Controlling Access in Kubernetes with RBAC

Role-based access control is an important component when it comes to managing a Kubernetes cluster securely. The more users and automated processes there are that need to interface with the Kubernetes API, the more important controlling access becomes.

Task:

You are working for BeeBox, a company that provides regular shipments of bees to customers. The company is in the process of building a Kubernetes-based infrastructure for some of their software.

Your developers frequently request that you provide information from the Kubernetes cluster, so you would like to give them the ability to read data from the cluster but not make any changes to it. Using Kubernetes role-based access control, ensure the dev user can read pod metadata and container logs from any pod in the beebox-mobile namespace.

A kubeconfig file for the dev user has already been created on the server. You can use this file to test your RBAC setup as the dev user like so:

```
kubectl get pods -n beebox-mobile --kubeconfig /home/cloud_user/dev-k8s-config
```

Create a Role for the dev User

We need to attempt to access the cluster as the dev user

- At the end of this lab we will want the dev user to be able to read data about pods

1. Verify access by attempting to list pods as the dev user

```
kubectl get pods -n beebox-mobile --kubeconfig dev-k8s-config
```

I got the “dev-k8s-config” portion because it is in the home directory. To check that just type “ls”

But our developers are not able to view anything

2. So since they can’t view, we need to create a spec file for a new role that we’ll use for our developers

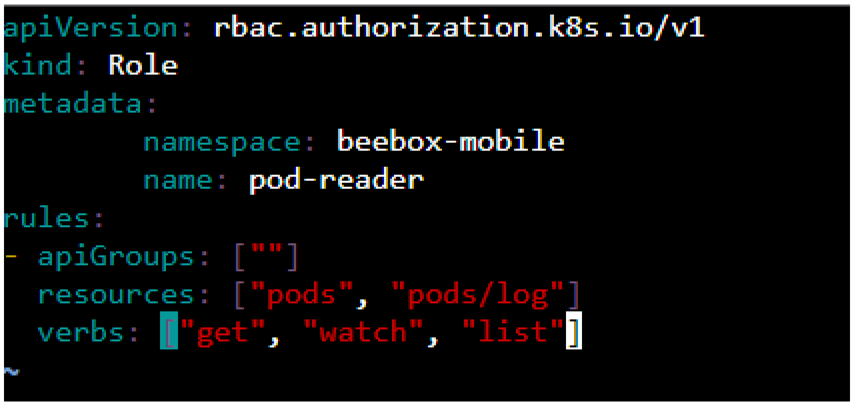
```
vi pod-reader-role.yml
```

Adding this in to the yml file

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: beebox-mobile
  name: pod-reader
rules:
- apiGroups: [""]
  resources: ["pods", "pods/log"]
  verbs: ["get", "watch", "list"]
```

^ Copy and paste may not work, so it may be best to type this out.

Figure 1-1



```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
        namespace: beebox-mobile
        name: pod-reader
rules:
- apiGroups: [""]
  resources: ["pods", "pods/log"]
  verbs: ["get", "watch", "list"]
~
```

^ This is what the instructor uses for any role. Once the basic settings are in place we need to implement the actual rules.
The resources we have set "pods" and "pods/log" the developers have access to that.
Also won't allow the developers to make changes to the pods by inputting the "get,watch,list" listed above in the command section.
- But since they can't make any changes any pods, they can still view and retrieve information about the pods, and pod logs

3. So once that is established, we need to go ahead and create that role with kubectl apply -f

