Chapitre 6
Persistance des données

1. Objectifs du chapitre et prérequis

Ce chapitre présente la persistance des données dans Kubernetes. En effet, le contenu d'un container n'a pas vocation à perdurer. Ce mécanisme est là pour permettre de conserver une information lorsque c'est nécessaire (bases de données, serveurs de fichiers, etc.).

Le chapitre abordera deux points de vue :

– l'utilisateur faisant appel aux volumes persistants,

– l'administrateur mettant en place ce mécanisme.

2. Persistance des données

2.1 Origine du besoin

Dans ce qui a précédé, vous avez pu aborder le cycle de vie du container dans Kubernetes. Un point important à retenir est qu'un container a une durée de vie relativement courte et, qu'en l'état, il n'est pas possible de conserver de la donnée.

Pour répondre à ce besoin, Kubernetes peut mettre à disposition des espaces de stockage externes au cluster. Ce mécanisme s'appuie sur la notion de volume de données persistant (*Persistent Volume*).

2.2 Utilisation d'un volume persistant externe

L'utilisation d'un volume persistant se fait au sein de la déclaration d'un pod. Une première solution pourrait être de passer par un service externe comme avec un point de montage NFS.

Cette déclaration de persistance de données se définira en deux parties :

– un référencement au niveau du pod dans le champ `volumes`,

– une indication de l'emplacement du montage au sein du container.

Le référencement d'un volume NFS au niveau d'un pod se présentera sous la forme suivante :

```
spec:
 volumes:
   - name: nfs
     nfs:
       # URL for the NFS server
       server: 192.168.0.1
       path: /
```

Le montage au niveau du container se présentera ainsi :

```
spec:
 containers:
   - name: mailhog
     image: mailhog/mailhog

     # Mount the NFS volume in the container
     volumeMounts:
       - name: nfs
         mountPath: /maildir
```

2.3 Volumes persistants

2.3.1 Structure du volume persistant

En plus de faire appel à des services externes, il est possible de référencer des objets de type volume persistant (*Persistent Volume*). Ces derniers ont la structure suivante :

- une version et un type (`apiVersion` et `kind`),
- des métadonnées (champ `metadata`),
- une spécification contenant :
  - le type d'accès,
  - la capacité de stockage,
  - la classe du volume persistant (positionner à `manual` pour l'exemple),
  - les caractéristiques du stockage.

Ci-dessous un exemple de stockage portant le nom de pv-mailhog s'appuyant sur un répertoire de la machine hôte. Ce volume a une capacité déclarée de 10 Mo :

```
apiVersion: v1
kind: PersistentVolume
metadata:
   name: pv-mailhog
spec:
   accessModes:
     - ReadWriteOnce
   storageClassName: manual
   capacity:
     storage: 10Mi
   hostPath:
     path: /tmp/pv-mailhog
```

Sauvegardez cette déclaration dans le fichier **pv-mailhog.yaml**.

2.3.2 Création du volume persistant

La déclaration réalisée, l'application de cette déclaration se fera à l'aide de la commande `kubectl` suivie des indications suivantes :

– le mot-clé apply,

– l'option -f suivie du nom de fichier.

Ci-dessous la commande correspondante :

```
$ kubectl apply -f pv-mailhog.yaml
```

Cette commande doit alors renvoyer la valeur suivante :

```
persistentvolume/pv-mailhog created
```

Le volume est déclaré. Reste maintenant à voir comment l'utiliser dans un container.

2.4 Persistance de données avec MailHog

2.4.1 Opérations à réaliser

Par défaut, lorsque MailHog reçoit un message, ce dernier le stocke en local dans un répertoire de travail. Malheureusement, lorsque le pod est relancé ou redémarré, la donnée se perd.

Afin de faire face à ce problème, ajoutez un point de montage dans le container de MailHog.

