Lab6 – RBAC

Brahim HAMDI

Introduction

Les RBAC contrôlent les accès dans Kubernetes, elles permettent de donner des droits à la fois à des utilisateurs humains mais aussi directement à des pods.

Nous allons voir les différentes options offertes par Kubernetes :

- Création d'un utilisateur
- Role / RoleBinding
- Accès depuis un Pod

Création d'un utilisateur

1. La notion d'utilisateur n'existe pas vraiment sur Kubernetes. Un utilisateur généralement représenté par un certificat x509 dont le CommonName est le nom de l'utilisateur. Pour que ce certificat soit valide, il doit avoir été signé par l'autorité de certification interne à Kubernetes.
    - Pour créer ce certificat, on va utiliser une fonction de kubeadm. Pour cela, connectez vous au nœud master et lancez les commandes suivantes :

kubectl get cm -n kube-system kubeadm-config -o jsonpath='{.data.ClusterConfiguration }' > cluster-configuration.yaml

sudo kubeadm kubeconfig user --client-name red --config=cluster-configuration.yaml > kubeconfig

Le *kubeconfig* généré contient les *crédentials* pour un nouveau user ***red***.

- Dans le fichier *kubeconfig*, remplacez l'IP de *eth0* (*10.0.2.15*) par celle de *eth1* (*192.168.56.10*) :

*server: https://**192.168.56.10**:6443*

- Listez les pods avec ce kubeconfig :

export KUBECONFIG=kubeconfig
kubectl get pod

Que constatez-vous ? Pourquoi ?

- Tapez la commande suivante pour récupérer la config par défaut, puis listez les pods :

unset KUBECONFIG
kubectl get pod

Role et RoleBinding

2. Nous allons donner des droits à l'utilisateur ***red***.
   - Créez et appliquez le fichier *role.yml* suivant:

```
---
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: access-pod-svc
rules:
- apiGroups: [""]
  resources: ["pods", "services"]
  verbs: ["get", "list"]
```

Ce Role donne les droits de get et list sur les Pods et les Services.

3. Maintenant on va lier ce Role à l'utilisateur ***red*** :
   - Créez et appliquez le fichier *rolebinding.yaml* suivant:

```
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: rb-red
subjects:
- kind: User
  name: red
roleRef:
  kind: Role
  name: access-pod-svc
  apiGroup: rbac.authorization.k8s.io
```

   - Pour le moment, on peut accéder au cluster et lister tous les objets sans erreurs :

```
kubectl get pod
kubectl get services
kubectl get deploy
```

4. Créons deux pods comme exemple :
   - Créez et appliquez le fichier *pods-color.yml* suivant :

```
apiVersion: v1
kind: Pod
metadata:
  name: blue
spec:
  containers:
  - name: web
    image: particule/helloworld
---
```

```
apiVersion: v1
kind: Pod
metadata:
  name: green
spec:
  containers:
  - name: web
    image: particule/helloworld
```

- Modifiez et ré-appliquez le Role :

```
---
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: access-pod-svc
rules:
- apiGroups: [""]
  resources: ["pods"]
  resourceNames: ["blue"]
  verbs: ["get", "list"]
```

- Testons le comportement en tant que l'utilisateur ***red*** :

```
export KUBECONFIG=kubeconfig
kubectl get pods
kubectl get pods blue
kubectl get pods green
```

- Que constatez vous ?

- Récupérez la config par défaut.

```
unset KUBECONFIG
```