```
kubectl apply -f pod-reader-role.yml
```

Bind the Role to the dev User and Verify Your Setup Works

Now that we have the role created we need to make sure the dev user can actually use that role, so we need to bind that role to the dev user.

4. We need to create this YAML spec file called "pod-reader-rolebinding.yml"

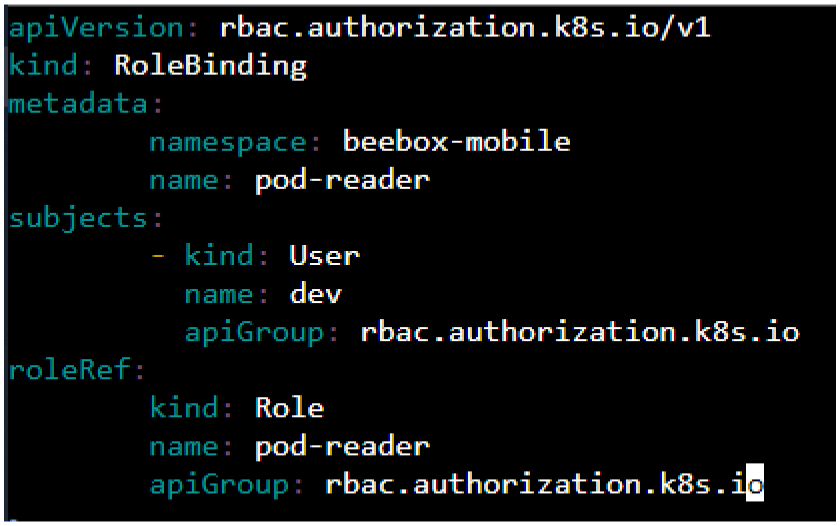
```
vi pod-reader-rolebinding.yml
```

5. And next we need to add the following to the yml file we just created

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: pod-reader
  namespace: beebox-mobile
subjects:
- kind: User
  name: dev
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: pod-reader
  apiGroup: rbac.authorization.k8s.io
```

^ Copy and paste may not work, so it may be best to type this out.

Figure 1-2



```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
        namespace: beebox-mobile
        name: pod-reader
subjects:
        - kind: User
          name: dev
          apiGroup: rbac.authorization.k8s.io
roleRef:
        kind: Role
        name: pod-reader
        apiGroup: rbac.authorization.k8s.io
```

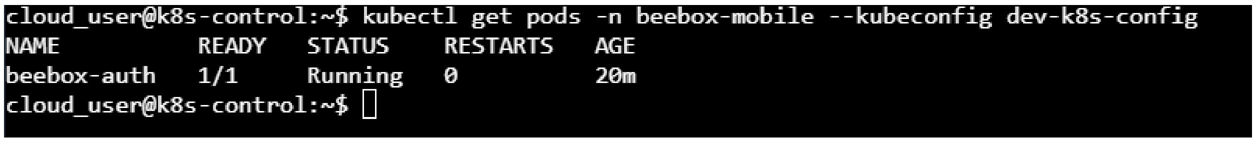
6. Once that is done create the RoleBinding (Similar to step 3)

kubectl apply -f pod-reader-rolebinding.yml

^ Copy and paste may not work, so it may be best to type this out.

7. So now we should be able to view the list of pods (Same command from step 1, by verifying)

Figure 1-3



```
cloud_user@k8s-control:~$ kubectl get pods -n beebox-mobile --kubeconfig dev-k8s-config
NAME          READY   STATUS    RESTARTS   AGE
beebox-auth   1/1     Running   0          20m
cloud_user@k8s-control:~$
```

8. See if dev user can read the logs

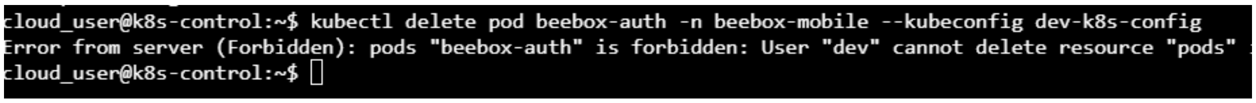
kubectl logs beebox-auth -n beebox-mobile --kubeconfig dev-k8s-config

This just pops up "Auth processing…"

9. So lets see if we can delete this pod as a dev user. Because remember we set those permissions under the verbs section in the yml file.

kubectl delete pod beebox-auth -n beebox-mobile --kubeconfig dev-k8s-config

Figure 1-4



```
cloud_user@k8s-control:~$ kubectl delete pod beebox-auth -n beebox-mobile --kubeconfig dev-k8s-config
Error from server (Forbidden): pods "beebox-auth" is forbidden: User "dev" cannot delete resource "pods"
cloud_user@k8s-control:~$
```

^ This is what we want, so we have completed the task of setting up our developer rule. That way developers can retrieve information about pods in that beebox-mobile namespace.