Mise en situation professionnelle	**EasyLinux**
	Cours : Kubernetes
Sujet : Cheat sheet	Numéro : - Version : 1.0

Objectifs :

Assistance

Prérequis :

aucun

Principales tâches à réaliser :

1 Pod

Un fichier de définition de Pod est un fichier du type : pod.yaml, la référence de ce fichier peut être trouvée sur :

https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.12/

```
apiVersion: v1
kind: Pod
metadata:
  name: Nom du pod
  labels:
    app: Nom de l'appli (permet de filtrer)
spec:
  containers:
La section containers contient l'ensemble des conteneurs de votre application
  - name: Nom du conteneur
    image: Nom de l'image
    command: Commande dans le conteneur voir CMD
    args: Argument(s) à passer à la commande
    env:  Les variables d'environnement pour le conteneur
    env:
      - name: Nom de la variable
        value: "sa valeur"
      - ...
    imagePullPolicy: IfNotPresent  Comment télécharger l'image (Always / Never / IfNotPresent)
    ports: Ports à l'écoute dans le conteneur
      - name : http
        containerPort : 8083
      - ...

  - name: Nom du conteneur
      …

  initContainers:  # optionnel voir plus bas
    - name: init-myservice
      image: busybox
      command: ['sh', '-c', 'until nslookup myservice; do echo waiting for myservice;
sleep']
```

NB : initContainers

c'est un ou plusieurs conteneurs lancés avant le démarrage du pod, en cas d'échec, ils bloquent le démarrage du Pod (sauf si restartPolicy : never)

Utilité :

- ✔ Initialiser des valeurs (sed, awk,. . .)
- ✔ Attente d'un service
- ✔ Initialiser des comptes utilisateurs

2 Deployment

Un Deployment est un des objets permettant de lancer des Pods. Dans les bonnes pratiques de Kubernetes il est encouragé d'utiliser des Deployments pour les applications stateless.

Lors de la création d'un Deployment le controller associé va créer un ReplicaSet à partir de votre configuration. Le controller associé au ReplicaSet va créer une série de Pod à partir de la configuration du ReplicaSet.

Les avantages d'utiliser un Deployment au lieu de créer directement un ReplicaSet sont :

- **Historisation de l'objet** : chaque changement dans l'objet (via un "apply" ou un "edit") va créer une sauvegarde de la version précédente.
- **Gestion du rollout et du rollback** : en lien avec le point précédent, vous pourrez revenir en arrière sur une configuration.

Le Deployment est un objet stable depuis la version 1.9 de Kubernetes, disponible sur l'api "apps/v1".

3 Namespace

Pour créer un namespace pour séparer le monitoring du cluster :

exemple création d'un namespace 'development'

```
apiVersion: v1
kind: Namespace
metadata:
  labels:
    name: development
  name: development
```

4 Deployment

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:  # ici les metadata liées au Deployment
  # Nom de l'objet, il doit être unique dans le namespace
  replicas: 3     # nombre de pod à lancer
  name: deployment-example
  namespace: default
spec:  # Ici les spécifications du Deployment
  selector:
    matchLabels:
      app: deployment-example
  template:
    metadata:  # ici les metadata lié aux Pods (à ne pas confondre avec les précédents)
      labels:
        app: deployment-example
    spec:  # Ici les spécifications des Pods
      containers:  # un Pod peut contenir plusieurs conteneurs
      - name: web
        image: nginx:1.10
```

5 Service

- **Port :** Port est le numéro de port qui rend un service visible aux autres services fonctionnant dans le même cluster K8s. En d'autres termes, si un service veut invoquer un autre service fonctionnant dans le même cluster Kubernetes, il pourra le faire en utilisant le port spécifié contre "port" dans le fichier de spécifications du service.
- **targetPort** : Le port cible est le port du POD sur lequel le service est exécuté.
- **Nodeport** : Node port est le port sur lequel le service peut être accédé à partir d'utilisateurs externes en utilisant Kube-Proxy. Jetez un coup d'oeil aux spécifications suivantes pour définir un exemple de service :

