CoreDNS and CNI

What is CoreDNS

CoreDNS is a server written in Go, that provides a flexible plugin to listen for DNS requests coming in over UDP/TCP traffic. It serves as DNS for a cluster and must be installed via the CoreDNS GitHub project here.

CoreDNS is a complete solution for DNS resolution of internal hosts that make it easy to manage while still providing security capabilities to the entire network.

All DNS services are run in the kube-system namespace. In order to view CoreDNS and its predecessor Kube-DNS execute the `kubectl get all -n kube-system` command.

```
dominickhrndz314@cloudshell:~$ kubectl get all -n kube-system
NAME                                   READY   STATUS    RESTARTS   AGE
pod/coredns-78fcd69978-15rjd           1/1     Running   0          44s
pod/etcd-minikube                      1/1     Running   0          58s
pod/kube-apiserver-minikube            1/1     Running   0          56s
pod/kube-controller-manager-minikube   1/1     Running   0          55s
pod/kube-proxy-sb9bh                   1/1     Running   0          44s
pod/kube-scheduler-minikube            1/1     Running   0          55s
pod/storage-provisioner                1/1     Running   0          54s

NAME               TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)                  AGE
service/kube-dns   ClusterIP   10.96.0.10   <none>        53/UDP,53/TCP,9153/TCP   56s

NAME                        DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR            AGE
daemonset.apps/kube-proxy   1         1         1       1            1           kubernetes.io/os=linux   56s

NAME                      READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/coredns   1/1     1            1           56s
```

Alternatively you can also search for CoreDNS specifically with the `kubectl get cm coredns -n kube-system` command.

```
dominickhrndz314@cloudshell:~$ kubectl get cm coredns -n kube-system
NAME      DATA   AGE
coredns   1      105s
dominickhrndz314@cloudshell:~$
```

CoreDNS vs Kube-DNS

CoreDNS runs on a single container per instance, vs kube-dns which uses three. **Kube-DNS** uses single threaded caching vs CoreDNS, which is multi-threaded since its written in Go. This simply means that CoreDNS can open multiple connections and measure the speeds across them simultaneously.

Hands on – View and Modify the CoreDNS config file

I. View the core file:

The core file for CoreDNS is a ConfigMap with a section that defines CoreDNS behaviors. You cannot modify the file directly so we must create our own ConfigMap to overwrite the settings. Start by viewing the core file.

```
# Please edit the object below. Lines beginning with a '#' will be ignored,
# and an empty file will abort the edit. If an error occurs while saving this file will be
# reopened with the relevant failures.
#
apiVersion: v1
data:
  Corefile: |
    .:53 {
        errors
        health {
           lameduck 5s
        }
        ready
        kubernetes cluster.local in-addr.arpa ip6.arpa {
           pods insecure
           fallthrough in-addr.arpa ip6.arpa
           ttl 30
        }
        prometheus :9153
        hosts {
           192.168.49.1 host.minikube.internal
           fallthrough
        }
        forward . /etc/resolv.conf {
           max_concurrent 1000
        }
        cache 30
        loop
        reload
        loadbalance
    }
kind: ConfigMap
metadata:
  creationTimestamp: "2022-03-21T14:14:40Z"
  name: coredns
  namespace: kube-system
  resourceVersion: "333"
  uid: 3ab43d09-e977-4bca-857b-180993f82b10
```

II. Create the ConfigMap:

We need to create a new config map and then apply it to the cluster. Once that’s done we can verify everything was created and then force our new Configmap to overwrite the old core file, by restarting the pod.

To restart a pod, simply delete the pod. Once this is done, check the kube-system namespace to verify that your new CoreDNS pod is running and Kube-DNS is not.

```
dominickhrndz314@cloudshell:~$ nano configmap-coredns.yaml
dominickhrndz314@cloudshell:~$ cat configmap-coredns.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: coredns-custom
  namespace: kube-system
data:
  example.server: | # All custom server files must have a ".server" file extension.
    # Change example.com to the domain you wish to forward.
    example.com {
      # Change 8.8.8.8 (Google) to your DNS resolver.
      forward . 8.8.8.8
    }
```

```
dominickhrndz314@cloudshell:~$ nano configmap-coredns.yaml
dominickhrndz314@cloudshell:~$ cat configmap-coredns.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: coredns-custom
  namespace: kube-system
data:
  example.server: | # All custom server files must have a ".server" file extension.
    # Change example.com to the domain you wish to forward.
    example.com {
      # Change 8.8.8.8 (Google) to your DNS resolver.
      forward . 8.8.8.8
    }
```

Now restart the pod.

```
dominickhrndz314@cloudshell:~$ kubectl delete pod --namespace kube-system -l k8s-app=kube-dns
pod "coredns-78fcd69978-15rjd" deleted
```

```
dominickhrndz314@cloudshell:~$ kubectl get pods -n kube-system
NAME                                READY   STATUS    RESTARTS   AGE
coredns-78fcd69978-lwqqh            1/1     Running   0          67s
etcd-minikube                       1/1     Running   0          31m
kube-apiserver-minikube             1/1     Running   0          31m
kube-controller-manager-minikube    1/1     Running   0          31m
kube-proxy-sb9bh                    1/1     Running   0          31m
kube-scheduler-minikube             1/1     Running   0          31m
storage-provisioner                 1/1     Running   0          31m
dominickhrndz314@cloudshell:~$
```

All Google requests to their 8.8.8.8 public IP will now be forwarded via the CoreDNS service. If you're deployment is running you have completed this part of the training.

What is Container Network Interface

In Kubernetes, networking is one of the central components, providing connectivity between pods within the same host and across hosts. To make networking easier, Kubernetes uses **CNI** to provide a unified interface for interaction between containers.

There are several CNI implementations, many of which are available as open source CNI 'plugins'.

These CNI plugins are apart of the pod network which includes one or more containers and a CNI plugin for each node. When configured, the containers connect dynamically to establish a network between pods and nodes.

For more information on CNI in GCP, please read the following articles:

https://medium.com/cloudzone/gke-networking-options-explained-demonstrated-5c0253415eba

https://medium.com/thermokline/how-to-choose-a-k8s-cni-plugin-771edf4842c0