Lab 4 : Communiquer avec les Pods via les Services ClusterIP et NodePort

À cette étape, la seule solution étudiée pour communiquer avec un `Pod` depuis l'extérieur de notre cluser K8s est d'utiliser la redirection de port avec l'outil **kubectl** et l'option `port-forward` (vue dans le Lab2). Toutefois, cette solution n'est pas envisageable pour une mise en production puisqu'elle nécessite l'accès au composant *API Server* (réservé à l'administrateur et au développeur) et surtout elle ne permet d'accéder qu'à un seul `Pod` à la fois. Ce dernier point est gênant puisque depuis le Lab3 nous avons appris à créer plusieurs `Pods` basés sur un même template.

Ce quatrième exercice adresse deux problèmes d'accès aux `Pods`. Le premier s'intéresse aux besoins de communication entre des `Pods` depuis l'intérieur du cluster K8s et le second s'intéresse aux besoins de communication depuis l'extérieur du cluster K8s vers des `Pods`. La solution technique proposée par Kubernetes est d'utiliser des objets de type `Service`. Il en existe plusieurs sortes et les principaux sont `ClusterIP`, `NodePort` et `LoadBalancer`. Nous allons étudier les deux premiers car le dernier `LoadBalancer` nécessite que le cluster K8s soit déployé dans un Cloud public.

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 une configuration `Service` de type `ClusterIP`
- Réaliser une communication entre deux `Pods`
- Écrire une configuration `Service` de type `NodePort`
- Réaliser une communication depuis l'extérieur du cluster K8s

Étapes à suivre

- Avant de commencer les étapes de cet exercice, assurez-vous que le `Namespace` créé dans l'exercice précédent `mynamespaceexercice2` soit supprimé.

```
$ kubectl delete namespace mynamespaceexercice2
namespace "mynamespaceexercice2" deleted
```

- Créer dans le répertoire *exercice3-service-clusterip-nodeport/* un fichier appelé *mynamespaceexercice3.yaml* en ajoutant le contenu suivant :

```
apiVersion: v1
kind: Namespace
metadata:
  name: mynamespaceexercice3
```

- Créer ce `Namespace` dans notre cluster :

```
$ kubectl apply -f exercice3-service-clusterip-
nodeport/mynamespaceexercice3.yaml
namespace/mynamespaceexercice3 created
```

- Créer dans le répertoire *exercice3-service-clusterip-nodeport/* un fichier appelé *mydeploymentforservice.yaml* qui décrit un `Deployment`. Ce dernier contiendra trois `Pods` basés sur l'image Docker Nginx :

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mydeploymentforservice
spec:
  replicas: 3
  selector:
    matchLabels:
      app: mypod
  template:
    metadata:
      labels:
        app: mypod
    spec:
      containers:
      - name: mycontainer
        image: nginx:latest
        ports:
        - containerPort: 80
        lifecycle:
          postStart:
            exec:
              command: ["/bin/sh", "-c", "echo $HOSTNAME >
/usr/share/nginx/html/index.html"]
```

Le paramètre `lifecycle` gère le cycle de vie du démarrage d'un conteneur. Dans notre cas, lorsque le conteneur démarre, une commande sera exécutée et modifiera le fichier *index.html*. Cela nous permettra de connaître le `Pod` qui répond à nos requêtes puisque le contenu du fichier *index.html* contient le nom du conteneur (`$HOSTNAME`).

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

```
$ kubectl apply -f exercice3-service-clusterip-
nodeport/mydeploymentforservice.yaml -n mynamespaceexercice3
deployment.apps/mydeploymentforservice created
```

- Pour vérifier que la page web par défaut de chaque `Pod` a été modifiée (penser à adapter la seconde commande en fonction des noms des `Pods` retournés par la première commande) :

```
$ kubectl get pod -n mynamespaceexercice3
NAME                                    READY   STATUS    RESTARTS   AGE
mydeploymentforservice-7f4dfcd55c-xrqz9   1/1     Running   0          3m27s
mydeploymentforservice-7f4dfcd55c-bsc58   1/1     Running   0          3m27s
mydeploymentforservice-7f4dfcd55c-8rctf   1/1     Running   0          3m27s
```

