Lab 3 : Créer et déployer une représentation logique de Pods avec les Deployments

Dans l'exercice précédent, nous avons montré comment créer des `Pods` manuellement. Toutefois, cette gestion manuelle peut devenir fastidieuse. Par exemple, lorsqu'un `Pod` est créé dans un cluster et qu'une panne intervient sur le nœud où est localisé le `Pod`, Kubernetes ne va pas essayer de trouver un état sain en recréant manuellement le `Pod` disparu. Dans un même ordre d'idée, si vous souhaitez démarrer des `Pods` à un intervalle régulier ou réaliser une mise à jour progressive des versions des `Pods` déjà en exécution.

Ce deuxième exercice s'intéresse aux objets de déploiement qui gèrent les `Pods` à votre place. Ces objets de déploiement permettent de donner une représentation logique de un ou plusieurs `Pods`. Ils sont également appelés également `Charge de Travail` (`Workload` en anglais) dans la documentation officielle de Kubernetes. Différents types d'objets de déploiement existent : `Deployment`, `StatefulSets`, `DaemonSets`, `Jobs` et `Cronjob`. Nous allons nous intéresser dans cet exercice aux objets de type `Deployment` qui aident à gérer des `Pods` de type *Stateless*. Ces objets sont capables de gérer la montée en charge horizontale (Scale Out) via l'utilisation d'un objet de type `ReplicaSet`.

Quelque soit le type d'installation choisi pour la mise en place de votre cluster Kubernetes, toutes les commandes ci-dessous devraient normalement fonctionner. Nous considérons qu'il existe un fichier `k3s.yaml` à la racine du dossier `kubernetes-tutorial/`, si ce n'est pas le cas, merci de reprendre la mise en place d'un cluster Kubernetes. Il est important ensuite de s'assurer que la variable `KUBECONFIG` soit initialisée avec le chemin du fichier d'accès au cluster Kubernetes (`export KUBECONFIG=$PWD/k3s.yaml`).

But

- Écrire un fichier de configuration `Deployment`
- Réaliser une montée en charge horizontale via les `ReplicaSet`
- Gérer une montée en version

Étapes à suivre

- Avant de commencer les étapes de cet exercice, assurez-vous que le `Namespace` créé dans l'exercice précédent `mynamespaceexercice1` soit supprimé.

```
$ kubectl delete namespace mynamespaceexercice1
namespace "mynamespaceexercice1" deleted
```

- Créer dans le répertoire *exercice2-deployment/* un fichier appelé *mynamespaceexercice2.yaml* en ajoutant le contenu suivant :

```
apiVersion: v1
kind: Namespace
metadata:
  name: mynamespaceexercice2
```

- Créer ce `Namespace` dans notre cluster :

```
$ kubectl apply -f exercice2-deployment/mynamespaceexercice2.yaml
namespace/mynamespaceexercice2 created
```

Nous allons créer un objet de type `Deployment` qui est responsable des `Pods` qu'il gère. Ainsi, si un `Pod` ne fonctionne pas correctement un nouveau `Pod` sera créé et le `Pod` qui ne fonctionnait pas correctement sera arrêté.

- Créer dans le répertoire *exercice2-deployment/* un fichier appelé *mydeployment.yaml* en ajoutant le contenu suivant :

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mydeployment
spec:
  selector:
    matchLabels:
      app: mypod
  template:
    metadata:
      labels:
        app: mypod
    spec:
      containers:
      - name: mycontainer
        image: nginx:1.19
        ports:
        - containerPort: 80
```

Dans cette configuration, nous retrouvons la description du `Pod` dans l'option `template` et les conteneurs de ce `Pod` dans l'option `containers`.

- Appliquer cette configuration pour créer ce `Deployment` dans le cluster Kubernetes :

```
$ kubectl apply -f exercice2-deployment/mydeployment.yaml -n
mynamespaceexercice2
deployment.apps/mydeployment created
```

- Vérifier que le `Pod` géré par ce `Deployment` est correctement créé :

