FILTRER LES SORTIES KUBERNETES

Introduction

Nous avons vu jusqu'ici, comment récupérer des informations sur vos ressources Kubernetes en utilisant soit la commande :

- `kubectl describe <RESOURCE_TYPE> [RESOURCE NAME]` : affiche une description détaillée des ressources sélectionnées, y compris les ressources connexes telles que les événements ou les contrôleurs. Vous pouvez sélectionner un seul objet par son nom, tous les objets de ce type, fournir un préfixe de nom ou un label selector.
- `kubectl get <RESOURCE_TYPE> [RESOURCE NAME]` : affiche dans un tableau les informations les plus importantes sur les ressources spécifiées. En spécifiant l'option `-o` qui prend en compte **différents formats de sorties**, on peut afficher les sorties de la commande selon le format souhaité.

Cependant, si vous souhaitez afficher ou **récupérer quelques informations spécifiques** sur vos ressources Kubernetes pour par exemple les réutiliser dans les variables de vos scripts ou bien les réutiliser dans votre propre outil de monitoring, dans ce cas, il va falloir **filtrer les sorties** de la commande `kubectl get` vu plus haut. Heureusement, pour nous que l'équipe kubernetes a pensé à nous, car il nous fournisse quelques options pour **filtrer les résultats** de cette commande. Nous allons donc voir dans ce chapitre les **différentes méthodes de filtrage** proposées par le projet Kubernetes.

Filtrage

jsonpath

À cet instant, notre but est de récupérer le nom des images des conteneurs de nos pods, en utilisant seulement la commande `kubectl get` ainsi que ses options. Pour le moment, lançons cette commande sans aucune option afin de lister nos différents pods.

```
kubectl get pods
```

Résultat :

```
apache-server   1/1       Running   0          21s
nginx-server    1/1       Running   0          1m
```

D'après le résultat, nous avons donc deux pods tournant sur notre cluster. Nous allons à présent, afficher encore plus d'informations sous le **format json** en exploitant l'option `-o` :

```
kubectl get pods -o json
```

Résultat :

```
{    "apiVersion": "v1",
    "items": [

    ...
    ...

    ],
    "kind": "List",
    "metadata": {
        "resourceVersion": "",
        "selfLink": ""
    }
}
```

Ensuite, nous allons utiliser la commande `grep` afin de vérifier si la clé `image` éxiste :

```
kubectl get pods -o json | grep -i image
```

Résultat :

```
"image": "httpd",
"imagePullPolicy": "IfNotPresent",
"image": "httpd:latest",
"imageID": "docker-pullable://httpd@sha256:ac6594daaa934c4c6ba66c562e96f2fb12f871415a9
"image": "nginx",
"imagePullPolicy": "IfNotPresent",
"image": "nginx:latest",
"imageID": "docker-pullable://nginx@sha256:50cf965a6e08ec5784009d0fccb380fc479826b6e0e
```

Nous avons bel et bien une clé nommée `image` que nous allons tenter de récupérer grâce à un format nommé `JSONPath`. Ce type de format va nous servir pour **filter le résultat de notre commande**.

On peut remarquer sur le résultat précédent au format json, que la première clé contenant une liste (reconnaissable grâce aux crochets `[]`) se nomme `items`. Nous allons donc afficher le contenu de cette liste :

```
kubectl get pods -o jsonpath='{.items[*]'}
```

Résultat :

```
map[apiVersion:v1 kind:Pod metadata:map[labels:map[run:apache-server] name:apache-serv
... image:nginx:latest]] hostIP:172.17.0.79 phase:Running]]
```

La encore nous avons encore énormément d'informations. Il va falloir donc affiner encore plus nos recherches. Nous allons pour le moment filtrer que le premier pod de notre cluster en remplaçant `*` par `0` (0 étant le premier index de notre liste) :

```
kubectl get pods -o jsonpath='{.items[0]'}
```

