Prototype: Example Setup using multiple vms

Kind Clusters

Cluster 1 - kind-fog1

```
NAME                   STATUS   ROLES           AGE   VERSION   INTERNAL-IP   EXTERNAL-IP   OS-IMAGE       KERNEL-VERSION       CONTAINER-RUNTIME
fog1-control-plane     Ready    control-plane   11h   v1.24.0   172.18.0.4    <none>        Ubuntu 21.10   4.15.0-194-generic   containerd://1.6.4
fog1-worker            Ready    <none>          11h   v1.24.0   172.18.0.3    <none>        Ubuntu 21.10   4.15.0-194-generic   containerd://1.6.4
fog1-worker2           Ready    <none>          11h   v1.24.0   172.18.0.2    <none>        Ubuntu 21.10   4.15.0-194-generic   containerd://1.6.4
```

Kind networking concepts : (https://gist.github.com/developer-guy/173347e71f92a61abbc017deb518b6cb)

Kind uses docker networking by creating seperate bridge network “kind”. Kind also provide advance capabilities to assign desired subnet for each cluster.

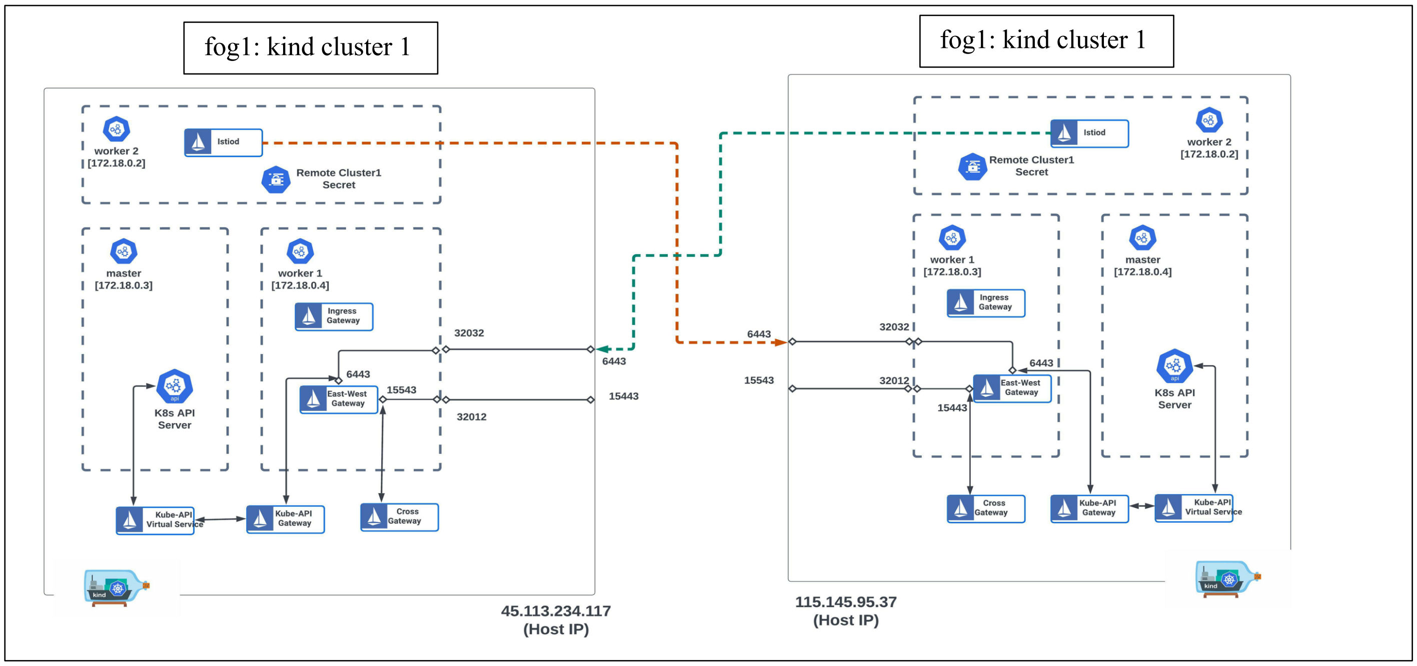
```
docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
e1ae88a27694        bridge              bridge              local
1853ebf0d3f5        host                host                local
c3ed1d8db4d3        kind                bridge              local
9647355c88b1        none                null                local

docker network inspect kind
    {
        "Name": "kind",
        "Id": "c3ed1d8db4d3eb00402ce0dc6a259958d301be57b35b0d66c5f6b7a2b9725d00",
        "Created": "2022-10-23T00:21:51.395143914Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": true,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.18.0.0/16",
                    "Gateway": "172.18.0.1"
                },
                {
                    "Subnet": "fc00:f853:ccd:e793::/64"
                }
            ]
        },
```

