Université de Picardie Jules Verne

Informatique – Master CCM

INSSET – Saint-Quentin

Conteneurs Applicatifs et Micro-Services M2

C. Drocourt

cyril.drocourt@u-picardie.fr

Cours 5 : Kubernetes - Services

V2023.02

Table des matières

1 - Introduction

Un déploiement va créer des « pods » qui possèdent chacun une adresse IP, qui peuvent naître et mourir, et donc changer d'adresses IPs. Par conséquent il n'est pas possible de les utiliser directement, mais il faut mettre en place la notion de service.

Un service va définir l'accès à l'application et donc aux « pods », en se basant sur la notion de « label », il existe plusieurs types d'accès réseau associés aux services :

- **ClusterIP** (default) : Expose le service sur une IP interne du cluster, et n'est donc accessible que depuis l'intérieur du Cluster, permet aussi d'utiliser un nom.
- **NodePort** : Expose le service sur le même port sur chaque nœud du cluster en utilisant le NAT. Le service est accessible depuis l'extérieur du cluster en utilisant <NodeIP>:<NodePort>. Surcouche de « ClusterIP ».

- **LoadBalancer** : Va créer un Load Balancer associé à une IP accessible depuis l'extérieur du cluster. Surcouche du « NodeIP ».
- **ExternalName** : Expose le service en utilisant un nom DNS arbitraire (externalName du champs « spec ») en utilisant un CNAME (≥ 1.7 de kube-dns).

2 - Service ClusterIP

Comme pour les « Pods » et les « Deployments », il est possible de demander à générer un fichier YAML :

```
root@maitre:~# kubectl expose deployment my-deploy --port 80 --dry-run -o yaml
```

Un service associé à l'application précédente pourrait être :

```
root@maitre:~# cat monservice1.yaml
apiVersion: v1
kind: Service
metadata:
  name: monservice1
  labels:
    run: monservice1
```

```
spec:
  ports:
  - port: 3080
    protocol: TCP
    targetPort: 80
  selector:
    run: monnginx
```

La description des différents ports est la suivante :

- ports : Les ports utilisés par le ClusterIP,
- targetPort : Le port utilisé par les pods,

Pour le créer :

```
root@maitre:~# kubectl create -f monservice1.yaml
service/monservice created
```

On vérifie en listant les services actuels :

```
root@maitre:~# kubectl get services
```

```
NAME          TYPE         CLUSTER-IP    EXTERNAL-IP   PORT(S)
AGE
kubernetes    ClusterIP    10.96.0.1     <none>        443/TCP
25h
monservice1 ClusterIP   10.101.5.33   <none>       3080/TCP   7s
```

On peut obtenir des informations supplémentaires :

```
root@maitre:~# kubectl describe  services monservice1
Name:              monservice1
Namespace:         default
Labels:            run=monservice1
Annotations:       <none>
Selector:          run=monnginx
Type:              ClusterIP
IP:                10.101.5.33
Port:              <unset>  3080/TCP
TargetPort:        80/TCP
Endpoints:         10.244.1.3:80,10.244.2.4:80
Session Affinity:  None
Events:            <none>
```

Et faire une requête sur le cluster :

```
root@maitre:~# curl -I 10.101.5.33:3080
HTTP/1.1 200 OK
Server: nginx/1.15.6
Date: Thu, 15 Nov 2018 09:21:20 GMT
Content-Type: text/html
Content-Length: 612
Last-Modified: Tue, 06 Nov 2018 13:32:09 GMT
Connection: keep-alive
ETag: "5be197d9-264"
Accept-Ranges: bytes
```

Comme nous avons créé le service après les pods, ils ne connaissent pas son existence :

```
root@maitre:~# kubectl exec monnginx-6d88c97b4c-pw8wk --
printenv | grep SERVICE
KUBERNETES_SERVICE_PORT=443
KUBERNETES_SERVICE_PORT_HTTPS=443
KUBERNETES_SERVICE_HOST=10.96.0.1
```

Nous pouvons y remédier en passant le nombre de pods à 0, puis en le repassant à 2, de la manière suivante :

```
root@maitre:~# kubectl scale deployment monnginx --replicas=0; kubectl scale deployment monnginx --replicas=2;
deployment.extensions/monnginx scaled
deployment.extensions/monnginx scaled
```

On vérifie maintenant :

```
root@maitre:~# kubectl exec monnginx-6d88c97b4c-87hb5 --printenv | grep SERVICE
KUBERNETES_SERVICE_HOST=10.96.0.1
KUBERNETES_SERVICE_PORT=443
KUBERNETES_SERVICE_PORT_HTTPS=443
MONSERVICE_PORT=tcp://10.101.5.33:3080
MONSERVICE_PORT_3080_TCP_PROTO=tcp
MONSERVICE_PORT_3080_TCP_ADDR=10.101.5.33
MONSERVICE_SERVICE_PORT=3080
MONSERVICE_PORT_3080_TCP_PORT=3080
MONSERVICE_PORT_3080_TCP=tcp://10.101.5.33:3080
MONSERVICE_SERVICE_HOST=10.101.5.33
```

