Kubernetes DNS: CoreDNS architecture, Deployment, Debugging

Check GitHub for helpful DevOps tools:

Michael Robotics
Hi, I'm Michal. I'm a Robotics Engineer and DevOps enthusiast. My mission is to create skill-learning platform that combats information overload by adhering to the set of principles: simplify, prioritize, and execute.

https://github.com/MichaelRobotics

Ask Personal AI Document assistant to learn interactively (FASTER)!

1 Download PDF

2 Go to website

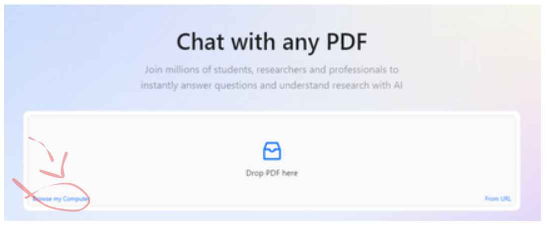
3 Browse file

4 Chat with Document

1 https://github.com/MichaelRobotics/DevOpsTools/blob/main/KubernetesDNS.pdf

Click there to go to ChatPdf website 2

3



Chat with any PDF

Join millions of students, researchers and professionals to instantly answer questions and understand research with AI

Ask questions about document! 4

Complety new to Linux and Networking?

Essential for this PDF is a thorough knowledge of networking. I highly recommend the HTB platform's networking module, which offers extensive information to help build a comprehensive understanding.

HTB - Your Cyber Performance Center

We provide a human-first platform creating and maintaining high performing cybersecurity individuals and organizations.

https://www.hackthebox.com/

What is Kubernetes?

Kubernetes is an open-source platform that automates the deployment, scaling, and management of containerized applications. It helps manage clusters of nodes running containers, ensuring efficient and reliable operation.

How Kubernetes clusters are made?

Kubernetes clusters consist of a control plane and multiple worker nodes. The control plane manages cluster operations, while worker nodes run the actual container workloads.

Why and When use Kubernetes

Kubernetes is ideal for deploying scalable, resilient, and automated containerized applications. It is used when managing multiple containers across different environments is necessary.

Example: Running a microservices-based e-commerce platform that scales up during peak hours.

System Requirements

- RAM: 2 GB per node (1 GB can work for testing but may lead to limited performance)
- 10 GB free storage
- Ubuntu

Kubernetes: Main components & packages

- **kube-apiserver:** Central management component that exposes the Kubernetes API; acts as the front-end for the cluster.
- **etcd:** Distributed key-value store for storing all cluster data, ensuring data consistency across nodes.
- **kube-scheduler:** Assigns pods to available nodes based on resource requirements and policies.
- **kube-controller-manager:** Manages core controllers that handle various functions like node status, replication, and endpoints.
- **kubelet:** Agent that runs on each node, responsible for managing pods and their containers.
- **kube-proxy:** Manages networking on each node, ensuring communication between pods and services within the cluster.

Kubernetes DNS: k8s DNS architecture, CoreDNS deployment

1) What is CoreDNS

CoreDNS is the default DNS server in Kubernetes, responsible for resolving service and pod domain names within the cluster. It is highly configurable, lightweight, and uses plugins to handle DNS-based service discovery and forwarding queries to external DNS servers.

2) Install CoreDNS

Check if CoreDNS is configured on your cluster:

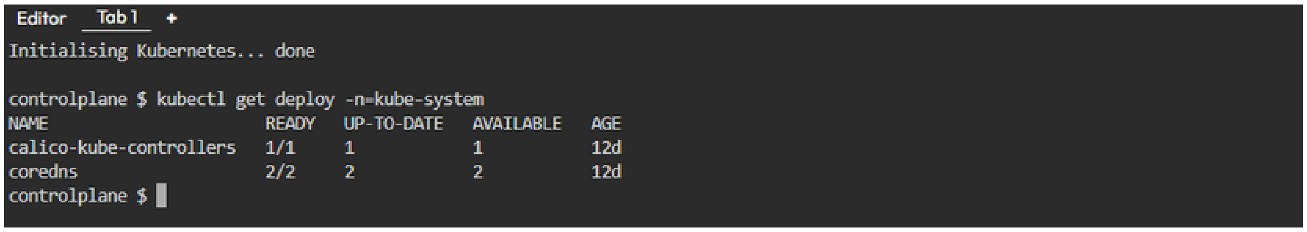
```
kubectl get deploy -n=kube-system
```

If there are 0 coredns deployments, make one:

```
kubectl scale deploy coredns --replicas=2 -n=kube-system
```

