ILKI
YOUR CLOUD DESIGNER

Formation Kubernetes

-

Pratique

kubernetes

Atelier 8

-

Volumes basiques

ILKI
YOUR CLOUD DESIGNER

Table des matières

1. Volumes

Nous avons évoqué le fait que les ressources sont partagées au sein d'un pod (CPU, RAM, réseau, etc). Malheureusement, les processus ne se partagent pas le stockage.

Pour rappel, chaque container possède son propre filesystem isolé, du fait que le filesystem provient de l'image source du container. Tout nouveau container démarre avec le même ensemble de fichiers ajouté lors de la construction de l'image.

De plus, pour rappel, les containers sont « stateless » par défaut. À chaque fois qu'un container redémarre, les données ne sont pas conservées.

Cependant, dans certains cas, vous pouvez avoir besoin, soit :

- Partager des données entre containers,
- Préserver des données en cas de redémarrage.

Kubernetes propose donc une ressource **Volume** permettant d'adresser ses besoins, en stockant les données en dehors des containers.

Les volumes **Kubernetes** ne peuvent pas être créés ou supprimés directement, autrement dit, il est obligatoire de les déclarer dans une spécification de pod. Une fois créé, un volume est accessible par tous les containers du pod, à condition de leur donner accès. Lorsque le pod est supprimé, le volume l'est aussi.

Un large panel de volumes est disponible : certains sont génériques, d'autres dépendent de la technologie de stockage sous-jacente.

Ci-dessous, quelques exemples de types volumes :

- **Quelques exemples de volumes disponibles :**
  - **emptyDir –** simple dossier vide utilisé pour stocker des données transitoires
  - **hostPath –** utilisé pour monter des dossiers du filesystem de la node
  - **gitRepo –** volume initialisé en checkant le contenu d'un repository Git
  - **nfs –** partage NFS
  - **gcePersitentDisk, awsElasticBlockStore, azureDisk… –** Stockage cloud
  - **Cinder, cephfs, iscsi, flocker, glusterfs, quobyte, rbd, vsphereVolume, photonPersistentDisk, scaleIO… –** autre stockage réseau
  - **configMap, secret, downwardAPI –** les volumes Kubernetes
  - **persistentVolume Claim –** une manière de pré/dynamiquement provisionner du stockage persistant

Dans cet atelier, seuls les volumes dit basiques seront traités.

1.1. Utilisation des volumes pour partager de la donnée en containers

1.1.1. Utilisation d'un volume emptyDir

Un volume de type **emptyDir** est particulièrement utile pour partager des fichiers entre des containers s'exécutant dans le même pod. Mais il peut également être utilisé par un seul container lorsqu'un container doit écrire des données sur le disque de manière temporaire (par exemple lors d'une opération de tri sur un jeu de données volumineux, qui ne peut pas être traité dans la mémoire disponible).

Les données peuvent également être écrites sur le système de fichiers du container lui-même (vous vous souvenez de la couche de lecture-écriture la plus haute dans un container?), mais il existe des différences subtiles entre les deux options. Le système de fichiers d'un container peut même ne pas être accessible en écriture (nous en parlerons à la fin du livre), l'écriture sur un volume monté peut alors être la seule option possible.

Vous allez donc réutiliser le pod chuck des ateliers précédents, mais en créant un volume permettant le partage de données entre les containers.

1) Créez un fichier yaml « pod-chuck-vol.yaml » avec le code suivant :

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: chuck
  labels:
    name: chuck
spec:
  containers:
  - name: web
    image: ilkiformation/webapache:v1
    ports:
      - containerPort: 80
      - containerPort: 22
    volumeMounts:
      - mountPath: /var/www/html
        name: html
  - name: generator
    image: ilkiformation/chuckgenvolume:v1
    volumeMounts:
      - mountPath: /tmp
        name: html
  volumes:
  - name: html
    emptyDir: {}
```

Ici, un volume **html** est créé dans lequel le container **generator** vient stocker la page HTML et le container **web** vient la chercher pour l'afficher.

2) Validez le bon fonctionnement du pod en effectuant la commande « curl http://IP_POD ».

1.2.Accès à des fichiers du filesystem de la node

Comme évoqué lors de l'atelier sur les **DaemonSets**, certains pods, agissant au niveau système, peuvent avoir besoin d'accéder aux fichiers se trouvant sur la node.

Cela est rendu possible grâce aux volumes de type *hostPath*. Ils permettent de monter un fichier ou dossier de la node directement dans le filesystem du container.

À noter que c'est le premier type de stockage persistant que vous allez traiter. En effet, si un pod est supprimé, le fichier ou le contenu du dossier n'est pas supprimé.

Au lieu de créer un nouveau pod, vous allez observer si des pods avec ce type de volume ne sont pas déjà déployés sur le cluster. Comme vous vous rappelez peut-être, plusieurs pods fonctionnent dans le namespace *kube-system*.

1) Listez les pods du namespace *kube-system* :

```
kubectl get po -n kube-system
```

2) Prenez le premier pod **fluentd** de la liste et étudiez les types de volumes qui lui sont associés :

```
kubectl -n kube-system describe po fluentd-gcp-v2.0.17-5r22n
```

Vous retrouvez la partie du fichier qui nous intéresse, ci-dessous :

```
Volumes:
  varlog:
    Type:          HostPath (bare host directory volume)
    Path:          /var/log
    HostPathType:
  varlibdockercontainers:
    Type:          HostPath (bare host directory volume)
    Path:          /var/lib/docker/containers
    HostPathType:
```

Vous pouvez voir que le pod utilise deux volumes *hostPath* pour accéder aux dossiers « */var/log* » et « */var/lib/docker/container* ».

En regardant le contenu de l'ensemble des pods de ce namespace, vous verrez qu'aucun d'eux n'utilise les volumes pour stocker leurs données, mais uniquement pour accéder aux données de la node.

Dans le cas précédent, le pod **fluentd** accède aux deux dossiers pour récupérer la liste des containers et les logs de la node.