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
- kubectl, minikube
- Compte Docker Hub

TP1 : Déploiement image sur Kubernetes

1.1 Démarrage minikube

Démarrer minikube en utilisant le pilote docker

```
minikube start
```

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
```

1.3 Manifeste kubernetes

Dans un nouveau workspace, récupérer le projet delivery-service fourni. C'est un projet SpringBoot classique n'ayant pas de dépendance sur SpringCloud.

Démarrer le projet et visualiser sa configuration actuator et en particulier le endpoint /health

Visualiser les manifestes Kubernetes fournis dans src/main/k8 ainsi que le script de déploiement .sh, exécuter un premier déploiement de la stack et vérifier son déploiement correct en accédant aux endpoints intéressant.

Modifier le fichier de ressource delivery-service.yml pour inclure les probes de liveness et readiness

TP2 : Outillage

2.1 Utilisation du plugin jKube

Ce TP est optionnel, il permet de voir ce que le plugin jKube propose mais nous utiliserons skafold et jib dans la suite des ateliers

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>
```

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

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: {}
```

Récupérer le répertoire ***k8s*** (répertoire par défaut pour le déploiement skaffold via kubectl) fourni et le mettre à la racine du projet

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

Redémarrer *skaffold* en utilisant les options ***--port-forward*** et ***--skip-tests=true***, accéder à l'application

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 nommée ***application-config*** avec une seule clé et reprenant le contenu du fichier ***application.yml*** fourni
  - La seconde nommée ***notification-config*** également avec un seule clé et reprenant le contenu du fichier ***notification-service.yml*** fourni

- Récupérer puis modifier le service *notification-service* afin qu'il charge ses propriétés à partir de Kubernetes lors de l'activation du profil ***kubernetes***
  Tenter un démarrage dans l'IDE en activant le profil ***kubernetes***
- 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/application-config.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 : Appels REST entre services

4.1 DiscoveryClient

- Reprendre le projet ***order-service*** fourni
- Mettre à jour le *pom.xml* en en ajoutant la dépendance à SpringCloudKubernetes
- Mettre à jour le bootstrap pour utiliser la config des ConfigMap de minikube
- Démarrer via skaffold
- Développer un contrôleur qui offre 2 ressources GET :
  - qui affichent tous les service disponibles
  - qui affiche toutes les instances d'un service passé en paramètre de la requête

4.2 FQDN et RestTemplate

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

Scaler notification-service et observer la répartition de charge

4.3 Répartition de charge avec Spring Cloud LoadBalancer

Ajouter le starter ***fabric8-loadbalancer*** dans ***order-service***

Utiliser l'annotation @LoadBalanced pour construire le *RestTemplate*

Observer la répartition de charge

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 : Déploiement dans le service mesh Istio

5.1 Publication des images

Ajouter les starters *zipkin-client* et *sleuth* sur les micro-services *order-service* et *notification-service*

Construire et pousser les images vers DockerHub

Vous pouver utiliser le plugin de spring-boot :

```
./mvnw clean package spring-boot:build-image
```

5.2 Installation Istio

Attention, il est recommandé d'installer *kind* : distribution kubernetes est plus performante que *minikube*, pour les ateliers suivants qui nécessitent beacoup de containers

https://kind.sigs.k8s.io/docs/user/quick-start/#installation

Installation Istio :

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

5.3 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 *order-service, delivery-service, zipkin et notification-service* en utilisant les images précédemment publiées

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 :

- *order-service*
- *delivery-service*

- *zipkin*

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

Vérifier le tout avec :

```
istioctl analyze
```

Accéder à des micro-services via la gateway :

minikube

```
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
```

kind

```
export INGRESS_HOST=$(kubectl get po -l istio=ingressgateway -n istio-system -o jsonpath='{.items[0].status.hostIP}')
```

```
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}')
```

Puis dans un navigateur :

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

5.4 Dasboard Istio/Kiali

Installer les add-ons istio : dans le répertoire d’istio :

```
kubectl apply -f samples/addons
```

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

TP6 : Canary Deployment avec Istio

https://www.digitalocean.com/community/tutorials/how-to-do-canary-deployments-with-istio-and-kubernetes