Check pods:

```
kubectl get deploy -n=kube-system
```



```
Initialising Kubernetes... done

controlplane $ kubectl get deploy -n=kube-system
NAME                      READY   UP-TO-DATE   AVAILABLE   AGE
calico-kube-controllers   1/1     1            1           12d
coredns                   2/2     2            2           12d
controlplane $ 
```

Check if network plugin is installed. CoreDNS needs network plugin to work correctly.

```
curl -L --remote-name https://github.com/cilium/cilium-cli/releases/latest/download/cilium-linux-amd64
chmod +x cilium-linux-amd64
sudo mv cilium-linux-amd64 /usr/local/bin/cilium
```

3) Test CoreDNS

deploy 2 ngnix pods for testing. First pod:

```
kubectl run nginx-pod-1 --image=nginx --restart=Never
kubectl expose pod nginx-pod-1 --type=NodePort --port=80
```

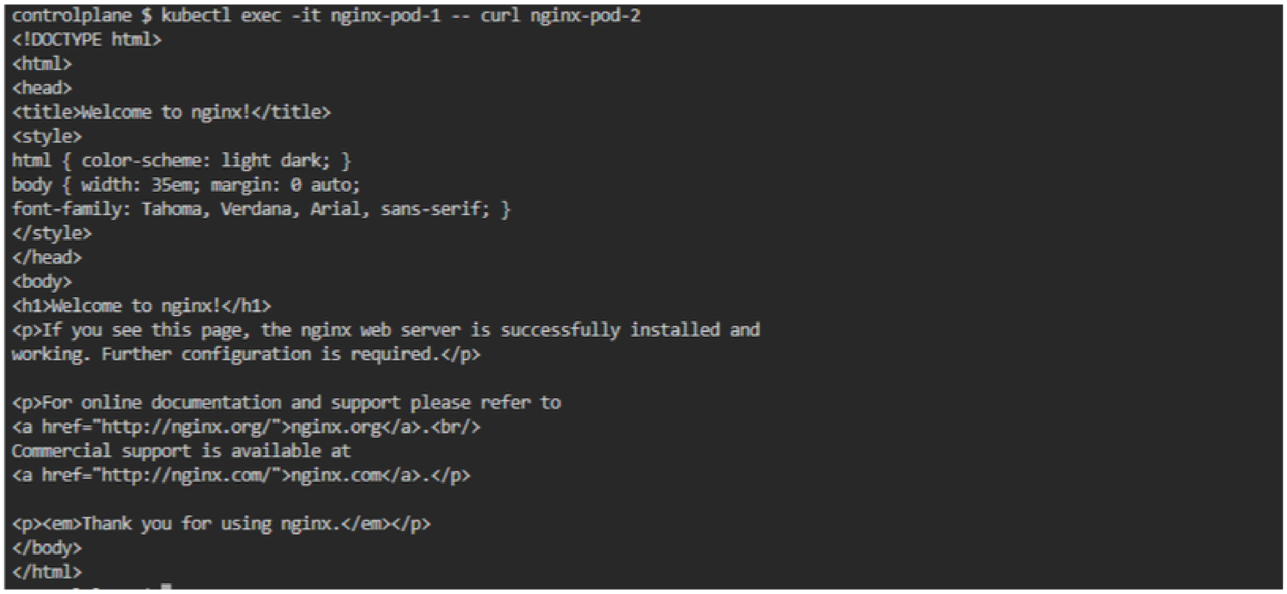
Second pod:

```
kubectl run nginx-pod-2 --image=nginx --restart=Never
kubectl expose pod nginx-pod-2 --type=NodePort --port=80
```

Test conectivity (exec into nginx-pod-1 pod and curl nginx-pod-2)

```
kubectl exec -it nginx-pod-1 -- curl nginx-pod-2
```

Output should be as follows:

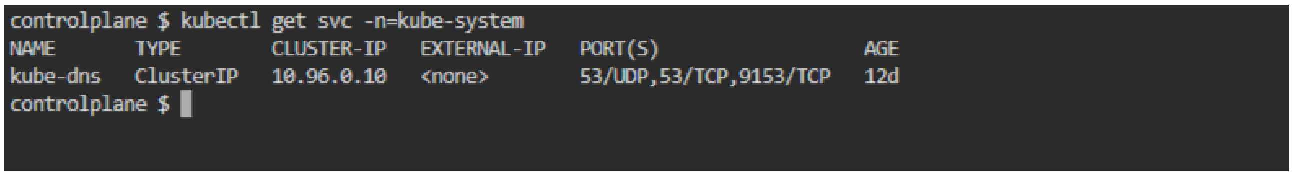


```
controlplane $ kubectl exec -it nginx-pod-1 -- curl nginx-pod-2
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

4) DNS server setup

To get CoreDNS server address:

