Issuing a Virtual IP to a Service Using MetalLB on Kubernetes

MetalLB is a load balancer implementation for bare metal Kubernetes clusters, using L2 advertisements. This tutorial will guide you through the process of setting up MetalLB in your Kubernetes cluster and assigning a virtual IP to a service.

Step 1: Install MetalLB

MetalLB can be installed via a manifest or using Helm. We'll use the manifest method here.

1. Apply the MetalLB manifest:

```
kubectl apply -f
https://raw.githubusercontent.com/metallb/metallb/v0.13.12/config/manifests/me
tallb-native.yaml
```

Note: Ensure you're using the latest version of MetalLB.

2. Verify the Installation.

`kubectl get pods -n metallb-system`

You should see the MetalLB pods running.

Step 2: Configure MetalLB

MetalLB can operate in either Layer 2 mode or BGP mode. We'll use Layer 2 mode for simplicity.

1. Create a ConfigMap for MetalLB: Define a range of IP addresses that MetalLB will manage. Create a file named `metallb-pool.yaml` with the following content:

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: ippool
  namespace: metallb-system
spec:
  addresses:
  - 192.168.1.200/32
  -  192.168.1.240-192.168.1.250
```

Replace IP ranges with your desired IP range.

Apply the Pool:

`kubectl apply -f metallb-pool.yaml`

2. Create a L2 Advertisement: When additional IP ranges are defined in the config- map, they need to be advertised on to the network. Create a file named `L2add.yaml` with the following content:

```
apiVersion: metallb.io/v1beta1
kind:  L2Advertisement
metadata:
  name: l2ads
  namespace: metallb-system
spec:
  ipAddressPools:
  - ippool
```

Apply the advertisement:

`kubectl apply -f L2add.yaml`

Step 3: Create a Service with a Virtual IP

Let's expose the wordpress application: Edit the service of type LoadBalancer on `wordpress-service.yaml`:

```
apiVersion: v1
kind: Service
metadata:
  name: wordpress
spec:
  selector:
    app: wordpress
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
  type: LoadBalancer
```

Save and apply it:

`kubectl apply -f wordpress-service.yaml`

Check the Service:

`kubectl get svc wordpress`

3. MetalLB will assign an external IP from the defined range to your service.

Step 4: Access the Service

- You can now access the wordpress server using the external IP provided by MetalLB. This IP is accessible within your network.

Troubleshoot

On a different VM than the master do the testing for ARP advertisements.

Remove !MetalLB (Only for the reference)

```
arp -a
ping 192.168.1.200
sudo apt install iputils-arping
arping 192.168.1.200
```

Remove !MetalLB (Only for the reference)

```
kubectl delete -f
https://raw.githubusercontent.com/metal
lb/metallb/v0.13.12/config/manifests/me
tallb-native.yaml
kubectl delete -f metallb-pool.yaml
kubectl delete -f L2add.yaml
kubectl get all -n metallb-system
```

Kubernetes Ingress. (Optional)

In a Kubernetes environment, if you want to use an Ingress resource to direct traffic to a service that's exposed via NodePort, while still allowing users to access the service using a standard port (like port 80) without specifying the NodePort, you can set it up as follows:

Step 1: Expose Your Service Using NodePort

1. Create a Service of Type NodePort for Your Web Application: Suppose you have a deployment named webapp. You'll need to create a service for it. Here's an example YAML for the service:

```
apiVersion: v1
kind: Service
metadata:
  name: webapp-nodeport-service
spec:
  type: NodePort
  selector:
    app: webapp
  ports:
    - port: 80
      targetPort: 80
      protocol: TCP
```

This service will expose your webapp on a NodePort.

- Apply the Service:

`kubectl apply -f [your-service-file].yaml`

Step 2: Set Up Ingress to Route to the NodePort Service

1. Define an Ingress Resource: Create an Ingress resource that routes traffic to your NodePort? service. Here's an example YAML for the Ingress:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: webapp-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  ingressClassName: nginx
  rules:
  - http:
      paths:
      - path: /webapp
        pathType: Prefix
        backend:
          service:
            name: webapp-nodeport-service
            port:
              number: 80
```

This Ingress resource routes traffic from nginx-ingress controller external IP to the webapp-nodeport-service on port 80.

- Apply the Ingress Resource:

`kubectl apply -f webapp-ingress.yaml`

Step 3: Ensure Ingress Controller is Set Up Correctly

Ensuring that your Ingress Controller is properly set up and accessible from outside the Kubernetes cluster involves several key steps. This setup is crucial for allowing external traffic to reach your services through the Ingress rules you've defined. Here's a breakdown of what this entails:

1. Deploying the Ingress Controller

- Choose an Ingress Controller: There are several Ingress Controllers available, such as NGINX, Traefik, HAProxy, etc. NGINIX is a popular choice due to its stability and feature set.
- Install the Ingress Controller: You need to deploy the Ingress Controller in your Kubernetes cluster. For NGINX, you might use a command like:

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-
nginx/controller-v1.8.2/deploy/static/provider/cloud/deploy.yaml
```

2. Exposing the Ingress Controller

Check IP issued from the load balancer

`kubectl get svc --all-namespaces`

Step 4: DNS Configuration

1. Configure DNS: Map the DNS record to the external IP address of one of your cluster nodes (if using NodePort for the Ingress Controller) or to the external IP provided by the LoadBalancer (if using LoadBalancer for the Ingress Controller).

Read More:

https://spacelift.io/blog/kubernetes-ingress

https://matthewpalmer.net/kubernetes-app-developer/articles/kubernetes-ingress-guide-nginx-example.html