Pour cela, deux modifications seront réalisées dans l'objet déploiement :

– utilisation d'un volume persistant,

– modification des options de lancement de MailHog.

En plus de ces modifications dans le déploiement, vous devrez également déclarer un objet permettant de référencer le volume persistant.

Cet objet fera le lien entre le volume persistant et la déclaration d'utilisation.

2.4.2 Déclaration de l'objet PersistentVolumeClaim

La demande de volume persistant (`PersistentVolumeClaim` ou son raccourci `pvc`) réclame un certain nombre d'informations :

– les champs `apiVersion` et `kind`,
– les métadonnées (`metadata`),
– les spécifications de l'objet :
  – le mode d'accès,
  – la classe précisée précédemment (`manual`),
  – la quantité de ressources demandées,
  – le nom du volume persistant.

Dans le cas d'un objet `PersistentVolumeClaim` portant le nom de `pvc-mailhog` pour un volume de 10 Mo, la déclaration sera la suivante :

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
   name: pvc-mailhog
spec:
   accessModes:
     - ReadWriteOnce
   storageClassName: manual
   resources:
     requests:
       storage: 10Mi
   volumeName: pv-mailhog
```

Stockez cette déclaration dans le fichier **pvc-mailhog.yaml** et appliquez cet objet à l'aide de la commande suivante :

```
$ kubectl apply -f pvc-mailhog.yaml
```

Cette commande doit alors renvoyer la valeur suivante :

```
persistentvolumeclaim/pvc-mailhog created
```

2.4.3 État des objets de volume persistant

La consultation de ces objets se fait traditionnellement avec les méthodes `get` ou `describe`.

Ci-dessous la commande pour consulter la liste des volumes persistants (`PersistentVolume` ou `pv`) :

```
$ kubectl get persistentvolume
```

Ci-dessous une première partie de la sortie de cette commande :

```
NAME          CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS ...
pv-mailhog    10Mi       RWX            Retain           Bound  ...
```

Et ci-dessous les champs suivants :

