Backup d'une base de données

Dans cet exercice, vous allez créer un *Job* qui servira à faire le dump d'une base de données *mongo*. Vous le modifierez ensuite pour en faire un *CronJob* et faire en sorte de réaliser ce dump à interval régulier.

Pour réaliser cet exercice, vous allez poser un label sur l'un des nodes de votre cluster et faire en sorte que chaque Pod soit déployé sur le node en question via la contrainte *nodeSelector*. Cela est nécessaire car vous allez utiliser un réprtoire de ce node pour échanger des données entre des Pods. Ce n'est pas un setup de production mais simplement une contrainte pour cet exercice.

Exercice

1. Création d'un label

Sur l'un de vos node, posez le label *tier=db*:

```
$ kubectl label node NODE_NAME tier=db
```

2. Création d'un Deployment Mongo

Dans un fichier *mongo-pod.yaml*, définissez la spécification d'un Pod basé sur l'image *mongo:3.6*. Ajouter l'instruction *nodeSelector* afin de déployer le Pod sur le node labelisé plus haut. Créez ensuite ce Pod.

3. Exposition de la base Mongo

Dans un fichier *mongo-svc.yaml*, définissez la spécification d'un Service, de type *clusterIP*, afin d'exposer le Pod précédent à l'intérieur du cluster, puis créez ensuite ce Service.

4. Définition d'un Job pour effectuer le dump de la base de données

Dans un fichier *mongo-dump-job.yaml*, définissez le spécification d'un Job, qui lance un Pod basé sur *mongo:3.6*.

Ajouter l'instruction *nodeSelector* afin de déployer le Pod sur le node labelisé plus haut.

Dans la spécification du Job, le Pod devra également définir un volume permettant de persister des données dans le répertoire */dump* du node. Vous utiliserez pour cela l'instruction *volumes* suivante:

```
volumes:
- name: dump
  hostPath:
    path: /dump
```

Le container mongo de ce Pod devra monter ce volume dans son répertoire */dump*. Vous utiliserez pour cela l'instruction *volumeMounts* dans la spécification du container:

```
volumeMounts:
- name: dump
  mountPath: /dump
```

De plus, vous ferez en sorte que le container de ce Pod lance la commande suivante afin de créer le fichier */dump/db.gz* contenant le dump de la base de données.

```
/bin/bash -c mongodump --gzip --host db --archive=/dump/db.gz
```

Note: cette commande utilise le binaire *mongodump* qui est présent dans l'image *mongo:3.6* et spécifie que le container se connectera au service *db* que vous avez lancé précédemment.

Lancez ensuite ce Job.

5. Restauration du dump

Définissez la spécification d'un nouveau Pod, nommé *test*, basé sur Mongo et montant le répertoire */dump* du node dans le répertoire */dump* du container (cela permettra au container d'avoir accès au dump créé précédemment).

Ajouter l'instruction *nodeSelector* afin de déployer le Pod sur le node labelisé plus haut.

Créez ensuite ce Pod, puis lancez un shell interactif dans ce Pod et, après avoir vérifié que le fichier */dump/db.gz* est présent, réalisez la restauration du dump en utilisant la commande

suivante:

```
$ mongorestore --drop  --archive=/dump/db.gz --gzip
```

6. Définition d'un CronJob pour effectuer le dump de la base de données à intervalle régulier

Dans un fichier *mongo-dump-cronjob.yaml*, définissez le spécification d'un CronJob qui lance un dump de mongo toutes les minutes.

Ajouter l'instruction *nodeSelector* afin de déployer le Pod sur le node labelisé plus haut.

Afin de conserver les différents dump, vous ferez en sorte que le container du Pod lance la commande suivante qui utilise un timestamp dans le nom du fichier.

```
/bin/bash -c mongodump --gzip --host db --archive=/dump/$(date
+"%Y%m%dT%H%M%S")-db.gz
```

Lancez ensuite ce CronJob.

7. Vérification des dump

Depuis le Pod *test*, vérifiez les dumps créés sur le node.

