Control Engine Setup and Testing

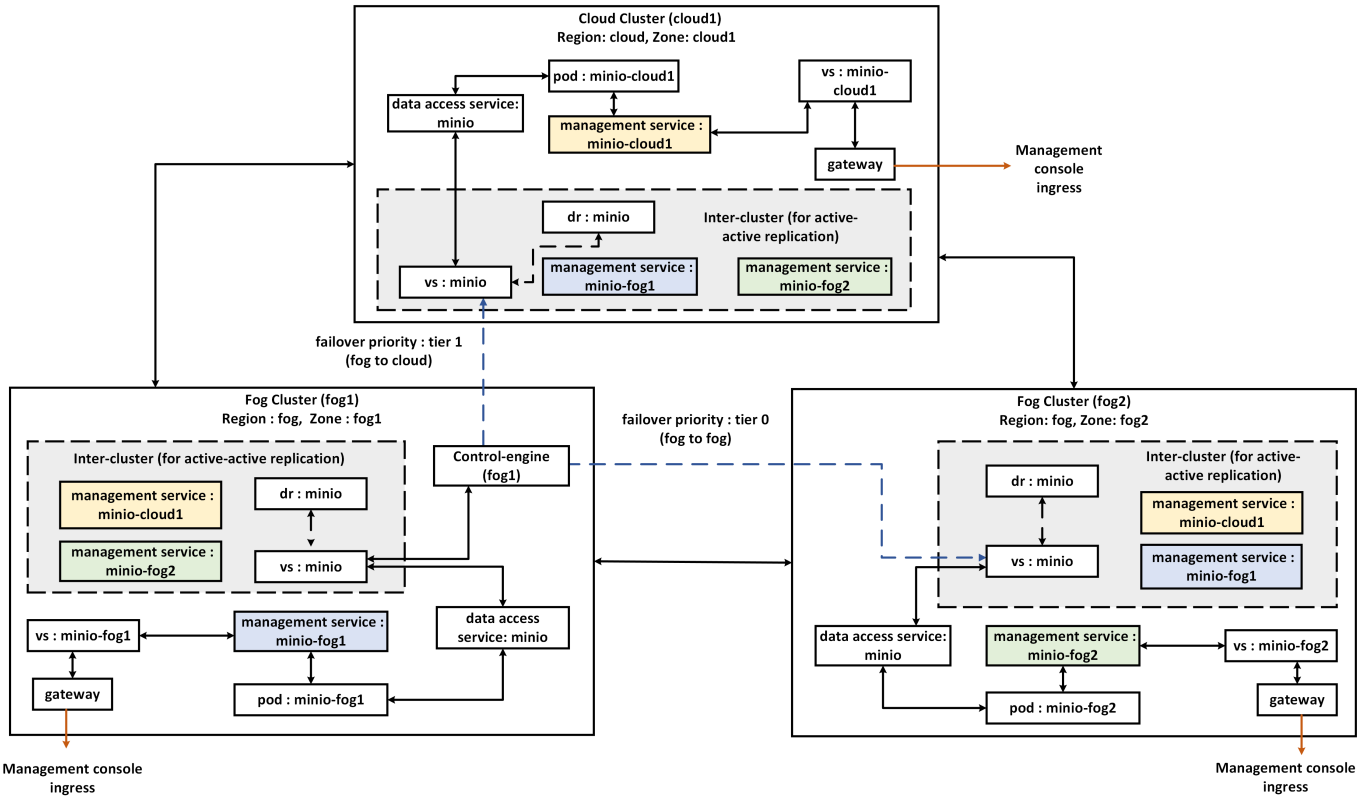
Pre-requisites : Clusters are setup with Kubernetes and Istio. Multi-cluster communication is working properly (in multi-cluster + mult-istio control plane mode).

1. Setup namespace with istio proxy enabled

```
kubectl create ns control-engine --context <cluster_name>
kubectl label --context <cluster_name> namespace control-engine \
    istio-injection=enabled
```

2. Setup data stores
   a. Redis Meta Data Store
   b. MinIO Yaml store - Yaml files for deployment (set up MinIO to store yaml files)
3. Deploying Control Engine
   a. Update config map

