Cahier de TP « Spring Boot et Kubernetes »

Pré-requis :

- Bonne connexion Internet
- Système d'exploitation recommandé : Linux, MacOs, Windows 10
- IDE Recommandés : STS 4, IntelliJIDEA, VSCode
- Docker
- Solution de virtualisation : Oracle Virtual Box
- kubectl
- minikube
- Compte Docker Hub

TP1 : Déploiement image sur Kubernetes

1.1 Démarrage minikube

Démarrer minikube en utilisant l'hyperviseur virtualbox

```
minikube start --driver=virtualbox
```

Démarrer le dashboard et visualiser les ressources du cluster

```
minikube dashboard
```

1.2 Déploiement à partir d'une image

Pour cet atelier utiliser votre propre image publiée sur DockerHub

```
# Créer un déploiement à partir d'une image docker
kubectl create deployment delivery-service \
 --image=dthibau/delivery-service:0.1.6

# Exposer le déploiement via un service
kubectl expose deployment delivery-service --type LoadBalancer \
  --port 80 --target-port 8080

# Vérifier exécution des pods
kubectl get pods

# Accès aux logs
kubectl logs <pod_id>

#Visulisation IP du service
kubectl get service delivery-service
```

#Forwarding de port
```
kubectl port-forward service/delivery-service 8080:80
```

Accès à l'application via localhost:8080/actuator/info

 # Mise à jour du déploiement
```
kubectl set image deployment/delivery-service \
 delivery-service=dthibau/delivery-service:0.1.5
```

 # Statut du roll-out
```
kubectl rollout status deployment/delivery-service
```

Accès à l'application : *http:<IP>/actuator/info*

#Visualiser les déploiements
```
kubectl rollout history deployment/nginx-deployment
```

#Effectuer un roll-back
```
kubectl rollout undo deployment/delivery-service
```

#Scaling
```
kubectl scale deployment/delivery-service --replicas=5
```

TP2 : Outillage

2.1 Utilisation du plugin jKube

Récupérer le projet ***delivery-service***

Modifier les fichiers Maven pour installer le plugin *jKube*

Utiliser la propriété Maven pour contrôler le nom de l'image et donc de pouvoir la pousser sur DockerHub. Ex :

```
<jkube.generator.name>dthibau/delivery-service</jkube.generator.name>
```

Effectuer les différentes commandes pour déployer le service

2.2 Skaffold et jib

Installer skaffold : https://skaffold.dev/docs/install/

Initialiser le projet avec ***skaffold init*** en choisissant l'option Dockerfile à *none*

Editer le fichier *skaffold.yml* pour ajouter la référence à ***jib***

```
build:
  artifacts:
  - image: dthibau/delivery-service
    context: .
    jib: {}
```

Modifier le *pom.xml* pour ajouter le plugin jib

```
          <plugin>
               <groupId>com.google.cloud.tools</groupId>
               <artifactId>jib-maven-plugin</artifactId>
               <version>1.8.0</version>
          </plugin>
```

Dans un terminal, exécuter : ***eval $(minikube docker-env)*** permettant de faire pointer le client docker vers le démon de Minikube

Tester en exécutant ***skaffold build***

Puis essayer la commande ***skaffold dev***

Modifier un fichier source et observer le redéploiement

Modifier la configuration afin que les tests ne soit pas exécutés

TP3 : Configuration centralisée

3.1 ConfigMap

- Créer un ***ClusterRole*** permettant la lecture des ***ConfigMaps***
  Affecter ce rôle au compte de service ***default:default***
- Créer 2 ressources ***ConfigMap*** :
  - La première avec une seule clé et reprenant le contenu adapté à Kubernetes de ***config/src/main/resources/shared/application.yml***

    - La seconde également avec un seule clé et reprenant le continu adapté à kubernetes de ***config/src/main/resources/shared/notification.yml***

- Récupérer puis modifier le service *notification-service* afin qu'il charge ses propriétés à partir de Kubernetes
  Tenter un démarrage dans l'IDE
- Mettre en place *skaffold* sur le projet
    - Déployer le service sur Kubernetes en utilisant ***skaffold dev --port-forward***
      Utiliser le service (voir curl.txt par exemple)
    - Déployer de façon permanente via *skaffold run*

3.2 Rechargement dynamique

- Modifier *conf/shared/application.xml* afin d'autoriser le rechargement automatique de la configuration lors du profil *kubernetes*

- Tester sur *notification-service*

3.3 Secrets

- Créer un secret avec les clés ***username*** et ***password*** correspondant aux identifiant du serveur SMTP

- Changer la ressources Kubernetes de notification-service afin qu'il utilise le secret
- Changer la configuration de ***notification-service*** pour activer les secrets
- Modifier ***MailConfigurationProperties*** afin qu'il affiche sur la console ses attributs
  Tester votre configuration

TP4 : Découverte de services

4.1 DiscoveryClient

- Ajouter la configuration pour ***account-service***

- Mettre en place l'environnement ***skaffold*** dans ***account-service***

- Supprimer les ACLs temporairement (*permitAll*)

- Développer un contrôleur qui offre une ressource GET qui affiche toutes les instances d'un service passé en paramètre de la requête

4.2 FQDN et RestTemplate

Dans le projet ***account-service***, définir une ressource GET qui effectue un appel à notification-service via un *RestTemplate* et le FQDN

4.3 Répartition de charge avec OpenFeign et LoadBalancer

Reprendre le pom.xml fourni pour ***account-service***.

Désactiver Ribbon

Scaler ***notification-service***

Observer la répartition de charge avec le fichier ***.jmx*** fourni

4.4 Circuit Breaker

Ajouter la dépendance sur ***spring-cloud-starter-circuitbreaker-resilience4j***

Configurer un *CircuitBreakerFactory* et encapsuler l'appel à la notification dans un *CircuitBreaker*

Observer

TP5 : Istio

5.1 Installation Istio

Voir https://istio.io/docs/setup/getting-started/#download

5.2 Déploiement stack avec istio enabled

Dans un premier temps désactiver Istio

```
kubectl edit namespace default
```

Mettre au point un script **/deployment.sh** pour déployer les micro-services a*ccount-service, delivery-service, zipkin et notification-service*

Vérifier le bon déploiement et le nombre de pods dans un contexte sans-istio

Activer ***istio*** dans le namespace par défaut

```
kubectl label namespace default istio-injection=enabled
```

Déployer la stack et regarder les pods

Définir une gateway istio permettant le routage vers les différents micro-services :

- *account-service*
- *delivery-service*

- *zipkin*

```
kubectl apply -f store-gateway.yaml
```

Vérifier le tout avec :

```
istioctl analyze
```

Accéder à des micro-services via la gateway :

```
export INGRESS_PORT=$(kubectl -n istio-system get service istio-ingressgateway -o jsonpath='{.spec.ports[?(@.name=="http2")].nodePort}')
```

```
export SECURE_INGRESS_PORT=$(kubectl -n istio-system get service istio-ingressgateway -o jsonpath='{.spec.ports[?(@.name=="https")].nodePort}')
```

```
export INGRESS_HOST=$(minikube ip)
```

Dans un autre terminal :

```
minikube tunnel
```

Puis dans un navigateur :

```
http://$INGRESS_HOST:$INGRESS_PORT/delivery-service/
```

Accès au tableau de bord kiali

```
istioctl dashboard kiali
```

Se logger avec admin/admin

Vérifier les connexions entre micro-services et en particulier avec zipkin.

Solliciter via un script JMeter