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 3
Déploiement d'applications avec Kubernetes

1. Contexte

1.1 Objectifs généraux

1.1.1 Exploiter la plateforme Kubernetes mise en place

Les précédents chapitres se sont attachés à montrer les différentes manières de monter un cluster Kubernetes, ainsi qu'à expliquer dans les détails les opérations de maintenance et d'exploitation de ce cluster une fois mis en place. Jusqu'à maintenant, donc, les opérations montrées relevaient plutôt de profils système ou opérationnels, dont le rôle est d'assurer la disponibilité de l'infrastructure.

Dans le présent chapitre, nous allons basculer dans un autre rôle, qui correspond à un profil d'exploitant du logiciel supporté par la plateforme, et donc plutôt consommateur de celle-ci. En résumé, après être rentré dans le détail de comment Kubernetes fonctionne, nous allons nous pencher sur ce que nous pouvons en faire. L'usage principal de ce type de plateforme étant de déployer des applications pouvant passer à l'échelle, nous utiliserons une application exemple que nous installerons sur un cluster Kubernetes et que nous manipulerons comme le ferait l'exploitant d'une structure similaire en production.

1.1.2 Remarque sur l'approche DevOps

Dans son expression-même, "DevOps" peut faire penser que le principe associé est qu'une même personne possède les deux rôles de développeur et d'opérationnel, ou à l'inverse qu'il convient de bien séparer ces deux rôles. En fait, le principe du DevOps est de faire en sorte d'éviter les barrières entre les personnes en charge de l'infrastructure et celles en charge des logiciels portés par cette dernière.

Traditionnellement, les approches sont en effet opposées car les enjeux sont très différents : pour l'opérationnel en charge de la plateforme, l'enjeu principal est que celle-ci fonctionne en continu, ce qui a pour conséquence naturelle une certaine aversion au changement. Après tout, un des commandements principaux dans l'informatique est de ne pas toucher à un système qui fonctionne. Tout changement est potentiellement vecteur d'un nouveau bug, d'un problème dans la procédure de mise à jour et n'importe quel administrateur système a donc comme réflexe de réduire au maximum ces interventions.

À l'inverse, les développeurs et les profils qui commanditent ces changements, que ce soient des Product Owners sur un produit ou le client/utilisateur en direct sur un projet, ont plutôt intérêt à ce que les fonctionnalités sortent le plus vite possible, pour apporter le maximum de valeur métier au logiciel. À l'inverse de l'administrateur du système, ils souhaitent donc généralement accélérer le rythme du changement. Cette différence dans les enjeux a longtemps été traitée par un rapport de force et une opposition entre les deux approches, les seconds obtenant bon gré mal gré des premiers des mises en production tandis que ceux-ci invoquaient toutes les raisons, bonnes ou mauvaises, pour ne pas les réaliser.

Comme le périmètre de recoupement entre les deux responsabilités est traditionnellement assez large, les difficultés étaient plus étendues. Par exemple, un développeur pouvait avoir besoin d'un OS particulier pour son application, alors que l'administrateur souhaitait ne plus le supporter. Dans certains cas, un administrateur pouvait décider de blocages de sécurité trop contraignants pour un applicatif. Bref, le champ d'interaction était très large entre les deux rôles, car il couvrait toutes les technologies de support de déploiement, de la machine physique jusqu'au choix de librairies partagées.

Une approche plus évoluée est de s'attaquer au problème lié à l'incertitude sur le changement elle-même. Les approches Lean et Agile recommandent de réaliser les changements les plus petits et granulaires possibles, de façon à ce que l'impact soit le plus limité possible. Une bonne approche d'assurance qualité et de tests automatisés renforce cette sécurité sur le changement. Enfin, les principes d'intégration et de déploiement continu se sont attaqués à un autre pan du problème, en faisant en sorte qu'une mise en production devienne une tâche la plus banale possible.

