Exercise 9.3: Working with CoreDNS

1. We can leverage **CoreDNS** and predictable hostnames instead of IP addresses. A few steps back we created the `service-lab` NodePort in the Accounting namespace. We will create a new pod for testing using `Ubuntu`. The pod name will be named `nettool`.

```
student@cp:~$ vim nettool.yaml
```

nettool.yaml

```
1  apiVersion: v1
2  kind: Pod
3  metadata:
4    name: nettool
5  spec:
6    containers:
7    - name: ubuntu
8      image: ubuntu:latest
9      command: [ "sleep" ]
10     args: [ "infinity" ]
```

2. Create the pod and then log into it.

```
student@cp:~$ kubectl create -f nettool.yaml
1 pod/ubuntu created
```

```
student@cp:~$ kubectl exec -it ubuntu -- /bin/bash
```

On Container

(a) Add some tools for investigating DNS and the network. The installation will ask you the geographic area and timezone information. Someone in Austin would first answer 2. America, then 37 for Chicago, which would be central time

```
root@ubuntu:/#  apt-get update ; apt-get install curl dnsutils -y
```

(b) Use the **dig** command with no options. You should see root name servers, and then information about the DNS server responding, such as the IP address.

```
root@ubuntu:/# dig
1  ; <<>> DiG 9.16.1-Ubuntu <<>>
2  ;; global options: +cmd
3  ;; Got answer:
4  ;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 3394
5  ;; flags: qr rd ra; QUERY: 1, ANSWER: 13, AUTHORITY: 0, ADDITIONAL: 1
6
7  <output_omitted>
8
9  ;; Query time: 4 msec
10 ;; SERVER: 10.96.0.10#53(10.96.0.10)
11 ;; WHEN: Thu Aug 27 22:06:18 CDT 2020
12 ;; MSG SIZE  rcvd: 431
```

(c) Also take a look at the /etc/resolv.conf file, which will indicate nameservers and default domains to search if no using a Fully Qualified Distinguished Name (FQDN). From the output we can see the first entry is `default.svc.cluster.local.`.

```
root@ubuntu:/# cat /etc/resolv.conf
```

```
nameserver 10.96.0.10
search default.svc.cluster.local svc.cluster.local cluster.local
c.endless-station-188822.internal google.internal
options ndots:5
```

(d) Use the **dig** command to view more information about the DNS server. Us the **-x** argument to get the FQDN using the IP we know. Notice the domain name, which uses `.kube-system.svc.cluster.local.`, to match the pod namespaces instead of `default`. Also note the name, `kube-dns`, is the name of a service not a pod.

```
root@ubuntu:/# dig @10.96.0.10 -x 10.96.0.10
```

```
...
;; QUESTION SECTION:
;10.0.96.10.in-addr.arpa.         IN        PTR

;; ANSWER SECTION:
10.0.96.10.in-addr.arpa. 30        IN        PTR        kube-dns.kube-system.svc.cluster.local.

;; Query time: 0 msec
;; SERVER: 10.96.0.10#53(10.96.0.10)
;; WHEN: Thu Aug 27 23:39:14 CDT 2020
;; MSG SIZE  rcvd: 139
```

(e) Recall the name of the `service-lab` service we made and the namespaces it was created in. Use this information to create a FQDN and view the exposed pod.

```
root@ubuntu:/# curl service-lab.accounting.svc.cluster.local.
```

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
...
```

(f) Attempt to view the default page using just the service name. It should fail as `nettool` is in the default namespace.

```
root@ubuntu:/# curl service-lab
```

```
curl: (6) Could not resolve host: service-lab
```

(g) Add the `accounting` namespaces to the name and try again. Traffic can access a service using a name, even across different namespaces.

```
root@ubuntu:/# curl service-lab.accounting
```

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<output_omitted>
```

(h) Exit out of the container and look at the services running inside of the `kube-system` namespace. From the output we see that the `kube-dns` service has the DNS serverIP, and exposed ports DNS uses.

```
root@ubuntu:/# exit
```

```
student@cp:~$ kubectl -n kube-system get svc
```

```
NAME       TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)                  AGE
kube-dns   ClusterIP   10.96.0.10   <none>        53/UDP,53/TCP,9153/TCP   42h
```

3. Examine the service in detail. Among other information notice the selector in use to determine the pods the service communicates with.

```
student@cp:~$ kubectl -n kube-system get svc kube-dns -o yaml
```

```
...
    labels:
      k8s-app: kube-dns
      kubernetes.io/cluster-service: "true"
      kubernetes.io/name: KubeDNS
...
    selector:
      k8s-app: kube-dns
    sessionAffinity: None
    type: ClusterIP
...
```

4. Find pods with the same labels in all namespaces. We see that infrastructure pods all have this label, including **?**coredns?

```
student@cp:~$ kubectl get pod -l k8s-app --all-namespaces
```

```
NAMESPACE     NAME                                       READY   STATUS    RESTARTS   AGE
kube-system   calico-kube-controllers-5447dc9cbf-275fs   1/1     Running   0          41h
kube-system   calico-node-6q74j                          1/1     Running   0          43h
kube-system   calico-node-vgzg2                          1/1     Running   0          42h
kube-system   coredns-f9fd979d6-4dxpl                    1/1     Running   0          41h
kube-system   coredns-f9fd979d6-nxfrz                    1/1     Running   0          41h
kube-system   kube-proxy-f4vxx                           1/1     Running   0          41h
kube-system   kube-proxy-pdxwd                           1/1     Running   0          41h
```

