Version - **2024.01**

Dernière mise-à-jour : 2024/12/15 06:53

DOF306 - Gestion des Volumes sous K8s

Ressources

Lab #1

- https://www.dropbox.com/scl/fi/jylunrftkra3csboubxvm/volume.yaml?rlkey=wz0ckbw31gbq4j8cm7lc1ny7h&dl=0
- https://www.dropbox.com/scl/fi/mv0z7jfqtd13q78m7716l/shared.yaml?rlkey=btadejbwv7i4hb98ap7ufrv1d&dl=0

Lab #2

- https://www.dropbox.com/scl/fi/9c5kn7yw9q5eftw7pi1sd/localdisk.yaml?rlkey=xwtiqso78ow84ww3sssq4hp1e&dl=0
- https://www.dropbox.com/scl/fi/fo5tfji3kxn9dtjitvek4/mypv.yaml?rlkey=cdt1g6dpwrzlihndzsrdreqsa&dl=0
- https://www.dropbox.com/scl/fi/ja2ddchsgunwdswc4cc92/mypvc.yaml?rlkey=80mmqg90y7ikdz8ifqpvb49pt&dl=0
- https://www.dropbox.com/scl/fi/lgwfi8tjbg5sq32zrqkpw/mypvcpod.yaml?rlkey=2v74mq9p8o63hiviwu595waj7&dl=0

Présentation

Volumes

Le système de fichiers d'un conteneur dans un pod est éphémère, à savoir qu'il n'existe que pendant le cycle de vie du conteneur. Si le conteneur est supprimé ou re-créé, le système de fichiers est perdu.

Les **volumes** permettent le stockage de données en dehors du système de fichiers du conteneur tout en permettant le conteneur d'y accéder.

Persistent Volumes

Un **Persistent Volume** (*Volume Persistant*) est une ressource abstraite qui peut être *consommer* par des pods. Pour accéder au Persistent Volume, le pod a besoin d'un **Persistent Volume Claim** (*Réclamation de Volume Persistant*) pour monter le Persistent Volume dans le pod.

Types de Volumes

Les Volumes et les Persistent Volumes ont un **Volume Type** (*Type de Volume*). Le Volume Type détermine le **Storage Method** (*Méthode de Stockage*) des données. Parmi les Storage Methods on trouve :

- NFS,
- AWS,
- Azure,

- GCP,
- ConfigMaps,
- Secrets,

Important : Pour plus d'information concernant les Storage Methods, consultez **cette page** de la documentation K8s.

LAB #1 - Utiliser des Volumes K8s

1.1 - Volumes et volumeMounts

Les Volumes sont configurés dans la spécification du **pod** et non le conteneur. Les deux Volume Types les plus important sont **hostPath** et **emptyDir** :

- **hostPath**,
  - Les données sont stockés localement dans un répertoire statique du noeud K8s,
- **emptyDir**,
  - Les données sont stockés localement dans un répertoire dynamique,
  - Le répertoire n'existe **que** pendant que le pod existe sur le noeud,
  - K8s supprime le répertoire et les données lors de la suppression ou le déplacement du pod,
  - Ce Volume Type est principalement utilisé pour partager des données entre deux conteneurs dans un pod.

Un **volumeMount** est configuré dans la spécification du **conteneur** et non le pod

Commencez par créer le fichier **volume.yaml** :

```
root@kubemaster:~# vi volume.yaml
root@kubemaster:~# cat volume.yaml
apiVersion: v1
kind: Pod
```

```
metadata:
  name: volumepod
spec:
  restartPolicy: Never
  containers:
  - name: busybox
    image: busybox
    command: ['sh', '-c', 'echo Success! > /output/success.txt']
    volumeMounts:
    - name: myvolume
      mountPath: /output
  volumes:
  - name: myvolume
    hostPath:
      path: /var/data
```

Important : Ce pod va écrire la chaîne **Success!** dans le fichier **/output/success.txt** à l'intérieur du conteneur puis s'arrêter car la valeur de **restartPolicy** est **Never**. Le volume **myvolume** sera monté sur **/output** dans le conteneur grâce à la configuration du volumeMount et sur **/var/data/** dans le noeud qui héberge le pod.

Créez le pod **volumepod** :

```
root@kubemaster:~# kubectl create -f volume.yaml
pod/volumepod created
```

Identifiez le noeud sur lequel s'exécute le pod :

```
root@kubemaster:~# kubectl get pod volumepod -o wide
NAME        READY   STATUS      RESTARTS   AGE     IP              NODE                      NOMINATED NODE
READINESS GATES
```

```
volumepod   0/1     Completed   0          3m10s   192.168.150.41   kubenode2.ittraining.loc   <none>
<none>
```

Connectez-vous au nœud identifié :

```
root@kubemaster:~# ssh -l trainee kubenode2
trainee@kubenode2's password: trainee
Linux kubenode2.ittraining.loc 4.9.0-19-amd64 #1 SMP Debian 4.9.320-2 (2022-06-30) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sun Sep  4 13:24:39 2022 from 192.168.56.2
```

