Lesson Plan

IP Pools and BGP Peering

Summary

1. Lab Setup
2. Cluster IP Address Ranges
3. Create IP Pools with Different Scopes
4. Configure BGP Peering
5. Configure a Namespace to Use External Routable IP Addresses

Lab Setup

This lab can be run on your cluster as is with Calico installed. During the lab we will be deploying an NGINX web server.

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

```
multipass shell host1
```

Cluster IP Address Ranges

Kubernetes configuration

There are two address ranges that Kubernetes is normally configured with that are worth understanding:

- The cluster pod CIDR is the range of IP addresses Kubernetes is expecting to be assigned to pods in the cluster.
- The services CIDR is the range of IP addresses that are used for the Cluster IPs of Kubernetes Sevices (the virtual IP that corresponds to each Kubernetes Service).

These are configured at cluster creation time (e.g. as initial kubeadm configuration).

You can find these values using the following command on host1:

```
kubectl cluster-info dump | grep -m 2 -E "service-cidr|cluster-cidr"
```

Example output:

```
"k3s.io/node-args": "[\"server\",\"--flannel-backend\",\"none\",\"--clu
ster-cidr\",\"198.19.16.0/20\",\"--service-cidr\",\"198.19.32.0/20\",\"--wr
ite-kubeconfig-m
ode\",\"664\",\"--disable-network-policy\"]",
```

Calico configuration

It is also important to understand the IP Pools that Calico has been configured with, which offer finer grained control of IP address ranges to be used by pods in the cluster.

```
calicoctl get ippools
```

Example output:

```
NAME                  CIDR             SELECTOR
default-ipv4-ippool   198.19.16.0/21   all()
```

In this cluster Calico has been configured to allocate IP addresses for pods from the 198.19.16.0/21 CIDR (which is a subset of the cluster pod CIDR, 198.19.16.0/20, configured on Kubernetes).

Overall we have the following address ranges:

CIDR	Purpose
198.19.16.0/20	Cluster Pod CIDR
198.19.16.0/21	Default IP Pool CIDR
198.19.32.0/20	Service CIDR

Create IP Pools with Different Scopes

One use of Calico IP Pools is to distinguish between different ranges of addresses with different routability scopes. If you are operating at very large scales then IP addresses are precious. You might want to have a range of IPs that is only routable within the cluster, and another range of IPs that is routable across the whole of your enterprise. Then you could choose which pods should get IPs from which range depending on whether workloads from outside of the cluster need to directly access the pods or not.

We'll simulate this use case in this lab by creating a second IP Pool to represent the externally routable pool, and we'll use host1 to represent a router that is "outside of the cluster". (And we've already configured the lab infrastructure to not allow routing of the existing IP Pool outside of the cluster.)

Create externally routable IP Pool

We're going to create a new pool for 198.19.24.0/21 that we want to be externally routable.

```
cat <<EOF | calicoctl apply -f -
---
apiVersion: projectcalico.org/v3
kind: IPPool
metadata:
  name: external-pool
spec:
  cidr: 198.19.24.0/21
  blockSize: 29
  ipipMode: Never
  natOutgoing: true
  nodeSelector: "!all()"
EOF
```

Let's check the new IP pools we now have:

```
calicoctl get ippools
```

Example output:

```
Successfully applied 1 'IPPool' resource(s)
NAME                  CIDR              SELECTOR
default-ipv4-ippool   198.19.16.0/21    all()
external-pool         198.19.24.0/21    !all()
```

We now have:

CIDR	Purpose
198.19.16.0/20	Cluster Pod CIDR
198.19.16.0/21	Default IP Pool CIDR
198.19.24.0/21	External Pool CIDR

198.19.32.0/20	Service CIDR

Configure Calico BGP peering

Examine BGP peering status

Some calicoctl commands need to be run locally on the corresponding node. The “calicoctl node status” command is one such command.

Switch to node1:

```
ssh node1
```

Check the status of Calico on the node:

```
sudo calicoctl node status
```

Example output:

```
Calico process is running.

IPv4 BGP status
+--------------+-------------------+-------+----------+-------------+
| PEER ADDRESS |     PEER TYPE     | STATE |  SINCE   |    INFO     |
+--------------+-------------------+-------+----------+-------------+
| 198.19.0.1   | node-to-node mesh | up    | 04:04:44 | Established |
| 198.19.0.3   | node-to-node mesh | up    | 04:04:48 | Established |
+--------------+-------------------+-------+----------+-------------+

IPv6 BGP status
No IPv6 peers found.
```

This shows that currently this node is only peering with the other nodes in the cluster and is not peering to any networks outside of the cluster.

