Today Lesson – Tuesday April 12, 2022

Service User Account – Managed by Kubernetes

RBAC – Role Based Access Control

- Role
- Cluster role
- Rolebinding
- Clusterrolebinding

Creating a service user account in Kubernetes - it called services because it is managed by Kubernetes.

Repeat the entire steps but this time create a static user, a clusterrole, and clusterrolebidning connected to the user.

Kubernetes service accounts let you give an identity to your Pods, which can be used to: **Authenticate Pods to the Kubernetes API server, allowing the Pods to read and manipulate Kubernetes API objects (for example, a CI/CD pipeline that deploys applications to your cluster). Now let work on Service account.**

We said in the previous sessions that serviceaccounts are used to authenticate processes, and machines that need to access the Kubernetes resources.

These machines include

1. Jenkins and jenkins agent configured in the cloud (Kubernetes)
2. ELK stack
3. Prometheus operator & Grafana
4. Portworx
5. PostgressSQL

Let create account for a new tester DevOps Engineer who just joined our team and need to access the Kubernetes resources.

Let start by creating a namespace called **Test**

Username: 3527585463
Password: HIPIPRAISE

echo -n "3527585463" | base64

echo -n " HIPIPRAISE " | base64

```
apiVersion: v1
kind: Secret
metadata:
  name: lucia-secret
type: Opaque
data:
  username: bHVjaWE=
  password: cGFzcw==
```

newdevopaccount.yml

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: 3527585463-secret
  namespace: test
type: Opaque
data:
  username: MzUyNzU4NTQ2Mw==
  password: IEhJUElQUkFJU0Ug
```

Create the secret

```
josh_kidfileapp@cloudshell:~ (prime-motif-337422)$ kubectl create -f  newdevopaccount.yml -n test
secret/3527585463-secret created
josh_kidfileapp@cloudshell:~ (prime-motif-337422)$ kubectl get secret
NAME                  TYPE                                  DATA   AGE
default-token-vdcm4   kubernetes.io/service-account-token   3      56m
josh_kidfileapp@cloudshell:~ (prime-motif-337422)$ kubectl get secret -n test
NAME                  TYPE                                  DATA   AGE
3527585463-secret     Opaque                                2      45s
default-token-ms8nn   kubernetes.io/service-account-token   3      4m3s
```

Get the secret created in the specific namespace

```
josh_kidfileapp@cloudshell:~ (prime-motif-337422)$ kubectl describe secret 3527585463-secret -n test
Name:         3527585463-secret
Namespace:    test
Labels:       <none>
Annotations:  <none>

Type:  Opaque

Data
====
password:  12 bytes
username:  10 bytes
```

Now we have created a static user account - it is important to know that user account is global.

Although we have created an account, we are yet to give the account the **ROLE**

Now let create a ROLE for the account

Note that pods (and many other resources) have an empty APIGROUP. This is because they are part of the core API group.

The RBAC concept. The actions on a resource that a role uses in its rules are the so-called verbs, such as the following: **get, list (read-only) create, update, patch, delete, deletecollection (read-write)**

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: default
  name: pod-reader
rules:
- apiGroups: [""] # "" indicates the core API group
  resources: ["pods"]
  verbs: ["get", "watch", "list"]
```

Create the role

```
josh_kidfileapp@cloudshell:~ (prime-motif-337422)$ cat role.yml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: test
  name: pod-reader
rules:
- apiGroups: [""] # "" indicates the core API group
  resources: ["pods"]
  verbs: ["get", "watch", "list"]
josh_kidfileapp@cloudshell:~ (prime-motif-337422)$ kubectl create -f role.yml -n test
role.rbac.authorization.k8s.io/pod-reader created
josh_kidfileapp@cloudshell:~ (prime-motif-337422)$
```