```
kubectl get svc -n=kube-system
```



```
controlplane $ kubectl get svc -n=kube-system
NAME       TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)                  AGE
kube-dns   ClusterIP   10.96.0.10   <none>        53/UDP,53/TCP,9153/TCP   12d
controlplane $
```

kube-dns is k8s service with respect to CoreDNS. Containers navigate towards DNS server, using its CLUSTER-IP

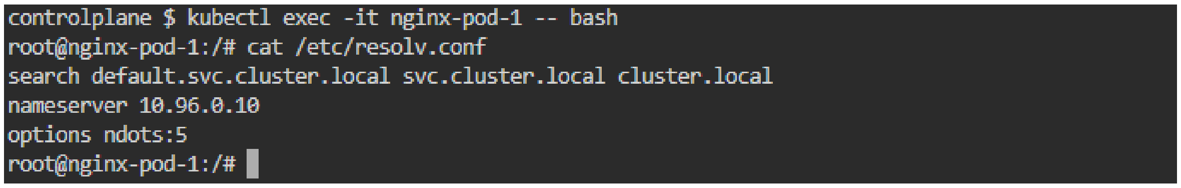
5) Container configuration

Enter to container

```
kubectl exec -it nginx -- bash
```

Check /etc/resolv.conf

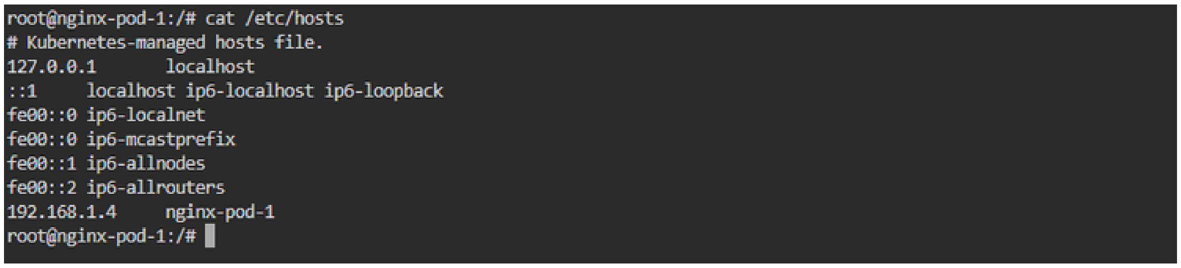
```
cat /etc/resolv.conf
```



```
controlplane $ kubectl exec -it nginx-pod-1 -- bash
root@nginx-pod-1:/# cat /etc/resolv.conf
search default.svc.cluster.local svc.cluster.local cluster.local
nameserver 10.96.0.10
options ndots:5
root@nginx-pod-1:/#
```

nameserver is DNS server ip. search parameter define domain for service discovery within the Kubernetes cluster.

Check /hosts file for container A records



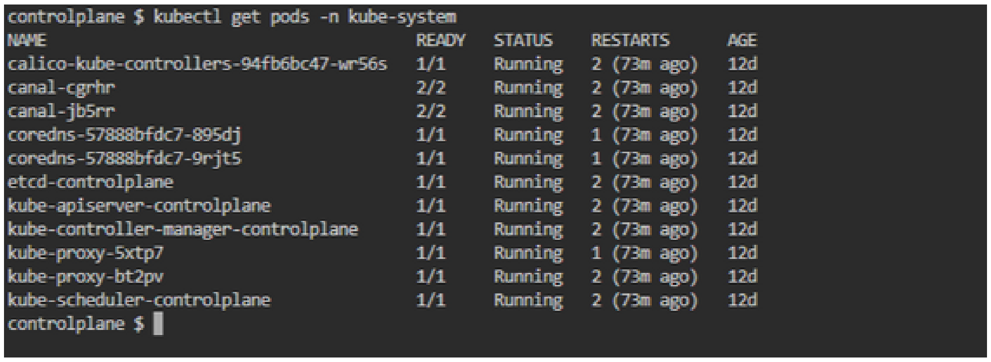
```
root@nginx-pod-1:/# cat /etc/hosts
# Kubernetes-managed hosts file.
127.0.0.1       localhost
::1     localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
fe00::0 ip6-mcastprefix
fe00::1 ip6-allnodes
fe00::2 ip6-allrouters
192.168.1.4     nginx-pod-1
root@nginx-pod-1:/#
```

As you see, A container (pod) usually has only an A record for itself. To resolve addresses of other pods or services, it relies on a DNS server like CoreDNS.

Kubernetes DNS: CoreDNS configuration

Get all pods in kube-system