```
apiVersion: v1
kind: Service
metadata:
  name: example-service
  labels:
    app: example-service
spec:
  ports:
  - port: 80              Port à l'écoute dans le conteneur
    targetPort: 4000      Port à l'écoute dans le cluster
  type: NodePort
  selector:
    app: example-pod
```

6 hostPath

Un volume hostPath monte un fichier ou un répertoire du système de fichiers du nœud hôte dans votre Pod. Ce n'est pas quelque chose dont la plupart des Pods ont besoin, mais il offre une solution puissante pour certaines applications.
Par exemple, certaines utilisations d'un hostPath :

- exécuter un conteneur qui a besoin d'accéder au fonctionnement interne de Docker ; utiliser un chemin hôte de /var/lib/docker ou /var/run/docker.sock
- exécuter cAdvisor dans un conteneur, pour accéder à /sys
- permettre à un Pod de spécifier si un hostPath donné doit exister avant l'exécution du Pod, s'il doit être créé, et qu'il doit exister.

En plus du chemin (obligatoire), l'utilisateur peut spécifier un type pour un volume hostPath.
Les valeurs prises en charge pour le type de zone sont :

Valeur	**Description**
vide	pour la rétrocompatibilité, ce qui signifie qu'aucun contrôle ne sera effectué avant le montage du volume hostPath.
DirectoryOrCreate	Si rien n'existe au niveau du chemin donné, un répertoire vide y sera créé selon les besoins avec l'autorisation fixée à 0755, ayant le même groupe et le même propriétaire avec Kubelet.
Directory	Un répertoire doit exister sur le chemin donné
FileOrCreate	Si rien n'existe sur le chemin donné, un fichier vide y sera créé au besoin avec la permission 0644, ayant le même groupe et le même propriétaire que Kubelet.
File	Un fichier doit exister au chemin donné
Socket	Un socket UNIX doit exister sur le chemin donné
CharDevice	Un périphérique de caractères doit exister sur le chemin donné
BlockDevice	Un périphérique de bloc doit exister sur le chemin donné

Attention lors de l'utilisation de ce type de volume, car :

- Les pods de configuration identique (comme ceux créés à partir d'un podTemplate) peuvent **se comporter différemment sur différents nœuds** en raison de fichiers différents sur les nœuds.
- lorsque Kubernetes ajoute une planification tenant compte des ressources, comme prévu, elle ne sera pas en mesure de rendre compte des ressources utilisées par un hostPath
- les fichiers ou répertoires créés sur les hôtes sous-jacents ne peuvent être écrits que par root. Vous devez soit exécuter votre processus en tant que root dans un conteneur privilégié, soit modifier les permissions du fichier sur l'hôte pour pouvoir écrire sur un volume hostPath.

Exemple de définition dans un pod Pod

```
apiVersion: v1
kind: Pod
metadata:
  name: test-pd
spec:
  containers:
  - image: k8s.gcr.io/test-webserver
    name: test-container
    volumeMounts:
    - mountPath: /test-pd
      name: test-volume
  volumes:
  - name: test-volume
    hostPath:
      # directory location on host
      path: /data
      # this field is optional
      type: Directory
```

7 PersitentVolume

Un PersistentVolume (PV) est un élément de stockage dans le cluster qui a été provisionné manuellement par un administrateur, ou dynamiquement provisionné par Kubernetes à l'aide d'une StorageClass.

Une PersistentVolumeClaim (PVC) est une demande de stockage par un utilisateur qui peut être satisfaite par un PersistentVolume.

PersistentVolumes et PersistentVolumeClaims sont indépendants du cycle de vie des Pods et préservent les données en redémarrant, reprogrammant et même en supprimant les Pods.

NB :

- De nombreux environnements de cluster ont une StorageClass par défaut installée. Lorsqu'une StorageClass n'est pas spécifiée dans la PersistentVolumeClaim, la StorageClass par défaut du cluster est utilisée à la place.
- Dans les clusters locaux, la StorageClass par défaut utilise le provisioner hostPath. Les volumes hostPath ne conviennent que pour le développement et les tests. Avec les volumes hostPath, vos données résident un répertoire sur le nœud sur lequel le Pod est planifié et ne se déplacent pas entre les nœuds. Si un Pod meurt et est programmé sur un autre nœud du cluster, ou si le nœud est redémarré, les données sont perdues.
- Sur un cluster qui a besoin d'utiliser le provisioner hostPath, l'option --enable-hostpath-provisioner doit être définie dans le composant controller-manager.

7.1 storageClass

Définir une classe de stockage pour séparer les données (dépend du fournisseur)

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  namespace: kube-system
  name: speed
  annotations:
    storageclass.beta.kubernetes.io/is-default-class: "true"
  labels:
    addonmanager.kubernetes.io/mode: EnsureExists
provisioner: k8s.io/minikube-hostpath (dépend du fournisseur)
```

7.2 volumeClaim

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: nextcloud-db-claim
  labels:
    app: nextcloud
spec:
  storageClassName: speed
  accessModes:
    - ReadWriteOnce
  volumeMode: Filesystem
  resources:
    requests:
      storage: 1Gi
```

7.3 Utilisation

```
apiVersion: apps/v1
kind: Deployment
…
spec:
…
  template:
…
    spec:
      containers:
…
        volumeMounts:
        - name: nextcloud-storage              (nom du volume cf plus bas)
          mountPath: /var/lib/mysql            (point de montage dans le pod)
      volumes:
      - name: nextcloud-storage                (défini le nom)
        persistentVolumeClaim:
          claimName: nextcloud-db-claim        (nom du pvc référencé)
```

8 secrets

```
apiVersion: v1
kind: Secret
metadata:
  name: next-conf                    Nom du secret
type: Opaque
data:
  DB: ZGIw                           Couple Clé / Valeur ( echo -n 'secret' | base64 )
  ADMIN: YWRtaW4=
```

8.1 Utiliser en tant que répertoire

