Creating a New Namespace

In this lesson, we will create a new Namespace and switch the kubectl context to it.

WE'LL COVER THE FOLLOWING

Understanding the Scenario

Currently, we're running the release 1.0 of the `go-demo-2` application. We can consider it the production release. Now, let's say that the team in charge of the application just made a new release. They ran unit tests and built the binary. They produced a new Docker image and tagged it as `vfarcic/go-demo-2:2.0`.

What they didn't do is run functional, performance, and other types of tests that require a running application. The new release is still not ready to be deployed to production so we cannot yet execute a rolling update and replace the production release with the new one. We need to finish running the tests, and for that we need the new release running in parallel with the old one.

Exploring the Options

We could, for example, create a new cluster that would be used only for testing purposes. While that is indeed a good option in some situations, in others it might be a waste of resources. Moreover, we'd face the same challenge in the testing cluster. There might be multiple new releases that need to be deployed and tested in parallel.

Another option could be to create a new cluster for each release that is to be

tested. That would create the necessary separation and maintain the freedom we strive for. However, that is slow. Creating a cluster takes time. Even though it might not look like much, wasting ten minutes (if not more) only on that is too much time. Even if you disagree and you think that ten minutes is not that much, such an approach would be too expensive.

Every cluster has a resource overhead that needs to be paid. While the overall size of a cluster affects the resource overhead, the number of clusters affects it even more. It's more expensive to have many smaller clusters than a big one. On top of all that, there is the operational cost. While it is often not proportional to the number of clusters, it still increases.

Having a separate cluster for all our testing needs is not a bad idea. We shouldn't discard it, just as we should consider creating (and destroying) a new cluster for each new release. However, before you start creating new Kubernetes clusters, we'll explore how we might accomplish the same goals with a single cluster and with the help of Namespaces.

Creating a Namespace

First things first. We need to create a new Namespace before we can use it.

```
kubectl create ns testing
kubectl get ns
```

The **output** of the latter command is as follows.

```
NAME              STATUS   AGE
default           Active   4h36m
kube-node-lease   Active   4h36m
kube-public       Active   4h36m
kube-system       Active   4h36m
testing           Active   3s
```

We can see that the new Namespace `testing` was created.

We can continue using the `--namespace` argument to operate within the newly created Namespace. However, writing `--namespace` with every command is tedious. Instead, we'll create a new context.

```
kubectl config set-context testing \
    --namespace testing \

    --cluster minikube \
    --user minikube
```

We created a new context called `testing`. It is the same as the `minikube` context, except that it uses the `testing` Namespace.

```
kubectl config view
```

The **output**, limited to the relevant parts, is as follows.

```
...
contexts:
- context:
    cluster: minikube
    user: minikube
  name: minikube
- context:
    cluster: minikube
    namespace: testing
    user: minikube
  name: testing
...
```

We can see that there are two contexts. Both are set to use the same `minikube` cluster with the same `minikube` user. The only difference is that one does not have the Namespace set, meaning that it will use the `default`. The other has it set to `testing`.

Context Switching

Now that we have two contexts, we can switch to `testing`.

```
kubectl config use-context testing
```

We switched to the `testing` context that uses the Namespace of the same name. From now on, all the `kubectl` commands will be executed within the context of the `testing` Namespace. That is, until we change the context again, or use the `--namespace` argument.

Verification

To be on the safe side, we'll confirm that nothing is running in the newly created Namespace.

```
kubectl get all
```

The **output** shows that `no resources` were `found`.

If we repeat the same command with the addition of the `--namespace=default` argument, we'll see that the `go-demo-2` objects we created earlier are still running.

In the next lesson, we will continue and deploy a new release to this newly created Namespace.