```
kubectl get pods -n kube-system
```

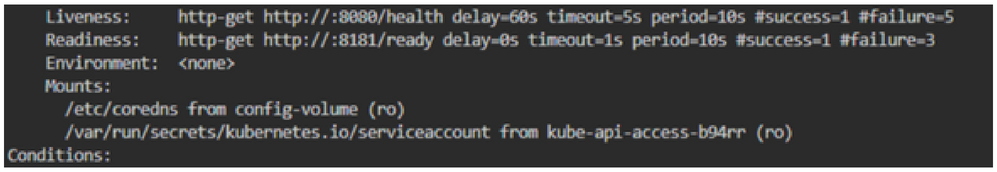


```
controlplane $ kubectl get pods -n kube-system
NAME                                       READY   STATUS    RESTARTS      AGE
calico-kube-controllers-94fb6bc47-wr56s    1/1     Running   2 (73m ago)   12d
canal-cgrhr                                2/2     Running   2 (73m ago)   12d
canal-jb5rr                                2/2     Running   2 (73m ago)   12d
coredns-57888bfdc7-895dj                   1/1     Running   1 (73m ago)   12d
coredns-57888bfdc7-9rjt5                   1/1     Running   1 (73m ago)   12d
etcd-controlplane                          1/1     Running   2 (73m ago)   12d
kube-apiserver-controlplane                1/1     Running   2 (73m ago)   12d
kube-controller-manager-controlplane       1/1     Running   2 (73m ago)   12d
kube-proxy-5xtp7                           1/1     Running   1 (73m ago)   12d
kube-proxy-bt2pv                           1/1     Running   2 (73m ago)   12d
kube-scheduler-controlplane                1/1     Running   2 (73m ago)   12d
controlplane $
```

Describe core dns pod

```
kubectl describe pod <pod-name> -n=kube-system
```

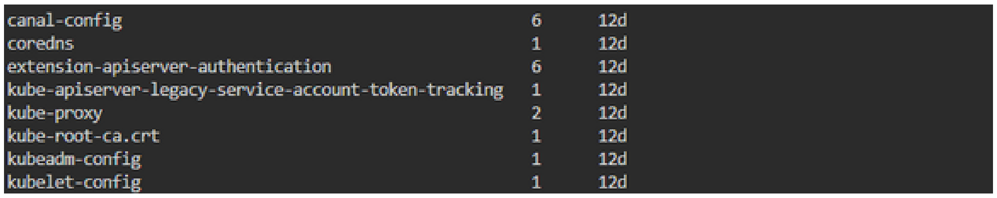
Search for ConfigMap.



```
    Liveness:     http-get http://:8080/health delay=60s timeout=5s period=10s #success=1 #failure=5
    Readiness:    http-get http://:8181/ready delay=0s timeout=1s period=10s #success=1 #failure=3
    Environment:  <none>
    Mounts:
      /etc/coredns from config-volume (ro)
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-b94rr (ro)
Conditions:
```

Config map is mounted as volume to CoreDNS pod. Get config map

```
kubectl get cm -n=kube-system
```



```
canal-config                                          6      12d
coredns                                               1      12d
extension-apiserver-authentication                    6      12d
kube-apiserver-legacy-service-account-token-tracking  1      12d
kube-proxy                                            2      12d
kube-root-ca.crt                                      1      12d
kubeadm-config                                        1      12d
kubelet-config                                        1      12d
```

Get info about this ConfigMap

```
kubectl describe cm coredns -n kube-system
```

Output should look as follows:

```
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
}



BinaryData
====

Events:  <none>
```

- **errors**: Logs errors.
- **health**: Health check endpoint (lameduck 5s for graceful shutdown).
- **ready**: Readiness probe for Kubernetes.
- **kubernetes** cluster.local in-addr.arpa ip6.arpa: Handles Kubernetes DNS (pods insecure, fallthrough in-addr.arpa ip6.arpa, ttl 30).
- **prometheus** :9153: Exposes Prometheus metrics.
- **forward** . /etc/resolv.conf: Forwards queries (max_concurrent 1000).
- **cache 30:** Caches DNS responses for 30s.
- **loop**: Prevents forwarding loops.
- **reload**: Auto-reloads on config changes.
- **loadbalance**: Balances query responses.

Kubernetes DNS: CoreDNS debugging

Testing if dns works properly