Exit back to host1:

```
exit
```

Add a BGP Peer

In this lab we will simulate peering to a network outside of the cluster by peering to host1. (We've set up host1 to act as if it were a router, and it is ready to accept new BGP peering requests.)

Add the new BGP Peer:

```
cat <<EOF | calicoctl apply -f -
---
apiVersion: projectcalico.org/v3
kind: BGPPeer
metadata:
  name: bgppeer-global-host1
spec:
  peerIP: 198.19.15.254
  asNumber: 64512
EOF
```

Example output:

```
Successfully applied 1 'BGPPeer' resource(s)
```

Examine the new BGP peering status

Switch to node1 again:

```
ssh node1
```

Check the status of Calico on the node:

```
sudo calicoctl node status
```

Example output:

```
Calico process is running.

IPv4 BGP status
+---------------+-------------------+-------+----------+-------------+
| PEER ADDRESS  |     PEER TYPE     | STATE |  SINCE   |    INFO     |
+---------------+-------------------+-------+----------+-------------+
| 198.19.0.1    | node-to-node mesh | up    | 04:04:45 | Established |
| 198.19.0.3    | node-to-node mesh | up    | 04:04:49 | Established |
| 198.19.15.254 | global            | up    | 04:09:36 | Established |
+---------------+-------------------+-------+----------+-------------+
```

```
IPv6 BGP status
No IPv6 peers found.
```

The output shows that Calico is now peered with host1 (198.19.15.254). This means Calico can share routes to and learn routes from host1. (Remember we are using host1 to represent a router in this lab.)

In a real-world on-prem deployment you would typically configure Calico nodes within a rack to peer with the ToRs (Top of Rack) routers, and the ToRs are then connected to the rest of the enterprise or data center network. In this way pods, if desired, can be addressed from anywhere on your network. You could even go as far as giving some pods public IP addresses and have them addressable from the internet if you wanted to.

We're done with adding the peers, so exit from node1 to return back to host1:

```
exit
```

Configure a Namespace to Use External Routable IP Addresses

Calico supports annotations on both namespaces and pods that can be used to control which IP Pool (or even which IP address) a pod will receive when it is created. In this example we're going to create a namespace to host out externally routable.

Create the namespace

Examine the namespaces we're about to create:

```
more 430-namespace.yaml
```

Notice the annotation that will determine which IP Pool pods in the namespace will use.

Apply the namespace:

```
cat <<EOF| kubectl apply -f -
---
apiVersion: v1
kind: Namespace
metadata:
  annotations:
    cni.projectcalico.org/ipv4pools: '["external-pool"]'
  name: external-ns
```

```
EOF
```

Example output:

```
namespace/external-ns created
```

Deploy an NGINX pod

Now deploy a NGINX example pod in the external-ns namespace, along with a simple network policy that allows ingress on port 80.

```
kubectl apply -f
https://raw.githubusercontent.com/tigera/ccol1/main/nginx.yaml
```

Example output:

```
deployment.apps/nginx created
networkpolicy.networking.k8s.io/nginx created
```

Access the NGINX pod from outside the cluster

Let's see what IP address was assigned:

```
kubectl get pods -n external-ns -o wide
```

Example output:

```
NAME                     READY   STATUS    RESTARTS   AGE     IP
NODE    NOMINATED NODE   READINESS GATES
nginx-8c44c96c6-wkzzp    1/1     Running   0          7m55s   198.19.28.208
node1   <none>           <none>
```

The output shows that the nginx pod has an IP address from the externally routable IP Pool.

Try to connect to it from host1:

```
curl 198.19.28.208
```

Example output:

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
```

```
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

This confirms that the NGINX pod is directly routable on the broader network. (In this simplified lab this means host1, but in a real environment this could be across the whole of the enterprise network if desired.)

Check Calico IPAM allocations statistics

Let's also take a quick look at the IP allocation stats from Calico-IPAM, by running the following command:

```
calicoctl ipam show
```

Example output:

```
+----------+----------------+-----------+------------+-------------+
| GROUPING |      CIDR      | IPS TOTAL | IPS IN USE |  IPS FREE   |
+----------+----------------+-----------+------------+-------------+
| IP Pool  | 198.19.16.0/21 |      2048 | 18 (1%)    | 2030 (99%)  |
| IP Pool  | 198.19.24.0/21 |      2048 | 1 (0%)     | 2047 (100%) |
+----------+----------------+-----------+------------+-------------+
```

It can be a good idea to periodically check IP allocations statistics to check you have sized your IP pools appropriately, for example if you aren't confident about the number of pods and whether your original sizing of the pools.