Vérifiez la présence et le contenu du fichier **/var/data/success.txt** :

```
trainee@kubenode2:~$ cat /var/data/success.txt
Success!
```

1.2 - Partager des volumes entre conteneurs

Retournez au **kubemaster** :

```
trainee@kubenode2:~$ exit
déconnexion
Connection to kubenode2 closed.
root@kubemaster:~#
```

Créez maintenant le fichier **shared.yaml** :

```
root@kubemaster:~# vi shared.yaml
root@kubemaster:~# cat shared.yaml
apiVersion: v1
kind: Pod
metadata:
  name: sharedvolume
spec:
  containers:
  - name: busybox1
    image: busybox
    command: ['sh', '-c', 'while true; do echo Success! > /output/output.txt; sleep 5; done']
    volumeMounts:
      - name: myvolume
        mountPath: /output
  - name: busybox2
    image: busybox
    command: ['sh', '-c', 'while true; do cat /input/output.txt; sleep 5; done']
    volumeMounts:
    - name: myvolume
      mountPath: /input
  volumes:
  - name: myvolume
    emptyDir: {}
```

Important : Ce fichier va créer deux pods. Le premier, **busybox1**, va écrire la chaîne **Success!** dans le fichier **/output/output.txt** du conteneur tous les 5 secondes. Le répertoire **/output** est connu en tant que **myvolume**. Ce même volume sera disponible au conteneur du deuxième pod, **busybox2** où il sera monté à **/input**. Le conteneur busybox2 va imprimer le contenu du fichier **/input/output.txt** sur la sortie standard tous les 5 secondes.

Créez les deux pods :

```
root@kubemaster:~# kubectl create -f shared.yaml
pod/sharedvolume created
```

Vérifiez que les deux pods sont en cours d'exécution :

```
root@kubemaster:~# kubectl get pods sharedvolume
NAME           READY   STATUS    RESTARTS   AGE
sharedvolume   2/2     Running   0          5m55s
```

Consultez maintenant les logs du deuxième conteneur :

```
root@kubemaster:~# kubectl logs sharedvolume -c busybox2
Success!
Success!
Success!
Success!
Success!
Success!
Success!
Success!
Success!
Success!
Success!
Success!
Success!
Success!
Success!
Success!
```

Important : Notez que busybox2 a imprimé le contenu du fichier **/input/output.txt** sur sa sortie standard

LAB #2 - Persistent Volumes

2.1 - Storage Classes

- **StorageClassName**,
  - Un StorageClassName est utilisé pour spécifier le **StorageClass**.
- **StorageClass**,
  - Un StorageClass est utilisé pour spécifier le type de service de stockage utilisé, par exemple, un disque local, le cloud etc,
  - Si la valeur du **allowVolumeExpansion** est true et le type de service de stockage le permet, un **PersistentVolumeClaim** peut être redimensionner à chaud.

Créez le fichier **localdisk.yaml** pour définir le **StorageClass** appelé **localdisk** :

```
root@kubemaster:~# vi localdisk.yaml
root@kubemaster:~# cat localdisk.yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: localdisk
provisioner: kubernetes.io/no-provisioner
allowVolumeExpansion: true
```

Important : Notez que la valeur du allowVolumeExpansion est true.

Créez le StorageClass **localdisk** :

```
root@kubemaster:~# kubectl create -f localdisk.yaml
```

2.2 - Persistent Volumes

Créez le fichier **mypv.yaml** pour définir le **PersistentVolume** appelé **mypv** :

```
root@kubemaster:~# vi mypv.yaml
root@kubemaster:~# cat mypv.yaml
kind: PersistentVolume
apiVersion: v1
metadata:
  name: mypv
spec:
  storageClassName: localdisk
  persistentVolumeReclaimPolicy: Recycle
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: /var/output
```

Important : Notez que la valeur de l'accessMode.

Il existe quatre types d'**accessModes** :

- **ReadWriteOnce** ou **RWO**,
  - le volume ne peut être monté que par un seul noeud,
- **ReadOnlyMany** ou **ROX**,
  - le volume peut être monté en lecture seule par plusieurs noeuds,
- **ReadWriteMany** ou **RWX**,
  - le volume peut être monté en lecture-écriture par plusieurs noeuds,
- **ReadWriteOncePod** ou **RWOP**,

- le volume ne peut être monté que par un seul pod.

La disponibilité de l'accessMode dépend du type de service de stockage. Le mode **ReadWriteOnce** étant toujours disponible. Pour plus d'information concernant les accessMode, consultez **cette page**.

Important : Notez que la valeur du persistentVolumeReclaimPolicy est Recycle.

Il existe trois types de PersistentVolumeReclaimPolicy :

- **Retain**,
  - Les données ne sont pas supprimées lors de la suppression d'un PersistentVolumeClaim,
- **Delete**,
  - Supprime automatiquement la **ressource de stockage** lors de la suppression d'un PersistentVolumeClaim,
  - A noter que **Delete** ne fonctionne qu'avec des services de clouds publiques tels AWS, GCP etc,
- **Recycle**,
  - Supprime automatiquement les données,
  - A noter que **Recycle** permet la réutilisation immédiate des ressources de stockage libérées lors de la suppression d'un PersistentVolumeClaim.

