LA SUPERVISION DANS KUBERNETES

Introduction

Dans ce chapitre nous allons parler du monitoring d'un cluster Kubernetes. Nous allons plus précisément voir comment il est possible de surveiller la consommation des ressources sur Kubernetes.

Il est possible avec kubernetes de **surveiller les métriques au niveau des nœuds et pods**. On peut par exemple inspecter le nombre de nœuds/pods dans le cluster, le nombre de nœuds/pods sains ainsi que les métriques de performances telles que la consommation de processeur, utilisation de la mémoire, du réseau et du disque.

Nous aurons besoin d'une solution qui surveillera toutes ces métriques et les stockera dans une base de données, tout en fournissant des analyses statistiques autour de ces données. Par défaut, Kubernetes ne propose pas de solution de monitoring intégrée complète. Cependant, il existe de nombreuses **solutions open sources** disponibles aujourd'hui, telles que Prometheus, Elastic Stack . Ainsi que des **solutions propriétaires** telles que DATADOG, dynatrace.

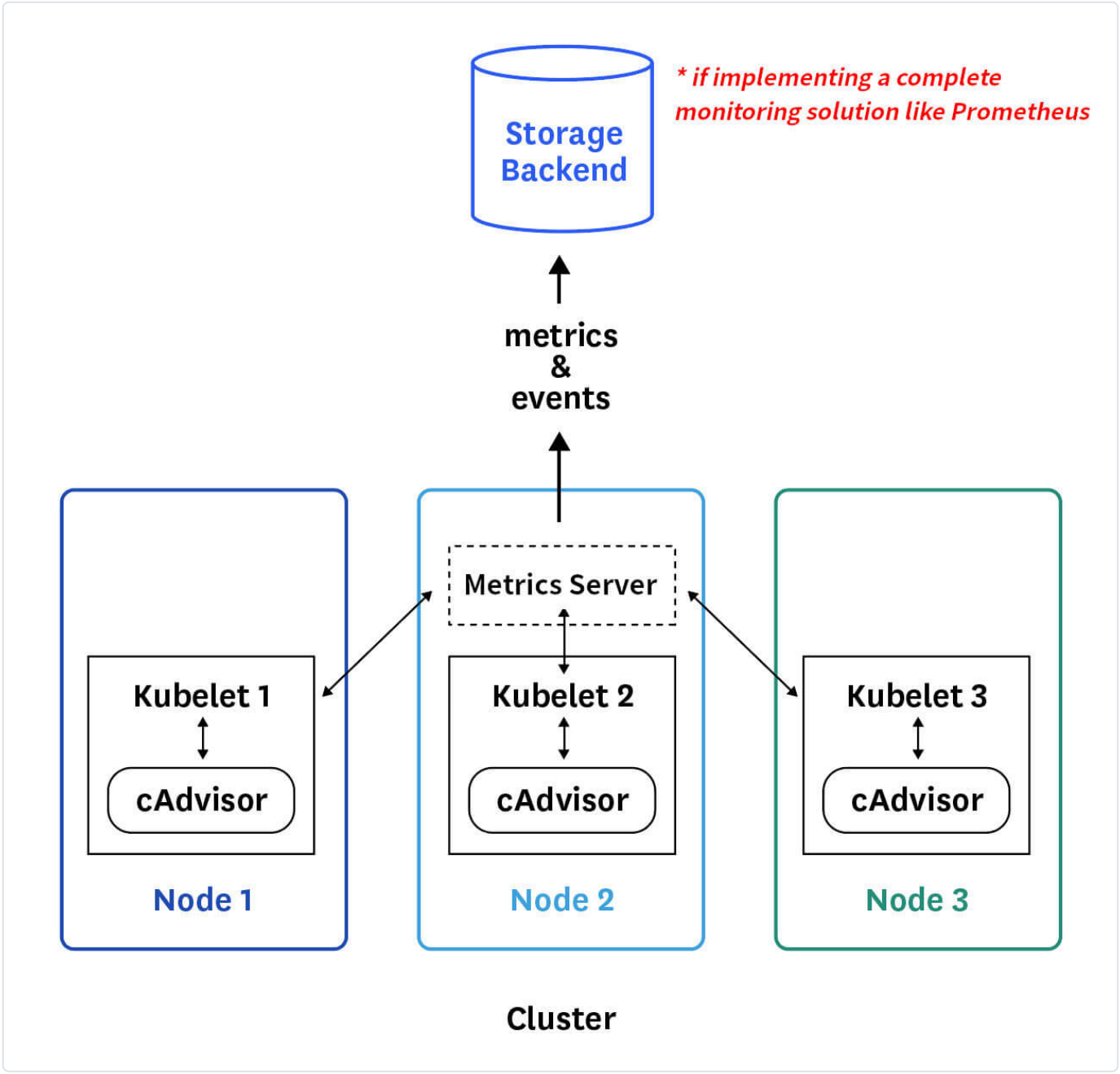
Heapster était l'un des projets originaux qui permettait d'activer la supervision et l'analyse de fonctionnalités pour Kubernetes. Vous verrez d'ailleurs beaucoup de références à Hipster lorsque vous recherchez des références sur la surveillance dans kubernetes. Cependant Heapster est maintenant obsolète, une version allégée a été conçue, plus connue sous le nom de **Metrics Server**.

