

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
```

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
```

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

- (d) Use the **dig** command to view more information about the DNS server. Use 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
```

```
1 ...
2 ;; QUESTION SECTION:
3 ;10.0.96.10.in-addr.arpa.      IN      PTR
4
5 ;; ANSWER SECTION:
6 10.0.96.10.in-addr.arpa. 30      IN      PTR      kube-dns.kube-system.svc.cluster.local.
7
8 ;; Query time: 0 msec
9 ;; SERVER: 10.96.0.10#53(10.96.0.10)
10 ;; WHEN: Thu Aug 27 23:39:14 CDT 2020
11 ;; 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.
```

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <title>Welcome to nginx!</title>
5 <style>
6     body {
7         width: 35em;
8         margin: 0 auto;
9         font-family: Tahoma, Verdana, Arial, sans-serif;
10     }
11 ...
```

- (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
```

```
1 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
```



```

1 <!DOCTYPE html>
2 <html>
3 <head>
4 <title>Welcome to nginx!</title>
5 <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
```

```

1 ...
2   labels:
3     k8s-app: kube-dns
4     kubernetes.io/cluster-service: "true"
5     kubernetes.io/name: KubeDNS
6 ...
7   selector:
8     k8s-app: kube-dns
9   sessionAffinity: None
10  type: ClusterIP
11 ...

```

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
```

```

1 ...
2 spec:
3   containers:
4   - args:
5     - -conf
6     - /etc/coredns/Corefile
7     image: k8s.gcr.io/coredns:1.7.0

```

```

8 ...
9   volumeMounts:
10   - mountPath: /etc/coredns
11     name: config-volume
12     readOnly: true
13 ...
14   volumes:
15   - configMap:
16     defaultMode: 420
17     items:
18     - key: Corefile
19       path: Corefile
20     name: coredns
21     name: config-volume
22 ...

```

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-1.21	1	43h
kubelet-config-1.22	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
```

```

1 apiVersion: v1
2 data:
3   Corefile: |
4     .:53 {
5       errors
6       health {
7         lameduck 5s
8       }
9       ready
10      kubernetes cluster.local in-addr.arpa ip6.arpa {
11        pods insecure
12        fallthrough in-addr.arpa ip6.arpa
13        ttl 30
14      }
15      prometheus :9153
16      forward . /etc/resolv.conf {
17        max_concurrent 1000
18      }
19      cache 30
20      loop
21      reload
22      loadbalance
23    }
24 kind: ConfigMap
25 ...

```

8. While there are many options and zone files we could configure, let's 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
```

```

1 apiVersion: v1
2 data:
3   Corefile: |
4     .:53 {
5       rewrite name regex (.*)\.test\.io {1}.default.svc.cluster.local    #<-- Add this line
6       errors
7       health {
8         lameduck 5s
9       }
10      ready
11      kubernetes cluster.local in-addr.arpa ip6.arpa {
12        pods insecure
13        fallthrough in-addr.arpa ip6.arpa
14        ttl 30
15      }
16      prometheus :9153
17      forward . /etc/resolv.conf {
18        max_concurrent 1000
19      }
20      cache 30
21      loop
22      reload
23      loadbalance
24    }

```

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
```

```

1 pod "coredns-f9fd979d6-s4j98" deleted
2 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
```

```
1 deployment.apps/nginx created
```

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

```
1 service/nginx expose
```

```
student@cp:~$ kubectl get svc
```

```

1 NAME          TYPE        CLUSTER-IP    EXTERNAL-IP  PORT(S)    AGE
2 kubernetes    ClusterIP   10.96.0.1     <none>       443/TCP    3d15h
3 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
```



```

1  ....
2  ;; QUESTION SECTION:
3  ;141.248.104.10.in-addr.arpa.      IN      PTR
4
5  ;; ANSWER SECTION:
6  141.248.104.10.in-addr.arpa. 30      IN      PTR      nginx.default.svc.cluster.local.
7  ....

```

- (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.
```

```

1  ....
2  ;; QUESTION SECTION:
3  ;nginx.default.svc.cluster.local. IN      A
4
5  ;; ANSWER SECTION:
6  nginx.default.svc.cluster.local. 30 IN      A      10.104.248.141
7  ....

```

- (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
```

```

1  ....
2  ;; QUESTION SECTION:
3  ;nginx.test.io.                  IN      A
4
5  ;; ANSWER SECTION:
6  nginx.default.svc.cluster.local. 30 IN      A      10.104.248.141
7  ....

```

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

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

```

1  ....
2  data:
3  Corefile: |
4  .:53 {
5      rewrite stop {                                     #<-- Edit this and following two lines
6          name regex (.*)\.test\.io {1}.default.svc.cluster.local
7          answer name (.*)\.default\.svc\.cluster\.local {1}.test.io
8      }
9      errors
10     health {
11
12     ....

```

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
```

```

1  pod "coredns-f9fd979d6-fv9qn" deleted
2  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
```

```
1 ....
2 ;; QUESTION SECTION:
3 ;nginx.test.io.                IN      A
4
5 ;; ANSWER SECTION:
6 nginx.test.io.                 30      IN      A      10.104.248.141
7 ....
```

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

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