```
NAME          ... CLAIM                  STORAGECLASS    ...
pv-mailhog    ... default/pvc-mailhog    manual          ...
```

L'ensemble de ces champs va donner les informations suivantes :

- `NAME` : nom du volume persistant.
- `CAPACITY` : taille du volume persistant.
- `ACCESS MODES` : types d'accès autorisés.
- `RECLAIM POLICY` : permet de spécifier le comportement à adopter en cas de suppression de l'objet `PersistentVolumeClaim` (garder le volume ou suppression automatique).
- `STATUS` : si le volume est associé (`Bound`) ou détaché (`Released`) vis-à-vis d'un objet `PersistentVolumeClaim`.
- `CLAIM` : nom de l'objet `PersistentVolumeClaim` associé.
- `STORAGECLASS` : classe utilisée par le volume.

Le volume persistant pv-mailhog est bien lié à l'objet `PersistentVolume-Claim pvc-mailhog`.

Chapitre 4
Les disques et le système de fichiers

1. Représentation des disques

Note préalable : les unités de mesure de stockage utilisées dans ce chapitre et dans l'ensemble de ce livre utilisent la représentation de l'usage traditionnel en kilo-octets, selon la règle 1 ko = 1024 octets (2^{10}), sauf indication contraire. Cette représentation se nomme théoriquement Kio (kibioctet).

1.1 Nomenclature

Ceci est un petit rappel des points déjà rencontrés dans le chapitre Présentation de Linux. Suivant le type de contrôleur et d'interface sur lesquels les disques sont connectés, Linux donne des noms différents aux fichiers spéciaux des périphériques disques.

Chaque disque est représenté par un fichier spécial de type bloc. Chaque partition aussi.

1.1.1 IDE

Cette section est conservée pour des raisons historiques, la norme SATA ayant remplacé la norme IDE sur la quasi-totalité des ordinateurs de bureau et portables depuis plus de dix ans. Les disques reliés à des contrôleurs IDE (appelés aussi PATA (*Parallel ATA*) ou ATAPI) se nomment hdX :

- hda : IDE0, Master
- hdb : IDE0, Slave

– hdc : IDE1, Master
– hdd : IDE1, Slave
– etc.

Contrairement aux idées reçues, il n'y a pas de limites au nombre de contrôleurs IDE, sauf le nombre de ports d'extension de la machine (slots PCI). De nombreuses cartes additionnelles et convertisseurs existent permettant de lire d'anciens disques IDE. Au-delà de quatre disques ou lecteurs, les fichiers se nomment hde, hdf, hdg, etc.

Les lecteurs CD-Rom, DVD et graveurs de type IDE/ATAPI sont vus comme des disques IDE et respectent cette nomenclature.

Les noyaux Linux utilisent maintenant par défaut une API appelée libata pour accéder à l'ensemble des disques IDE, SCSI, USB, Firewire, etc. La nomenclature reprend celle des disques SCSI, abordée au point suivant.

1.1.2 SCSI, SATA, USB, FIREWIRE, etc.

Les disques reliés à des contrôleurs SCSI, SCA, SAS, FibreChannel, USB, Firewire, thunderbolt (et probablement d'autres interfaces exotiques) se nomment sdX. L'énumération des disques reprend l'ordre de détection des cartes SCSI et des adaptateurs (hosts) associés, puis l'ajout et la suppression manuelle des autres via hotplug ou udev.

– sda : premier disque SCSI
– sdb : deuxième disque SCSI
– sdc : troisième disque SCSI
– etc.

La norme SCSI fait une différence entre les divers supports. Aussi les lecteurs CD-Rom, DVD, HD-DVD, Blu-ray et les graveurs associés n'ont pas le même nom. Les lecteurs et graveurs sont en srX (sr0, sr1, etc.). Vous pouvez aussi trouver scd0, scd1, etc. mais ce sont généralement des liens symboliques vers sr0, sr1, etc.

Au-delà de sdz, l'énumération redémarre à sdaa, sdab, etc.

La commande **lsscsi** permet d'énumérer les périphériques SCSI. Notez que les disques sont bien en sdX, tandis que le lecteur dvd est en srX.

```
$ lsscsi
[4:0:0:0]    disk    ATA      ST380011A         8.01    /dev/sda
[5:0:0:0]    cd/dvd  LITE-ON  COMBO SOHC-4836V  S9C1    /dev/sr0
[31:0:0:0]   disk    USB2.0   Mobile Disk       1.00    /dev/sdb
```

1.2 Cas spéciaux

1.2.1 Contrôleurs spécifiques

Certains contrôleurs ne suivent pas cette nomenclature. C'est par exemple le cas de certains contrôleurs RAID matériels. C'est du cas par cas. Un contrôleur Smart Array sur un serveur HP, utilisant le pilote cciss, place ses fichiers de périphériques dans /dev/cciss sous les noms cXdYpZ, où X est le slot, Y le disque et Z la partition. Les nouveaux contrôleurs utilisent le pilote hpsa, exploitant la couche SCSI du noyau et donc un nommage standard des périphériques.

1.2.2 Virtualisation

La représentation des disques des systèmes invités (*guests*) virtualisés dépend du type de contrôleur simulé. La plupart sont de type IDE ou SCSI, et dans les deux cas bien souvent avec la libata ils sont vus comme du SCSI. Cependant certains systèmes comme KVM ou XEN (ainsi que les environnements cloud les utilisant, comme AWS) proposant de la paravirtualisation offrent un contrôleur spécifique présentant les disques sous le nom vdX (virtual disk x), ou xvdX :

– vda : premier disque virtualisé, ou xvda,
– vdb : deuxième disque virtualisé, ou xvdb,
– etc.

1.2.3 SAN, iSCSI, multipathing

Les disques raccordés via un SAN (*Storage Area Network*, généralement en fibre optique) ou par iSCSI sont vus comme des disques SCSI et conservent cette nomenclature. Cependant les systèmes de gestion des chemins multiples (*multipathing*) se plaçant par-dessus fournissent d'autres noms. Powerpath nommera les disques emcpowerx (emcpowera, emcpowerb, etc.) tandis que le système par défaut de Linux appelé multipath les nommera mpathx (mpath0, mpath1, etc.) ou tout autre nom choisi par l'administrateur.

2. Manipulations de bas niveau

2.1 Informations

La commande **hdparm** permet d'effectuer un grand nombre de manipulations directement sur les périphériques disques gérés par la bibliothèque libata, c'est-à-dire tous les disques SATA, ATA (IDE) et SAS. La commande **sdparm** peut faire à peu près la même chose pour les disques SCSI. Notez que bien que les noms de périphériques de la libata soient identiques à ceux du SCSI, il est fort probable que de nombreuses options de configuration de hdparm ne fonctionnent pas sur des disques SCSI, la réciproque étant vraie pour sdparm avec les disques SATA ou IDE. La suite se base sur hdparm.

Pour obtenir des informations complètes sur un disque, utilisez les paramètres `-i` ou `-I`. Le premier récupère les informations depuis le noyau et obtenues au moment du boot, le second interroge directement le disque. Préférez le `-I` qui donne des informations très détaillées.

