

Communicating between Namespaces

In this lesson, we will establish communication between Namespaces.

WE'LL COVER THE FOLLOWING ^

- Creating a Pod
- Establishing the Communication

Creating a Pod

We'll create an `alpine`-based Pod that we'll use to demonstrate communication between Namespaces.

```
kubectl config use-context minikube

kubectl run test \
  --image=alpine \
  --generator "run-pod/v1" \
  sleep 10000
```



We switched to the `minikube` context (`default` Namespace) and created a Pod with a container based on the `alpine` image. We let it `sleep` for a long time. Otherwise, the container would be without a process and would stop almost immediately.

Before we proceed, we should confirm that the Pod is indeed running.

```
kubectl get pod test
```



The **output** is as follows.

```
NAME READY STATUS  RESTARTS AGE
test 1/1   Running 0       10m
```



Please wait a few moments if, in your case, the Pod is not yet ready.

Establishing the Communication

Before we proceed, we'll install `curl` inside the container in the `test` Pod.

```
kubectl exec -it test \
  -- apk add -U curl
```



We already explored communication between objects in the same Namespace. Since the `test` Pod is running in the `default` Namespace, we can, for example, reach the `go-demo-2-api` Service by using the Service name as a DNS name.

```
kubectl exec -it test -- curl \
  "http://go-demo-2-api:8080/demo/hello"
```



The **output** is as follows.

```
hello, release 1.0!
```



We got the response from the release 1.0 because that's the one running in the same Namespace. Does that mean that we cannot reach Services from other Namespaces?

When we create a Service, it creates a few DNS entries. One of them corresponds to the name of the Service.

So, the `go-demo-2-api` Service created a DNS based on that name. Actually, the full DNS entry is `go-demo-2-api.svc.cluster.local`. Both resolve to the same service `go-demo-2-api` which, in this case, runs in the `default` Namespace.

The third DNS entry we got is in the format `<service-name>.<namespace-name>.svc.cluster.local`. In our case, that is `go-demo-2-api.default.svc.cluster.local`. Or, if we prefer a shorter version, we could use `go-demo-2-api.default`.

In most cases, there is no good reason to use the `<service-name>.<namespace-name>` format when communicating with Services within the same

Namespace

Namespace.

The primary objective behind the existence of the DNSes with the Namespace name is when we want to reach services running in a different Namespace.

If we'd like to reach `go-demo-2-api` running in the `testing` Namespace from the `test` Pod in the `default` Namespace, we should use the `go-demo-2-api.testing.svc.cluster.local` DNS or, even better, the shorter version `go-demo-2-api.testing`.

```
kubectl exec -it test -- curl \
  "http://go-demo-2-api.testing:8080/demo/hello"
```



This time, the **output** is different.

```
hello, release 2.0!
```



Kube DNS used the DNS suffix `testing` to deduce that we want to reach the Service located in that Namespace. As a result, we got the response from the release 2.0 of the `go-demo-2` application.

In the next lesson, we will turn on the destructive mode and delete the Namespace alongwith everything associated to it.