3 - Le DNS avec « kube-dns »

Le service est également accessible via un nom DNS géré par le service kube-dns :

```
root@lemaitre:~# kubectl get services kube-dns --namespace=kube-system
NAME         TYPE          CLUSTER-IP   EXTERNAL-IP   PORT(S)
AGE
kube-dns   ClusterIP   10.96.0.10   <none>        53/UDP,53/TCP
26h
```

Le nom donné est aussi connu à l'intérieur des « pods » :

```
root@maitr:~# kubectl exec my-nginx -- curl -sI monservice1:3080
HTTP/1.1 200 OK
Server: nginx/1.25.3
Date: Wed, 13 Dec 2023 14:23:00 GMT
Content-Type: text/html
Content-Length: 615
Last-Modified: Tue, 24 Oct 2023 13:46:47 GMT
Connection: keep-alive
...
```

Il est donc possible d'utiliser le nom dans un fichier de description « yml », par exemple pour l'utilisation d'une base de donnée !!

4 - Service NodePort

```
root@maitre:~# cat monservice2.yaml
apiVersion: v1
kind: Service
metadata:
  name: monservice2
  labels:
    run: monservice2
spec:
  type: NodePort
  ports:
  - port: 5080
    protocol: TCP
    targetPort: 80
  selector:
    run: monnginx
```

Puis on peut créer le service :

```
root@maitre:~# kubectl create -f monservice2.yaml
service/monservice1 created
```

On peut à vérifier :

```
root@maitre:~# kubectl get services monservice1
NAME         TYPE        CLUSTER-IP    EXTERNAL-IP    PORT(S)          AGE
monservice1 NodePort     10.96.46.172   <none>      5080:30397/TCP   60s
```

Donc on peut accéder via chaque node sur le port 30397 :

```
root@maitre:~# curl -I <IP>:30397
HTTP/1.1 200 OK
Server: nginx/1.15.6
Date: Thu, 15 Nov 2018 09:00:44 GMT
Content-Type: text/html
Content-Length: 612
Last-Modified: Tue, 06 Nov 2018 13:32:09 GMT
Connection: keep-alive
ETag: "5be197d9-264"
Accept-Ranges: bytes
```

Il est possible de spécifier un port entre dans la plage de ports autorisés de 30000 et 32767 à l'aide de la directive « nodePort: XXXXX » dans la section « ports » des spécifications « spec ».

5 - Service LoadBalancer

Il faut modifier le fichier de la manière suivante :

```
root@maitre:~# cat monservice3.yaml
apiVersion: v1
kind: Service
metadata:
  name: monservice3
  labels:
    run: monservice3
spec:
  type: LoadBalancer
  externalIPs:
  - 10.3.134.130
  ports:
  - port: 6080
    protocol: TCP
    targetPort: 80
  selector:
    run: monnginx
```

Puis on peut créer le service :

```
root@maitre:~# kubectl create -f monservice3.yaml
service/monservice2 created
```

On vérifie :

```
root@maitre:~# kubectl get services monservice3
NAME           TYPE           CLUSTER-IP     EXTERNAL-IP  PORT(S)
AGE
monservice3      LoadBalancer      10.97.192.197      10.3.134.130
6080:30477/TCP 8s
```

Si le LoadBalancer n'a pas réussi à choisir la bonne adresse IP (external IP en <pending>), il faut lui préciser. Pour cela nous allons éditer le service :

```
root@maitre:~# kubectl edit svc monservice3
...
  type: LoadBalancer
  externalIPs:
    - 10.3.134.130
status:
  loadBalancer: {}
...
```

On peut à nouveau vérifier :

```
root@maitre:~# kubectl get services monservice3
NAME        TYPE         CLUSTER-IP    EXTERNAL-IP    PORT(S)        AGE
monservice2 LoadBalancer 10.97.192.197 10.3.134.130 6080:30477/TCP 6m52s
```

Et faire une requête sur le LoadBalancer :

```
root@maitre:~# curl -I 10.3.134.130:6080
HTTP/1.1 200 OK
Server: nginx/1.15.6
Date: Thu, 15 Nov 2018 10:00:44 GMT
Content-Type: text/html
Content-Length: 612
Last-Modified: Tue, 06 Nov 2018 13:32:09 GMT
Connection: keep-alive
ETag: "5be197d9-264"
Accept-Ranges: bytes
```