Experiment: Calico Networking & Kind

In this experiment, we will deploy Calico CNI with on Kind.

As we've learning instead of using VMs or physical hosts as the Kubernetes nodes, Kind spins up docker containers that look like VMs and installs Kubernetes on them. Getting a cluster up and running with Kind is super fast, which makes it an excellent tool for creating test clusters on your laptop.

Kind has a default Container Networking Interface (CNI) plugin called `kindnet`, which is a minimal implementation of a CNI plugin. If we wanted to have a more full featured implementation of CNI for Kind we'd need to switch that out. For our experiment we'll monitor our pods with Calico. That will require setup of Calico as the CNI plugin for Kind clusters.

To use Calico as the CNI plugin in Kind clusters, we need to do the following:

1. **Disable the installation of `kindnet`**
2. **Configure the pod subnet of the cluster**
3. **Install Calico on the cluster**
4. **Tweak Calico's configuration**

Create Calico Kind Configuration

Kind clusters can be customized using a configuration file that exposes a variety of knobs. In our case, we need to disable `kindnet` and set the pod subnet to Calico's default subnet.

To do so, create a `calico-kind.yaml` file that contains the following:

```
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
networking:
  disableDefaultCNI: true # disable kindnet
  podSubnet: 192.168.0.0/16 # set to Calico's default subnet
```

Create your Kind cluster, passing the configuration file using the `--config` flag:

Note: calico-kind.yaml is available in the labs/calico folder of the course GitHub repository.

Create our Cluster

$ kind create cluster --config calico-kind.yaml --image kindest/node:v1.21.12

Verify Kind Cluster

$ kind get kubeconfig

$ kind get clusters
```
kind
```

$ kind get nodes
```
kind-control-plane
```

Once the cluster is up, list the pods in the `kube-system` namespace to verify that `kindnet` is not running:

Note: In older versions of Kind you would have to specify the KUBECONFIG as $ export KUBECONFIG="$(kind get kubeconfig-path –name="kind")". This is not required in the version that we should be running for our experiments.

Verify default CNI is disabled

$ kubectl get pods -n kube-system

`kindnet` should be missing from the list of pods:

```
NAME                                         READY   STATUS    RESTARTS   AGE
coredns-5c98db65d4-dgfs9                     0/1     Pending   0          77s
coredns-5c98db65d4-gg4fh                     0/1     Pending   0          77s
etcd-kind-control-plane                      1/1     Running   0          16s
kube-apiserver-kind-control-plane            1/1     Running   0          24s
kube-controller-manager-kind-control-plane 1/1     Running   0          41s
kube-proxy-qsxp4                             1/1     Running   0          77s
kube-scheduler-kind-control-plane            1/1     Running   0          10s
```

Note: The pair of coredns pods are in the *`pending`* state. This is expected. They will remain in the *`pending`* state until a CNI plugin is installed.

Install Calico

Use the following command to install Calico:

$ kubectl apply -f https://docs.projectcalico.org/v3.8/manifests/calico.yaml
```
Warning: apiextensions.k8s.io/v1beta1 CustomResourceDefinition is deprecated in
v1.16+, unavailable in v1.22+; use apiextensions.k8s.io/v1
CustomResourceDefinition
customresourcedefinition.apiextensions.k8s.io/felixconfigurations.crd.projectca
lico.org created
customresourcedefinition.apiextensions.k8s.io/ipamblocks.crd.projectcalico.org
created
customresourcedefinition.apiextensions.k8s.io/blockaffinities.crd.projectcalico
.org created
customresourcedefinition.apiextensions.k8s.io/ipamhandles.crd.projectcalico.org
created
customresourcedefinition.apiextensions.k8s.io/ipamconfigs.crd.projectcalico.org
```

```
created
customresourcedefinition.apiextensions.k8s.io/bgppeers.crd.projectcalico.org
created
customresourcedefinition.apiextensions.k8s.io/bgpconfigurations.crd.projectcali
co.org created
customresourcedefinition.apiextensions.k8s.io/ippools.crd.projectcalico.org
created
customresourcedefinition.apiextensions.k8s.io/hostendpoints.crd.projectcalico.o
rg created
customresourcedefinition.apiextensions.k8s.io/clusterinformations.crd.projectca
lico.org created
customresourcedefinition.apiextensions.k8s.io/globalnetworkpolicies.crd.project
calico.org created
customresourcedefinition.apiextensions.k8s.io/globalnetworksets.crd.projectcali
co.org created
customresourcedefinition.apiextensions.k8s.io/networkpolicies.crd.projectcalico
.org created
customresourcedefinition.apiextensions.k8s.io/networksets.crd.projectcalico.org
created
clusterrole.rbac.authorization.k8s.io/calico-kube-controllers created
clusterrolebinding.rbac.authorization.k8s.io/calico-kube-controllers created
clusterrole.rbac.authorization.k8s.io/calico-node created
clusterrolebinding.rbac.authorization.k8s.io/calico-node created
daemonset.apps/calico-node created
serviceaccount/calico-node created
deployment.apps/calico-kube-controllers created
serviceaccount/calico-kube-controllers created
```

Wait for Calico to be Ready

$ kubectl wait --namespace kube-system --for=condition=ready pod --selector=k8s-app=calico-kube-controllers --timeout=90s

```
pod/calico-kube-controllers-7f6768fdfb-5cjr5 condition met
```

Relax Calico's RPF Check Configuration

By default, Calico pods fail if the Kernel's Reverse Path Filtering (RPF) check is not enforced. This is a security measure to prevent endpoints from spoofing their IP address.

The RPF check is not enforced in Kind nodes. Thus, we need to disable the Calico check by setting an environment variable in the `calico-node` DaemonSet:

$ kubectl -n kube-system set env daemonset/calico-node FELIX_IGNORELOOSERPF=true

```
ERPF=true
daemonset.apps/calico-node env updated
```

Verify Calico Is Up

To verify that `calico-node` is running, list the pods in the `kube-system` namespace:

$ kubectl -n kube-system get pods | grep calico-node

You should see the `calico-node` pod running and ready (1/1 containers ready):

```
calico-node-slr2g          1/1     Running   0          71s
```

You should also see the CoreDNS pods running if you get a full listing of pods in the `kube-system` namespace.

$ kubectl get pods -n kube-system

Cleanup

$ kind get clusters
kind

$ kind delete cluster
Deleting cluster "kind" ...