Comment ça marche ?

Le serveur de métriques récupère les mesures de chacun des nœuds et des pods Kubernetes, il les regroupe ensuite et les stocke en mémoire. Notez que le Metric Server n'est qu'une solution de surveillance en mémoire et ne stocke en aucun cas les données récupérées dans le disque. Par conséquent, vous ne pouvez pas voir l'historique des données de performances. Pour cela, vous devez compter sur l'une des solutions de monitoring avancées dont nous avons parlé au tout début de ce chapitre.

On peut d'ores et déjà se poser la question suivante ? **Comment les métriques sont-elles générées pour les POD sur ses nœuds** ? En réalité Kubernetes exécute un agent sur chaque nœud connu sous le nom de kubelet. Cet agent est responsable de la réception des ordres du serveur API kubernetes, qui lui demande d'exécuter des pods sur les nœuds disponibles dans le cluster.

L'agent kubelet contient également un sous composant appelé **cAdvisor** ou **Container Advisor**, qui est responsable de récupérer les métriques de performance des pods et les exposer via l'API de kubelet afin de les rendre disponibles pour le serveur de métriques.



Implémentation d'un Metrics Server

Si vous utilisez minikube, il suffit alors d'exécuter la commande suivante:

```
minikube addons enable metrics-server
```

Pour ceux qui utilisent une autre solution, vous devez d'abord cloner le projet git Metrics Server :

```
git clone https://github.com/kubernetes-incubator/metrics-server.git
```

Après avoir cloné le projet depuis GitHub, vous allez par la suite, créer un ensemble de ressources disponibles dans le dossier **deploy/1.8+/** afin de permettre au serveur de métriques d'interroger les métriques de performance des nœuds du cluster.

```
kubectl create -f deploy/1.8+/
```

Vérifiez que votre pod de métriques est bien à l'état **running** :

```
kubectl get po -n kube-system |grep metrics
```

Résultat :

```
metrics-server-77dd877444-hcqrt   1/1     Running   0          105s
```

Vérifiez aussi les logs du conteneur dans le pod afin de s'assurer du bon fonctionnement de ce dernier :

```
kubectl logs metrics-server-77dd877444-hcqrt  -n kube-system
```

Résultat :

```
I0911 08:10:10.242962       1 serving.go:312] Generated self-signed
al.config/certificates/apiserver.key)
I0911 08:10:11.933898       1 secure_serving.go:116] Serving secure
```

Attendez un peu, le temps de collecter et traiter les données. Une fois traitées, les performances du cluster peuvent être visualisées par les commandes suivantes :

Dans le cas d'une **supervision des nœuds :**

```
kubectl top nodes
```

Résultat :

```
NAME     CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
master   134m         3%     999Mi           52%
worker-1   40m        1%     644Mi           16%
```

Dans le cas d'une **supervision des pods :**

Je vais premièrement créer un Deployment avec une réplique de trois pods :

```
kubectl run nginx --image=nginx -r=3
```

Deuxièmement, je les supervise avec la commande suivante :

```
kubectl top pod
```

Résultat :

```
NAME                     CPU(cores)   MEMORY(bytes)
nginx-7db9fccd9b-4bbq5   0m           1Mi
nginx-7db9fccd9b-7mgb5   0m           1Mi
nginx-7db9fccd9b-d4d5p   0m           1Mi
```

Bonus (dépannage)

> **Attention**
>
> Cette manipulation est non recommandée pour une utilisation en production.

Si jamais vous travaillez sur un environnement de test et que vous rencontrez des problèmes avec le pod de métriques, alors dans ce cas de figure, rajoutez les informations ci-dessous au fichier manifeste yaml situé dans **deploy/1.8+/metrics-server-deployment.yaml** juste après la ligne `imagePullPolicy: Always` :

```
containers:
- name: metrics-server
```

```
      image: k8s.gcr.io/metrics-server-amd64:v0.3.1
      imagePullPolicy: Always
      # début des informations à rajouter...
      command:
      - /metrics-server
      - --metric-resolution=30s
      - --kubelet-insecure-tls
      - --kubelet-preferred-address-types=InternalIP
      # fin des informations à rajouter...
      volumeMounts:
      - name: tmp-dir
        mountPath: /tmp
```

Voici la **définitions des différentes options de metrics-server** que vous allez ajouter :

- `--kubelet-insecure-tls` : ignorer la vérification des certificats Kubelet CA.
- `--metric-resolution` : intervalle auquel les mesures seront extraites de Kubelets (valeur par défaut de 60).
- `kubelet-preferred-address-types` : type de communication avec les différents nœuds, dans notre cas on utilise les adresses IP interne.

Enfin, appliquez vos changements en relançant la commande suivante :

```
kubectl apply -f deploy/1.8+/
```