Creating multiple clusters help mimic multi-network multi-region scenarios, which is helpful in initial evaluation of the placement algorithms with federated edge-cloud scenarios.

Thus we present a sample scenario consisting of 2 edge clusters. Creating both clusters in a single node is trivial. Hence, we explain a scenario where we create two clusters in two different devices and use Istio and Kubernetes to establish seamless connectivity between the two. This provides capability to have more resources to create clusters and also help simulate inter cluster communication with realistic network delays.

Physical Architecture



Remote Setup - Using kind clusters

1. Created melbourne research cloud instances with public ips (16VCPU, 64GB ram and 30GB storage). Each instance represent a cluster (created using Kind to mimic resource heterogeneity)
2. Make sure openssl is installed properly

3. Security groups : ssh, and allow traffic for ports mapped to istio-ingress gateway (80 - for external traffic from users) and istio-eastwest gateways (15443 - for traffic between the two clusters, and 6443 for inter-cluster authentication)
4. Install Docker (https://docs.docker.com/engine/install/ubuntu/), kubectl (https://kubernetes.io/docs/tasks/tools/install-kubectl-linux/) and kind (https://kind.sigs.k8s.io/docs/user/quick-start/#installation)
5. Setup Istio (https://istio.io/latest/docs/setup/getting-started/)
6. Create kind cluster
    1. To get traffic into the cluster and also to maintain connection with the other clusters Extra Port Mapping is used (https://kind.sigs.k8s.io/docs/user/configuration/#extra-port-mappings)
    2. Three ports are mapped (container port to host port) - One for Istio Ingress Gateway and Two for Istio EastWest Gateway
7. Label nodes with their Region and Zone (used for failover purposes by data stores in later sections)
8. Update Kubernetes API Certificate to add the VM IP address. This enables the other clusters to access API server through vmip:6443
9. Setup Kubernetes Metric Server
10. Install MetalLB load balancer with public ip address of the instance. (ip pool should be updated to include the public IP of the VM)
11. Install Istio (Loadbalancer ip is updated to the public ip of the VM)
12. Deploy cross-gateway, and authentication gateways (gateway and virtual service)
13. Update mesh network configmap to include all connected networks.
14. Generate secrets

