ILKI
YOUR CLOUD DESIGNER

Formation Kubernetes

-

Pratique

kubernetes

Atelier 9

-

ConfigMaps & Secrets

ILKI
YOUR CLOUD DESIGNER

Table des matières

1. ConfigMaps et Secrets

Kubernetes permet de séparer des options de configuration dans un autre objet nommé **ConfigMap**, qui est une map qui contient des couples clé/valeur. À noter que les valeurs peuvent aller jusqu'à un fichier de configurations complet.

Une application n'aura pas besoin de lire la **ConfigMap** directement, car le contenu sera passé en tant que variables d'environnement ou fichiers dans un volume.

Évidemment, s'il est nécessaire que l'application lise le contenu d'une **ConfigMap**, cela reste possible en interrogeant l'API **Kubernetes**.

1.1. Création de ConfigMaps

Pour cet atelier, vous allez utiliser un pod web apache et lui passer des configurations par le biais de plusieurs **ConfigMaps**.

Il existe plusieurs méthodes pour créer des **ConfigMaps**, dont :

- Création à partir d'un fichier yaml
- Création à partir de la commande **kubectl create configmap**

1.1.1. Création à partir d'un fichier yaml

1) Pour commencer, créez un fichier yaml nommé « config-vhost.yaml », qui décrira une **ConfigMap** permettant de changer le root du serveur web :

>> ConfigMap avec un grand C et apiversion : v1 et non apps/v1

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: frontweb-vhost
data:
  000-default.conf: |
    <VirtualHost *:80>
        ServerAdmin webmaster@ilki.fr
        DocumentRoot /var/www/simpleweb
        ErrorLog ${APACHE_LOG_DIR}/error.log
        CustomLog ${APACHE_LOG_DIR}/access.log combined
    </VirtualHost>
```

2) Créez la **ConfigMap** :

```
kubectl create -f config-vhost.yaml
```

3) Listez la **ConfigMap** créée :

```
kubectl get configmap
```

1.1.2. Création à partir en ligne de commande

1) Vous allez créer plusieurs **ConfigMaps** à partir du contenu d'un dossier. Pour cela, commencez par récupérez le dossier en question :

```
git clone https://github.com/ilkilab/simpleweb.git
```

2) Vous pouvez maintenant créer trois **ConfigMaps**, à partir du contenu du dossier précédemment téléchargé (contenu du frontweb), grâce aux commandes suivantes :

```
kubectl create configmap frontweb-html --from-file=simpleweb/
kubectl create configmap frontweb-css --from-file=simpleweb/css/
kubectl create configmap frontweb-templates --from-file=simpleweb/templates/
```

Note : Par défaut, la création de **Configmaps** à partir de dossier ne se fait pas de manière récursive, d'où la création de 3 **ConfigMaps** dans votre contexte.

3) Listez les **ConfigMaps** créées :

```
kubectl get configmap
```

4) Affichez le contenu d'une **ConfigMap** grâce à la commande suivante :

```
kubectl describe configmap frontweb-vhost
```

1.2. Création de pod avec des ConfigMaps

Maintenant que les quatre **ConfigMaps** sont créées, il vous est possible de créer le pod ***frontweb*** en lui injectant l'ensemble des configurations.

1) Pour cela, créez un fichier « frontweb.yaml », en injectant les configurations par le biais :

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: frontweb
  labels:
    name: frontweb
spec:
  containers:
  - name: frontweb
    image: ilkiformation/simplefrontweb:v1
    ports:
      - containerPort: 80
    volumeMounts:
      - name: vhost
        mountPath: /etc/apache2/sites-available/
      - name: html
        mountPath: /var/www/simpleweb/
      - name: css
        mountPath: /var/www/simpleweb/css/
      - name: templates
        mountPath: /var/www/simpleweb/templates
  volumes:
    - name: vhost
      configMap:
        name: frontweb-vhost
    - name: html
      configMap:
        name: frontweb-html
    - name: css
      configMap:
        name: frontweb-css
    - name: templates
      configMap:
        name: frontweb-templates
```

2) Déployez le pod en question puis repérez son IP et la node sur lequel il s'exécute.