```
$ kubectl get pods -n mynamespaceexercice2 -o wide
NAME                            READY   STATUS    RESTARTS   AGE   IP          NODE
mydeployment-59f769b799-hpfvs   1/1     Running   0          21s   10.42.2.8   k3d-mycluster-
agent-1
```

Nous remarquons que le nom du `Pod` n'est pas celui que nous avons donné dans le fichier de configuration, cela est normal car un `Deployment` peut créer plusieurs `Pods` basés sur un même patron (option `template`). Nous reviendrons sur cet aspect quand nous étudierons la montée en charge horizontale (`ReplicatSet`).

- Afficher également les informations de ce `Deployment` :

```
$ kubectl get deployments.apps -n mynamespaceexercice2
NAME           READY   UP-TO-DATE   AVAILABLE   AGE
mydeployment   1/1     1            1           2m14s
```

- Nous pouvons également donner un détail complet de ce `Deployment` via l'option `describe` :

```
$ kubectl describe deployments.apps -n mynamespaceexercice2 mydeployment
Name:                   mydeployment
Namespace:              mynamespaceexercice2
CreationTimestamp:      Tue, 07 Feb 2023 15:11:21 +0100
Labels:                 <none>
Annotations:            deployment.kubernetes.io/revision: 1
Selector:               app=mypod
Replicas:               1 desired | 1 updated | 1 total | 1 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  app=mypod
  Containers:
   mycontainer:
    Image:        nginx:1.19
    Port:         80/TCP
    Host Port:    0/TCP
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Available      True    MinimumReplicasAvailable
  Progressing    True    NewReplicaSetAvailable
OldReplicaSets:  <none>
NewReplicaSet:   mydeployment-59f769b799 (1/1 replicas created)
Events:
  Type    Reason             Age    From                   Message
  ----    ------             ----   ----                   -------
  Normal  ScalingReplicaSet  2m29s  deployment-controller  Scaled up replica set
mydeployment-59f769b799 to 1
```

La configuration d'un objet de type `Deployment` peut inclure le nombre de `Pods` à créér qui est de `un` (1) actuellement. Cette information permet donc de gérer la montée en charge horizontale en utilisant un objet de type `ReplicatSet`. Il est possible d'écrire une configuration de type `ReplicatSet` au même titre que celles que nous avons déjà faites pour `Pod` ou `Deployment`. Toutefois, `Deployment` peut inclure cette information directement dans sa configuration. C'est de cette manière que nous présenterons `ReplicatSet`.

- Modifier le fichier de configuration *exercice2-deployment/mydeployment.yaml* de façon à intégrer les informations de réplication (clé `replicas`) :

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mydeployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: mypod
  template:
    metadata:
      labels:
        app: mypod
    spec:
      containers:
      - name: mycontainer
        image: nginx:1.19
        ports:
        - containerPort: 80
```

Cette configuration de `Deployment` déclare maintenant que trois `Pods` basés sur le template nommé `mypod` devront être créés dans le cluster Kubernetes. Toute la création est déléguée à Kubernetes qui se chargera de trouver les bons nœuds pour héberger les nouveaux `Pods`.

- Appliquer la nouvelle configuration :

```
$ kubectl apply -f exercice2-deployment/mydeployment.yaml -n
mynamespaceexercice2
deployment.apps/mydeployment configured
```

- Vérifions que les nouveaux `Pods` ont été créés :

```
$ kubectl get pods -n mynamespaceexercice2 -o wide
NAME                          READY   STATUS    RESTARTS   AGE    IP          NODE
mydeployment-59f769b799-hpfvs   1/1     Running   0          4m9s   10.42.2.8   k3d-mycluster-
agent-1
mydeployment-59f769b799-xkjhz   1/1     Running   0          18s    10.42.0.5   k3d-mycluster-
server-0
mydeployment-59f769b799-kbxwd   1/1     Running   0          18s    10.42.1.5   k3d-mycluster-
agent-0

$ kubectl get deployments.apps -n mynamespaceexercice2
NAME           READY   UP-TO-DATE   AVAILABLE   AGE
mydeployment   3/3     3            3           4m35s
```