```
$ kubectl exec -it -n mynamespaceexercice3 mydeploymentforservice-7f4dfcd55c-xrqz9 -- more /usr/share/nginx/html/index.html
mydeploymentforservice-7f4dfcd55c-xrqz9
```

La première commande affiche la liste des `Pods` depuis notre `Namespace`. En considérant le premier `Pod` de la liste, nous exécutons une commande sur le conteneur pour afficher le contenu du fichier */usr/share/nginx/html/index.html*. Si le résultat est similaire à `mydeploymentforservice-XXXXXXXXX-YYYYY`, c'est que la commande au démarrage du conteneur a été correctement exécutée.

Nous allons créer notre premier `Service` de type `ClusterIP` qui est le `Service` de base dans K8s. `ClusterIP` est un `Service` accessible uniquement à l'intérieur d'un cluster. Il expose un `Service` sur une IP et un CNAME (Canonical Name) et distribue les requêtes vers l'adresse IP d'un `Pod`. S'il existe plusieurs `Pods`, le `Service ClusterIP` distribue aléatoirement les requêtes vers les `Pods`. L'utilisation du nom du `Service` impose que les `Pods` soient dans le même `Namespace`.

- Créer dans le répertoire *exercice3-service-clusterip-nodeport/* un fichier appelé *myclusteripservice.yaml* qui décrit un `Service` de type `ClusterIP` :

```
apiVersion: v1
kind: Service
metadata:
  name: myclusteripservice
spec:
  selector:
    app: mypod
  type: ClusterIP
  ports:
    - protocol: TCP
      targetPort: 80
      port: 8080
```

Le `Service` va rediriger les requêtes reçues vers les `Pods` identifiés par le paramètre `selector`. Ce `Service` devra être créé dans le même `Namespace` que les `Pods` du `Deployment mydeploymentforservice`. Si ce n'était pas le cas, aucun `Pod` ne serait identifié. Le paramètre `targetPort 80` correspond au port utilisé par les `Pods`. Le paramètre `port 8080` précise le port pour communiquer avec le `Service`. Si vous êtes familiarisé avec Docker, c'est sensiblement équivalent à la redirection de port par le paramètre `-p port:targetPort`.

- Appliquer cette configuration de `Service` dans le cluster Kubernetes :

```
$ kubectl apply -f exercice3-service-clusterip-nodeport/myclusteripservice.yaml -n mynamespaceexercice3
service/myclusteripservice created
```

- Afficher la liste des `Services` du `Namespace mynamespaceexercice3`:

```
$ kubectl get service -n mynamespaceexercice3 -o wide
NAME                TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE   SELECTOR
myclusteripservice  ClusterIP   10.43.173.137   <none>        8080/TCP   7s    app=mypod
```

Le `Service myclusteripservice` est disponible. L'accès à ce `Service` se fera via l'IP `10.43.173.137` ou via le CNAME `myclusteripservice`.

Pour tester notre `Service` qui va distribuer des requêtes aux `Pods` de notre `Deployment`, nous allons créer un `Pod` de test basé sur l'image Docker Alpine (penser à adapter l'adresse IP du `ClusterIP`) :

```
$ kubectl run podtest --image=alpine:latest -- /bin/sh -c "while true; do
wget -qO- 10.43.173.137:8080; sleep 1; done"
pod/podtest created

$ kubectl logs podtest -f
mydeploymentforservice-6bb797546-fh28g
mydeploymentforservice-6bb797546-lx28c
mydeploymentforservice-6bb797546-fh28g
mydeploymentforservice-6bb797546-nld6h
mydeploymentforservice-6bb797546-lx28c
mydeploymentforservice-6bb797546-lx28c
...
CTRL+C
```

Nous constatons que l'accès au `Service` via son IP permet de distribuer aléatoirement la requête sur les trois `Pods`.

- Essayons maintenant d'utiliser le nom (CNAME) du `Service` à la place de son IP pour l'identifier :

```
$ kubectl delete pod podtest
pod "podtest" deleted

$ kubectl run podtest --image=alpine:latest -- /bin/sh -c "while true; do
wget -qO- myclusteripservice:8080; sleep 1; done"
pod/podtest created