3) Connectez-vous à la node et effectuez une requête HTTP sur le pod pour valider que la page web retournée n'est pas celle par défaut. (remplacez l'IP par celle de votre pod)

```
curl http://10.28.12.17
```

Vous devriez voir ce contenu apparaître :

```
<?php include "templates/header.php"; ?>

<ul>
        <li><a href="create.php"><strong>Create</strong></a> - add a
user</li>
        <li><a href="read.php"><strong>Read</strong></a> - find a user</li>
        <li><a href="update.php"><strong>Update</strong></a> - edit a
user</li>
        <li><a href="delete.php"><strong>Delete</strong></a> - delete a
user</li>
</ul>

<?php include "templates/footer.php"; ?>
```

1.3. Création de Secrets

Jusqu'ici, les informations de configuration ont été considérées comme non sensibles. Autrement dit, l'ensemble de celles-ci ont été transmises en clair.

Il est possible de visualiser la ConfigMap « frontweb-html » via la commande suivante :

```
kubectl get cm frontweb-html -o yaml
```

En analysant le résultat, vous pouvez remarquer qu'un fichier *« config.php »* a été transmis avec le contenu suivant :

```
config.php: |-
  <?php

  /**
   * Configuration for database connection
   *
   */

  $host       = "localhost";
  $username   = "root";
  $password   = "rootroot";
  $dbname     = "test";
  $dsn        = "mysql:host=$host;dbname=$dbname";
  $options    = array(
                  PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION
```

Le fichier contient donc les informations de connexion à une base de données. Logiquement, vous souhaiteriez que ces informations ne soient pas en clair, pour des raisons de sécurité.

Cela est possible grâce à la ressource **Secrets**. Cette ressource est très similaire à la ressource **ConfigMaps** mais elle est prévue pour détenir des informations sensibles.

Deux informations importantes sur les **Secrets** :

- Ils sont toujours stockés en RAM sur les nodes, jamais écrits sur le stockage physique
  - Contrairement aux **ConfigMaps**
- Ils sont stockés de manière chiffrée dans l'etcd

1.3.1. Création de Secrets

1) Pour commencer, supprimez le pod de l'exercice précédent.

2) Suivez les étapes ci-dessous pour créer votre « secret ».

   a. Modifiez votre fichier « *config.php* » comme suit :

```php
<?php

/**
 * Configuration for database connection
 *
 */

$host       = "simpleweb-bdd";
$username   = "root";
$password   = "azerty";
$dbname     = "test";
$dsn        = "mysql:host=$host;dbname=$dbname";
$options    = array(
                PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION
              );
```

   b. Recréez la **ConfigMap frontweb-html**

```
kubectl create configmap frontweb-html --from-file=simpleweb/
```

3) Passez à la création du **Secret** à partir du fichier « *config.php* »

```
kubectl create secret generic frontweb-config --from-file=config.php
```

4) Validez la bonne création du **Secret**

```
kubectl get secret
```

5) Affichez le contenu du **Secret**

```
kubectl describe secret frontweb-config
```

Que remarquez-vous ?

1.1.Création de pod avec des ConfigMaps

Maintenant que les **ConfigMaps** et Secrets sont prêts, il vous est possible de créer le pod **frontweb-secret** en lui injectant l'ensemble des configurations.

1) Pour cela, créez un fichier « frontweb-secret.yaml », en injectant les configurations par le biais :

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: frontweb
  labels:
    name: frontweb
spec:
  containers:
  - name: frontweb
    image: ilkiformation/apache:v1
    ports:
      - containerPort: 80
    volumeMounts:
      - name: vhost
        mountPath: /etc/apache2/sites-available
      - name: html
        mountPath: /var/www/simpleweb
      - name: css
        mountPath: /var/www/simpleweb/css
      - name: templates
        mountPath: /var/www/simpleweb/templates
      - name: config
        mountPath: /var/www/simpleweb/config
        readOnly: true
  volumes:
    - name: vhost
      configMap:
        name: frontweb-vhost
    - name: html
      configMap:
        name: frontweb-html
    - name: css
      configMap:
        name: frontweb-css
    - name: templates
      configMap:
        name: frontweb-templates
    - name: config
      secret:
        secretName: frontweb-config
```

2) Déployez le pod en question puis repérez son IP et la node sur lequel il s'exécute.

3) Connectez-vous à la node et effectuez une requête HTTP sur le pod pour valider que la page web retournée n'est pas celle par défaut.