Check whether the role has been created

```
josh_kidfileapp@cloudshell:~ (prime-motif-337422)$ kubectl get roles -n test
NAME         CREATED AT
pod-reader   2022-04-12T22:16:50Z
josh_kidfileapp@cloudshell:~ (prime-motif-337422)$
```

Now it is time to create a Role binding that is going to bind the role with the user account that we just created for the new tester DevOps

```
apiVersion: rbac.authorization.k8s.io/v1
# This role binding allows " 3527585463-secret" to read pods in the "test" namespace.
# You need to already have a Role named "pod-reader" in that namespace.
kind: RoleBinding
metadata:
  name: read-podsbinding
  namespace: test
subjects:
# You can specify more than one "subject"
- kind: User # Group or ServiceAccount
  name: 3527585463-secret

 # "name" is case sensitive
  apiGroup: rbac.authorization.k8s.io
roleRef:
  # "roleRef" specifies the binding to a Role / ClusterRole
  kind: Role #this must be Role or ClusterRole
```

name: pod-reader *# this must match the name of the Role or ClusterRole you wish to bind to*
apiGroup: rbac.authorization.k8s.io

Create the role binding

newdevopaccount.yml | role.yml | rolebinding.yml

rolebinding.yml

```yaml
apiVersion: rbac.authorization.k8s.io/v1
# This role binding allows " 3527585463-secret" to read pods in the "test" namespace.
# You need to already have a Role named "pod-reader" in that namespace.
kind: RoleBinding
metadata:
  name: read-podsbinding
  namespace: test
subjects:
# You can specify more than one "subject"
- kind: User
  name: 3527585463-secret
 # "name" is case sensitive
  apiGroup: rbac.authorization.k8s.io
roleRef:
  # "roleRef" specifies the binding to a Role / ClusterRole
  kind: Role #this must be Role or ClusterRole
  name: pod-reader # this must match the name of the Role or ClusterRole you wish to bind to
  apiGroup: rbac.authorization.k8s.io

```

Problems | prime-motif-337422

```
josh_kidfileapp@cloudshell:~ (prime-motif-337422)$ kubectl create -f rolebinding.yml -n test
rolebinding.rbac.authorization.k8s.io/read-podsbinding created
josh_kidfileapp@cloudshell:~ (prime-motif-337422)$
```

Check whether the role binding have been created

```
josh_kidfileapp@cloudshell:~ (prime-motif-337422)$ kubectl get rolebinding -n test
NAME               ROLE              AGE
read-podsbinding   Role/pod-reader   3m53s
josh_kidfileapp@cloudshell:~ (prime-motif-337422)$
```

