Lab4 – ConfigMap et les Secrets

Brahim HAMDI

Introduction

Dans ce Lab, nous allons utiliser les objets Secrets et ConfigMap :

- Créer et utiliser une ConfigMap en variable d'environnement
- Utiliser une ConfigMap en tant que volume
- Créer et utiliser un Secret en tant que secret
- Utiliser un Secret en tant que volume

ConfigMap

1. On va définir deux données *log.level* et *log.location* dans une *ConfigMap*.
   - Créer et appliquer le fichier *configmap.yaml* :

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: log-config
data:
  log.level: WARNING
  log.location: LOCAL
```

```
brahim@Training:~/Lab5$ kubectl apply -f configmap.yaml
configmap/log-config created
brahim@Training:~/Lab5$
brahim@Training:~/Lab5$ kubectl get cm
NAME               DATA   AGE
kube-root-ca.crt   1      3d18h
log-config         2      5s
brahim@Training:~/Lab5$
brahim@Training:~/Lab5$ kubectl describe cm log-config
Name:         log-config
Namespace:    default
Labels:       <none>
Annotations:  <none>

Data
====
log.level:
----
WARNING
log.location:
----
LOCAL

BinaryData
====

Events:  <none>
brahim@Training:~/Lab5$
```

2. Maintenant on va utiliser les données de cette *configmap* comme des variables d'environnement dans un déploiement.

- Créer et appliquer le déploiement suivant (fichier *deploy-cm-env.yaml*):

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: helloworld-cm-env
  labels:
    app: helloworld-cm-env
spec:
  replicas: 1
  selector:
    matchLabels:
      app: helloworld-cm-env
  template:
   metadata:
      labels:
        app: helloworld-cm-env
    spec:
     containers:
     - name: helloworld
       image: particule/helloworld:1.0.0
       ports:
       - containerPort: 80
       env:
       - name: LOG_LEVEL
         valueFrom:
          configMapKeyRef:
           name: log-config
           key: log.level
       - name: LOG_LOCATION
         valueFrom:
          configMapKeyRef:
           name: log-config
           key: log.location
```

```
brahim@Training:~/Lab5$ kubectl  apply -f deploy-cm-env.yaml
deployment.apps/helloworld-cm-env created
brahim@Training:~/Lab5$
brahim@Training:~/Lab5$ kubectl get all
NAME                                      READY   STATUS              RESTARTS   AGE
pod/helloworld-cm-env-7856b9786d-d6z6w    1/1     Terminating         0          48s
pod/helloworld-cm-env-7856b9786d-ljbb9    0/1     ContainerCreating   0          5s

NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   32m

NAME                                 READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/helloworld-cm-env    0/1     1            0           5s

NAME                                            DESIRED   CURRENT   READY   AGE
replicaset.apps/helloworld-cm-env-7856b9786d    1         1         0       5s
brahim@Training:~/Lab5$
```

- En *kubectl exec*, regarder les variables d'environnement disponibles dans le conteneur.

```
brahim@Training:~/Lab5$ kubectl exec -ti pod/helloworld-cm-env-7856b9786d-ljbb9 -- sh
/ #
/ # echo $LOG_LEVEL
WARNING
/ # echo $LOG_LOCATION
LOCAL
/ # brahim@Training:~/Lab5$
brahim@Training:~/Lab5$
```

3. Dans cette partie, on va utiliser *ConfigMap* en tant que volume.

◦ Supprimez tous l'ancien déploiement. Ensuite créez et appliquez le déploiement suivant (fichier *deploy-cm-vol.yaml*) :

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: helloworld-cm-vol
  labels:
    app: helloworld-cm-vol
spec:
  replicas: 1
  selector:
    matchLabels:
      app: helloworld-cm-vol
  template:
    metadata:
      labels:
        app: helloworld-cm-vol
    spec:
      containers:
      - name: helloworld
        image: particule/helloworld:1.0.0
        ports:
        - containerPort: 80
        volumeMounts:
        - name: config-volume
          mountPath: /etc/config
      volumes:
      - name: config-volume
        configMap:
          name: log-config
```

◦ Observez le comportement à l'intérieur du conteneur.

```
brahim@Training:~/Lab5$ kubectl  apply -f deploy-cm-vol.yaml
deployment.apps/helloworld-cm-vol unchanged
brahim@Training:~/Lab5$
brahim@Training:~/Lab5$ kubectl get all
NAME                                      READY   STATUS    RESTARTS   AGE
pod/helloworld-cm-vol-5dbd6c77db-c6t64    1/1     Running   0          27s

NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   7m5s

NAME                                  READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/helloworld-cm-vol     1/1     1            1           27s

NAME                                             DESIRED   CURRENT   READY   AGE
replicaset.apps/helloworld-cm-vol-5dbd6c77db     1         1         1       27s
brahim@Training:~/Lab5$
brahim@Training:~/Lab5$ kubectl exec -ti pod/helloworld-cm-vol-5dbd6c77db-c6t64 -- sh
/ # ls /etc/config/
log.level     log.location
/ #
/ # cat /etc/config/log.level
WARNING/ #
/ # brahim@Training:~/Lab5$
```

Les Secrets

Les *Secrets* fonctionnent exactement de la même manière que les *ConfigMap* à l'exception qu'ils sont stockés encodés en *base64*.

4. Les valeurs stockées dans un *Secret* au format yaml doivent être encodées au préalable.

◦ Encoder les deux mots "admin" et "password".

echo -n "admin" | base64
echo -n "password" | base64

