GÉRER ET MANIPULER LES DEPLOYMENTS KUBERNETES

Introduction

Jusqu'ici, nous avions étudié deux types d'objets Kubernetes, à savoir les `Pods` et les `ReplicaSets`. Mais, il existe encore un autre niveau plus haut d'abstraction qui permet de gérer ces deux derniers en un seul même objet. Ce nouvel objet Kubernetes se nomme le `Deployment` (en FR : "déploiement"), que nous allons étudier à travers ce chapitre.

Comment ça marche ? Vous décrivez un état souhaité de vos `Pods` et de vos `ReplicaSets` dans un `Deployment` et le `Deployment controller` (en FR : "contrôleur de déploiement") modifie l'état actuel à l'état souhaité.

Manipulation des Deployments

Créer un Deployment

Sans un template YAML

Vous pouvez **créer un Deployment sans utiliser de template au format YAML**, avec la commande `kubectl run`. Dans cet exemple, nous allons créer trois répliques de Pods utilisant l'image `alpine`, possédant le label `app: alpine` et exécutant la commande `sleep 1d` afin que les Pods soient actifs pendant 1 jour :

```
kubectl run alpine --image=alpine -r=3 -l="app=alpine" -- sleep 1d
```

Résultat :

```
deployment.apps/alpine created
```

Ensuite **vérifions la liste des Deployments Kubernetes** en cours de création :

```
kubectl get deployment
```

Le résultat est similaire à celui-ci :

```
NAME     READY   UP-TO-DATE   AVAILABLE   AGE
alpine-deployment   3/3     3            3           3m59s
```

Avec le template YAML

Voici un autre exemple de **Deployment créé depuis un template YAML**, afin d'exécuter un ReplicaSet de trois Pods de type `nginx` :

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

Le squelette du template YAML, reste le même que celui des ReplicaSets avec le champ `replicas` ou nous spécifions le nombre de Pods et le champ `selector` ou nous définissons comment le Deployment trouve les Pods à gérer. Dans ce cas, il ne

gérera que les Pods ayant le label `app: nginx`.

Créons maintenant notre Deployment, à l'aide de la commande ci-dessous :

```
kubectl create -f nginx-pod.yaml
```

Revérifions ensuite la liste de nos Deployments :

```
kubectl get deployment
```

Résultat :

```
NAME                 READY   UP-TO-DATE   AVAILABLE   AGE
alpine-deployment    3/3     3            0           9s
nginx-deployment     3/3     3            3           3m43s
```

Récupérer des informations sur votre Deployment

Comme pour les Pods et les ReplicaSets, vous pouvez **inspecter** votre objet Kubernetes à l'aide de la commande `kubectl inspect` :

```
kubectl describe deploy nginx-deployment
```

En lançant cette commande, vous récupérerez pas mal d'informations comme les spécifications de votre Deployment, et **les différents événements qu'a subis votre Deployment**.

```
Name:                   nginx-deployment
Namespace:              default
CreationTimestamp:      Sun, 25 Aug 2019 03:39:51 +0200
Labels:                 app=nginx
Annotations:            deployment.kubernetes.io/revision: 5
...
Events:
  Type    Reason             Age                    From                   Message
  ----    ------             ----                   ----                   -------
  Normal  ScalingReplicaSet  30h                    deployment-controller  Scaled u
  Normal  ScalingReplicaSet  30h                    deployment-controller  Scaled u
  Normal  ScalingReplicaSet  30h                    deployment-controller  Scaled d
```

```
  Normal  ScalingReplicaSet  30h                      deployment-controller  Scaled u
  Normal  ScalingReplicaSet  30h                      deployment-controller  Scaled d
  Normal  ScalingReplicaSet  <invalid> (x2 over 30h)  deployment-controller  Scaled u
  Normal  ScalingReplicaSet  <invalid> (x2 over 30h)  deployment-controller  Scaled d
  Normal  ScalingReplicaSet  <invalid> (x2 over 30h)  deployment-controller  Scaled u
  Normal  ScalingReplicaSet  <invalid> (x2 over 30h)  deployment-controller  Scaled d
  Normal  ScalingReplicaSet  <invalid> (x8 over 30h)  deployment-controller  (combine
```

Retour en arrière

Vous avez la possibilité de revenir sur une version précédente de votre Deployment en utilisant la méthode kubectl rollout. Dans cet exemple nous allons mettre à jour notre Deployment `nginx-deployment`, en mettant le nom de l'image des Pods de notre Deployment en nginx version 1.9.1 au lieu de la version latest :

```
kubectl set image deployment/nginx-deployment nginx=nginx:1.9.1
```

