# CoreDNS

In Kubernetes, resources uses DNS names to communicate with each other.

## Kube-dns

Kube-dns is CoreDNS’ Service.

Pods should have in the /etc/resolv.conf file a Cluster IP of the kube-dns (or coreds) Service so they can use the kube-dns for resolving DNS names of other Pods / Services.

In CoreDNS 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 for example the firewall rules.

## CoreDNS configMap

In the CoreDNS configMap, next to the ‘forward’ keyword, there is specified an IP address of the DNS server which will be used for resolving DNS names which CoreDNS can’t resolve on its own.

If there is specified a file path:

A black screen with white text

AI-generated content may be incorrect.

That means that the DNS server specified at this path on a Node (a host) will be used.

If there is an IP address:

A black screen with white text

AI-generated content may be incorrect.

Then that means that this IP address will be used.

That DNS server needs to be able to resolve public domains like google.com. If it can’t, then that’s a problem.

## Kubernetes services DNS names

In Kubernetes, all Services are accessible via DNS at:

* <service-name>.<namespace>.svc

Any Pod can use that DNS name to connect to other Pods which uses that Service.

Kubernetes API server always uses that DNS name to communicate with resources through a Service.

If we can’t resolve that DNS name from inside of a Pod, then that indicates an issue with Kubernetes.

# Containerd

Kubernetes uses containerd as a runtime for running containers. Kubelet is communicating with it.

## Containerd.sock

It is a file which is a Unix domain socket. It is containerd’s endpoint used by clients (like Kubernetes (kubelet) and other CLI tools) for communication with containerd.

Path of that file uniquely identifies that endpoint and is used by clients to connect to containerd (like an IP address).

## Crictl

It is a CLI tool used to interact with container runtimes like containerd.

## Crictl.yaml

It is a configuration file used by the circtl. Amoung the others it specifies a path to the socket (the .sock file) to use.

## Cgroup management – sytemdCgroup configuration

Cgroup is a Linux kernel feature that controls and limits the resource usage (CPU, RAM etc) of groups of processes.

The sytemdCgroup option in the containerd configuration indicates whether or not use systemd for cgroup management.

When sytemdCgroup = true then systemd will be used for cgroup management.

When systemdCgroup = false then containerd will manage cgroups using the cgoupsfs interface independently of systemd.

sytemdCgroup = true is usually required for Kubernetes. It is required if kubelet is using systemd for cgroup management.

# CNI

CNI stands for Container Network Interface. It’s a specification that provides network connectivity to containers in Kubernetes.

Kubernetes uses CNI plugins such as:

* Calico
* Flannel
* Cilium

Those plugins handles networking, for example:

* Assignins IP addresses to Pods
* Setting up network interface inside Pods
* Routing traffic

# Kubernetes config file

The Kubernetes config file (Kubeconfig 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.

We use the KUBECONFIG environment variable to specify which Kubeconfig file will be used by the current Linux user. More info in the ‘KUBECONFIG environment’ section.

Some kubectl or helm commands might fail silently if we are using wrong Kubeconfig file.

## KUBECONFIG variable

This environment variable indicates a path to the Kubernetes configuration file (admin.conf) which will be used by the current Linux user.

If it is not set up then Kubernetes will look for that config file in the default location ~/.kube/config.

# Kube context

A kube context is a named entry in the kubeconfig file that tells kubectl which cluster to talk to, as which user, and in which namespace by default.

It contains:

* Cluster – Kubernetes API server endpoint
* User – Auth credentials
* Namespace – Default namespace
* Name – Name of the kube context

All the kube contexts are saved in the Kubeconfig file (~./kube/config).

# Service

## Labels and selector

In order to attach a Service to Pods, we need to define a label in all those Pods, for example:



And then specify in the Service a selector matching those labels:



## Cluster IP

Every Service get assigned a Cluster IP from the CIDR range (defined by the CNI when setting up a cluster).

Kube-proxy forwards requests sent to a Service’s Cluster IP, to Pods attached to that Service.

It is used only for the internal cluster communication. It can’t be reached from outside of the cluster.

That means that from inside of a Pod we can connect to other Pods using their Service’s Cluster IP, but we can’t do that from outside of a Pod.

# Webhooks

Webhook uses TLS certificates for secure HTTPS communication between Kubernetes API server and Webhook server. It is saved as files on the Webhook server and as the caBundle field in the MutatingWebhookConfiguration or ValidatingWebhookConfiguration.