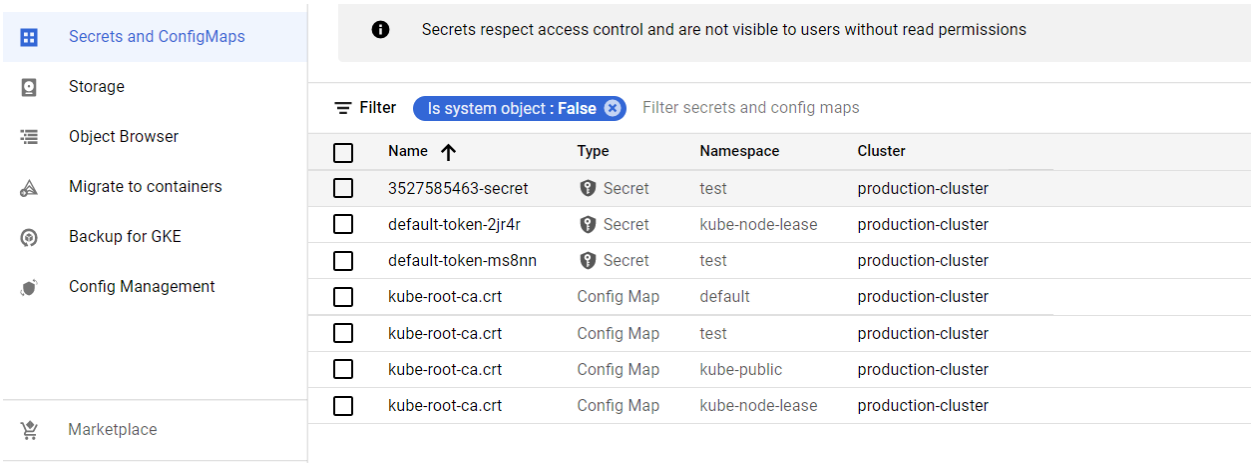
Now let go and get the token associated with the user account that we just created

```
josh_kidfileapp@cloudshell:~ (prime-motif-337422)$ kubectl describe secret $(kubectl describe secret 3527585463-secret --namespace=test |
grep Token | awk '{print $2}') --namespace=test
Name:         3527585463-secret
Namespace:    test
Labels:       <none>
Annotations:  <none>
```

```
Labels:       <none>
Annotations:  <none>

Type:  Opaque

Data
====
password:  12 bytes
username:  10 bytes


Name:         default-token-ms8nn
Namespace:    test
Labels:       <none>
Annotations:  kubernetes.io/service-account.name: default
              kubernetes.io/service-account.uid: 084c2b67-144b-48d2-b237-3aaca12f4336

Type:  kubernetes.io/service-account-token

Data
====
ca.crt:     1509 bytes
namespace:  4 bytes
```

```
Type:  kubernetes.io/service-account-token

Data
====
ca.crt:     1509 bytes
namespace:  4 bytes
token:      eyJhbGciOiJSUzI1NiIsImtpZCI6IjZHTkV3a3Nrc2JSS0JuWXVWbUdsalJGaGRTeUdIdE9PRG5KcC0wUXFac1kifQ.eyJpc3MiOiJrdWJlcm5ldGVzL3NlcnZpY2VhY2NvdW50Iiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9uYW1lc3BhY2UiOiJ0ZXN0Iiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9zZWNyZXQubmFtZSI6ImRlZmF1bHQtdG9rZW4tbXM4bm4iLCJrdWJlcm5ldGVzLmlvL3NlcnZpY2VhY2NvdW50L3NlcnZpY2UtYWNjb3VudC5uYW1lIjoiZGVmYXVsdCIsImt1YmVybmV0ZXMuaW8vc2VydmljZWFjY291bnQvc2VydmljZS1hY2NvdW50LnVpZCI6IjA4NGMyYjY3LTE0NGItNDhkMi1iMjM3LTNhYWNhMTJmNDMzNiIsInN1YiI6InN5c3RlbTpzZXJ2aWNlYWNjb3VudDp0ZXN0OmRlZmF1bHQifQ.qaOb-REHM-CE3wboOc5vXaCTBtXckjZrZZEOcLPtOJ3ZgKm21hmBYO4t9wZnPWL-3vkIkSg9AjID84YGMdouPLaTFPu2IJwGsY3dCeI86A1L4wMvzNKGdD0Ql7VyIx4y
FFpw0bnPZoteHtnrfsZVDoXFwzk3x3gqx5vn6AfA7plU_I6nyailI_Z0PtQ9As3kLUllFPEXb2Er7V55RJy5m2f5bjQmGXnIOfZYE8D1DVaX15NLWnHVorvkte-g3ooDe-MsZcmCDl
iTHQsMVlKB93SJ3MUpUS7UV6XnGB16QMTTG85mwaOlAjGObxfc2McmmX0u05cYTcWxLXmfgJMcgA
```

Let us now find out all the resources we have created in the config map and secrets



Next Lesson – Thursday – April 14, 2022

Users in Kubernetes

All Kubernetes clusters have two categories of users: service accounts managed by Kubernetes, and normal users.

It is assumed that a cluster-independent service manages normal users in the following ways:

- an administrator distributing private keys
- a user store like Keystone or Google Accounts
- a file with a list of usernames and passwords