MinIO Yaml store setup



1. Distributed across clusters and replicated to maintain consistency among MinIO servers.
2. Fault-tolerance : in case of failure in own cluster, can access the MinIO servers from other clusters.
3. Architecture :
   a. Consists of two traffic routing layers.
      i. Management layer: For replication traffic (through api port 9000 of Minio Server) and to console access through ingress gateway (through port 9090 of Minio Server)
      ii. Data access layer: Control Engine access the Minio server through port 9000 to get the yaml files required for application deployment.
4. Example setup

```
kubectl apply -f minio_edge1.yaml --context kind-edge1 -n control-
engine
kubectl apply -f minioManagementService_edge1.yaml --context kind-edge1
-n control-engine

kubectl apply -f minioVS_edge1.yaml --context kind-edge1 -n control-
engine
kubectl apply -f minioGW.yaml --context kind-edge1 -n control-engine

kubectl apply -f minioService.yaml --context kind-edge1 -n control-
engine

kubectl apply -f minio_dr.yaml -n control-engine --context kind-edge1
```

```
kubectl apply -f minio_edge2.yaml --context kind-edge2 -n control-
engine
kubectl apply -f minioManagementService_edge2.yaml --context kind-edge2
-n control-engine

kubectl apply -f minioVS_edge2.yaml --context kind-edge2 -n control-
engine
kubectl apply -f minioGW.yaml --context kind-edge2 -n control-engine

kubectl apply -f minioService.yaml --context kind-edge2 -n control-
engine

kubectl apply -f minio_dr.yaml -n control-engine --context kind-edge2
```

```
kubectl apply -f minioManagementService_edge1.yaml --context kind-edge2
-n control-engine
kubectl apply -f MinIOManagementService_edge2.yaml --context kind-edge1
-n control-engine
```

- Login to Minio console through ingress gateway and create yaml data bucket (bucket name : `microfog-app-metadata` - with versioning)
- Configure replication (two-way replications : https://min.io/docs/minio/linux/administration/bucket-replication/enable-server-side-two-way-bucket-replication.html)

- Target Url : <management_service_name>.control-engine.svc.cluster.local:9000
  eg : minio-edge1.control-engine.svc.cluster.local:9000

- Disable tls
- Use username & password or access keys if set

- Upload data to one of the servers (and will be replicated across configured servers in other clusters)
- Folder structure

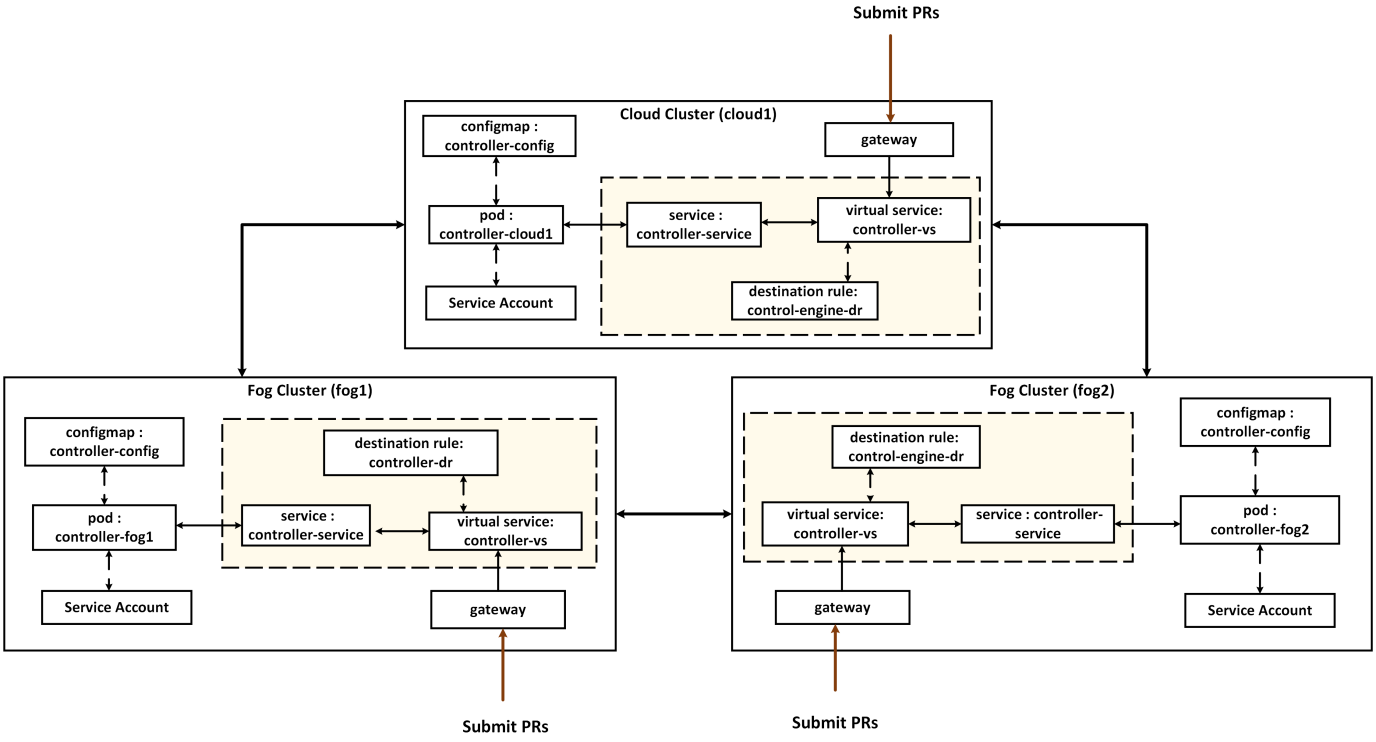
```
microfog-app-metadata (bucket)
|-- bookinfo (folder name should match with the application Id used
in redis meta data store)
    |-- xxxx.yaml (kubernetes and istio configuration yaml files)
```