Résultat :

```
map[status:map[containerStatuses:[map[lastState:map[] name:apache-server ready:true re
...
image:httpd imagePullPolicy:IfNotPresent name:apache-server
...
```

Comme annoncé plus haut, nous allons encore plus affiner nos recherches en recherchant la map (reconnaissable grâce à l'opérateur `map`) `spec` :

```
kubectl get pods -o jsonpath='{.items[0]'.spec}
```

Résultat :

```
map[priority:0 restartPolicy:Always terminationGracePeriodSeconds:30 containers:[map[i
...
```

Nous remarquons dans le résultat, qu'il existe une autre map nommée `containers` . Ceci est intéressant car on peut très vite remarquer que la map `spec` comporte une arborescence identique à celle de notre fichier manifest YAML, qui dans mon cas ressemble à cela :

```
apiVersion: v1
kind: Pod
metadata:
  labels:
  run: apache-server
```

```
  name: apache-server
spec:
  containers:
  - image: httpd
    name: apache-server
```

Nous allons donc suivre alors la même arborescence à savoir **spec (type : list) > containers (type : list) > image (type: string)** soit :

```
kubectl get pods -o jsonpath='{.items[0].spec.containers[*].image}'
```

Résultat :

```
httpdmaster
```

Le but est presque atteint, la dernière étape consiste à lister maintenant le nom de l'image des conteneurs de **TOUS** nos pods. Pour cela, rien de compliqué, il suffit juste de remplacer `item[0]` par `item[*]` :

```
kubectl get pods -o jsonpath='{.items[*].spec.containers[*].image}'
```

Résultat :

```
httpd nginx
```

Exércice :

Je vais vous complexifier un peu la tâche, on vous donnant un nouvel objectif : ***"Récupérer les noms des nodes et leur capacité cpu avec un saut à la ligne"***. Le résultat doit donc ressembler à ceci :

```
master node01
4      4
```

Conseil

Pour récupérer la capacité cpu regarder sur la map `status`

1

2

3

...

Solution !

```
kubectl get nodes -o jsonpath='{.items[*].metadata.name}{"\n"}{.items[*].status.capaci
```

Les boucles

Vous pouvez aussi utiliser les opérateurs `range` et `end` pour **itérer sur des listes**. Cette démarche va vous permettre de mieux manipuler les élements de votre liste. Nous allons dans cet exemple réutiliser l'exemple précédent mais cette fois-ci avec les boucles, de façon à afficher le nom des nœuds sur la premiere colonne et la capacité cpu sur la deuxième colonne :

```
kubectl get nodes -o jsonpath='{range .items[*]}{.metadata.name}{"\t"}{.status.capacit
```

Résultat :

```
master  4
node01  4
```

custom-columns

Le format `custom-columns` comme son nom l'indique permet d'afficher un

tableau en utilisant une **liste de colonnes personnalisées** séparées par des virgules et contenant comme valeur forme d'une expression JSONPath. Nous allons dans cette-fois-ci afficher sous forme de tableau que le nom d'un Persistent-Volume ainsi que sa capacité de stockage. Ce qui nous donnera la commande ci-dessous :

```
kubectl get pv -o=custom-columns=NAME:.metadata.name,CAPACITY:.spec.capacity.storage
```

Résultat :

```
NAME       CAPACITY
pv-log-4   40Mi
pv-log-1   100Mi
pv-log-2   200Mi
pv-log-3   300Mi
```

sort-by

L'option `--sort-by` permet de **trier vos ressources dans l'ordre croissant**, il prend comme valeur la forme d'une expression JSONPath. Nous allons réutiliser l'exemple précedent en rajoutant l'option `--sort-by` de manière à trier les PVs dans l'ordre croissant selon leur capacité de stockage.

```
kubectl get pv --sort-by=.spec.capacity.storage  -o=custom-columns=NAME:.metadata.name
```

Résultat :

```
NAME       CAPACITY
pv-log-4   40Mi
pv-log-1   100Mi
pv-log-2   200Mi
pv-log-3   300Mi
```