$ kubectl logs podtest -f
wget: bad address 'myclusteripservice:8080'
wget: bad address 'myclusteripservice:8080'
```

Le nom du `Service` (via son CNAME) ne peut être utilisé que si les `Pods` qui souhaitent communiquer sont dans le même `Namespace` que ce `Service`.

- Corrigeons ce problème en créant le `Pod` de test dans le même `Namespace` que notre `Deployment`:

```
$ kubectl delete pod podtest

$ kubectl run podtest -n mynamespaceexercice3 --image=alpine:latest --
/bin/sh -c "while true; do wget -qO- myclusteripservice:8080; sleep 1; done"
```

```
$ kubectl logs podtest -n mynamespaceexercice3 -f
mydeploymentforservice-6bb797546-lx28c
mydeploymentforservice-6bb797546-lx28c
mydeploymentforservice-6bb797546-lx28c
mydeploymentforservice-6bb797546-nld6h
mydeploymentforservice-6bb797546-lx28c
...
CTRL+C
```

Vous aurez deviné qu'il est plus pratique d'utiliser le nom (CNAME) que l'IP pour identifier un `Service`. Toutefois, il faudra faire attention d'être dans le même `Namespace`.

Un `Service` de type `ClusterIP` est accessible uniquement à l'intérieur d'un cluster. Par conséquent, on peut accéder à ce `Service` depuis les nœuds (machine virtuelle) de notre cluster uniquement via une identification par l'IP (penser à adapter l'adresse IP du `ClusterIP`) :

Via K3d

```
$ docker exec -it k3d-mycluster-server-0 wget -qO- 10.43.173.137:8080
mydeploymentforservice-6bb797546-lx28c
```

Si vous souhaitez cependant accéder à ce `Service` depuis l'extérieur, il y a une solution, mais **elle ne doit être utilisée qu'à des fins de tests**. Cette solution, basée sur l'utilisation de l'outil **kubectl**, permet de créer un `proxy` entre la machine locale et le cluster.

- Depuis l'invite de commande *kubectl* :

```
$ kubectl proxy --port=8080
Starting to serve on 127.0.0.1:8080
```

L'option `proxy` permet de créer un proxy entre la machine locale (depuis le port 8080) et le cluster. À l'éxécution de cette commande, le processus **kubectl** devient bloquant. Toutes les opérations suivantes se feront par l'intermédiaire de l'outil cURL pour interroger l'API Rest fournie par le cluster Kubernetes. Nous pourrons obtenir des informations sur les `Pods`, `Namespaces`, `Deployment` et les `Services`. Avec ces derniers, on pourra envoyer des requêtes.

- Ouvrir une nouvelle invite de commande :

```
$ curl -s http://localhost:8080/api/v1/namespaces | jq -r
'.items[].metadata.name'
default
kube-system
kube-public
kube-node-lease
mynamespaceexercice3
```

Par exemple, avec cette requête, nous obtenons tous les `Namespaces` du cluster Kubernetes.

```
$ curl
http://localhost:8080/api/v1/namespaces/mynamespaceexercice3/services/myclu
steripservice:8080/proxy/
mydeploymentforservice-6bb797546-nld6h
```

Avec cette commande, nous envoyons une requête au `Service` qui distribuera aléatoirement à tous les `Pods` de notre `Deployment`. À noter que l'identifiant du `Service` ne peut pas être son IP, il faut obligatoirement que le nom du `Service` soit utilisé (`myclusteripservice`). Cette solution est pratique si vous souhaitez interagir avec vos `Pods` sans vouloir les exposer.

- Avant de continuer, stopper le proxy (CTRL+C), supprimer le `Pod podtest` et supprimer le `Service myclusteripservice`.