Deploying Redis Meta Data Store

```
#For Primary
kubectl apply -f redis-primary.yaml -n control-engine
#For Replica
kubectl apply -f redis-replica.yaml -n control-engine

#For all
kubectl apply -f redis-service.yaml -n control-engine
kubectl apply -f redis-service-primary.yaml -n control-engine
kubectl apply -f redis-vs.yaml -n control-engine
kubectl apply -f redis-dr.yaml -n control-engine
```

Deploying Control Engine



Updating Config Map for Control engine

Quarkus apps access config map for configurations instead of application.properties by setting following configurations

```
quarkus.kubernetes-config.enabled=true
quarkus.kubernetes-client.namespace=control-engine
quarkus.kubernetes-config.config-maps=controller-config
```

Important configurations

Operation mode (Centralised or Distributed)

1. For distributed both parameters set to “true”
2. For centralised `controlengine.operationmode.distributed = false` and the second parameter set depending on the role of the CE

```
controlengine.operationmode.distributed =true
controlengine.operationmode.primary = true
```

Placement mode (Periodic or Event-Driven)

```
controlengine.placementmode.periodic = true
controlengine.period = 1
```

Placement algorithms

```
controlengine.placementalgotype.internal= true
controlengine.placementalgo = DISTRIBUTED_PLACEMENT
controlengine.placementalgourl = <set for externam algorithms)
controlengine.placementalgo.version  = V2
```

Yaml store (Minio) configurations

```
minio.minioEndPoint = http://minio.control-engine.svc.cluster.local:9000
minio.accesskey = <user_name>
minio.secretkey = <passwor>
minio.bucketname = microfog-app-metadata
```

Redis Meta Data Store

```
redis.url = redis://redis.control-engine.svc.cluster.local:6379
controlengine.populatemetadata = false  // if primary redis instance,
then set to true
```

Placement request forwarding related configs

```
#distributed controller communication
cloud.adjacent.cluster[0] = cloud1 //connected cloud clusters
fog.adjacent.cluster[0] = cluster2 //connected edge clusters
fog.adjacent.cluster[1] = cluster3

controlengine.forwardpolicy = random_selec // how to forward prs
controlengine.forwardpolicy = to_fog_cloud
controlengine.forwardUrl = http://control-engine.control-engine.svc.
cluster.local:8080
```

Load Balancing Policy

```
controlengine.loadblancing.enabled = true
controlengine.loadblancing.policy = weighted_round_robin
```

Steps to deploy :

```
kubectl apply -f controller-service-account.yaml -n control-engine --
context kind-edge2
kubectl apply -f controller-service-account.yaml -n control-engine --
context kind-edge1
kubectl apply -f controller-service-account.yaml -n control-engine --
context kind-cloud1

kubectl apply -f permission.yaml -n control-engine

kubectl apply -f controller-configmap-edge2.yaml -n control-engine --
context kind-edge2
kubectl apply -f controller-configmap-edge1.yaml -n control-engine --
context kind-edge1
kubectl apply -f controller-configmap-cloud1.yaml -n control-engine --
context kind-cloud1

kubectl apply -f controller-edge2.yaml -n control-engine --context kind-
edge2
kubectl apply -f controller-edge1.yaml -n control-engine --context kind-
edge1
kubectl apply -f controller-cloud1.yaml -n control-engine --context
kind-cloud1

kubectl apply -f controller-service.yaml -n control-engine

kubectl apply -f controller-vs.yaml -n control-engine

kubectl apply -f controller-gateway.yaml -n control-engine

kubectl apply -f controller-dr.yaml -n control-engine
```