```
brahim@Training:~/Lab5$ echo -n "admin" | base64
YWRtaW4=
brahim@Training:~/Lab5$ echo -n "password" | base64
cGFzc3dvcmQ=
brahim@Training:~/Lab5$
```

◦ Créer et appliquer le contenu suivant du fichier *secret.yaml* :

```
apiVersion: v1
kind: Secret
metadata:
  name: my-secret
```

```
type: Opaque
data:
  username: YWRtaW4=
  password: cGFzc3dvcmQ=
```

```
brahim@Training:~/Lab5$ vim secret.yaml
brahim@Training:~/Lab5$ kubectl  apply -f secret.yaml
secret/my-secret created
brahim@Training:~/Lab5$
brahim@Training:~/Lab5$ kubectl get secret
NAME        TYPE     DATA   AGE
my-secret   Opaque   2      6s
brahim@Training:~/Lab5$
brahim@Training:~/Lab5$ kubectl describe secret my-secret
Name:         my-secret
Namespace:    default
Labels:       <none>
Annotations:  <none>

Type:  Opaque

Data
====
username:  5 bytes
password:  8 bytes
brahim@Training:~/Lab5$
```

5. Maintenant on va utiliser les données de ces données comme des variables d'environnement dans un déploiement.

◦ Modifier le déploiement utilisé dans la partie de ConfigMap en ajoutant une variable d'environnement à partir d'un *Secret*, puis appliquer (fichier deploy-secret-env.yaml) :

```
- name: SECRET_USERNAME
  valueFrom:
    secretKeyRef:
      name: my-secret
      key: username
```

```
brahim@Training:~/Lab5$ kubectl apply -f deploy-secret-env.yaml
deployment.apps/helloworld-cm-env created
brahim@Training:~/Lab5$
brahim@Training:~/Lab5$ kubectl get all
NAME                                      READY   STATUS              RESTARTS   AGE
pod/helloworld-cm-env-78cd8d55c8-db47x    0/1     ContainerCreating   0          7s

NAME                 TYPE        CLUSTER-IP    EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.96.0.1     <none>        443/TCP   2m3s

NAME                                  READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/helloworld-cm-env     0/1     1            0           7s

NAME                                             DESIRED   CURRENT   READY   AGE
replicaset.apps/helloworld-cm-env-78cd8d55c8     1         1         0       7s
```

◦ Observer ensuite le comportement dans le pod avec *kubectl exec*.

```
brahim@Training:~/Lab5$ kubectl exec -ti pod/helloworld-cm-env-78cd8d55c8-db47x -- sh
/ # echo $SECRET_USERNAME
admin
/ # brahim@Training:~/Lab5$
brahim@Training:~/Lab5$
```

6. Il est possible aussi d'utiliser le Secret en tant que volume.

◦ Reprenez le déploiement utilisés pour les *ConfigMap* et ajoutez un volume à partir d'un secret (fichier *deploy-secret-vol.yaml*).

```yaml
spec:
  containers:
  - name: helloworld
    image: particule/helloworld
    ports:
    - containerPort: 80
    volumeMounts:
    - name: config-volume
      mountPath: /etc/config
    - name: secret-volume
      mountPath: /etc/secret
  volumes:
  - name: config-volume
    configMap:
      name: log-config
  - name: secret-volume
    secret:
      secretName: my-secret
```

◦ Appliquez et testez

```
brahim@Training:~/Lab5$ kubectl apply -f deploy-secret-vol.yaml
deployment.apps/helloworld-cm-vol unchanged
brahim@Training:~/Lab5$ kubectl get all
NAME                                     READY   STATUS    RESTARTS   AGE
pod/helloworld-cm-vol-7d78476cf-qdtwc    1/1     Running   0          99s

NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   2m44s

NAME                                    READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/helloworld-cm-vol       1/1     1            1           99s

NAME                                               DESIRED   CURRENT   READY   AGE
replicaset.apps/helloworld-cm-vol-7d78476cf        1         1         1       99s
brahim@Training:~/Lab5$
brahim@Training:~/Lab5$ kubectl exec -ti pod/helloworld-cm-vol-7d78476cf-qdtwc -- sh
/ # ls /etc/secret
password  username
/ # cat /etc/secret/password
password/ #
/ # brahim@Training:~/Lab5$
```

7. Maintenant on va changer le mot de passe du Secret et vérifier si les pods du déploiement sont à jour.

◦ Changez la valeur du mot de passe de « password » à « newpassword » et réappliquez le *Secret* (fichier *newsecret.yaml*). Vérifiez le nouveau paramètre dans le pod.

```
brahim@Training:~/Lab5$ echo -n "newpassword" | base64
bmV3cGFzc3dvcmQ=
brahim@Training:~/Lab5$ vim newsecret.yaml
brahim@Training:~/Lab5$ kubectl apply -f newsecret.yaml
secret/my-secret unchanged
brahim@Training:~/Lab5$ kubectl get pod
NAME                                  READY   STATUS    RESTARTS   AGE
helloworld-cm-vol-7d78476cf-qdtwc     1/1     Running   0          11m
brahim@Training:~/Lab5$ kubectl exec -ti pod/helloworld-cm-vol-7d78476cf-qdtwc -- sh
/ # cat /etc/secret/password
newpassword/ # brahim@Training:~/Lab5$
brahim@Training:~/Lab5$
```