# CoreDNS

Kubernetes API server uses the service name + namespace domain names to communicate with resources. So Pods need to be able to resolve that domain name.

In its logs we can see some information about the traffic.

The iptables is used to control traffic inside the cluster, including communication to the DNS server. We can use it to see the firewall rules.

In the CoreDNS configMap, next to the ‘forward’ keyword, is specified either an IP addresses or a path to the file (like /etc/resolv.conf) with IP addresses of the DNS servers which will be resolving DNS names which CoreDNS can’t resolve itself.

Its IP address should be used in the /etc/resolv.conf file of Pods indicating that this server will be resolving DNS names.

## Testing DNS

We can create simple Pod using public image and use nslookup to check if DNS works properly.

Create a Pod. We will get access to its terminal and that Pod will be deleted after exiting it:

* kubectl run dns-test --image=busybox:1.28 --restart=Never -it --rm -- /bin/sh

From the Pod’s terminal run:

* nslookup kubernetes.default – DNS name for API server
* nslookup google.com – check a public domain

## resolv.conf file

/etc/resolv.conf file contains information about IP addresses of the DNS servers which are used for translating DNS names.

It is contained in every Pod and is used for resolving domain (DNS) names.

In the resolv.conf file in a Pod there might be written:

* search domain-name-1 domain-name-2 …
* Options ndots:x

That means there need to be at least x dots in the domain name in order to be treated as FQDN.

So if Pod tries to resolve a domain name which has less than x dots, then it will try to append domain names from the ‘search’ field to the domain name which it is trying to resolve.

## Useful commands

* iptables – check firewall rules. We can use it to check if rules for CoreDNS and kube-proxy are set up properly.
* EDITOR=nano kubectl edit configmap coredns -n kube-system – Use nano in order to edit a configMap called ‘coredns’ in the ‘kube-system’ namespace.
* kubectl rollout restart deployment coredns -n kube-system – restart a coredns deployment.

# Pods

* kubectl get pod <pod-name> -n <namespace> -o yaml – Get the Pod specification in YAML format.
* kubectl get pods -n namespace – list Pods in the specified namespace
* kubectl get pods –all-namespaces – List Pods in all namespaces
* kubectl delete pod <pod-name> - Delete the Pod.
* kubectl exec -it <pod-name> -- /bin/bash – Get access to a bash session in a running Pod.
* kubectl run -it <pod-name> --image=your-docker-image -- /bin/bash – Run a Pod and get access to a bash session in it.

# Logs

* kubectl logs -n <namespace\_name> <pod-name> - check logs of a Pod from a given namespace

# Nodes

* kubectl get nodes – list nodes

# Services

* kubectl get svc – list services

# Roles

* kubectl get rolebinding,clusterrolebinding --all-namespaces -o wide | grep <name> – get roles assigned to all the users, groups and Service Accounts. ‘grep’ filters outputs to find for a specific string.

# Other commands

* kubectl apply -f path\_to\_yaml\_manifest – deploy resource using its manifest YAML
* kubectl rollout restart deployment <deployment-name> -n <namespace>– restart a deployment
* Kubectl get …. -o YAML – get information about a YAML specification of some resource.
* Kubectl describe <resource-type> <resource-name> -n <namespace> - Describe given resource. Similar to YAML specification but there are additionally for example events (warnings, errors).
* EDITOR=nano kubectl edit <resource-type> <resource-name> -n <namespace> – Use nano in order to edit a specified resource.
* Kubectl patch <resource-type> <resource-name> -n <namespace> … - Modify a specified resource (there are different options for doing this).

# Setting up Kubernetes

Here are usefull tools for debugging.

* crictl ps -a – list the running and stopped containers.
* Crictl logs <containerID> - check logs for a given container
* ss -tuln | grep 6443 – check processes listening on the 6443 port
* ps aux | grep kube-apiserver – check processes in which the ‘kube-apiserver’ string appears.
* journalctl -u kubelet -f – view logs from the kubelet process
* kubectl get pods -n kube-system – Check pods in the kube-system namespace
* telnet 10.0.1.4 6443 – try to connect over TCP to the server with the 10.0.1.4 IP address over the 6443 port.

## Kubernetes config file

This file contains configuration files that kubectl uses to connect and authenticate to Kubernetes and it is located usually at /etc/kubernetes/admin.conf.

When Linux user uses kubectl, then those configuration files specifies which Kubernetes user will be used for executing kubectl commands, like creating resources.

Depending on which configuration files are used, user might see different resources in Kubernetes because of permissions assigned to the Kubernetes user which is used.

## KUBECONFIG variable

This variable indicates a path to the Kubernetes configuration file (admin.conf). If it is not set up then Kubernetes will look for that config file in the default location ~/.kube/config.