Lesson Plan
Advertising Services

Summary

1. Lab Setup
2. Advertise Cluster IP range
3. Advertise Individual Cluster IPs
4. Advertise Individual External IPs

Lab Setup

If you haven't already done so - add host1 as a BGP peer as described in the Week 3 - Configure BGP Peering module.

This lab assumes you already have the "Yet Another Online Bank" (yaobank) installed, as created in the "Installing the Sample Application" module in Week 1. If you don't already have it installed then go back and do so now.

For reference, this is architecture of YAOBank:

If you are not already on host1, you can enter host1 by using the multipass shell command.

```
multipass shell host1
```

Advertise Cluster IP Range

Advertising services over BGP allows you to directly access the service without using NodePorts or a cluster Ingress Controller.

Examine routes

Let's start by taking a look at the state of routes on host1:

```
ip route
```

Example output:

```
default via 192.168.159.113 dev eth0 proto dhcp src 192.168.159.121 metric
100
192.168.159.112/28 dev eth0 proto kernel scope link src 192.168.159.121
192.168.159.113 dev eth0 proto dhcp scope link src 192.168.159.121 metric
100
198.19.0.0/20 dev eth0 proto kernel scope link src 198.19.15.254
198.19.28.208/29 via 198.19.0.2 dev eth0 proto bird
```

If you completed the previous lab you'll see one route that was learned from Calico that provides access to the nginx pod that was created in the externally routable namespace (the route ending in "proto bird", the last line in this example output). In this lab we will advertise Kubernetes services (rather than individual pods) over BGP.

Update Calico BGP configuration

The serviceClusterIPs clause tells Calico to advertise the cluster IP range.

Apply the configuration:

```
cat <<EOF | calicoctl apply -f -
---
apiVersion: projectcalico.org/v3
kind: BGPConfiguration
metadata:
  name: default
spec:
  serviceClusterIPs:
  - cidr: "198.19.32.0/20"
EOF
```

Verify the BGPConfiguration update worked and contains the serviceClusterIPs key:

```
calicoctl get bgpconfig default -o yaml
```

Example output:

```
apiVersion: projectcalico.org/v3
kind: BGPConfiguration
metadata:
  creationTimestamp: "2020-10-19T21:16:09Z"
  name: default
  resourceVersion: "30335"
  uid: 2bd1a883-4425-4274-a8ce-fe706de98e6a
spec:
  serviceClusterIPs:
  - cidr: 198.19.32.0/20
```

Examine routes

Examine the routes again on host1:

```
ip route
```

Example output:

```
default via 192.168.159.113 dev eth0 proto dhcp src 192.168.159.117 metric
100
192.168.159.112/28 dev eth0 proto kernel scope link src 192.168.159.117
192.168.159.113 dev eth0 proto dhcp scope link src 192.168.159.117 metric
100
198.19.0.0/20 dev eth0 proto kernel scope link src 198.19.15.254
198.19.28.208/29 via 198.19.0.2 dev eth0 proto bird
198.19.32.0/20 proto bird
        nexthop via 198.19.0.1 dev eth0 weight 1
        nexthop via 198.19.0.2 dev eth0 weight 1
        nexthop via 198.19.0.3 dev eth0 weight 1
```

You should now see the cluster service cidr 198.19.32.0/20 advertised from each of the kubernetes cluster nodes. This means that traffic to any service's cluster IP address will get load balanced across all nodes in the cluster by the network using ECMP (Equal Cost Multi Path). Kube-proxy then load balances the cluster IP across the service endpoints (backing pods) in exactly the same way as if a pod had accessed a service via a cluster IP.

Verify we can access cluster IPs

Find the cluster IP for the customer service:

```
kubectl get svc -n yaobank customer
```

Example output:

```
NAME       TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
customer   NodePort   198.19.35.118   <none>        80:30180/TCP   33m
```

In this example output it is 198.19.35.118. Your IP may be different.

Confirm we can access it from host1:

```
curl 198.19.35.118
```

Advertising Cluster IPs in this way provides an alternative to accessing services via Node Ports (simplifying client service discovery without clients needing to understand DNS SRV records) or external network load balancers (reducing overall equipment costs).

Advertise Individual Cluster IPs

You can set `externalTrafficPolicy:Local` on a Kubernetes service to request that external traffic to a service should only be routed via nodes which have a local service endpoint (backing pod). With kube-proxy's service handling this preserves the client source IP and avoids the second hop associated NodePort or ClusterIP services when kube-proxy load balances to a service endpoint (backing pod) on another node.

WIth Calico service IP advertisement, traffic to the cluster IP for a service with `externalTrafficPolicy:Local` will always be routed to nodes with endpoints for that service.

Add external traffic policy

Update the customer service to add externalTrafficPolicy: Local.

```
kubectl patch svc -n yaobank customer -p
'{"spec":{"externalTrafficPolicy":"Local"}}'
```

Examine routes

Examine the new routes on host1:

```
ip route
```

Example output:

```
default via 192.168.159.113 dev eth0 proto dhcp src 192.168.159.117 metric
100
192.168.159.112/28 dev eth0 proto kernel scope link src 192.168.159.117
192.168.159.113 dev eth0 proto dhcp scope link src 192.168.159.117 metric
100
198.19.0.0/20 dev eth0 proto kernel scope link src 198.19.15.254
198.19.28.208/29 via 198.19.0.2 dev eth0 proto bird
198.19.32.0/20 proto bird
        nexthop via 198.19.0.1 dev eth0 weight 1
        nexthop via 198.19.0.2 dev eth0 weight 1
        nexthop via 198.19.0.3 dev eth0 weight 1
198.19.38.11 via 198.19.0.2 dev eth0 proto bird
```

You should now have a /32 route for the yaobank customer service (198.19.38.11 in the above example output) advertised from the node hosting the customer service pod (node1, 198.19.0.2 in this example output).

