MetalLB

Features

- *Load-balancer implementation for bare metal Kubernetes clusters*
- ***Can function either in layer 2 or layer 3 modes***
- ***Supports multiple IP pool configuration***
- ***Supports LoadBalancer IP re-use in cases where there are few IP addresses that can be used for load-balancing.***

N.B If you are using Calico or any other BGP based CNI (e.g. kube-router), you might consider advertising the service IPs via the in-built BGP mechanism, in essence you may not need metalLB.

METALLB IN LAYER 2 MODE

- LoadBalancer IP is assigned to just one node (leader-elected node), IP should be in one of the physical host network for routing purposes.
- Works where there is no provision for routers

Limitations:

- Only one node can respond to traffic requests, this means there might be bottle-neck
- Slow failover when the leader-elected node goes down

METALLB IN LAYER 3 MODE

- Uses BGP to advertise LoadBalancer IP.
- Can use ECMP if network routers supports it.

Limitations:

- Requires dedicated router that can speak BGP
- Load balancing behavior is dependent on the network router features. The common behavior is to balance per-connection, based on a packet hash. Two typical options are 3-tuple and 5-tuple hashing. 3-tuple uses (protocol, source-ip, dest-ip) as the key, meaning that all packets between two unique IPs will go to the same backend. 5-tuple hashing adds the source and destination ports to the mix, which allows different connections from the same clients to be spread around the cluster.

LAB 1 Installation

```
kubectl apply -f
https://raw.githubusercontent.com/metallb/metallb/v0.9.5/manifests/name
space.yaml
```

# On first install only

```
kubectl create secret generic -n metallb-system memberlist
--from-literal=secretkey="$(openssl rand -base64 128)"
```

```
kubectl apply -f
https://raw.githubusercontent.com/metallb/metallb/v0.9.5/manifests/meta
llb.yaml
```

Lab 2 - Layer 2

```
cat > metallb.yaml <<EOF
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
    address-pools:
    - name: ip-space-1
      protocol: layer2
      addresses:
      - 10.10.100.200/32
      - 10.10.100.201/32
EOF
```

Lab 2 - Layer 2

Create deployment and expose it as a LB type service

```
kubectl create deploy nginx-metallb --image nginx:stable-alpine
```

```
kubectl expose deploy nginx-metallb --port 80 --type=LoadBalancer
```

Check metallb logs to see which worker node will be responding to ARP

Check connectivity from the Alpine container

Lab 2 - Layer 2

Make sure metalLB controller POD replica is more than one and it is spread across the worker nodes, if replica is just one then there is possibility that controller POD may be running in the node that fails, this will make fail-over a bit challenging since the controller will be down.

```
kubectl -n metallb-system scale deploy controller --replicas=2
```

Test for Failover:

```
kubectl create deploy nginx-metallb-failover --image
nginx:stable-alpine --replicas=4
```

```
kubectl expose deploy nginx-metallb-failover --port 80
--type=LoadBalancer
```

Shutdown the interface of the worker node that is responding to the ARP.

```
sudo ip link set dev enp0s8 down
```

Layer 3 - LAB

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
    peers:
    - my-asn: 64500
      peer-asn: 64500
      peer-address: 10.10.100.1
    address-pools:
    - name: ip-pool-1
      protocol: bgp
      addresses:
      - 172.19.100.0/24
```

Layer 3 - LAB

Configure Vyos

```
set protocols bgp 64500 neighbor 10.10.100.3 remote-as '64500'

set protocols bgp 64500 neighbor 10.10.100.4 remote-as '64500'
```

Create deployment and expose it via LB service type

```
kubectl create deploy nginx-metallb-l3 --image nginx:stable-alpine

kubectl expose deploy nginx-metallb-l3 --port 80 --type=LoadBalancer
```

Layer 3 - Multiple IP Pools LAB

```
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
    peers:
    - my-asn: 64500
      peer-asn: 64500
      peer-address: 10.10.100.1
    address-pools:
    - name: ip-pool-1
      protocol: bgp
      addresses:
      - 172.19.100.0/24
    - name: ip-pool-2
      protocol: bgp
      addresses:
      - 172.19.101.0/24
```

Layer 3 - Multiple IP Pools LAB

Create deployment

```
kubectl create deploy nginx-metallb-l3-ip-pool --image
nginx:stable-alpine
```

Create the service manifest.

```
kubectl expose deploy nginx-metallb-l3-ip-pool --port 80
--type=LoadBalancer --dry-run=client -o yaml >
metallb-ip-pool-demo.yaml
```

Add IP pool annotations to the service manifest and create it.

```
metallb.universe.tf/address-pool: ip-pool-2
```

Layer 3 - IP Sharing LAB

We will use the exisitng coreDNS deployment, 2 new services type LoadBalancer will be created, one service for the TCP port 53 and the other one for UDP port 53. Each of the service will use the same LB IP address.

coreDNS TCP service template

```
apiVersion: v1
kind: Service
metadata:
  name: coredns-tcp-svc
  namespace: kube-system
  annotations:
    metallb.universe.tf/allow-shared-ip: gns3-ip
spec:
  ports:
  - port: 53
    protocol: TCP
    targetPort: 53
  selector:
    k8s-app: kube-dns
  type: LoadBalancer
```

Layer 3 - IP Sharing LAB

We will use the existing coreDNS deployment, 2 new services type LoadBalancer will be created, one service for the TCP port 53 and the other one for UDP port 53. Each of the service will use the same LB IP address.

coreDNS UDP service template

```
apiVersion: v1
kind: Service
metadata:
  name: coredns-udp-svc
  namespace: kube-system
  annotations:
    metallb.universe.tf/allow-shared-ip: gns3-ip
spec:
  ports:
  - port: 53
    protocol: UDP
    targetPort: 53
  selector:
    k8s-app: kube-dns
  type: LoadBalancer
  loadBalancerIP: 172.19.100.0
```

Layer 3 - IP Sharing LAB

Test DNS resolution via the load balancer using both UDP and TCP

UDP: `dig +short coredns-tcp-svc.kube-system.svc.cluster.local @LB_IP`

TCP: `dig +short +tcp coredns-udp-svc.kube-system.svc.cluster.local @LB_IP`

N.B - You can install dig in alpine via : `apk add bind-tools`

References

https://metallb.universe.tf/usage/

https://github.com/metallb/metallb/tree/main/manifests

https://metallb.universe.tf/installation/

https://metallb.universe.tf/concepts/