5. Look at the details of one of the `coredns` pods. Read through the pod spec and find the image in use as well as any configuration information. You should find that configuration comes from a configmap.

```
student@cp:~$ kubectl -n kube-system get pod coredns-f9fd979d6-4dxpl -o yaml
```

```
...
spec:
  containers:
  - args:
    - -conf
    - /etc/coredns/Corefile
    image: k8s.gcr.io/coredns:1.7.0
```

```
...
    volumeMounts:
    - mountPath: /etc/coredns
      name: config-volume
      readOnly: true
...
  volumes:
  - configMap:
      defaultMode: 420
      items:
      - key: Corefile
        path: Corefile
      name: coredns
    name: config-volume
...
```

6. View the configmaps in the `kube-system` namespace.

```
student@cp:~$ kubectl -n kube-system get configmaps
```

```
NAME                                 DATA   AGE
calico-config                        4      43h
coredns                              1      43h
extension-apiserver-authentication   6      43h
kube-proxy                           2      43h
kubeadm-config                       2      43h
kubelet-config-1.20                  1      43h
kubelet-config-1.21                  1      41h
```

7. View the details of the `coredns` configmap. Note the `cluster.local` domain is listed.

```
student@cp:~$ kubectl -n kube-system get configmaps coredns -o yaml
```

```
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
        forward . /etc/resolv.conf {
           max_concurrent 1000
        }
        cache 30
        loop
        reload
        loadbalance
    }
kind: ConfigMap
...
```

8. While there are many options and zone files we could configure, lets start with simple edit. Add a rewrite statement such that `test.io` will redirect to `cluster.local` More about each line can be found at coredns.io.

```
student@cp:~$ kubectl -n kube-system edit configmaps coredns
```

```
apiVersion: v1
data:
  Corefile: |
    .:53 {
        rewrite name regex (.*)\.test\.io {1}.default.svc.cluster.local   #<-- Add this line
        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
        forward . /etc/resolv.conf {
           max_concurrent 1000
        }
        cache 30
        loop
        reload
        loadbalance
    }
```

9. Delete the coredns pods causing them to re-read the updated configmap.

```
student@cp:~$ kubectl -n kube-system delete pod coredns-f9fd979d6-s4j98 coredns-f9fd979d6-xlpzf
pod "coredns-f9fd979d6-s4j98" deleted
pod "coredns-f9fd979d6-xlpzf" deleted
```

10. Create a new web server and create a `ClusterIP` service to verify the address works. Note the new service IP to start with a reverse lookup.

```
student@cp:~$ kubectl create deployment nginx --image=nginx
deployment.apps/nginx created
```

```
student@cp:~$ kubectl expose deployment nginx --type=ClusterIP --port=80
service/nginx expose
```

```
student@cp:~$ kubectl get svc
NAME         TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.96.0.1        <none>        443/TCP   3d15h
nginx        ClusterIP   10.104.248.141   <none>        80/TCP    7s
```

11. Log into the `ubuntu` container and test the URL rewrite starting with the reverse IP resolution.

```
student@cp:~$ kubectl exec -it ubuntu -- /bin/bash
```

On Container

(a) Use the **dig** command. Note that the service name becomes part of the FQDN.

```
root@ubuntu:/# dig -x 10.104.248.141
```

```
....
;; QUESTION SECTION:
;141.248.104.10.in-addr.arpa.        IN        PTR

;; ANSWER SECTION:
141.248.104.10.in-addr.arpa. 30        IN        PTR        nginx.default.svc.cluster.local.
....
```

(b) Now that we have the reverse lookup test the forward lookup. The IP should match the one we used in the previous step.

```
root@ubuntu:/# dig nginx.default.svc.cluster.local.
```

```
....
;; QUESTION SECTION:
;nginx.default.svc.cluster.local. IN        A

;; ANSWER SECTION:
nginx.default.svc.cluster.local. 30 IN        A        10.104.248.141
....
```

(c) Now test to see if the rewrite rule for the `test.io` domain we added resolves the IP. Note the response uses the original name, not the requested FQDN.

```
root@ubuntu:/# dig nginx.test.io
```

```
....
;; QUESTION SECTION:
;nginx.test.io.                        IN        A

;; ANSWER SECTION:
nginx.default.svc.cluster.local. 30 IN        A        10.104.248.141
....
```

12. Exit out of the container then edit the configmap to add an `answer` section.

```
student@cp:~$ kubectl -n kube-system edit configmaps coredns
```

```
....
data:
  Corefile: |
    .:53 {
        rewrite stop {                                          #<-- Edit this and following two lines
           name regex (.*)\.test\.io {1}.default.svc.cluster.local
           answer name (.*)\.default\.svc\.cluster\.local {1}.test.io
           }
        errors
        health {

....
```

13. Delete the coredns pods again to ensure they re-read the updated configmap.

```
student@cp:~$ kubectl -n kube-system delete pod coredns-f9fd979d6-fv9qn coredns-f9fd979d6-lnxn5
```

```
pod "coredns-f9fd979d6-fv9qn" deleted
pod "coredns-f9fd979d6-lnxn5" deleted
```

14. Log into the ubuntu container again. This time the response should show the FQDN with the requested FQDN.

```
student@cp:~$ kubectl exec -it ubuntu -- /bin/bash
```

On Container

```
root@ubuntu:/# dig nginx.test.io
```

```
....
;; QUESTION SECTION:
;nginx.test.io.                         IN        A

;; ANSWER SECTION:
nginx.test.io.                 30        IN        A        10.104.248.141
....
```

15. Exit then delete the DNS test tools container to recover the resources.

```
student@cp:~$ kubectl delete -f nettool.yaml
```