Correction

2. Création d'un Deployment Mongo

La spécification suivante définit un Pod basé sur *mongo:3.6*.

```
apiVersion: v1
kind: Pod
metadata:
  name: db
  labels:
    app: db
spec:
```

```
    nodeSelector:
      tier: db
    containers:
    - name: mongo
      image: mongo:3.6
```

Copiez cette spécification dans *mongo-pod.yaml* et lancez ce Pod avec la commande:

```
$ kubectl apply -f mongo-pod.yaml
```

3. Exposition de la base Mongo

La spécification suivante définit un Service de type *ClusterIP*. Ce service permet d'exposer le Pod précédent à l'intérieur du cluster.

```
apiVersion: v1
kind: Service
metadata:
  name: db
spec:
  selector:
    app: db
  type: ClusterIP
  ports:
  - port: 27017
```

Copiez cette spécification dans *mongo-svc.yaml* et lancez ce Service avec la commande:

```
$ kubectl apply -f mongo-svc.yaml
```

4. Définition d'un Job pour effectuer le dump de la base de données

La spécification suivante définit un Job qui effectue le dump de la base de données.

```
apiVersion: batch/v1
kind: Job
metadata:
  name: dump
```

```
spec:
  template:
    spec:
      nodeSelector:
        tier: db
      restartPolicy: Never
      containers:
      - name: mongo
        image: mongo:3.6
        command:
        - /bin/bash
        - -c
        - mongodump --gzip --host db --archive=/dump/db.gz
        volumeMounts:
        - name: dump
          mountPath: /dump
      volumes:
      - name: dump
        hostPath:
          path: /dump
```

Copiez cette spécification dans *mongo-dump-job.yaml* et lancez ce Job avec la commande:

```
$ kubectl apply -f mongo-dump-job.yaml
```

5. Restauration du dump

La spécification suivante définit un Pod basé sur *mongo:3.6*. Par l'utilisation d'un volume, le contenu du répertoire */dump* du node est monté dans le répertoire du même nom dans le container.

```
apiVersion: v1
kind: Pod
metadata:
  name: test
spec:
  nodeSelector:
    tier: db
  containers:
  - name: mongo
    image: mongo:3.6
    volumeMounts:
    - name: dump
      mountPath: /dump
  volumes:
    - name: dump
      hostPath:
```

```
          path: /dump
```

Copiez cette spécification dans *mongo-pod-test.yaml* et lancez ce Pod avec la commande:

```
$ kubectl apply -f mongo-pod-test.yaml
```

La commande suivante permet de lancer un shell interactif dans le container *mongo* du Pod *test*.

```
$ kubectl exec -ti test -- bash
```

Une fois dans ce container, on a accès au fichier de dump créé précédemment:

```
$ ls /dump
db.gz
```

La restauration de ce dump est réalisée avec la commande suivante:

```
$ mongorestore --drop  --archive=/dump/db.gz --gzip
2019-02-22T13:16:44.507+0000    preparing collections to restore from
2019-02-22T13:16:44.532+0000    done
```

6. Définition d'un CronJob pour effectuer le dump de la base de données à intervalle régulier

La spécification suivante définit un CronJob qui effectue le dump de la base de données, accessible via le service nommé *db*, toutes les minutes.

```
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: dump
spec:
  schedule: "*/1 * * * *"
  jobTemplate:
    spec:
```

```
      template:
        spec:
          nodeSelector:
            tier: db
          containers:
          - name: mongo
            image: mongo:3.6
            command:
            - /bin/bash
            - -c
            - mongodump --gzip --host db --archive=/dump/$(date
+"%Y%m%dT%H%M%S")-db.gz
            volumeMounts:
            - name: dump
              mountPath: /dump
          restartPolicy: OnFailure
          volumes:
          - name: dump
            hostPath:
              path: /dump
```

Copiez cette spécification dans *mongo-dump-cronjob.yaml* et lancez ce CronJob avec la commande:

```
$ kubectl apply -f mongo-dump-cronjob.yaml
```

7. Vérification des dump

Lancez un shell interactif dans le container du pod *test* afin d'observer les dumps créés.

```
$ kubectl exec test -- ls /dump
20190417T144810-db.gz
20190417T144909-db.gz
20190417T145010-db.gz
20190417T145110-db.gz
20190417T145210-db.gz
20190417T145310-db.gz
20190417T145410-db.gz
20190417T145510-db.gz
db.gz
```