Créez le PersistentVolume **mypv** :

```
root@kubemaster:~# kubectl create -f mypv.yaml
persistentvolume/mypv created
```

Vérifiez le statut du PersistentVolume :

```
root@kubemaster:~# kubectl get pv
NAME   CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM   STORAGECLASS   REASON   AGE
mypv   1Gi        RWO            Recycle          Available           localdisk                5m28s
```

Important : Notez que la valeur du STATUS est Available.

2.3 - Persistent Volume Claims

- Un PersistentVolumeClaim représente la demande d'un utilisateur d'une ressource de stockage,
  - Le PersistentVolumeClaim spécifie un **StorageClassName**, un **AccessMode** et une **taille**,
- Lors sa création, le PersistentVolumeClaim recherche un **PersistentVolume** capable de satisfaire la demande formulée,
  - Si le résultat de la recherche est positive, le PersistentVolumeClaim est automatiquement lié au PersistentVolume,
  - Dans le cas contraire, le PersistentVolumeClaim **reste en attente** jusqu'à la création d'un PersistentVolume capable de satisfaire la demande formulée,
  - Pour redimensionner, sans interruption de service, un PersistentVolumeClaim il convient de modifier la valeur du **spec.resources.requests.storage** du fichier yaml.

Créez le fichier **mypvc.yaml** pour définir le **PersistentVolumeClaim** appelé **my-pvc** :

```
root@kubemaster:~# vi mypvc.yaml
root@kubemaster:~# cat mypvc.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-pvc
spec:
  storageClassName: localdisk
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 100Mi
```

Important : Notez que la valeur du storageClassName est localdisk.

Créez le PersistentVolumeClaim **my-pvc** :

```
root@kubemaster:~# kubectl create -f mypvc.yaml
persistentvolumeclaim/my-pvc created
```

Vérifiez le statut du PersistentVolume :

```
root@kubemaster:~# kubectl get pv
NAME   CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM            STORAGECLASS   REASON   AGE
mypv   1Gi        RWO            Recycle          Bound    default/my-pvc   localdisk               9m33s
```

Important : Notez que la valeur du STATUS est Bound. Notez aussi qu'un PersistentVolume ne peut être associé qu'à un seul PersistentVolumeClaim à la fois.

Vérifiez le statut du PersistentVolumeClaim :

```
root@kubemaster:~# kubectl get pvc
NAME     STATUS   VOLUME   CAPACITY   ACCESS MODES   STORAGECLASS   AGE
my-pvc   Bound    mypv     1Gi        RWO            localdisk      72s
```

Important : Notez que la valeur du STATUS est Bound.

2.4 - Utiliser un PersistentVolumeClaim dans un pod

Créez le fichier **mypvcpod.yaml** pour définir le **pod** appelé **pv-pod** :

```
root@kubemaster:~# vi mypvcpod.yaml
```

```
root@kubemaster:~# cat mypvcpod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: pv-pod
spec:
  restartPolicy: Never
  containers:
  - name: busybox
    image: busybox
    command: ['sh', '-c', 'echo Success! > /output/success.txt']
    volumeMounts:
    - name: pv-storage
      mountPath: /output
  volumes:
  - name: pv-storage
    persistentVolumeClaim:
      claimName: my-pvc
```

Créez le pod **pv-pod** :

```
root@kubemaster:~# kubectl create -f mypvcpod.yaml
pod/pv-pod created
```

Important : Notez que le pod utilise le persistentVolumeClaim my-pvc qui est monté sur /output dans le conteneur busybox.

2.5 - Redimensionnement d'un PersistentVolumeClaim

Modifiez la valeur du **storage:** du PersistentVolumeClaim :

```
root@kubemaster:~# kubectl edit pvc my-pvc --record
...
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 200Mi
  storageClassName: localdisk
  volumeMode: Filesystem
  volumeName: mypv
...
```

Sauvegardez la modification :

```
root@kubemaster:~# kubectl edit pvc my-pvc --record
Flag --record has been deprecated, --record will be removed in the future
persistentvolumeclaim/my-pvc edited
```

Important : Notez que le message de confirmation de l'édition.

Supprimez le pod **pv-pod** ainsi que le PersistentVolumeClaim **my-pvc** :

```
root@kubemaster:~# kubectl delete pod pv-pod
pod "pv-pod" deleted

root@kubemaster:~# kubectl delete pvc my-pvc
persistentvolumeclaim "my-pvc" deleted
```

Vérifiez le statut du PersistentVolume :

```
root@kubemaster:~# kubectl get pv
NAME   CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM   STORAGECLASS   REASON   AGE
mypv   1Gi        RWO            Recycle          Available           localdisk               23m
```

Important : Notez que la valeur du STATUS est de nouveau Available.