```
curl http://10.28.12.17
```

Vous devriez voir ce contenu apparaître :

```
<?php include "templates/header.php"; ?>

<ul>
        <li><a href="create.php"><strong>Create</strong></a> - add a
user</li>
        <li><a href="read.php"><strong>Read</strong></a> - find a user</li>
        <li><a href="update.php"><strong>Update</strong></a> - edit a
user</li>
        <li><a href="delete.php"><strong>Delete</strong></a> - delete a
user</li>
</ul>

<?php include "templates/footer.php"; ?>
```

4) Connectez-vous au container et affichez le contenu du fichier *« config.php »*. Que remarquez-vous ?

2. DownwardAPI

La downward API permet de rendre disponible dans le pod les informations qui lui sont attribuées par Kubernetes. Vous allez découvrir les limites que nous traiterons dans un prochain atelier.

2.1. Variables d'environnement

1) Cherchez comment fixer les limites et réservations suivantes sur le pod ***frontweb*** :
    a. Réservations
        i. CPU = 15m
        ii. Mémoire = 100Ki
    b. Limites
        i. CPU = 1 00m
        ii. Mémoire = 4Mi

2) Lancez le pod et vérifiez la bonne configuration des limites grâce à la commande « ***kubectl describe*** »

3) Modifiez le manifeste du pod dans le but de faire passer un certain nombre d'informations en variables d'environnement. Pour cela ajoutez donc au manifeste les éléments suivants :

```
env:
- name: POD_NAME
  valueFrom:
    fieldRef:
      fieldPath: metadata.name
- name: POD_NAMESPACE
  valueFrom:
    fieldRef:
      fieldPath: metadata.namespace
- name: CONTAINER_CPU_REQUEST_MILLICORES
  valueFrom:
    resourceFieldRef:
      resource: requests.cpu
      divisor: 1m
- name: CONTAINER_RAM_LIMIT_MEGABYTES
  valueFrom:
    resourceFieldRef:
      resource: limits.memory
      divisor: 1Mi
```

4) Appliquez vos modifications et visualisez les variables d'environnement présentes dans le container grâce à la commande suivante :

```
kubectl exec frontweb env
```

5) Cherchez maintenant comment récupérer l'adresse IP du pod et le nom de la node sur laquelle le pod s'exécute et passer ces informations en tant que variables d'environnement.

6) Modifiez votre manifeste et relancez le pod.

7) Validez vos modifications grâce à la commande ***« kubectl apply »***.

2.2. Volumes

Vous allez cette fois-ci passer les informations précédentes dans un volume (et non en tant que variables d'environnement).

1) Pour cela, modifiez votre manifeste de la manière suivante :

```yaml
volumes:
  - name: downward
    downwardAPI:
      items:
      - path: "pod-name"
        fieldRef:
          fieldPath: metadata.name
      - path: "podNamespace"
        fieldRef:
          fieldPath: metadata.namespace
      - path: "labels"
        fieldRef:
          fieldPath: metadata.labels
      - path: "annotations"
        fieldRef:
          fieldPath: metadata.annotations
      - path: "containerCpuRequestMilliCores"
        resourceFieldRef:
          containerName: main
          resource: requests.cpu
          divisor: 1m
      - path: "containerMemoryLimitBytes"
        resourceFieldRef:
          containerName: main
          resource: limits.memory
          divisor: 1
```

1) Appliquez vos modifications et visualisez les informations présentes dans le container grâce à la commande suivante :

```
kubectl exec frontweb ls -lL /etc/downward
```