**Service Discovery**

Kubernetes service discovery by example

Service discovery is the process of figuring out how to connect to a service. While there is a service discovery option based on [environment variables](https://kubernetes.io/docs/concepts/services-networking/connect-applications-service/#environment-variables) available, the DNS-based service discovery is preferable. Note that DNS is a [cluster add-on](https://github.com/kubernetes/kubernetes/blob/master/cluster/addons/dns/README.md) so make sure your Kubernetes distribution provides for one or install it yourself.

Let’s create a [service](https://github.com/openshift-evangelists/kbe/blob/master/specs/sd/svc.yaml) named thesvc and an [RC](https://github.com/openshift-evangelists/kbe/blob/master/specs/sd/rc.yaml) supervising some pods along with it:

$ kubectl apply -f https://raw.githubusercontent.com/openshift-evangelists/kbe/master/specs/sd/rc.yaml

$ kubectl apply -f https://raw.githubusercontent.com/openshift-evangelists/kbe/master/specs/sd/svc.yaml

Now we want to connect to the thesvc service from within the cluster, say, from another service. To simulate this, we create a [jump pod](https://github.com/openshift-evangelists/kbe/blob/master/specs/sd/jumpod.yaml) in the same namespace (default, since we didn’t specify anything else):

$ kubectl apply -f https://raw.githubusercontent.com/openshift-evangelists/kbe/master/specs/sd/jumpod.yaml

The DNS add-on will make sure that our service thesvc is available via the FQDN thesvc.default.svc.cluster.local from other pods in the cluster. Let’s try it out:

$ kubectl exec -it jumpod -c shell -- ping thesvc.default.svc.cluster.local

PING thesvc.default.svc.cluster.local (172.30.251.137) 56(84) bytes of data.

...

The answer to the ping tells us that the service is available via the cluster IP 172.30.251.137. We can directly connect to and consume the service (in the same namespace) like so:

$ kubectl exec -it jumpod -c shell -- curl http://thesvc/info

Note that the IP address 172.17.0.5 above is the cluster-internal IP address of the jump pod.

To access a service that is deployed in a different namespace than the one you’re accessing it from, use a FQDN in the form $SVC.$NAMESPACE.svc.cluster.local.

Let’s see how that works by creating:

1. a [namespace](https://github.com/openshift-evangelists/kbe/blob/master/specs/sd/other-ns.yaml) other
2. a [service](https://github.com/openshift-evangelists/kbe/blob/master/specs/sd/other-svc.yaml) thesvc in namespace other
3. an [RC](https://github.com/openshift-evangelists/kbe/blob/master/specs/sd/other-rc.yaml) supervising the pods, also in namespace other

If you’re not familiar with namespaces, check out the namespace examples first.

$ kubectl apply -f https://raw.githubusercontent.com/openshift-evangelists/kbe/master/specs/sd/other-ns.yaml

$ kubectl apply -f https://raw.githubusercontent.com/openshift-evangelists/kbe/master/specs/sd/other-rc.yaml

$ kubectl apply -f https://raw.githubusercontent.com/openshift-evangelists/kbe/master/specs/sd/other-svc.yaml

We’re now in the position to consume the service thesvc in namespace other from the default namespace (again via the jump pod):

$ kubectl exec -it jumpod -c shell -- curl http://thesvc.other/info

Summing up, DNS-based service discovery provides a flexible and generic way to connect to services across the cluster.

You can destroy all the resources created with:

$ kubectl delete pods jumpod

$ kubectl delete svc thesvc

$ kubectl delete rc rcsise

$ kubectl delete ns other

Keep in mind that removing a namespace will destroy every resource inside.