```
# hdparm -I /dev/sda

/dev/sda:

ATA device, with non-removable media
    Model Number:       VBOX HARDDISK
    Serial Number:      VB91a2e953-933cdc65
    Firmware Revision:  1.0
Standards:
    Used: ATA/ATAPI-6 published, ANSI INCITS 361-2002
    Supported: 6 5 4
Configuration:
    Logical        max      current
    cylinders      16383    16383
    heads          16       16
    sectors/track  63       63
    --
    CHS current addressable sectors:    16514064
    LBA    user addressable sectors:    63152320
    LBA48  user addressable sectors:    63152320
    Logical/Physical Sector size:           512 bytes
    device size with M = 1024*1024:        30836 MBytes
    device size with M = 1000*1000:        32333 MBytes (32 GB)
    cache/buffer size  = 256 KBytes (type=DualPortCache)
Capabilities:
    LBA, IORDY(cannot be disabled)
    Queue depth: 32
```

```
    Standby timer values: spec'd by Vendor, no device specific minimum
    R/W multiple sector transfer: Max = 128        Current = 128
    DMA: mdma0 mdma1 mdma2 udma0 udma1 udma2 udma3 udma4 udma5 *udma6
         Cycle time: min=120ns recommended=120ns
    PIO: pio0 pio1 pio2 pio3 pio4
         Cycle time: no flow control=120ns  IORDY flow control=120ns
Commands/features:
    Enabled    Supported:
        *    Power Management feature set
        *    Write cache
        *    Look-ahead
        *    48-bit Address feature set
        *    Mandatory FLUSH_CACHE
        *    FLUSH_CACHE_EXT
        *    Gen2 signaling speed (3.0Gb/s)
        *    Native Command Queueing (NCQ)
Checksum: correct
```

2.2 Modification des valeurs

Plusieurs paramètres des disques peuvent être modifiés. Attention cependant ! Certaines options de hdparm peuvent se révéler être dangereuses tant pour les données contenues sur le disque que pour le disque lui-même. La plupart des paramètres sont en lecture et écriture. Si aucune valeur n'est précisée hdparm affiche l'état du disque (ou du bus) pour cette commande. Voici quelques exemples d'options intéressantes.

- **-c** : largeur du bus de transfert EIDE sur 16 ou 32 bits. 0=16, 1=32, 3=32 compatible.
- **-d** : utilisation du DMA. 0=pas de DMA, 1=DMA activé.
- **-X** : modifie le mode DMA (mdma0 mdma1 mdma2 udma0 udma1 udma2 udma3 udma4 udma5). Vous pouvez utiliser l'un des modes précédents ou des valeurs numériques : 32+n pour les modes mdma (n variant de 0 à 2) et 64+n pour les modes udma.
- **-C** : statut de l'économie d'énergie sur le disque (unknown, active/idle, standby, sleeping). L'état peut être modifié avec -S, -y, -Y et -Z.
- **-g** : affiche la géométrie du disque.
- **-M** : indique ou modifie l'état du Automatic Acoustic Management (AAM). 0=off, 128=quiet et 254=fast. Tous les disques ne le supportent pas.
- **-r** : passe le disque en lecture seule.

– **-T** : bench de lecture du cache disque, idéal pour tester les performances de transfert entre Linux et le cache du disque. Il faut relancer la commande deux ou trois fois.
– **-t** : bench de lecture du disque, hors cache. Mêmes remarques que l'option précédente.

Ainsi la commande suivante passe le bus de transfert en 32 bits, active le mode DMA en mode Ultra DMA 5 pour le disque sda :

