# Kubernetes Networking

# Few useful commands

# ip route show -

# ip neigh show - To list your machines current arp/neighbor cache/table

# ip route list

# route -n - Display the list of routes currently configured

# netstat -rn

# 

# POD to POD Networking

# a) Installing calicoctl as a Kubernetes pod

# kubectl apply -f <https://docs.projectcalico.org/manifests/calicoctl.yaml>

# b) How to find Veth interface

# kubectl exec -ti -n kube-system calicoctl -- /calicoctl get wep -n default -o wide

# 

# Calico CNI work flow

# Every worker node in a Kubernetes cluster have Kubelet running. When a POD is to be created, Kubelet calls down to the CNI using CNI specification. It connects to etcd to get the JSON artifact which defines the CNI specification and calls (ADD, DELETE) the CNI plugin to add the POD namespace into the host.

# Calico uses Etcd for synchronization of the Calico data. When a new POD comes up the orchestrator (Kubelet) calls the CNI using Calico CNI plugin (Calico for Exp). It gives a unique IP address for the POD from the IPAM and write the POD IP address into the Etcd datastore.

# As part of the Calico plugin, on each Node, an agent (Felix) is running which receives a notification from Etcd that a new POD is created on that node. It gets the POD ip address part of the notification and adds a route entry in the Linux routing for the newly created POD with the POD ip address and the virtual ethernet. Along with Felix, there is a daemon (which is running on the Nodes) comes to know about this and need to make sure all nodes in the cluster knows about this newly added IP. It uses BGP protocol and broadcast the details to all the Nodes. This is how it exchanges routes between nodes.

# 

# Tcpdump

# tcpdump for containers inside Kubernetes pods

# Find the container name and node your app is running on.

# Run the below kubectl from the master node

# [root@ip-172-31-29-219 webapp] kubectl get pod backend-deployment-7c84dfbf47-bz622 -o json | grep -i "hostIP"

# "hostIP": "172.31.31.122",

# [root@ip-172-31-29-219 webapp] kubectl get pod backend-deployment-7c84dfbf47-bz622 -o json | grep -i "containerID"

# 

# "containerID": "docker://9f051aa9fc0335867fbdfa753ca3573041ad1e822a991963a19973cd3986e874"

# Connect to the worker with the IP "172.31.31.122", find the pods unique network interface index inside it's container.

# [root@ip-172-31-29-219 webapp] docker exec 9f051aa9fc0335867fbdfa753ca3573041ad1e822a991963a19973cd3986e874 /bin/bash -c 'cat /sys/class/net/eth0/iflink'

# 159

# Locate the interface from the Worker node

# [root@ip-172-31-29-219 webapp] ip link |grep ^159:

# 159: cali07c73a9e5ad@if4: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1440 qdisc noqueue state UP mode DEFAULT group default

# Tcpdump

The output shows “cali07c73a9e5ad” is the virtual interface pointing to the unique network interface index of the running container.  We can use that interface to perform network packet capture's on.

[root@ip-172-31-29-219 webapp]tcpdump -i cali07c73a9e5ad

Invoke a “ping from the any node on the POD ip” ie ping < IP of the POD used) and you will be able to see the dump.

Output

listening on cali07c73a9e5ad@if4, link-type EN10MB (Ethernet), capture size 262144 bytes

09:08:16.419369 IP ip-192-168-121-64.ap-southeast-1.compute.internal > ip-192-168-24-91.ap-southeast-1.compute.internal: ICMP echo request, id 7001, seq 1, length 64

09:08:16.419787 ARP, Request who-has 169.254.1.1 tell ip-192-168-24-91.ap-southeast-1.compute.internal, length 28

09:08:16.419790 ARP, Reply 169.254.1.1 is-at ee:ee:ee:ee:ee:ee (oui Unknown), length 28

09:08:16.419795 IP ip-192-168-24-91.ap-southeast-1.compute.internal > ip-192-168-121-64.ap-southeast-1.compute.internal: ICMP echo reply, id 7001, seq 1, length 64

09:08:17.439204 IP ip-192-168-121-64.ap-southeast-1.compute.internal > ip-192-168-24-91.ap-southeast-1.compute.internal: ICMP echo request, id 7001, seq 2, length 64

# Access ETCD

Connect to the server where ETCD is running (Master in case of Kubeadm)

[root@k8masterMLB vagrant]# docker ps | grep etcd

Select the container id with ‘etcd --advertise-cl…’

[root@k8masterMLB vagrant]# docker exec -it a08243cc05b1 /bin/sh

Set the environment variables

# ETCDCTL\_API=3

# etcdctl --endpoints 127.0.0.1:2379 --cacert /etc/kubernetes/pki/etcd/ca.crt --cert /etc/kubernetes/pki/etcd/server.crt --key /etc/kubernetes/pki/etcd/server.key get / calico

Etcd is the backend data store for all the information Calico needs.

etcdctl --endpoints 127.0.0.1:2379 --cacert /etc/kubernetes/pki/etcd/ca.crt --cert /etc/kubernetes/pki/etcd/server.crt --key /etc/kubernetes/pki/etcd/server.key ls / calico

• /calico/ipam

• /calico/v1

• /calico/bgp

**References:**

https://www.tigera.io/blog/kubernetes-networking-with-calico/

https://sookocheff.com/post/kubernetes/understanding-kubernetes-networking-model/

<https://gokube.io/media/Diving-Deep-Into-Kubernetes-Networking.pdf>

<https://www.praqma.com/stories/debugging-kubernetes-networking/>

<https://www.youtube.com/watch?v=dFsrx5AxgyI&t=636s>

# 

# How does Calico find out the destination IP?

# Connect to the POD using kubectl from the master node

# [root@ip-172-31-29-219 webapp] kubectl exec -it backend-deployment-7c84dfbf47-bz622 /bin/sh

# # ip addr

# 1: lo: <LOOPBACK,UP,LOWER\_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000

# link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00

# inet 127.0.0.1/8 scope host lo

# valid\_lft forever preferred\_lft forever

# 2: tunl0@NONE: <NOARP> mtu 1480 qdisc noop state DOWN group default qlen 1000

# link/ipip 0.0.0.0 brd 0.0.0.0

# 4: eth0@if159: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1440 qdisc noqueue state UP group default

# link/ether 8a:2a:dd:c9:34:78 brd ff:ff:ff:ff:ff:ff

# inet 192.168.24.91/32 scope global eth0

# valid\_lft forever preferred\_lft forever