Vérifions ensuite l'historique de ce Deployment :

```
kubectl rollout history deployment/nginx-deployment
```

Résultat :

```
deployments "nginx-deployment"
REVISION    CHANGE-CAUSE
1           kubectl create -f nginx-pod.yaml
2           kubectl set image deployment/nginx-deployment nginx=nginx:1.9.1
```

Nous allons vérifier si les Pods de notre Deployment possèdent bel et bien la nouvelle version 1.9.1 de l'image nginx :

```
kubectl describe deploy nginx-deployment | grep Image
```

Et c'est bien le cas :

```
Image:    nginx:1.9.1
```

Supposons, que nous décidons maintenant d'**annuler le Deployment actuel et revenir à la révision précédente**, soit à la révision numéro 1. Pour ce faire, nous exécuterons la commande suivante :

```
kubectl rollout undo deployment/nginx-deployment --to-revision=1
```

Revérifions une nouvelle fois la version de notre image nginx :

```
kubectl describe deploy nginx-deployment | grep Image
```

Résultat :

```
Image:    nginx
```

Mise à l'échelle d'un Deployment

Vous pouvez **mettre à l'échelle un Deployment** à l'aide de la commande suivante :

```
kubectl scale deployment nginx-deployment --replicas=6
```

Résultat :

```
deployment.extensions/nginx-deployment scaled
```

Vérifions la liste des Deployments disponibles dans notre cluster :

```
kubectl get deploy
```

Résultat :

```
NAME               READY   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment   6/6     6            6           2m37
```

Vous pouvez configurer une mise à l'échelle horizontale automatique des Pods de votre Deployment et choisir le nombre minimal et maximal de Pods que vous

souhaitez exécuter en fonction de l'utilisation du processeur de vos Pods existants.

Dans cet exemple nous allons créer un autoscaler d'un Deployment avec 2 Pods au minimum et 5 Pods au maximum.

```
kubectl autoscale deployment nginx-deployment  --min=2 --max=5
```

Ce qui aura tendance à créer un objet Kubernetes de type `HorizontalPodAutoscaler`, que nous verrons dans un autre chapitre.

```
kubectl get hpa
```

Résultat :

```
NAME               REFERENCE                     TARGETS          MINPODS   MAXPODS
nginx-deployment   Deployment/nginx-deployment   <unknown>/80%    2         10
```

Comme un Deployment contient à la fois des objets de types ReplicaSets et Pods, alors vous pouvez utiliser les mêmes commandes de manipulation de ces derniers, vu dans le chapitre consacré aux ReplicaSets mais aussi le chapitre consacré aux Pods.

Supprimer votre Deployment

Voici la commande qui permet de supprimer votre Deployment :

```
kubectl delete deploy nginx-deployment
```

Résultat :

```
deployment.extensions "nginx-deployment" deleted
```

Conclusion

Le Deployment nous permet de gérer plus facilement nos Pods et nos ReplicaSets depuis un seul objet Kubernetes. Comme à mon habitude, voici un **récapitulatif des commandes liées aux Deployments**.

```
# Afficher la liste des Deployments
kubectl get deployment [En option <DEPLOYMENT NAME>]
    -o wide : récupérer en plus, le nom de l'image et le sélecteur

# Créer un Deployment
kubectl create -f <template.yaml>

# Supprimer un Deployment
kubectl delete deployment <DEPLOYMENT NAME>

# Appliquer des nouveaux changements à votre Deployment sans le détruire
kubectl apply -f <template.yaml>

# Modifier et appliquer les changements de votre Deployment instantanément sans le de
kubectl edit deployment <DEPLOYMENT NAME>

# Afficher les détails d'un Deployment
kubectl describe deployment <DEPLOYMENT NAME>

# Mettre à jour l'image des Pods de votre Deployment
kubectl set image deployment/<DEPLOYMENT NAME> <CONTAINER NAME>=<NEW IMAGE NAME>

# Afficher le status du rolling update de votre Deployment
kubectl rollout status deployment/<DEPLOYMENT NAME>

# Afficher l'historique des révisions de votre Deployment
kubectl rollout history deployment/<DEPLOYMENT NAME>

# Revenir à une version précédente
kubectl rollout undo deployment/nginx-deployment --to-revision=<REVISION NUMBER>
```