```
kubectl exec -i -t <pod_name> -- nslookup kubernetes.default
```

Working properly, means this output:

```
Server:    10.0.0.10
Address 1: 10.0.0.10

Name:      kubernetes.default
Address 1: 10.0.0.1
```

If the nslookup command fails, check the following:

Take a look inside the resolv.conf file

```
kubectl exec -ti dnsutils -- cat /etc/resolv.conf
```

Verify that the search path and name server are set up like the following

```
search default.svc.cluster.local svc.cluster.local cluster.local google.internal
c.gce_project_id.internal
nameserver 10.0.0.10
options ndots:5
```

Check if the DNS pod is running_

```
kubectl get pods --namespace=kube-system -l k8s-app=kube-dns
```

```
NAME                       READY    STATUS    RESTARTS   AGE
...
coredns-7b96bf9f76-5hsxb   1/1      Running   0          1h
coredns-7b96bf9f76-mvmmt   1/1      Running   0          1h
```

Check for errors in the DNS pod

```
kubectl logs --namespace=kube-system -l k8s-app=kube-dns
```

Here is an example of a healthy CoreDNS log:

```
2018/08/15 14:37:17 [INFO] CoreDNS-1.2.2
2018/08/15 14:37:17 [INFO] linux/amd64, go1.10.3, 2e322f6
CoreDNS-1.2.2
linux/amd64, go1.10.3, 2e322f6
```

Is DNS service up?

```
kubectl get svc --namespace=kube-system
```

```
NAME        TYPE        CLUSTER-IP    EXTERNAL-IP   PORT(S)          AGE
...
kube-dns    ClusterIP   10.0.0.10     <none>        53/UDP,53/TCP    1h
```

Are DNS endpoints exposed?

```
kubectl get endpoints kube-dns --namespace=kube-system
```

```
NAME       ENDPOINTS                       AGE
kube-dns   10.180.3.17:53,10.180.3.17:53   1h
```

If you do not see the endpoints, see the endpoints section in the debugging Services documentation.

For additional Kubernetes DNS examples, see the cluster-dns examples in the Kubernetes GitHub repository

common troubleshooting

1) CoreDNS Pods Not Running

Cause: CoreDNS Deployment issues or insufficient resources.
Solution: Check the CoreDNS Deployment with kubectl get deployment -n kube-system. Ensure there are sufficient CPU/memory resources and that node taints don't prevent scheduling. Restart CoreDNS pods if needed.

2) DNS Resolution Failing for Cluster Services

Cause: CoreDNS configuration issues or connectivity problems.
Solution: Verify the Corefile configuration in the kube-system namespace using kubectl -n kube-system get configmap coredns -o yaml. Check CoreDNS logs for errors (kubectl logs -n kube-system -l k8s-app=kube-dns). Test DNS resolution with kubectl exec and nslookup or dig.

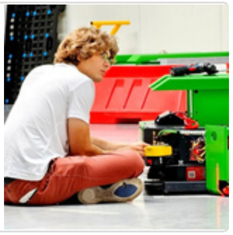
3) External DNS Queries Timing Out

Cause: CoreDNS cannot forward queries to external DNS servers.
Solution: Verify the forward or proxy plugin configuration in the Corefile. Ensure external DNS servers (e.g., 8.8.8.8) are reachable from the cluster. Check network policies and firewalls.

4) Check my Kubernetes Troubleshooting series:

Michael Robotics
Hi, I'm Michal. I'm a Robotics Engineer and DevOps enthusiast. My mission is to create skill-learning platform that combats skill information overload by adhering to the set of principles: simplify, prioritize, and execute.

https://github.com/MichaelRobotics



Learn more about Kubernetes

Check Kubernetes and piyushsachdeva - great docs!

Setup a Multi Node Kubernetes Cluster

kubeadm is a tool to bootstrap the Kubernetes cluster

https://github.com/piyushsachdeva/CKA-2024/tree/main/Resources/Day27

Kubernetes Documentation

This section lists the different ways to set up and run Kubernetes

https://kubernetes.io/docs/setup/

Share, comment, DM and check GitHub for scripts & playbooks created to automate process.

Check my GitHub

Michael Robotics

Hi, I'm Michal. I'm a Robotics Engineer and DevOps enthusiast. My mission is to create skill-learning platform that combats skill information overload by adhering to the set of principles: simplify, prioritize, and execute.

https://github.com/MichaelRobotics

PS.

If you need a playbook or bash script to manage KVM on a specific Linux distribution, feel free to ask me in the comments or send a direct message!