```
# hdparm -c1 -d3 -X udma5 /dev/sda
```

Voici quelques autres exemples :

```
# hdparm -c /dev/sda

/dev/sda:
 IO_support = 0 (default 16-bit)

# hdparm -C /dev/sda

/dev/sda:
 drive state is: active/idle

# hdparm -g /dev/sda

/dev/sda:
 geometry = 3931/255/63, sectors = 63152320, start = 0

# hdparm -T /dev/sda

/dev/sda:
 Timing cached reads: 23868 MB in 2.00 seconds = 11950.45 MB/sec
# hdparm -t /dev/sda

/dev/sda:
 Timing buffered disk reads: 308 MB in 3.02 seconds = 101.87 MB/sec
```

3. Choisir un système de fichiers

3.1 Principe

3.1.1 Définition

L'action de « formater » un disque, une clé ou tout support de données consiste uniquement à créer sur un support de mémoire secondaire (volume de stockage) l'organisation logique permettant d'y placer des données. Le mot « formatage » sous Linux est utilisé pour décrire la création d'un système de fichiers. On parle donc de système de fichiers qui est à la fois l'organisation logique des supports au niveau le plus bas comme au niveau de l'utilisateur.

Les informations ne sont pas écrites n'importe comment sur les disques. Une organisation est nécessaire pour y placer tant les informations sur les fichiers qui y sont stockés que les données. Ce sont le système de fichiers et les pilotes associés qui définissent cette organisation. Si les principes de base sont souvent les mêmes entre les divers systèmes présents sous Linux, les implémentations et les organisations logiques des données sur le disque varient fortement. Aussi il n'existe pas un type de système de fichiers, mais plusieurs, au choix de l'utilisateur ou de l'administrateur système.

Tous les systèmes de fichiers Linux doivent respecter les normes POSIX. Comme POSIX définit un ensemble de règles de base, un système de fichiers peut aller au-delà de cette norme en proposant des extensions. La plupart de celles-ci concernent des éléments de sécurité, comme les ACL ou selinux.

Le principe de base d'un système de fichiers est d'associer un nom de fichier à son contenu et d'y permettre l'accès : création, modification, suppression, déplacement, ouverture, lecture, écriture, fermeture. Suivant ce principe, le système de fichiers doit gérer ce qui en découle : mécanismes de protection des accès (les permissions, les propriétaires), les accès concurrents, etc.

3.1.2 Représentation

Outre l'organisation et le stockage des informations et des données sur les fichiers, le système de fichiers doit fournir à l'utilisateur une vision structurée de ses données, permettant de les distinguer, de les retrouver, de les traiter et de les manipuler sous forme de fichiers au sein d'une arborescence de répertoires avec les commandes associées. De même, chaque système de fichiers doit fournir le nécessaire pour que les programmes puissent y accéder.