Combining Role Bindings with Namespaces

In this lesson, we will combine Role Bindings with Namespaces and create a user-specific Namespace.

WE'LL COVER THE FOLLOWING

The Solution

The new request demanding more freedom provides an excellent opportunity to combine Namespaces with Role Bindings.

We can create a `dev` Namespace and allow a selected group of users to do almost anything in it. That should give developers enough freedom within the `dev` Namespace while avoiding the risks of negatively impacting the resources running in others.

Looking into the Definition

Let’s take a look at the `auth/rb-dev.yml` definition.

```
cat auth/rb-dev.yml
```

The **output** is as follows.

```
apiVersion: v1
kind: Namespace
metadata:
  name: dev

---

apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: dev
  namespace: dev
subjects:
- kind: User
  name: jdoe
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: admin
  apiGroup: rbac.authorization.k8s.io
```

Line 1-4: The first section defines the `dev` Namespace.

Line 8-20: The second section specifies the binding with the same name. Since we're using `RoleBinding` (not `ClusterRoleBinding`), the effects will be limited to the `dev` Namespace. At the moment, there is only one subject (the User `jdoe`). We can expect the list to grow with time.

Line 17-20: `roleRef` uses `ClusterRole` (not `Role`) `kind`. Even though the Cluster Role is available across the whole cluster, the fact that we are combining it with `RoleBinding` will limit it to the specified Namespace.

The Cluster Role `admin` has an extensive set of resources and verbs, and the Users (at the moment only `jdoe`) will be able to do almost anything within the `dev` Namespace.

Creating the Resources

Let's create the new resources.

```
kubectl create -f auth/rb-dev.yml \
    --record --save-config
```

The **output** is as follows.

```
namespace/dev created
rolebinding.rbac.authorization.k8s.io/dev created
```

We can see that the Namespace and the Role Binding were created.

Verification

Let's verify that, for example, `jdoe` can create and delete Deployments.

```
kubectl --namespace dev auth can-i \
    create deployments --as jdoe

kubectl --namespace dev auth can-i \
    delete deployments --as jdoe
```

In both cases, the output was `yes`, confirming that `jdoe` can perform at least `create` and `delete` actions with Deployments. Since we already explored the list of resources defined in the Cluster Role `admin`, we can assume that we'd get the same response if we'd check other operations.

Still, there are a few permissions that are not granted to John. Only the `cluster-admin` role covers all the permissions. The Cluster Role `admin` is very wide, but it does not include all the resources and verbs. We can confirm that with the command that follows.

```
kubectl --namespace dev auth can-i \
    "*" "*" --as jdoe
```

The **output** is `no`, indicating that there are still a few operations forbidden to John within the `dev` Namespace. Those operations are mostly related to cluster administration that is still in our control.

John is happy. He and his fellow developers have a segment of the cluster where they can do almost anything without affecting other Namespaces.

John is a team player, but he'd also like to have space for himself. Now that he knows how easy it was to create a Namespace for developers, he's wondering whether we could generate one only for him. We cannot deny the fact that his new request makes sense.

User Specific Namespace

It should be easy to create his personal Namespace, so why not grant him that wish.

Looking into the Definition

Let's take a look at yet another YAML definition.

```
cat auth/rb-jdoe.yml
```

The **output** is as follows.

```
apiVersion: v1
kind: Namespace
metadata:
  name: jdoe

---

apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: jdoe
  namespace: jdoe
subjects:
- kind: User
  name: jdoe
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: cluster-admin
  apiGroup: rbac.authorization.k8s.io
```

This definition is not much different from the previous one. The important change is that the Namespace is `jdoe`, and that John is likely to be its only user, at least until he decides to add someone else.

By referencing the role `cluster-admin`, he's given full permissions to do whatever he wants within that Namespace. He might deploy something cool and give others permissions to see it. It's his Namespace, and he should be able to do anything he likes inside it.

Creating the Resources

Let's create the new resources.

```
kubectl create -f auth/rb-jdoe.yml \
    --record --save-config
```

Verification

Before we move on, we’ll confirm that John can indeed do anything he likes in the `jdoe` Namespace.

```
kubectl --namespace jdoe auth can-i \
    "*" "*" --as jdoe
```

As expected, the response is `yes`, indicating that John is a god-like figure in his own little galaxy.

In the next lesson, we will grant access to the user to act as a release manager.