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@ckad-1:~/app2$ cd

student@ckad-1:~$ kubectl get secrets
NAME                  TYPE                                  DATA   AGE
default-token-c4rdg   kubernetes.io/service-account-token   3      4d16h
lfsecret              Opaque                                1      6m5s

student@ckad-1:~$ kubectl get secrets --all-namespaces
NAMESPACE     NAME
 TYPE                                  DATA   AGE
default       default-token-c4rdg
 kubernetes.io/service-account-token   3      4d16h
kube-public   default-token-zqzbg
 kubernetes.io/service-account-token   3      4d16h
kube-system   attachdetach-controller-token-wxzvc
 kubernetes.io/service-account-token   3      4d16h
<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@ckad-1:~$ vim serviceaccount.yaml
```

serviceaccount.yaml

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

```
student@ckad-1:~$ kubectl create -f serviceaccount.yaml
serviceaccount/secret-access-sa created

student@ckad-1:~$ kubectl get serviceaccounts
NAME              SECRETS   AGE
default           1         1d17h
secret-access-sa  1         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@ckad-1:~$ kubectl get clusterroles
```

```
NAME                                                   AGE
admin                                                  1d17h
calico-cni-plugin                                      1d17h
calico-kube-controllers                                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@ckad-1:~$ kubectl get clusterroles admin -o yaml
<output_omitted>

student@ckad-1:~$ kubectl get clusterroles cluster-admin -o yaml
<output_omitted>
```

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

```
student@ckad-1:~$ vim clusterrole.yaml
```

clusterrole.yaml

```
1  apiVersion: rbac.authorization.k8s.io/v1beta1
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@ckad-1:~$ kubectl create -f clusterrole.yaml
clusterrole.rbac.authorization.k8s.io/secret-access-cr created

student@ckad-1:~$ 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@ckad-1:~$ vim rolebinding.yaml
```

rolebinding.yaml

```
1  apiVersion: rbac.authorization.k8s.io/v1beta1
2  kind: RoleBinding
```

```yaml
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@ckad-1:~$ kubectl create -f rolebinding.yaml
rolebinding.rbac.authorization.k8s.io/secret-rb created

student@ckad-1:~$ kubectl get rolebindings
NAME        AGE
secret-rb   17s
```

9. View the `secondapp` pod and **grep** for secret settings. Note that it uses the default settings.

```
student@ckad-1:~$ kubectl describe pod secondapp |grep -i secret
   /var/run/secrets/kubernetes.io/serviceaccount from
default-token-c4rdg (ro)
   Type:        Secret (a volume populated by a Secret)
   SecretName:  lfsecret
   Type:        Secret (a volume populated by a Secret)
   SecretName:  default-token-c4rdg
```

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

```
student@ckad-1:~$ vim ~/app2/second.yaml
```

second.yaml

```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. View what the secret is by default.

```
student@ckad-1:~$ kubectl delete pod secondapp ; kubectl create -f ~/app2/second.yaml
pod "secondapp" deleted
pod/secondapp created

student@ckad-1:~$ kubectl describe pod secondapp | grep -i secret
    /var/run/secrets/kubernetes.io/serviceaccount from
 secret-access-sa-token-wd7vm (ro)
  secret-access-sa-token-wd7vm:
   Type:        Secret (a volume populated by a Secret)
   SecretName:  secret-access-sa-token-wd7vm
```