# CoreDNS

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

Pods need to be able to resolve those domain names to communicate with other resources. In order to do that, they should have in the /etc/resolv.conf file an IP address of CoreDNS service. That indicates that CoreDNS is used for resolving DNS names.

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.

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.

## Kubernetes services DNS names

In Kubernetes, all services are accessible via DNS at:

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

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.

# 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.

# 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 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.

# 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