Updating Kubernetes API Server Certificate (https://blog.scottlowe.org/2019/07/30/adding-a-name-to-kubernetes-api-server-certificate/ , https://blog.devgenius.io/updating-kind-kubernetes-api-certificate-after-reboot-1521b43f7574)

```
kubectl -n kube-system get configmap kubeadm-config -o jsonpath='{.data.ClusterConfiguration}' >
kubeadm.yaml

add SANs ip (ip or dns - match this with the network configs in other clusters done in spet 13 and
secrets in step 14)

mkdir certTemp
sudo docker cp fog2-control-plane:/etc/kubernetes/pki/apiserver.crt certTemp/
sudo docker cp fog1-control-plane:/etc/kubernetes/pki/apiserver.crt certTemp/
sudo docker cp cloud1-control-plane:/etc/kubernetes/pki/apiserver.crt certTemp/

sudo docker cp fog2-control-plane:/etc/kubernetes/pki/apiserver.key certTemp/
sudo docker cp fog1-control-plane:/etc/kubernetes/pki/apiserver.key certTemp/
sudo docker cp cloud1-control-plane:/etc/kubernetes/pki/apiserver.key certTemp/

sudo docker exec fog2-control-plane rm -rf /etc/kubernetes/pki/apiserver.crt
sudo docker exec fog1-control-plane rm -rf /etc/kubernetes/pki/apiserver.crt
sudo docker exec cloud1-control-plane rm -rf /etc/kubernetes/pki/apiserver.crt

sudo docker exec fog2-control-plane rm -rf /etc/kubernetes/pki/apiserver.key
sudo docker exec fog1-control-plane rm -rf /etc/kubernetes/pki/apiserver.key
sudo docker exec cloud1-control-plane rm -rf /etc/kubernetes/pki/apiserver.key

sudo docker cp kubeadm.yaml fog2-control-plane:/etc/kubernetes/pki/
sudo docker cp kubeadm.yaml fog1-control-plane:/etc/kubernetes/pki/
sudo docker cp kubeadm.yaml cloud1-control-plane:/etc/kubernetes/pki/
sudo docker exec fog2-control-plane kubeadm init phase certs apiserver --config
/etc/kubernetes/pki/kubeadm.yaml
sudo docker exec fog1-control-plane kubeadm init phase certs apiserver --config
/etc/kubernetes/pki/kubeadm.yaml
sudo docker exec cloud1-control-plane kubeadm init phase certs apiserver --config
/etc/kubernetes/pki/kubeadm.yaml

sudo docker exec fog2-control-plane chown root:root /etc/kubernetes/pki/apiserver.crt
/etc/kubernetes/pki/apiserver.key
sudo docker exec fog1-control-plane chown root:root /etc/kubernetes/pki/apiserver.crt
/etc/kubernetes/pki/apiserver.key
sudo docker exec cloud1-control-plane chown root:root /etc/kubernetes/pki/apiserver.crt
/etc/kubernetes/pki/apiserver.key
```

Openssl

```
sudo apt-get update
sudo apt install build-essential checkinstall zlib1g-dev -y
cd /usr/local/src/
sudo wget https://www.openssl.org/source/openssl-3.0.5.tar.gz
sudo tar -xf openssl-3.0.5.tar.gz
cd openssl-3.0.5
sudo ./config --prefix=/usr/local/ssl --openssldir=/usr/local/ssl shared zlib
sudo make
sudo make test
```

Labelling the nodes

When locality-based failover is implemented, Envoy proxy do the fail over based on the following priority order:

1. same Sub zone (`topology.istio.io/subzone`)
2. same region (`topology.kubernetes.io/zone`)
3. same zone (`topology.kubernetes.io/region`)

For MicroFog current implementation region (cloud or fog) and zone (which cluster) is enough for priority-base failover

Failover Rule : If the instance in the cluster is failed, move to the next one in same region (fog clusters prioritise connected fog clusters and use cloud only if fog cluster instances are unreachable.)

```
kubectl get nodes --show-labels

// region : Fog or Cloud || zone : cluster in this case
kubectl label nodes fog2-control-plane topology.kubernetes.io/region=fog
kubectl label nodes fog2-control-plane topology.kubernetes.io/zone=fog2
```

Updating mesh network configmap

```
kubectl -n istio-system edit configmap/istio

  meshNetworks: |-
    networks:
      network1:
        endpoints:
          - fromRegistry: kind-fog1
        gateways:
          - address: <Public_IP>
            port: 15443
      network2:
        endpoints:
          - fromRegistry: kind-fog2
        gateways:
          - address: <Public_IP>
            port: 15443
      network3:
        endpoints:
          - fromRegistry: kind-cloud1
        gateways:
          - address: <Public_IP>
            port: 15443
```

Commands for troubleshooting

Verify the connectivity among

```
kubectl -n istio-system logs service/istiod
```