**Objective:** Dive deep into service mesh principles using AWS App Mesh.

**Tasks:**

1. Set up App Mesh in EKS.

2. Configure virtual nodes and virtual routers.

3. Observe service-to-service communication enhancements.

**Documentation:**

- Service mesh principles.

- Benefits of AWS App Mesh.

- Configuring App Mesh in EKS.

**Prerequisites:** Before you begin, ensure that you have the following prerequisites in place:

* An AWS account.
* The AWS CLI installed and configured.
* kubectl installed and configured to interact with your EKS cluster.
* eksctl installed for creating and managing EKS clusters.
* Docker

**Tasks:**

**Task 1:** Set up App Mesh in EKS

Create an EKS Cluster: If you don’t have an EKS cluster, create one using eksctl. Here is an example command to create a cluster:

| eksctl create cluster \  --name=eks \  --version=1.28 \  --region=us-east-1 \  --spot \  --node-type=t3.small \  --nodes=1 \  --nodes-min=1 \  --nodes-max=2 \  --node-volume-size=25 \  --nodegroup-name=eks-group \  --managed |
| --- |

Manual Clone: As a workaround, you can manually clone the repository and then apply the CRDs:

Install the App Mesh CRDs (Custom Resource Definitions):

| git **clone** **https**://github.com/aws/eks-charts.git cd eks-charts/stable/appmesh-controller/crds kubectl apply -k . |
| --- |

| kubectl apply -k "https://github.com/aws/eks-charts/stable/appmesh-controller/crds?ref=master" |
| --- |

Create the appmesh-system namespace:

| kubectl **create** ns appmesh-**system** |
| --- |

Add the EKS charts repository to Helm:

| helm repo **add** eks https://aws.github.io/eks-charts |
| --- |

Install the App Mesh controller:

| helm install appmesh-controller eks/appmesh-controller --**namespace** appmesh-**system** |
| --- |

Verify that the controller is up and running:

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

You should see the appmesh-controller pod in the RUNNING state.

**Step 2:** Enable App Mesh on a Namespace

Create a new namespace for your applications:

| kubectl **create** ns my-app-mesh-**namespace** |
| --- |

Label the namespace to enable App Mesh:

| kubectl **label** namespace my-app-mesh-namespace appmesh.k8s.aws/sidecarInjectorWebhook=enabled |
| --- |

| **kubectl label namespace my-app-mesh-namespace mesh=my-mesh** |
| --- |

**Tasks:**

**Task 2: Configure Virtual Nodes and Virtual Routers**

Define app-mesh

| apiVersion: appmesh.k8s.aws/v1beta2 kind: Mesh metadata:  name: my-mesh spec:  namespaceSelector:  matchLabels:  mesh: my-mesh |
| --- |

**Define Virtual Nodes:** Create YAML definitions for your virtual nodes. Here is an example:

| apiVersion: appmesh.k8s.aws/v1beta2 kind: VirtualNode metadata:  namespace: my-app-mesh-namespace  name: my-app-virtual-node spec:  awsName: my-app-virtual-node  podSelector:  matchLabels:  app: my-app  listeners:  - portMapping:  port: 8080  protocol: http  serviceDiscovery:  dns:  hostname: my-app.my-app-mesh-namespace.svc.cluster.local |
| --- |

**Define Virtual Routers:** Similarly, create YAML definitions for your virtual routers and apply them.

| apiVersion: appmesh.k8s.aws/v1beta2 kind: VirtualRouter metadata:  namespace: my-app-mesh-namespace  name: my-app-virtual-router spec:  awsName: my-app-virtual-router  listeners:  - portMapping:  port: 8080  protocol: http |
| --- |

**Deploy app: Nginx**

| apiVersion: v1 kind: Service metadata:  name: my-service-a  namespace: my-app-mesh-namespace  labels:  app: my-app-1 spec:  selector:  app: my-app-1  ports:  - protocol: TCP  port: 80  targetPort: 80 --- apiVersion: apps/v1 kind: Deployment metadata:  name: my-service-a  namespace: my-app-mesh-namespace  labels:  app: my-app-1 spec:  replicas: 3  selector:  matchLabels:  app: my-app-1  template:  metadata:  labels:  app: my-app-1  spec:  serviceAccountName: my-service-a  containers:  - name: nginx  image: nginx  ports:  - containerPort: 80 |
| --- |

**Tasks:**

**Task 3:** Observe Service-to-Service Communication Enhancements

1. **Deploy Your Applications:** Deploy the applications that will be part of the service mesh.
2. **Examine Metrics and Logs:** Use AWS CloudWatch or other monitoring tools to examine the metrics and logs from your service mesh. Look for improvements in latency, error rates, and other performance indicators.
3. **Experiment with Routing and Resilience Features:** Use App Mesh to experiment with various routing and resilience features like canary deployments, circuit breakers, and retry policies.

**Documentation**

**Service Mesh Principles**

A service mesh is a dedicated infrastructure layer that facilitates service-to-service communication in a microservices architecture. It provides features like load balancing, service discovery, health checks, encryption, and more. App Mesh applies these features at the application level, rather than the network level, providing more granular control.

* **Enhanced Visibility:** App Mesh provides end-to-end visibility into the performance of your applications.
* **Traffic Control:** It allows you to easily implement advanced traffic routing for A/B testing, canary releases, and other deployment strategies.
* **Resilience:** App Mesh improves the resilience of your applications with features like circuit breakers and retries.
* **Security:** With App Mesh, you can enforce consistent policies across your services, ensuring secure communication.