Exercise 6.3: Working with ServiceAccounts

We can use `ServiceAccounts` to assign cluster roles, or the ability to use particular HTTP verbs. In this section we will create a new `ServiceAccount` and grant it access to view secrets.

1. Begin by viewing secrets, both in the default namespace as well as all.

```
student@cp:~/app2$ cd
```

```
student@cp:~$ kubectl get secrets
```

```
NAME                  TYPE                                  DATA   AGE
lfsecret              Opaque                                1      6m5s
```

```
student@cp:~$ kubectl get secrets --all-namespaces
```

```
NAMESPACE     NAME                     TYPE                                  DATA   AGE
default       lfsecret                 Opaque                                1      69s
kube-system   bootstrap-token-j6r4vk   bootstrap.kubernetes.io/token         7      58m

<output_omitted>
```

2. We can see that each agent uses a secret in order to interact with the API server. We will create a new `ServiceAccount` which will have access.

```
student@cp:~$ vim serviceaccount.yaml
```

serviceaccount.yaml

```
1 apiVersion: v1
2 kind: ServiceAccount
3 metadata:
4  name: secret-access-sa
```

```
student@cp:~$ kubectl create -f serviceaccount.yaml
```

```
serviceaccount/secret-access-sa created
```

```
student@cp:~$ kubectl get serviceaccounts
```

```
NAME              SECRETS   AGE
default           0         1d17h
secret-access-sa  0         34s
```

3. Now we will create a `ClusterRole` which will list the actual actions allowed cluster-wide. We will look at an existing role to see the syntax.

```
student@cp:~$ kubectl get clusterroles
```

```
NAME                                                   AGE
admin                                                  1d17h
cilium                                                 1d17h
```

```
cilium-operator                                        1d17h
cluster-admin                                          1d17h
<output_omitted>
```

4. View the details for the `admin` and compare it to the `cluster-admin`. The `admin` has particular actions allowed, but `cluster-admin` has the meta-character '*' allowing all actions.

```
student@cp:~$ kubectl get clusterroles admin -o yaml
```

```
<output_omitted>
```

```
student@cp:~$ kubectl get clusterroles cluster-admin -o yaml
```

```
<output_omitted>
```

5. Using some of the output above, we will create our own file.

```
student@cp:~$ vim clusterrole.yaml
```

clusterrole.yaml

```
1  apiVersion: rbac.authorization.k8s.io/v1
2  kind: ClusterRole
3  metadata:
4   name: secret-access-cr
5  rules:
6  - apiGroups:
7    - ""
8    resources:
9    - secrets
10   verbs:
11   - get
12   - list
```

6. Create and verify the new `ClusterRole`.

```
student@cp:~$ kubectl create -f clusterrole.yaml
```

```
clusterrole.rbac.authorization.k8s.io/secret-access-cr created
```

```
student@cp:~$ kubectl get clusterrole secret-access-cr -o yaml
```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  creationTimestamp: 2018-10-18T19:27:24Z
  name: secret-access-cr
<output_omitted>
```

7. Now we bind the role to the account. Create another YAML file which uses `roleRef:`:

```
student@cp:~$ vim rolebinding.yaml
```

rolebinding.yaml

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
 name: secret-rb
subjects:
- kind: ServiceAccount
  name: secret-access-sa
roleRef:
 kind: ClusterRole
 name: secret-access-cr
 apiGroup: rbac.authorization.k8s.io
```

8. Create the new `RoleBinding` and verify.

```
student@cp:~$ kubectl create -f rolebinding.yaml
```

```
rolebinding.rbac.authorization.k8s.io/secret-rb created
```

```
student@cp:~$ kubectl get rolebindings
```

```
NAME        AGE
secret-rb   17s
```

9. View the `secondapp` pod and **grep** for the current `serviceAccount`. Note that it uses the default account.

```
student@cp:~$ kubectl get pod secondapp -o yaml |grep serviceAccount
```

```
serviceAccount: default
serviceAccountName: default
    - serviceAccountToken:
```

10. Edit the `second.yaml` file and add the use of the `serviceAccount`.

```
student@cp:~$ vim $HOME/app2/second.yaml
```

second.yaml

```
....
  name: secondapp
spec:
  serviceAccountName: secret-access-sa  #<-- Add this line
  securityContext:
    runAsUser: 1000
....
```

11. We will delete the `secondapp` pod if still running, then create it again. Note that the `serviceAccount` is no longer the `default`.

```
student@cp:~$ kubectl delete pod secondapp ; kubectl create -f $HOME/app2/second.yaml
```

```
pod "secondapp" deleted
pod/secondapp created
```

```
student@cp:~$ kubectl get pod secondapp -o yaml | grep serviceAccount
```

```
  serviceAccount: secret-access-sa
  serviceAccountName: secret-access-sa
      - serviceAccountToken:
```