```
$ kubectl delete pods -n mynamespaceexercice3 podtest
pod "podtest" deleted
$ kubectl delete service -n mynamespaceexercice3 myclusteripservice
service "myclusteripservice" deleted
```

Nous allons maintenant nous intéresser aux `Services` de type `NodePort` qui contrairement à `ClusterIP` sont accessibles depuis l'extérieur du cluster K8s. Le principe de fonctionnement du `Service NodePort` est en deux temps. Dans un premier temps, ce `Service` est exposé sur une IP et un CNAME (Canonical Name) puis distribue les requêtes vers l'adresse IP d'un `Pod` (identique à `ClusterIP`). Un `Service NodePort` s'appuie donc sur un `Service ClusterIP`. Dans un second temps, ce `Service` est exposé sur l'IP de chaque nœud à un port statique (appelé `nodePort`). La plage du `nodePort` est comprise entre `30000` et `32767`, cela laisse de la marge, mais n'est pas illimité. Ainsi, pour accéder à un `Service NodePort` depuis l'extérieur, il n'y aura plus qu'à requêter l'adresse `<IP Nœud>:<nodePort>`.

- Créer dans le répertoire *exercice3-service-clusterip-nodeport/* un fichier appelé *mynodeportservice.yaml* qui décrit un `Service` de type `ClusterIP` :

```
apiVersion: v1
kind: Service
metadata:
  name: mynodeportservice
spec:
  selector:
    app: mypod
  type: NodePort
  ports:
    - protocol: TCP
      targetPort: 80
      port: 8080
      nodePort: 30001
```

Le `Service` va rediriger les requêtes reçues vers les `Pods` identifiés par le paramètre `selector`. Ce `Service` sera créé dans le même `Namespace` que les `Pods` du `Deployment mydeploymentforservice`. Si ce n'était pas le cas, aucun `Pod` ne serait identifié. Les paramètres `targetPort` et `port` sont identiques à ceux utilisés pour le `Service` de type `ClusterIP`. Le port défini par le paramètre `nodePort` est celui qui sera exposé au niveau de chaque nœud du cluster Kubernetes.

- Appliquer cette configuration de `Service` dans le cluster Kubernetes :

```
$ kubectl apply -f exercice3-service-clusterip-
nodeport/mynodeportservice.yaml -n mynamespaceexercice3
service/mynodeportservice created
```

- Afficher la liste des `Services` du `Namespace` `mynamespaceexercice3`:

```
$ kubectl get service -n mynamespaceexercice3 -o wide
NAME                TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE   SELECTOR
mynodeportservice   NodePort   10.43.145.109   <none>        8080:30001/TCP   35s   app=mypod
```

Nous constatons que ce `Service NodePort` est bien basé sur un `Service ClusterIP` puisqu'une IP interne a été définie `10.43.192.164`. La valeur du `nodePort` se retrouve dans la colonne `PORT(s)`. Le `Service NodePort` va donc recevoir une requête sur le port `30001` qu'il va retourner au `Service ClusterIP` sur le port `8080` qui à son tour va distribuer aléatoirement sur tous les `Pods`.

- Depuis l'invite de commande *kubectl* :

Via K3d

- Lister l'ensemble des conteneurs disponibles :

```
$ docker ps
CONTAINER ID   IMAGE                                  COMMAND                  CREATED        STATUS
PORTS                             NAMES
218cfd215045   ghcr.io/k3d-io/k3d-tools:5.4.7   "/app/k3d-tools noop"     4 hours ago    Up 4
hours                                  k3d-mycluster-tools
7e507d8c048f   ghcr.io/k3d-io/k3d-proxy:5.4.7   "/bin/sh -c nginx-pr…"    4 hours ago    Up 4
hours   80/tcp, 0.0.0.0:61868->6443/tcp   k3d-mycluster-serverlb
fc1b05755fa1   rancher/k3s:v1.25.6-k3s1          "/bin/k3d-entrypoint…"    4 hours ago    Up 4
hours                                  k3d-mycluster-agent-1
881b02b75046   rancher/k3s:v1.25.6-k3s1          "/bin/k3d-entrypoint…"    4 hours ago    Up 4
hours                                  k3d-mycluster-agent-0
ccb827ca9afd   rancher/k3s:v1.25.6-k3s1          "/bin/k3d-entrypoint…"    4 hours ago    Up 4
hours                                  k3d-mycluster-server-0
```

- Actuellement aucun conteneur du cluster K8s ne peut répondre à une requête sur le port `30001`. Modifier le cluster K3d afin d'ajouter l'écoute sur ce port :