Pour arriver à ce résultat, et lutter contre l'impression de "balancer une release par-dessus le mur", l'automatisation est certes une part de l'équation, avec la capacité de revenir facilement en arrière, mais il est également important de travailler sur les aspects management et méthodologie. C'est l'approche DevOps qui trouve son origine dans cette volonté de rendre les interactions simples entre les développeurs et les administrateurs de la plateforme logicielle déployée. Or, en plus d'un état d'esprit à insuffler, il convient de disposer également d'outils rendant possible et simple ce partage d'une réalité qui doit être gérée en commun au lieu de faire l'objet de conflits.

Kubernetes s'inscrit fortement dans cet objectif véhiculé de manière plus globale par les approches de gestion par conteneurs en fournissant une solution qui permet aux deux rôles de se répartir au mieux les tâches tout en leur offrant une réalité partagée, un contrat simple qui leur permet de travailler ensemble, mais en offrant le moins possible de prise à des conflits.

Ce qu'attendent principalement ces rôles d'une plateforme Kubernetes est qu'elle les rendent les plus indépendants possible des technologies liées à l'infrastructure : gestion du réseau physique, des machines physiques ou virtuelles, de l'ajout de ressources matérielles avec tout ce que cela comporte en termes de branchement, mais aussi de ressources "secondaires", comme la climatisation ou l'électricité, qu'il faut en outre sécuriser sur leur approvisionnement. L'idée est donc que l'utilisateur du cluster Kubernetes puisse simplement connaître l'adresse de celui-ci et les informations nécessaires pour s'y connecter, mais qu'il soit le moins possible exposé à cette complexité qui est liée au rôle de l'administrateur.

Le contrat résultant entre le producteur de la plateforme et le client de celle-ci repose au final uniquement sur l'adresse du cluster Kubernetes (nous reviendrons plus loin sur les portions techniques de cette "adresse" ou "identifiant", en reprenant les concepts de contexte, cluster et espace de nommage dans le cadre de l'exploitation logicielle de la plateforme Kubernetes). Le rôle d'administrateur consiste à s'occuper de la disponibilité des ressources matérielles, de la robustesse du cluster, de son élasticité, etc. et fournit une adresse pour le cluster. Le rôle d'exploitant logiciel utilise cette adresse pour déployer des applicatifs dans une configuration telle que souhaitée, et ce sans dépendance à l'administrateur autre que la connaissance de l'adresse du cluster. Il s'occupe des technologies à mettre en place dans le conteneur Docker, des librairies en dépendance, de la sécurité logicielle, etc., et fournit au cluster une boîte noire qui expose un port réseau avec des fonctionnalités en échange de ressources livrées de manière standard.

Dans la pratique, il est évident que des échanges sont nécessaires en plus, ne serait-ce que pour équilibrer les coûts du cluster avec les gains attendus par l'application que celui-ci supporte. Toutefois, ces enjeux sont désormais suffisamment éloignés de la technique pour qu'ils ne constituent pas un sujet de conflit immédiat. L'ajout de ressources étant simple – et surtout sans aucun impact sur la partie logicielle, il est relativement aisé de fournir une plateforme à coût réduit et de la faire grossir au fur et à mesure des besoins justifiés économiquement par les gains du logiciel. Cette souplesse rend également moins conflictuel le rapport entre les deux parties.

1.1.3 Principaux enseignements à attendre

Comme le présent chapitre est axé sur le consommateur de la plateforme, les sujets principaux relèveront de la façon de structurer une application pour la déployer sur Kubernetes, puis sur les méthodes utilisables pour réaliser ce déploiement et enfin sur celles à disposition pour maintenir le logiciel dans un état fonctionnel.

De façon à rendre plus compréhensibles tous les points d'attention (et ils sont nombreux) pour réaliser ceci, une première manipulation sera réalisée de manière manuelle et la plus détaillée possible, pour montrer tous les recoins de la technologie et bien expliquer les concepts associés.

Ensuite seulement, un second exemple montrera une réalisation plus automatisée et donc plus conforme à ce qu'on peut s'attendre à voir dans la réalité, sur des systèmes en production.

