

Compte rendu du TP 2- Déploiement d'une application Nginx sur Kubernetes

Objectif : L'objectif de ce TP était de déployer une application simple utilisant Nginx sur un cluster Kubernetes. Nous avons configuré les ressources Kubernetes nécessaires, telles que des Pods, des ReplicaSets, des Deployments et un Service NodePort, pour exposer l'application en dehors du cluster.

1. Création des Pods

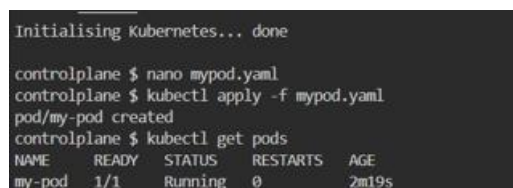
La première étape consistait à créer un Pod simple exécutant l'image Nginx. Un fichier YAML a été utilisé pour définir la configuration du Pod. Ce Pod a permis de vérifier que l'application Nginx fonctionne correctement dans le cluster Kubernetes.

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  containers:
    - name: nginx-container
      image: nginx:latest
      ports:
        - containerPort: 80
```

Après la création du Pod avec la commande `kubectl apply`, nous avons vérifié son état avec :

```
kubectl get pods
```

Le Pod était en état "Running", ce qui montre que Nginx fonctionne correctement.



```
Initialising Kubernetes... done
controlplane $ nano mypod.yaml
controlplane $ kubectl apply -f mypod.yaml
pod/my-pod created
controlplane $ kubectl get pods
NAME      READY   STATUS    RESTARTS   AGE
my-pod    1/1     Running   0           2m19s
```

2. Création d'un ReplicaSet

Ensuite, nous avons créé un ReplicaSet pour assurer la disponibilité continue de l'application avec plusieurs répliques du Pod Nginx. Un fichier YAML a été utilisé pour définir cette ressource.

apiVersion: apps/v1

kind: ReplicaSet

metadata:

name: my-replicaset

spec:

replicas: 3

selector:

matchLabels:

app: nginx

template:

metadata:

labels:

app: nginx

spec:

containers:

- name: nginx

image: nginx:latest

ports:

- containerPort: 80

Ce ReplicaSet a lancé trois répliques du Pod Nginx, garantissant la haute disponibilité de l'application.

```
controlplane $ nano myreplicaset.yaml
controlplane $ kubectl apply -f myreplicaset.yaml
replicaset.apps/my-replicaset created
controlplane $ kubectl get pods
NAME                READY   STATUS    RESTARTS   AGE
my-pod              1/1     Running   0           3m11s
my-replicaset-bltjs 1/1     Running   0           13s
my-replicaset-lvrc8 1/1     Running   0           13s
my-replicaset-p7nr7 1/1     Running   0           13s
```

3. Création d'un Deployment

Pour faciliter les mises à jour et la gestion à grande échelle, un Deployment a été créé. Ce Deployment gère les réplicas et permet des déploiements progressifs d'une nouvelle version de l'application.

```
apiVersion: apps/v1

kind: Deployment

metadata:
  name: my-deployment

spec:
  replicas: 3

  selector:
    matchLabels:
      app: nginx

  template:
    metadata:
      labels:
        app: nginx

    spec:
      containers:
        - name: nginx
          image: nginx:latest

      ports:
        - containerPort: 80
```

Le Deployment a été appliqué et nous avons vérifié le bon fonctionnement avec `kubectl get deployments`.

```
controlplane $ nano mydeployment.yaml
controlplane $ kubectl apply -f mydeployment.yaml
deployment.apps/my-deployment created
controlplane $ kubectl get deployments
NAME          READY   UP-TO-DATE   AVAILABLE   AGE
my-deployment 3/3      3            3           10s
```

4. Exposition de l'application avec un Service NodePort

Pour rendre l'application accessible depuis l'extérieur du cluster, un Service de type NodePort a été créé. Ce Service expose l'application sur un port spécifique de tous les nœuds du cluster, permettant d'accéder à l'application Nginx via un navigateur ou curl.

apiVersion: v1

kind: Service

metadata:

name: my-service

spec:

type: NodePort

selector:

app: nginx

ports:

- protocol: TCP

port: 80

targetPort: 80

nodePort: 30001

Nous avons utilisé la commande suivante pour vérifier que le Service était correctement configuré :

kubectl get svc

```
controlplane $ nano myservice.yaml
controlplane $ kubectl apply -f myservice.yaml
service/my-service created
controlplane $ kubectl get services
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)        AGE
kubernetes   ClusterIP   10.96.0.1     <none>        443/TCP        20d
my-service   NodePort    10.109.120.11 <none>        80:30001/TCP   9s
```



```
controlplane $ kubectl get svc
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)        AGE
kubernetes   ClusterIP   10.96.0.1     <none>        443/TCP        20d
my-service   NodePort    10.109.120.11 <none>        80:30001/TCP   15m
```

5. Test de l'application

Enfin, l'application a été testée à l'aide de curl pour vérifier l'accès localement sur le cluster Kubernetes :

curl <http://localhost:30001/>

```
controlplane $ curl http://localhost:30001/
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

Cela a renvoyé la page d'accueil de Nginx, confirmant que l'application est correctement déployée et accessible.

Conclusion

Ce TP a permis de comprendre le processus de déploiement d'une application sur Kubernetes, ainsi que la gestion des réplicas et l'exposition de services. Les différentes étapes ont été réalisées avec succès, et l'application Nginx est accessible via un Service NodePort.