Creating Role Bindings

In this lesson, we will create a Role Binding to let the user have the viewing access to all the objects in our default Namespace.

WE'LL COVER THE FOLLOWING

Creating Role Bindings

Role Bindings bind a User (or a Group, or a Service Account) to a Role (or a Cluster Role). Since John wants more visibility to our cluster, we'll create a Role Binding that will allow him to view (almost) all the objects in the `default` namespace. That should be a good start of our quest to give John just the right amount of privileges.

```
kubectl create rolebinding jdoe \
    --clusterrole view \
    --user jdoe \
    --namespace default \
    --save-config

kubectl get rolebindings
```

We created a Role Binding called `jdoe`. Since the Cluster Role `view` already provides, more or less, what we need, we used it instead of creating a whole new Role.

The **output** of the latter command proved that the new Role Binding `jdoe` was indeed created.

This is a good moment to clarify that a Role Binding does not need to be used

only with a Role, but that it can also be combined with a Cluster Role (as in our example). As the rule of thumb, we define Cluster Roles when we think that they might be used cluster-wide (with Cluster Role Bindings) or in multiple Namespaces (with Role Bindings).

The scope of the permissions is defined with the type of binding, not with the type of role. Since we used Role Binding, the scope is limited to a single Namespace which, in our case, is the `default`.

Looking into the Role Binding

Let's take a look at the details of the newly created Role Binding.

```
kubectl describe rolebinding jdoe
```

```
Name:         jdoe
Labels:       <none>
Annotations:  kubectl.kubernetes.io/last-applied-configuration:
                {"kind":"RoleBinding","apiVersion":"rbac.authorization.k8s.io/v1","metadata":
Role:
  Kind:  ClusterRole
  Name:  view
Subjects:
  Kind  Name  Namespace
  ----  ----  ---------
  User  jdoe
```

We can see that the Role Binding `jdoe` has a single subject with the User `jdoe`. It might be a bit confusing that the Namespace is empty and you might think that the Role Binding applies to all Namespaces. Such an assumption would be false. Remember, a Role Binding is always tied to a specific Namespace, and we just described the one created in the `default` Namespace.

Checking the Scope

The same Role Binding should not be available anywhere else. Let's confirm that.

```
kubectl --namespace kube-system \
    describe rolebinding jdoe
```

We described the Role Binding `jdoe` in the Namespace `kube-system`.

The **output** is as follows.

```
Error from server (NotFound): rolebindings.rbac.authorization.k8s.io "jdoe" not found
```

The Namespace `kube-system` does not have that Role Binding. We never created it.

It might be easier to verify that our permissions are set correctly through the `kubectl auth can-i` command.

```
kubectl auth can-i get pods \
    --as jdoe

kubectl auth can-i get pods \
    --as jdoe --all-namespaces
```

The first command validated whether the user `jdoe` can `get pods` from the `default` Namespace. The answer was `yes`. The second checked whether the Pods could be retrieved from all the Namespaces and the answer was `no`. Currently, John can only see the Pods from the `default` Namespace, and he is forbidden from exploring those from the other Namespaces.

Deleting the Role Binding

From now on, John should be able to view the Pods in the `default` Namespace. However, he works in the same company as we do and we should have more trust in him. Why don't we give him permissions to view Pods in any Namespace? Why not apply the same permissions cluster-wide? Before we do that, we'll delete the Role Binding we created and start over.

```
kubectl delete rolebinding jdoe
```

In the next lesson, we'll change John's `view` permissions so that they are applied across the whole cluster.