Kubernetes est souvent présenté comme une plateforme qui permet de simplifier fortement le déploiement, les mises à jour et les passages à l'échelle des applications web, mais cette simplicité ne va pas sans un coût initial pour la masquer. Ce coût est en partie supporté par la technologie elle-même, mais également par le paramétrage de l'application cible en amont. Bref, Kubernetes rend effectivement plus simple et robuste la vie d'une application, mais cela suppose de traiter une partie de la complexité inhérente à cette activité lors du montage de la solution. C'est toute cette complexité que nous allons détailler ci-dessous, de façon qu'une fois la plateforme fonctionnelle, elle le soit de manière extrêmement robuste et évolutive.

1.2 Outillage

1.2.1 Cluster Azure Kubernetes Services

Il ressort de toute l'introduction au présent chapitre qu'un cluster Kubernetes doit être prêt pour une utilisation de façon à réaliser les exercices ci-dessous. Nous pourrions laisser à l'utilisateur le choix du mode d'installation vu que plusieurs ont été montrés auparavant. Toutefois, deux raisons font que nous reviendrons sur une installation d'un cluster.

La première est que la première partie du présent ouvrage est entrée en détail dans la structure d'un cluster Kubernetes, en expliquant les détails de mise en œuvre et toutes les étapes nécessaires aux réalisations les plus complexes. Toutefois, il existe également des méthodes pour bénéficier d'un cluster "clé en main", et ce suite à quelques clics sur des interfaces web simples. Comme il était nécessaire de disposer d'une plateforme pour les exemples, autant en profiter pour montrer une nouvelle approche, qui constituera une corde de plus à l'arc de l'administrateur, bien que très rudimentaire. Il aurait bien sûr été possible d'utiliser Google Kubernetes Engine, mais comme Kubernetes vient lui-même de Google, la séparation entre ce qui vient de la plateforme et ce qui est fourni par le cloud de Google aurait été plus complexe à tracer, comme expliqué en introduction.

La seconde raison est que certains lecteurs ne seront potentiellement pas intéressés par l'installation ou le maintien en condition opérationnelle d'un cluster Kubernetes. Il ne serait donc pas approprié de les renvoyer à ces sections. Une solution rapide et efficace devait donc être fournie pour que ce type de lecteur puisse obtenir rapidement une plateforme pour ses tests, et se concentrer ensuite comme il le souhaite sur l'utilisation de Kubernetes elle-même.

Le choix des auteurs s'est porté sur la plateforme Azure et en particulier le service Azure Kubernetes Service, abrégé à partir de maintenant en AKS, pour sa simplicité de mise en œuvre. Microsoft fournit en effet plusieurs offres gratuites sur Azure. Celles-ci sont bien sûr limitées en ressources, mais elles ont l'avantage supplémentaire pour certaines d'entre elles de ne même pas nécessiter de carte de crédit pour s'inscrire, ce qui facilite l'inscription et réduit à zéro un éventuel risque de coûts cachés en cas de dépassement. La très bonne intégration avec la forge logicielle Azure DevOps et le registre de conteneurs Azure sont également des arguments pour commencer sur ce fournisseur de cloud.

Remarque

Par souci de transparence, le fait qu'un des auteurs soit Microsoft Most Valuable Professionnal sur Azure et bénéficie de nombreuses ressources gratuites pour les tests liés à l'écriture de ce livre a également pesé dans le choix du cloud de Microsoft. Des offres gratuites existent également sur le cloud d'Amazon ou sur le Google Container Engine/Google Kubernetes Engine, mais leur limitation en taille rendait difficile la mise au point de tous les exercices du présent ouvrage.

L'objectif, ici, étant de montrer la solution la plus simple et rapide possible pour bénéficier d'un cluster Kubernetes fonctionnel, nous passerons par l'interface graphique pour piloter la création de la ressource dans Azure. Cette interface est disponible sur https://portal.azure.com.