Nous remarquons d'une part que deux nouveaux `Pods` ont été ajoutés (information liée à l'âge) et d'autre part que Kubernetes a choisi d'équilibrer le déploiement sur l'ensemble des nœuds. Cela est cohérent car les ressources matérielles affectées aux nœuds à notre cluster sont assez réduites (1 Go de mémoire) et que K8s essaye de maximiser la disponibilité des `Pods` au cas où un nœud deviendrait indisponible. Notons également que les trois `Pods` ont des noms assez proches.

Nous allons nous intéresser à la problématique de la montée en version des images Docker. Dans le fichier de configuration *mydeployment.yaml* la version de l'image Docker Nginx est actuellement de `1.19`. Il existe de nouvelles versions plus récentes, nous allons migrer progressivement vers celles-ci tout en s'assurant que notre dépoiement est disponible et qu'il est possible de revenir sur un déploiement précédent. Quand une montée en version est réalisée, un enroulement (*rollup*) est provoqué et référencé dans une revision (un numéro unique). Pour chaque révision la cause du changement peut être renseignée dans un texte libre. Cet enroulement pourra être consulté et utilisé pour revenir sur une révision donnée. À noter que cela ne concerne pas uniquement le changement des versions d'image, les changements sur les variables d'environnement et les ressources sont également considérées. Cependant, le changement sur la valeur de `replicas` ne sera pas enroulé.

Nous considérons dans la suite que nous disposons d'un `Deployment` qui gère trois `Pods` basés sur l'image Docker `1.19`.

- Pour afficher l'historique de l'enroulement (*rollup*) en cours pour le `Deployment mydeployment` :

```
$ kubectl rollout history deployment -n mynamespaceexercice2 mydeployment
deployment.apps/mydeployment
REVISION  CHANGE-CAUSE
1         <none>
```

- Comme constaté, il n'y a pas d'information sur la cause du changement. Pour renseigner la cause du changement :

```
$ kubectl annotate deployments.apps -n mynamespaceexercice2 mydeployment
kubernetes.io/change-cause="Image en 1.19"
deployment.apps/mydeployment annotated
```

- Afficher de nouveau l'historique de l'enroulement (*rollup*) en cours pour le `Deployment mydeployment` pour vérifier que la description dans la colonne `CHANGE-CAUSE` a été modifiée :

```
$ kubectl rollout history deployment -n mynamespaceexercice2 mydeployment
deployment.apps/mydeployment
REVISION  CHANGE-CAUSE
1         Image en 1.19
```

Un seule révision est présente pour l'instant dans cet enroulement (*rollup*).

- La version de l'image Docker va être modifiée pour passer de la version `1.19` à la version `1.20`. Nous utiliserons l'option `set` de l'outil **kubectl** pour effecuter la modification :

```
$ kubectl set image -n mynamespaceexercice2 deployment mydeployment
mycontainer=nginx:1.20
deployment.apps/mydeployment image updated
$ kubectl annotate deployments.apps -n mynamespaceexercice2 mydeployment
kubernetes.io/change-cause="Image en 1.20"
deployment.apps/mydeployment annotated
```

- Afficher de nouveau l'historique de l'enroulement (*rollup*) pour le `Deployment` `mydeployment` :

```
$ kubectl rollout history deployment -n mynamespaceexercice2 mydeployment
deployment.apps/mydeployment
REVISION  CHANGE-CAUSE
1         Image en 1.19
2         Image en 1.20
```

Deux révisions sont actuellement dans l'enroulement du `Deployment` `mydeployment`.

