link Here](https://helm.sh/docs/intro/install/)

| sudo apt-get **update** curl https://baltocdn.**com**/helm/signing.asc | gpg --dearmor | sudo tee /usr/share/keyrings/helm.gpg > /dev/null  sudo apt-get install apt-transport-https --yes  **echo** "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/helm.gpg] https://baltocdn.com/helm/stable/debian/ all main" | sudo tee /etc/apt/sources.**list**.d/helm-stable-debian.**list**  sudo apt-get **update**  sudo apt-get install helm |
| --- |

**Verify Check Install Helm**

| helm list |
| --- |

#### Install App Mesh Controller: [Link Here](https://archive.eksworkshop.com/advanced/330_servicemesh_using_appmesh/appmesh_installation/install_appmesh/) or [second link](https://docs.aws.amazon.com/app-mesh/latest/userguide/getting-started-kubernetes.html#install-controller)

| # Create the namespace kubectl create ns appmesh-system  # Install the App Mesh CRDs kubectl apply -k "github.com/aws/eks-charts/stable/appmesh-controller//crds?ref=master"  # Create your OIDC identity provider for the cluster eksctl utils associate-iam-oidc-provider \  --cluster eksworkshop-eksctl \  --approve  # Download the IAM policy for AWS App Mesh Kubernetes Controller curl -o controller-iam-policy.json https://raw.githubusercontent.com/aws/aws-app-mesh-controller-for-k8s/master/config/iam/controller-iam-policy.json  # Create an IAM policy called AWSAppMeshK8sControllerIAMPolicy aws iam create-policy \  --policy-name AWSAppMeshK8sControllerIAMPolicy \  --policy-document file://controller-iam-policy.json  # Create an IAM role for the appmesh-controller service account eksctl create iamserviceaccount --cluster eksworkshop-eksctl \  --namespace appmesh-system \  --name appmesh-controller \  --attach-policy-arn arn:aws:iam::$ACCOUNT\_ID:policy/AWSAppMeshK8sControllerIAMPolicy \  --override-existing-serviceaccounts \  --approve |
| --- |

| export CLUSTER\_NAME= eks-labs export AWS\_REGION=us-east-2 |
| --- |

| eksctl utils associate-iam-oidc-provider \  --region=$AWS\_REGION \  --cluster $CLUSTER\_NAME \  --approve |
| --- |

| helm upgrade -i appmesh-controller eks/appmesh-controller \  --namespace appmesh-system \  --set region=$AWS\_REGION \  --set serviceAccount.create=false \  --set serviceAccount.name=appmesh-controller |
| --- |

**Update the Helm Repo**

| helm repo update |
| --- |

Install App Mesh Controller into the appmesh-system namespace

| helm upgrade -i appmesh-controller eks/appmesh-controller \  --namespace appmesh-system \  --set region=$AWS\_REGION \  --set serviceAccount.create=false \  --set serviceAccount.name=appmesh-controller \  --set tracing.enabled=true \  --set tracing.provider=x-ray |
| --- |

Confirm that the controller version is v1.0.0 or later.

| kubectl get deployment appmesh-controller \  -n appmesh-system \  -o json | jq -r ".spec.template.spec.containers[].image" | cut -f2 -d ':' |
| --- |

**Confirm all the App Mesh CRDs are created in the Cluster**

| **kubectl get crds | grep appmesh** |
| --- |

**gatewayroutes.appmesh.k8s.aws 2020-11-02T16:02:14Z  
meshes.appmesh.k8s.aws 2020-11-02T16:02:15Z  
virtualgateways.appmesh.k8s.aws 2020-11-02T16:02:15Z  
virtualnodes.appmesh.k8s.aws 2020-11-02T16:02:15Z  
virtualrouters.appmesh.k8s.aws 2020-11-02T16:02:15Z  
virtualservices.appmesh.k8s.aws 2020-11-02T16:02:15Z**

**Create a Mesh:**

| **aws appmesh create-mesh --mesh-name my-mesh** |
| --- |

**Create Virtual Nodes:**

**For each of your microservices, you would create a virtual node:**

| **aws appmesh create-virtual-node --mesh-name my-mesh --spec file://virtual-node-spec.json --virtual-node-name my-service-node** |
| --- |

**Create Virtual Services and Routers:**

This ties together the nodes and routes traffic:

| aws appmesh create-virtual-router --mesh-name my-mesh --spec file://virtual-router-spec.json --virtual-router-name my-router  aws appmesh create-route --mesh-name my-mesh --spec file://route-spec.json --route-name my-route --virtual-router-name my-router |
| --- |

Here, **virtual-router-spec.json** and **route-spec.json** are configuration files for the router and **route**.

**Get all the resources created in appmesh-system Namespace**

| **kubectl -n appmesh-system get all** |
| --- |

**virtual-node.yaml**

| **apiVersion: appmesh.k8s.aws/v1beta2 kind: VirtualNode metadata:  name: my-microservice-vn spec:  listeners:  - portMapping:  port: 80  protocol: http** |
| --- |

**virtual-service.yaml:**

| **apiVersion: appmesh.k8s.aws/v1beta2 kind: VirtualService metadata:  name: my-microservice-vs spec:  awsName: my-microservice.default.svc.cluster.local  provider:  virtualNode:  virtualNodeName: my-microservice-vn** |
| --- |

**To add X-Ray Tracing, this command will help**

| **helm upgrade -i appmesh-controller eks/appmesh-controller \ --namespace appmesh-system \ --set tracing.enabled=true \ --set tracing.provider=x-ray** |
| --- |

| **aws servicediscovery list-services** |
| --- |

| **aws appmesh list-meshes** |
| --- |

**Task 3:**

**Validate service discovery and load balancing**

**1. Service Discovery Testing:**

| **kubectl run -i --tty --rm debug --image=busybox -- sh # Inside pod nslookup my-microservice.default.svc.cluster.local** |
| --- |

2. Deploy a test microservice to send traffic to your original microservices.

3. Monitor the traffic:

Using AWS CloudWatch, you can monitor the traffic flow and see how App Mesh handles service discovery and load balancing.

**Documentation:**

1. Service Mesh basics:

A Service Mesh is an infrastructure layer for microservices applications. It facilitates communication between services by ensuring that it is fast, reliable, and secure. The mesh provides features like load balancing, service discovery, traffic management, and more.

**2. AWS App Mesh essentials:**

AWS App Mesh is a managed service mesh provided by AWS. It standardizes the way your services communicate, giving end-to-end visibility and ensuring high availability for your applications.

* Virtual Node: Represents a logical pointer to a particular task group, such as a Kubernetes deployment.
* Virtual Service: An abstraction of an actual microservice, used by the mesh to route traffic to the correct virtual nodes.
* Virtual Router: Processes traffic that is sent to the virtual service to determine where to send it next.
* Route: Specifies rules for routing traffic to service backends.

**3. Microservice communication patterns:**

With microservices, there's a need to handle service-to-service communication efficiently. Patterns can include:

* Synchronous HTTP/REST: A direct call from one service to another.
* Asynchronous Messaging: Using message brokers like RabbitMQ or Kafka.
* Service Discovery: Dynamically discover service instances.