```
apiVersion: extensions/v1beta1
kind: Deployment
...
spec:
...
  template:
...
    spec:
      containers:
        - name: mypod
          image: easylinux/kubernetes:nginx
          volumeMounts:
            - mountPath: "/etc/password"      répertoire où sera monté les secrets
              name: credential                Nom du volume à mapper
      volumes:
        - name: credential                    Nom du volume
          secret:
            secretName: next-conf             Nom du secret
```

8.2 Utiliser en tant que variable

```
apiVersion: extensions/v1beta1
kind: Deployment
...
  template:
 ...
    spec:
      containers:
        - name: mypod
          image: easylinux/kubernetes:nginx
          env:
            - name: NEXT_DB                   Nom de la variable (dans le conteneur)
              valueFrom:
                secretKeyRef:
                  name: next-conf             Nom de l'objet secret
                  key: NEXT_DB                Nom de la variable dans l'object
```

9 configMap

```
apiVersion: v1
kind: ConfigMap
data:
  game.properties: \
    enemies=aliens
    lives=3
    enemies.cheat=true
    enemies.cheat.level=noGoodRotten
    secret.code.passphrase=UUDDLRLRBABAS
    secret.code.allowed=true
    secret.code.lives=30
  ui.properties: \
    color.good=purple
    color.bad=yellow
    allow.textmode=true
    how.nice.to.look=fairlyNice
metadata:
  name: game-config-env-file
  namespace: default
```

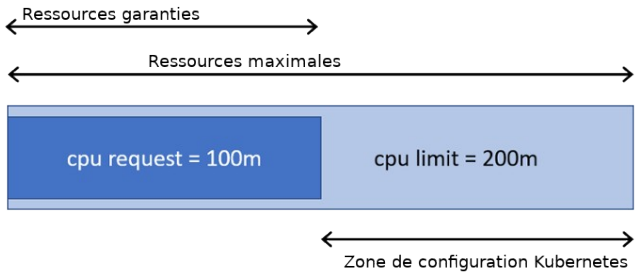
9.1 Utilisation

Idem secrets

10 Limiter les ressources

Pour limiter les ressources utilisées par un conteneur, on peut ajouter dans la définition :

```
apiVersion: extensions/v1beta1
kind: Deployment
...
    resources:  Limitation des ressources
      limits:
        cpu:  100m
        memory:  500Mi
      requests:
        cpu:  100m
        memory:  500Mi
 ...
```



11 Autoscaling

Kubernetes ajustera automatiquement le nombre de pod en fonction de l'usage

```
$ kubectl autoscale deployment foo --min=2 –max=10 --cpu-percent=80
```

12 Gestion des droits

```
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  namespace: default
  name: pod-reader
rules:
- apiGroups: [""]                  # "" indique le groupe d'API
  resources: ["pods"]              # Type de ressource
  verbs: ["get", "watch", "list"]
```

Un ClusterRole peut être utilisé pour accorder les mêmes permissions qu'un Role, mais ils sont données pour le cluster, ils peuvent en conséquence aussi être utilisés pour donner accès à :

- les ressources à l'échelle du cluster (comme les nœuds)
- critères d'évaluation non liés aux ressources (comme "/healthz")
- ressources namespaced (comme les pods) dans tous les espaces de noms (nécessaires pour exécuter kubectl get pods --all-namespaces, par exemple)

Le ClusterRole suivant peut être utilisé pour autoriser l'accès en lecture aux secrets dans n'importe quel espace de nommage particulier, ou dans tous les espaces de nommage (selon la façon dont il est lié) :

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  # "namespace" omitted since ClusterRoles are not namespaced
  name: secret-reader
rules:
- apiGroups: [""]
  resources: ["secrets"]
  verbs: ["get", "watch", "list"]
```

NB :
verbes: ["get", "list", "watch", "create", "update", "patch", "delete"]

12.1 RoleBinding et ClusterRoleBinding

Un role binding accorde les permissions définies dans un rôle à un utilisateur ou à un ensemble d'utilisateurs. Il contient une liste de sujets (utilisateurs, groupes ou comptes de services) et une référence au rôle accordé. Les permissions peuvent être accordées dans un espace de noms avec un RoleBinding, ou à l'échelle du cluster avec un ClusterRoleBinding.

```
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: read-pods
  namespace: default
subjects:
- kind: User
  name: jane         # Name est sensible à la casse
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role         # ce doit être Role ou ClusterRole
  name: pod-reader   # doit correspondre au Role ou ClusterRole auquel on se réfère
  apiGroup: rbac.authorization.k8s.io
```

On peut aussi créer des groupes

```
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: read-secrets-global
subjects:
- kind: Group
  name: manager       # le nom est sensible à la casse
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: secret-reader
  apiGroup: rbac.authorization.k8s.io
```