```
$ k3d cluster edit mycluster --port-add 30001:30001@server:0
INFO[0000] Renaming existing node k3d-mycluster-serverlb to k3d-mycluster-serverlb-AtDbL...
INFO[0000] Creating new node k3d-mycluster-serverlb...
INFO[0000] Stopping existing node k3d-mycluster-serverlb-AtDbL...
INFO[0010] Starting new node k3d-mycluster-serverlb...
INFO[0010] Starting Node 'k3d-mycluster-serverlb'
INFO[0017] Deleting old node k3d-mycluster-serverlb-AtDbL...
INFO[0017] Successfully updated mycluster
```

```
$ docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED             STATUS
PORTS                                                NAMES
71daed0bd9cd   ef33158baf49                      "/bin/sh -c nginx-pr…"   About a minute ago   Up
About a minute    80/tcp, 0.0.0.0:30001->30001/tcp, 0.0.0.0:61868->6443/tcp    k3d-mycluster-
serverlb
218cfd215045   ghcr.io/k3d-io/k3d-tools:5.4.7   "/app/k3d-tools noop"    4 hours ago         Up
4 hours                                                  k3d-mycluster-tools
fc1b05755fa1   rancher/k3s:v1.25.6-k3s1         "/bin/k3d-entrypoint…"   4 hours ago         Up
4 hours                                                  k3d-mycluster-
agent-1
881b02b75046   rancher/k3s:v1.25.6-k3s1         "/bin/k3d-entrypoint…"   4 hours ago         Up
4 hours                                                  k3d-mycluster-
agent-0
ccb827ca9afd   rancher/k3s:v1.25.6-k3s1         "/bin/k3d-entrypoint…"   4 hours ago         Up
4 hours                                                  k3d-mycluster-
server-0
```

Le port `30001` du cluster K8s est maintenant exposé sur le port `30001` du poste de développeur.

- Toutes requêtes sur le port `30001` du poste du développeur est transférée vers le `Service NodePort` :

```
$ curl localhost:30001
mydeploymentforservice-6bb797546-fh28g
```

Pour information, il est possible de combiner dans un même fichier plusieurs configurations. Dans l'exemple qui va suivre, nous allons combiner le précédent `Deployment` et le `Service` de type `NodePort`.

- Créer dans le répertoire *exercice3-service-clusterip-nodeport/* un fichier appelé *mydeploymentwithservice.yaml* qui décrit dans un même fichier un `Deployment` et un `Service` de type `NodePort` :

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mydeploymentforservice
spec:
  replicas: 3
  selector:
    matchLabels:
      app: mypod
  template:
    metadata:
      labels:
        app: mypod
    spec:
      containers:
      - name: mycontainer
        image: nginx:latest
        ports:
        - containerPort: 80
        lifecycle:
          postStart:
            exec:
              command:    ["/bin/sh",    "-c",    "echo    $HOSTNAME    >
/usr/share/nginx/html/index.html"]

---

apiVersion: v1
kind: Service
metadata:
  name: mynodeportservice
spec:
  selector:
    app: mypod
  type: NodePort
  ports:
    - protocol: TCP
      targetPort: 80
      port: 8080
      nodePort: 30001
```

- Appliquer ces deux configurations dans le cluster Kubernetes :

```
$ kubectl apply -f exercice3-service-clusterip-
nodeport/mydeploymentwithservice.yaml -n mynamespaceexercice3
deployment.apps/mydeploymentforservice unchanged
service/mynodeportservice unchanged
```

Comme les identifiants sont les mêmes que ceux utilisés pendant l'exercice, aucun changement n'a été découvert par K8s.

Bilan de l'exercice

À cette étape, vous savez :

- créer des `Services` de type `ClusterIP` et `NodePort` ;
- faire la différence entre les deux types de `Services` ;
- mettre en place un proxy pour interroger un cluster ;
- créer des configurations avec plusieurs objets Kubernetes.

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` ;
- créer un `Service` de type `ClusterIP` pour ce `Deployment` ;
- créer un `Service` de type `NodePort` pour ce `Deployment`.