Lesson Plan

Calico Installation

Summary

Introduction to Calico Installation Methods

There are many different ways to install Calico. The Calico docs site includes recommended install options across a range of environments, so you don't need to be an expert to get started.

Broadly there are 4 different approaches to install.

Manifest

This is the most basic method for installing Calico. The Calico docs include a range of manifests for different environments. If you are an advanced user, you can customize the manifests to give you ultimate flexibility and control over your installation.

Operator

Calico 3.15 introduces the option to install Calico using an open-source operator, created by Tigera. This offers simplification for installing and configuring Calico without needing to customize manifests. Additionally, the operator allows you to have a uniform, self-healing environment. Using the Tigera operator is highly recommended.

Managed Kubernetes Services

Support for Calico is included with many of the most popular managed Kubernetes services (e.g. EKS, AKS, GKE, IKS), either enabled by default, or optionally enabled using the cloud provider's management consoles or command line tools, depending on the specific managed Kubernetes service.

Kubernetes Distros and Installers

Many Kubernetes distros and installers include support for installing Calico. (e.g. kops, kubespray, microk8s, etc). Most of these currently use manifest based installs under the covers.

Installing Calico for the Lab

We will be using the Tigera Operator to install and configure Calico.

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

```
multipass shell host1
```

The command below will install the operator onto our lab kubernetes cluster:

```
kubectl create -f
https://docs.projectcalico.org/archive/v3.16/manifests/tigera-operator.yaml
```

Validating the Operator installation

Following the operator being installed, we will validate that the operator is running:

```
kubectl get pods -n tigera-operator
```

The output from this command should indicate that the operator pod is running:

```
NAME                                READY   STATUS    RESTARTS   AGE
tigera-operator-64f448dfb9-d2fdq    1/1     Running   0          2m33s
```

Installing Calico

After the operator is in a Running state, we will configure an Installation kind for Calico, specifying the IP Pool that we would like below.

Note that throughout the course we make use of inline manifests (piping stdin to kubectl) to make it easier for you to follow what each manifest does. In most cases it would be a more normal practice to use a vanilla kubectl command with a manifest file (e.g. kubectl apply -f my-installation.yaml). We recommend taking a minute to read through and make sure you understand the contents of each manifest we apply in this way throughout the rest of the course to get the most out of each example.

```
cat <<EOF | kubectl apply -f -
```

```
apiVersion: operator.tigera.io/v1
kind: Installation
metadata:
  name: default
spec:
  calicoNetwork:
    containerIPForwarding: Enabled
    ipPools:
    - cidr: 198.19.16.0/21
      natOutgoing: Enabled
      encapsulation: None
EOF
```

Validating the Calico installation

Following the configuration of the installation resource, Calico will begin deploying onto your cluster. This can be validated by running the following command:

```
kubectl get tigerastatus/calico
```

The output from the command when the installation is complete is:

```
NAME      AVAILABLE   PROGRESSING   DEGRADED   SINCE
calico    True        False         False      10m
```

We can review the environment now by invoking:

```
kubectl get pods -A
```

Example output:

```
NAMESPACE         NAME                                       READY   STATUS
RESTARTS   AGE
tigera-operator   tigera-operator-84c5c5d6df-zb49b           1/1     Running
0          5m48s
calico-system     calico-typha-868bb997ff-l22n7              1/1     Running
0          4m6s
calico-system     calico-typha-868bb997ff-fvmws              1/1     Running
0          3m24s
calico-system     calico-typha-868bb997ff-8qt45              1/1     Running
0          3m24s
calico-system     calico-node-r94mp                          1/1     Running
```

```
0          4m6s
calico-system     calico-node-w5ptt                          1/1     Running
0          4m6s
calico-system     calico-node-zgrvb                          1/1     Running
0          4m6s
kube-system       helm-install-traefik-t68vd                 0/1
Completed   0          35m
kube-system       metrics-server-7566d596c8-pccvz            1/1     Running
2          35m
kube-system       local-path-provisioner-6d59f47c7-gh97b     1/1     Running
2          35m
calico-system     calico-kube-controllers-89cf65556-c7gz7    1/1     Running
3          4m6s
kube-system       coredns-7944c66d8d-f4q6g                   1/1     Running
0          35m
kube-system       svclb-traefik-9bxg2                        2/2     Running
0          32s
kube-system       svclb-traefik-pb72f                        2/2     Running
0          32s
kube-system       svclb-traefik-l6mzn                        2/2     Running
0          32s
kube-system       traefik-758cd5fc85-8hcdx                   1/1     Running
0          32s
```

Reviewing Calico pods

Let's take a look at the Calico pods that have been installed by the operator.

```
kubectl get pods -n calico-system
```

Example Output:

```
NAME                                       READY   STATUS    RESTARTS   AGE
calico-typha-5d788c654b-56wp9              1/1     Running   0
4h28m
calico-node-2bkv5                          1/1     Running   0
4h28m
calico-kube-controllers-5dcfdbc5f4-vpgx5   1/1     Running   0
4h28m
calico-node-8465h                          1/1     Running   0
4h26m
calico-typha-5d788c654b-wn7gf              1/1     Running   0
4h24m
calico-node-qmq5j                          1/1     Running   0
3h57m
calico-typha-5d788c654b-rd8kl              1/1     Running   0
3h56m
```

From here we can see that there are different pods that are deployed:

- **calico-node**: Calico-node runs on every Kubernetes cluster node as a DaemonSet. It is responsible for enforcing network policy, setting up routes on the nodes, plus managing any virtual interfaces for IPIP, VXLAN, or WireGuard.
- **calico-typha**: Typha is as a stateful proxy for the Kubernetes API server. It's used by every calico-node pod to query and watch Kubernetes resources without putting excessive load on the Kubernetes API server. The Tigera Operator automatically scales the number of Typha instances as the cluster size grows.
- **calico-kube-controllers**: Runs a variety of Calico specific controllers that automate synchronization of resources. For example, when a Kubernetes node is deleted, it tidies up any IP addresses or other Calico resources associated with the node.

Reviewing Node Health

Finally, we can review the health of our Kubernetes nodes by invoking the kubectl command.

```
kubectl get nodes -A
```

Example output:

```
NAME      STATUS   ROLES    AGE   VERSION
control   Ready    master   37m   v1.18.10+k3s1
node2     Ready    <none>   16m   v1.18.10+k3s1
node1     Ready    <none>   31m   v1.18.10+k3s1
```

Now we can see that our Kubernetes nodes have a status of Ready and are operational. Calico is now installed on your cluster and you may proceed to the next module: Installing the Sample Application.