- Créer un nouveau fichier de configuration `Deployment` nommé *exercice2-deployment/mydeployment121.yaml* :

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mydeployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: mypod
  template:
    metadata:
      labels:
        app: mypod
    spec:
      containers:
      - name: mycontainer
        image: nginx:1.21
        ports:
        - containerPort: 80
```

- Appliquer cette nouvelle configuration `Deployment` :

```
$ kubectl apply -f exercice2-deployment/mydeployment121.yaml -n
mynamespaceexercice2
deployment.apps/mydeployment configured
$ kubectl annotate deployments.apps -n mynamespaceexercice2 mydeployment
kubernetes.io/change-cause="Image en 1.21"
deployment.apps/mydeployment annotated
```

Même s'il s'agit d'un nouveau fichier de configuration, le nom du `Deployment` est le même que celui du fichier de configuration *exercice2-deployment/mydeployment.yaml*. Kubernetes va identifier qu'il s'agit du même `Deployment` et appliquer cette configuration. La seule modification apportée concerne la version de l'image Docker qui passe de `1.20` à `1.21`.

- Afficher de nouveau l'historique de l'enroulement (*rollup*) pour le `Deployment mydeployment` :

```
$ kubectl rollout history deployment -n mynamespaceexercice2 mydeployment
deployment.apps/mydeployment
REVISION  CHANGE-CAUSE
1         Image en 1.19
2         Image en 1.20
3         Image en 1.21
```

Une troisième révision est ajoutée à l'enroulement du `Deployment mydeployment`.

- Afficher la description détaillée du `Deployment mydeployment` :

```
$ kubectl describe deployments.apps -n mynamespaceexercice2 mydeployment
Name:                   mydeployment
Namespace:              mynamespaceexercice2
CreationTimestamp:      Tue, 07 Feb 2023 15:11:21 +0100
Labels:                 <none>
Annotations:            deployment.kubernetes.io/revision: 3
                        kubernetes.io/change-cause: Image en 1.21
Selector:               app=mypod
Replicas:                3 desired | 3 updated | 3 total | 3 available | 0
unavailable
StrategyType:           RollingUpdate
…
```

La version de l'image Docker est bien à `1.21`. Dans la partie `Events` nous constatons les différentes opérations réalisées pour passer de la version `1.19` jusqu'à la version `1.21`.

- Utilisons l'option `rollout` pour revenir à une révision antérieure :

```
$ kubectl rollout undo deployment mydeployment -n mynamespaceexercice2 --to-
revision=1
deployment.apps/mydeployment rolled back
```

Ces opérations sont importantes si vous constatez qu'une mise à jour d'une image Docker ne s'est pas bien déroulée. Vous pouvez à tout moment revenir sur des versions antérieures.

Par ailleurs, nous ne l'avons pas détaillé, mais une stratégie de déploiement est utilisée à chaque passage d'une révision supérieure ou inférieure. Par défaut il s'agit de `Rolling Update` qui consiste à changer la version d'un `Pod` un par un. Si un échec se produit, Kubernetes s'assure de revenir dans un état cohérent. D'autres stratégies existent comme `Recreate` qui consiste à supprimer tous les `Pods` d'un `Deployment` puis de recréer de nouveaux `Pods`. Cette stratégie peut rendre indisponible votre applicatif pendant un certain moment.

Bilan de l'exercice

À cette étape, vous savez :

- écrire un fichier de configuration `Deployment` ;
- réaliser une montée en charge horizontale via les `ReplicaSet` ;
- gérer l'enroulement (*rollup*) des modifications d'un `Deployment` ;
- changer la révision d'un enroulement.

Pour continuer sur les concepts présentés dans cet exercice, nous proposons les expérimentations suivantes :

- créer un `Deployment` basé sur une image Docker Apache HTTP et définir trois `ReplicaSets` ;
- changer la stratégie de montée en charge en `Recreate` ;
- changer la version de l'image Docker Apache HTTP pour des versions antérieures tout en détaillant la cause du changement `CHANGE-CAUSE`.