Exploring Virtual Clusters

In this lesson, we will start off by seeking the details of running system-level objects and then explore virtual clusters.

WE'LL COVER THE FOLLOWING

Seeking for System-Level Objects

Almost all of the system services are running as Kubernetes objects. Kube DNS is a deployment. Minikube Addon Manager, Dashboard, Storage Controller, and nginx Ingress are a few of the system Pods that are currently running in our Minikube cluster.

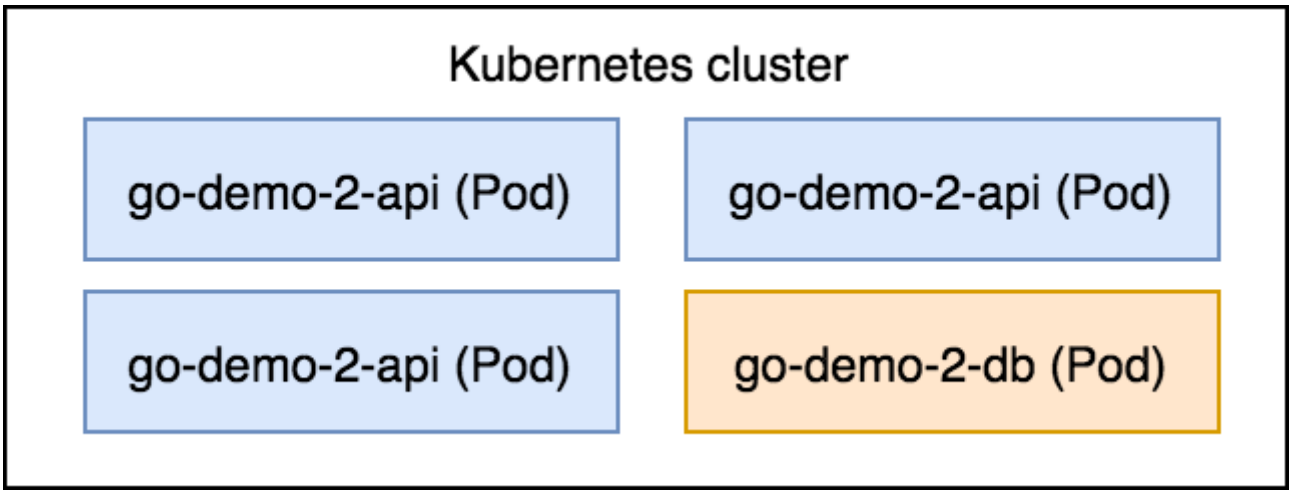
Still, we haven't seen them yet. Even though we executed `kubectl get all` quite a few times, there was not a trace of any of those objects. How can that be? Will we see them now if we list all the objects?

Let's check it out.

```
kubectl get all
```

The **output** shows only the objects we created. There are `go-demo-2` Deployments, ReplicaSets, Services, and Pods. The only system object we can observe is the `kubernetes` Service.

Judging from the current information, if we limit our observations to Pods, our cluster can be described through the following illustration.



The cluster with go-demo-2 Pods

All in all, our cluster runs a mixture of system-level objects and the objects we created, but only the latter is visible. You might be compelled to execute `kubectl get --help` hoping that there is an argument that will allow you to retrieve the information about system level objects.

You might think that these objects are hidden from you by default. That's not the case. They are not hidden. Instead, they *do not live* in the *Namespace* we're looking at.

The Virtual Clusters

Kubernetes uses Namespaces to create virtual clusters. When we created the Minikube cluster, we got **three Namespaces**. In a way, each Namespace is a cluster within the cluster. They provide scope for names.

So far our experience tells us that we cannot have two of the same types of objects with the same name. There cannot be, for example, two deployments named `go-demo-2-api`. However, that rule applies only within a Namespace.

> Inside a cluster, we can have many of same object types with the same name as long as they belong to different Namespaces.

The Default Namespace

So far, we had the impression that we are operating on the level of a Minikube Kubernetes cluster. That was a wrong assumption. All this time we were

inside one Namespace of all the possible Namespaces in the cluster. To be more concrete, all the commands we executed thus far created objects in the `default` Namespace.

Namespaces are so much more than scopes for object names.

- They allow us to split a cluster among different groups of users.
- Each of those Namespaces can have different permissions and resources quotas.
- There are quite a few other things we can do if we combine Namespaces with other Kubernetes services and concepts.

However, we'll ignore permissions, quotas, policies, and other things we did not yet explore. We'll focus on Namespaces alone.

We'll start by exploring the pre-defined Namespaces first.

In the next lesson, we will explore the existing Namespaces in a cluster.