Mise en situation professionnelle	**EasyLinux**
	Cours : Kubernetes
Sujet : Exploiter le Cluster	Numéro : 12 à 14 Version : 1.0

Objectifs :

Exploiter le cluster

Prérequis :

aucun

Principales tâches à réaliser :

Exploiter le cluster

12 Affinité sur un nœud

- Récupérer le projet Monitor et le lancer sur le cluster
- Ajouter un label sur le nœud Master
- Imposer le lancement de InfluxDb sur le nœud master

13 Contrainte sur une machine

Créer un déploiement pour registry.formation.local:5000/easylinux/kuard , demander 128Mo et limiter la mémoire à 256Mo

14 Mise en œuvre du HealtCheck

Mettre des contrôles de santé sur le deploiement Kuard.

Pour tester si le conteneur est prêt, faire un test httpGet sur /ready (port 8080) démarrant immédiatement avec une périodicité de 2 secondes. Mise en erreur si plus de 3 échecs et OK dès la première réponse positive

Pour tester si le conteneur fonctionne, faire un test httpGet sur /healthy (port 8080) démarrant au bout de 5 secondes avec une périodicité de 10 secondes. Mise en erreur timeout de plus d'une seconde echec dés 3 essais infructueux.

15 Sécurisation

Créer un NameSpace Dev,

créer un userAccount Developpeur et

lui assigner les droits sur Dev

16 Solution

Installer git (selon votre poste-voir avec formateur)

```
$ sudo apt-get install git
...
```

Récupérer le projet

```
$ git clone http://registry.formation.local:3000/snoel/Monitor.git
Cloning into 'Monitor'...
remote: Counting objects: 40, done.
remote: Compressing objects: 100% (39/39), done.
remote: Total 40 (delta 19), reused 0 (delta 0)
Unpacking objects: 100% (40/40), done.
vagrant@Master:~$
```

16.1 Mettre en œuvre un stockage NFS

Modifier les persistentVolume pour les rendre compatible (il sont en mode Minikube)

Le persistentVolume NFS

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: nfs-pv
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Retain
  nfs:
    path: /nfs
    server: <ip formateur>
    readOnly: false
```

Le persistentVolumeClaim

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: nfs-pvc
spec:
  accessModes:
  - ReadWriteMany
  resources:
     requests:
       storage: 1Gi
```

16.2 Affinité sur un nœud

Ajouter un label au nœud master

```
kubectl label nodes master DiskType=Ssd
```

Puis dans la définition du deployment ajouter

```
    imagePullPolicy: IfNotPresent
  nodeSelector:
    DiskType: Ssd
```

16.3 Limites de mémoire

```
containers:
- image: nginx
  imagePullPolicy: Always
  name: default-mem-demo-ctr
  resources:
    limits:
      memory: 512Mi
    requests:
      memory: 256Mi
```

Dans l'application, faire une requête de 512Mo, que se passe-t-il ?

16.4 Probe

Pour valider le bon démarrage du programme :

```
…
  spec:
    containers:
      …
      readinessProbe:
        httpGet:
          path: /ready
          port: 8080
        periodSeconds: 2
        initialDelaySeconds: 2
        failureThreshold: 3
        successThreshold: 1
```

16.5 Liveness

```
    livenessProbe:
      httpGet:
        path: /ready
        port: 8080
      initialDelaySeconds: 5
      timeoutSeconds: 1
      periodSeconds: 10
      failureTreshold: 3
```

Provoquer des erreurs dans Kuard, que se passe-t-il ?

16.6 Sécurisation

a Créer le namespace

```
kubectl create namespace dev
```

b Créer le user account

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: developpeur
  namespace: dev
```

c Lui donner les droits

```
kind: Role
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: dev-user-full-access
  namespace: dev
rules:
- apiGroups: ["", "extensions", "apps"]
  resources: ["*"]
  verbs: ["*"]
- apiGroups: ["batch"]
  resources:
  - jobs
  - cronjobs
  verbs: ["*"]

---
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: dev-user-view
  namespace: dev
subjects:
- kind: ServiceAccount
  name: developpeur
  namespace: dev
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: dev-user-full-access
```

Se connecter avec un utilisateur Developpeur (à créer sur le poste) et récupérer la configuration Kubectl

Récupérer les secrets générés :

```
kubectl describe sa developpeur -n dev
```

Récupérer le userToken

```
kubectl get secret developpeur-token-xxxxx -n dev -o "jsonpath={.data.token}" | base64 -
D
```

Récupérer le certificat

```
kubectl get secret developpeur-token-xxxxx -n dev -o "jsonpath={.data['ca\.crt']}"
```

NB :

On pourra récupérer le certificat dans un fichier, ex dev.crt puis y faire référence dans le fichier config ici-bas

d Générer le fichier config pour kubectl

On peut maintenant générer le fichier config

```
apiVersion: v1
kind: Config
preferences: {}

# Define the cluster
clusters:
- cluster:
    certificate-authority-data: PLACE CERTIFICATE HERE
    # You'll need the API endpoint of your Cluster here:
    server: https://YOUR_KUBERNETES_API_ENDPOINT
  name: dev

# Define the user
users:
- name: developpeur
  user:
    as-user-extra: {}
    client-key-data: PLACE CERTIFICATE HERE
    token: PLACE USER TOKEN HERE

# Define the context: linking a user to a cluster
contexts:
- context:
    cluster: my-cluster
    namespace: dev
    user: developpeur
  name: dev

# Define current context
current-context: dev
```

Placer le fichier config dans ~/.kube/ (on aura ~/.kube/config)