Utilisation d'un ConfigMap pour la configuration d'un reverse proxy

1. Context

Dans cette mise en pratique nous allons voir l'utilisation de l'objet ConfigMap pour fournir un fichier de configuration à un reverse proxy très simple que nous allons baser sur *nginx*.

Nous allons configurer ce proxy de façon à ce que les requètes reçues sur le endpoint */whoami* soient forwardées sur un service nommé *whoami*, tournant également dans le cluster. Ce service expose le endpoint / et renvoie simplement le nom du container qui a traité la requète.

1. Création de l'application whoami

La spécification suivante définie un Pod, contenant un unique container basé sur l'image *lucj/whoami*, ainsi qu'un service de type *ClusterIP* dont le rôle est d'exposer ce Pod à l'intérieur du cluster.

```
apiVersion: v1
kind: Pod
metadata:
  name: whoami
  labels:
    app: whoami
spec:
  containers:
  - name: whoami
    image: lucj/whoami
---
apiVersion: v1
kind: Service
metadata:
  name: whoami
  labels:
    app: whoami
spec:
  selector:
    app: whoami
  type: ClusterIP
  ports:
  - port: 80
    targetPort: 80
```

Copiez cette spécification dans un fichier *whoami.yaml* puis créez le Pod et le Service avec la commande suivante:

```
kubectl create -f whoami.yaml
```

Vérifiez ensuite que ces 2 objets ont été correctement lancés:

```
$ kubectl get po,svc
NAME                                READY     STATUS    RESTARTS   AGE
pod/whoami                          1/1       Running   0          26s

NAME                 TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)
AGE
service/whoami       ClusterIP   10.11.243.238   <none>        80/TCP
18s
```

2. Création d'une ConfigMap

Nous allons utiliser la configuration ci-dessous pour le serveur nginx que nous mettrons en place dans la suite.

```
user nginx;
worker_processes 4;
pid /run/nginx.pid;
events {
   worker_connections 768;
}
http {
  server {
    listen *:80;
    location = /whoami {
      proxy_pass http://whoami/;
    }
  }
}
```

Après avoir copié cette configuration dans un fichier *nginx.conf*, lancez la commande suivante pour créer la configMap *proxy-config*:

```
kubectl create configmap proxy-config --from-file=./nginx.conf
```

3. Spécification du reverse-proxy

La spécification suivante définie un Pod, contenant un unique container basé sur l'image *nginx*, ainsi qu'un service de type *NodePort* dont le rôle est d'exposer ce Pod à l'extérieur du cluster. C'est à ce service que l'on enverra une requête HTTP dans la suite.

Comme on peut le voir, la spécification définie un volume qui est utilisé pour monter la ConfigMap *proxy-config* dans le container *proxy* et donc le configurer

```
apiVersion: v1
kind: Pod
metadata:
  name: proxy
  labels:
    app: proxy
spec:
  containers:
  - name: proxy
    image: nginx:1.14-alpine
    volumeMounts:
    - name: config
      mountPath: "/etc/nginx/"
  volumes:
  - name: config
    configMap:
      name: proxy-config
---
apiVersion: v1
kind: Service
metadata:
  name: proxy
  labels:
    app: proxy
spec:
  selector:
    app: proxy
  type: NodePort
  ports:
  - port: 80
    targetPort: 80
    nodePort: 31600
```

Copiez cette spécification dans un fichier *proxy.yaml* puis créez le Pod et le Service avec la

commande suivante:

```
kubectl create -f proxy.yaml
```

Vérifiez ensuite que ces 2 objets ont été correctement lancés:

```
$ kubectl get po,svc
NAME                                READY     STATUS    RESTARTS   AGE
pod/proxy                           1/1       Running   0          7s
pod/whoami                          1/1       Running   0          m

NAME                    TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)
AGE
service/proxy           NodePort    10.11.255.14    <none>        80:31600/TCP
7s
service/whoami          ClusterIP   10.11.243.238   <none>        80/TCP
8m
```

4. Test de l'application

Depuis l'IP d'une des machines du cluster, nous pouvons alors envoyer une requête GET sur le endpoint */whoami* et voir que celle-ci est bien traitée par l'application *whoami*, elle renvoie *whoami*, le nom du container.

```
$ curl HOST_IP:31600/whoami
whoami
```

En résumé

Nous avons donc utilisé un objet ConfigMap que nous avons monté dans le container nginx du Pod faisant office de reverse-proxy. Cela permet de découpler la configuration et l'application.