For each active service with `externalTrafficPolicy:Local`, Calico advertises the IP for that service as a /32 route from the nodes that have endpoints for that service. This means that external traffic to the service will get load balanced across all nodes in the cluster that have a service endpoint (backing pod) for the service by the network using ECMP (Equal Cost Multi Path). Kube-proxy (or Calico eBPF native service handling) then load balance equally to the local backing pods if there is more than one on the node).

The two main advantages of using `externalTrafficPolicy:Local` in this way are:

- There is a network efficiency win avoiding potential second hop of kube-proxy load balancing to another node.
- The client source IP addresses are preserved, which can be useful if you want to restrict access to a service to specific IP addresses using network policy applied to the backing pods. (This is an alternative approach to that explored earlier in the Advanced Policy lab, where we used Calico host endpoint preDNAT policy to restrict external traffic to the services.)

The downsides of this approach are that the evenness of the network ECMP based load balancing is topology dependent. It's therefore common to use anti-affinity rules when scheduling pods that aback these services. You can learn more about in this short video: Everything you need to know about Kubernetes Services Networking.

Note that, as covered in the previous module, Calico's eBPF dataplane native service handling always preserves source IP, and supports DSR eliminates the second hop for return traffic. This is

topology independent, so easy to operationalize, but if you have particularly latency sensitive workloads you may still want to make use of `externalTrafficPolicy:Local` in combinations with Calico service advertisement to eliminate the second hop for inbound traffic.

Advertise External IPs

If you want to advertise a service using an IP address outside of the service cluster IP range, you can configure the service to have one or more external-IPs.

Examine the existing services

Before we begin, examine the kubernetes services in the yaobank kubernetes namespace:

```
kubectl get svc -n yaobank
```

```
NAME       TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
database   ClusterIP   198.19.45.32    <none>        2379/TCP       4m2s
summary    ClusterIP   198.19.38.28    <none>        80/TCP         4m2s
customer   NodePort    198.19.35.118   <none>        80:30180/TCP   4m1s
```

Note that none of them currently have an EXTERNAL-IP.

Update BGP configuration

Update the Calico BGP configuration to advertise a service external IP CIDR range of 198.19.48.0/20:

```
calicoctl patch BGPConfig default --patch \
   '{"spec": {"serviceExternalIPs": [{"cidr": "198.19.48.0/20"}]}}'
```

Note that serviceExternalIPs is a list of CIDRs, so you could for example add individual /32 IP addresses if there were just a small number of specific IPs you wanted to advertise.

Examine routes on host1:

```
ip route
```

Example output:

```
default via 192.168.159.113 dev eth0 proto dhcp src 192.168.159.117 metric
100
192.168.159.112/28 dev eth0 proto kernel scope link src 192.168.159.117
192.168.159.113 dev eth0 proto dhcp scope link src 192.168.159.117 metric
100
```

```
198.19.0.0/20 dev eth0 proto kernel scope link src 198.19.15.254
198.19.28.208/29 via 198.19.0.2 dev eth0 proto bird
198.19.32.0/20 proto bird
        nexthop via 198.19.0.1 dev eth0 weight 1
        nexthop via 198.19.0.2 dev eth0 weight 1
        nexthop via 198.19.0.3 dev eth0 weight 1
198.19.38.11 via 198.19.0.2 dev eth0 proto bird
198.19.48.0/20 proto bird
        nexthop via 198.19.0.1 dev eth0 weight 1
        nexthop via 198.19.0.2 dev eth0 weight 1
        nexthop via 198.19.0.3 dev eth0 weight 1
```

You should now have a route for the external ID CIDR (198.19.48.0/20) with next hops to each of our cluster nodes.

Assign the service external IP

Assign the service external IP 198.19.48.10/20 to the customer service.

```
kubectl patch svc -n yaobank customer -p  '{"spec": {"externalIPs":
["198.19.48.10"]}}'
```

Examine the services again to validate everything is as expected:

```
kubectl get svc -n yaobank
```

```
NAME       TYPE        CLUSTER-IP      EXTERNAL-IP    PORT(S)        AGE
database   ClusterIP   198.19.42.125   <none>         2379/TCP       32m
summary    ClusterIP   198.19.32.103   <none>         80/TCP         32m
customer   NodePort    198.19.38.11    198.19.48.10   80:30180/TCP   32m
```

You should now see the external ip (198.19.48.10) assigned to the customer service. We can now access the customer service from outside the cluster using the external ip address (198.19.48.10) we just assigned.

Verify we can access the service's external IP

Connect to the customer service from the standalone node using the service external IP 198.19.48.10:

```
curl 198.19.48.10
```

As you can see the service has been made available outside of the cluster via bgp routing and network load balancing.

Recap

We've covered five different ways for connecting to your pods from outside the cluster during this Module.

- Via a standard NodePort on a specific node. (This is how you connected to the YAO Bank web front end when you first deployed it.)
- Direct to the pod IP address by using externally routable IP Pools.
- Advertising the service cluster IP range. (And using ECMP to load balance across all nodes in the cluster.)
- Advertising individual cluster IPs. (Services with externalTrafficPolicy: Local, using ECMP to load balance only to the nodes hosting the pods backing the service.)
- Advertising service external-IPs. (So you can use service IP addresses outside of the cluster IP range.)