Cahier de TPs

Eclipse MicroProfile

Pré-requis :
Poste développeur avec accès réseau Internet libre
Linux (Recommandé) ou Windows 10
Pré-installation de :

- Git
- JDK11
- Docker
- Distribution Kubernetes : kind recommandé
- IDE : Eclipse, VSCode ou IntelliJ, lombok
- Quarkus CLI

Solutions des ateliers :

https://github.com/dthibau/microprofile-solutions

Atelier 1 : CDI

.

Objectifs

- Annotations CDI
- Exemple d'intercepteur

1. 1 Création de projet

Installer Quarkus CLI :
`https://quarkus.io/guides/cli-tooling#installing-the-cli`

Créer un projet ***delivery-service*** avec l'extension resteasy-reactive

```
quarkus create app org.formation:delivery-service \
    --extension=resteasy-reactive
cd delivery-service
quarkus dev
```

Accéder à la page d'accueil et :

- Visiter la devui
- La ressource /hello

1.2 Récupération des sources

Mettre à jour votre ***pom.xml*** afin qu'il référence une bibliothèque de sérialisation JSON et lombok

Si vous êtes avec Eclipse, vous devez configurer le démarrage d'Eclipse afin qu'il prenne en compte lombok

Récupérer les sources fournis

1.3 Annotations

Annotez les sources fournis afin que le endpoint ***localhost:8080/livraisons*** renvoie une liste de Livraison en JSON

Atelier 2 : Configuration

Objectifs

- Configuration propriétés Quarkus
- Objets de configuration
- Injection de propriété
- Profils de configuration

2.1 Configuration

Via *src/main/resources/application.yml*, configurer Quarkus afin que le serveur écoute sur le port 8000

Définir une interface qui définit les propriétés d'un service de notification :

- protocole (http ou https)
- Base URL
- port
- URI
- API Token

Ajouter des contraintes de validation et vérifier quelles sont effectives

S'injecter l'objet configuration dans le bean ***LivraisonServiceImpl***

2.2 Profils

Définir un autre port pour le profil de ***integration***

La particularité de Quarkus est que le profil d'exécution est défini au moment du build

Par exemple :

```
./mvnw package -Dquarkus.profile=integration
```

Démarrer l'application via :
java -jar target/quarkus-app/quarkus-run.jar

Vérifier l'activation du profil

Atelier 3 : API Rest

.

Objectifs

- Annotations JAX-RS
- Utilisation JSON-P
- Problématiques classiques API Rest
- Invoquer un autre service REST

3.1 Mapping RestEasy-Reactive

Préfixer toutes les urls de l'application par "/api/v1"

Implémenter une API CRUD permettant

- Accéder à une livraison
- Créer/mettre à jour supprimer une livraison

Vérifier les endpoints par la DevUI

Implémenter un endpoint avec le modèle réactif.

Observer les threads utilisés pour les différents endpoints

3.2 Annotations JSON-B

Modifier la dépendance afin dans le *pom.xml* afin d'utiliser JSON-B à la place de Jackson

A l'aide des annotations JSON-B définir 1 formats de sérialisation pour la classe Livraison à utiliser lors de l'affichage de la liste (on pourra utiliser une classe DTO)

Annoter afin d'obtenir ce résultat :

```
{
  id: 1,
  creation-date: "23/05/2022 09:29",
  no-commande: "1",
  status: "DISTRIBUE",
  id-livreur: 2
}
```

3.3 OpenAPI

Ajouter la dépendance

```
<dependency>
  <groupId>io.quarkus</groupId>
  <artifactId>quarkus-smallrye-openapi</artifactId>
</dependency>
```

pour obtenir la documentation Swagger/OpenAPI

Accéder aux URLS :

- http://localhost:8000/q/swagger-ui/
- http://localhost:8000/q/openapi

Atelier 4 : APIs pour la répartition

4.1 Rest Client

Récupérer les sources du projet ***notification-service,*** le démarrer et visualiser la documentation Swagger

Dans *delivery-service*, appeler le service de notification via un client REST lors de la création d'une Livraison

4.2 Fault-tolerance

Ajouter la dépendance :

```
<dependency>
  <groupId>io.quarkus</groupId>
  <artifactId>quarkus-smallrye-fault-tolerance</artifactId>
</dependency>
```

Mettre en place les annotations *@Retry*, *@CircuitBreaker* et *@FallBack* sur la connexion entre *delivery-service* et *notification-service*

Vous pouvez utiliser le script JMeter fourni pour tester

4.3 : Émission de message vers topic Kafka

Importer les sources fournis du projet ***order-service*** et visualiser les dépendances

Démarrer le projet et observer le démarrage d'une image docker

Configurer un canal de sortie *order-out* pointant vers le topic Kafka *order*

Émettre un message de type *OrderEvent* lors de la création d'un objet Order.

La méthode via les déclaration de channels est recommandée car elle permet de profiter des sérialiseurs JSON automatiquement

4.4 Réception de message

Du côté du projet ***product-service*** :

- Ajouter l'extension pour Kafka
- Définir un channel pointant sur le topic précédent
- S'abonner au channel et effectuer le traitement suivant :
  - Création d'un ticket
  - Publication sur le topic *order* d'un OrderEvent avec le statut PENDING

4.5 Réception de message et SSE (Optionnel)

Revenir dans le projet ***order-service***

- Dans le controller, ajouter un endpoint */**status*** renvoyant un *Multi<OrderEvent>* correspondant au topic order au format MediaType.SERVER_SENT_EVENTS
- Récupérer le fichier *orders.html* et le placer dans src/main/resources/META-INF/resources
- Le comprendre

Créer via le swagger de order-service une commande et visualiser les événements dans la page html (http://<server>/orders.html

Vous pouvez également utiliser le script JMETER fourni

Atelier 6 : Observabilité

6.1 Health

Ajouter la dépendance :

```
<dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-smallrye-health</artifactId>
</dependency>
```

Accéder aux URLs de santé (/q/health).

Ecrire un indicateur indiquant si l'espace disque restant est supérieru ç un seuil configuré

6.2 Open Tracing

Ajouter la dépendance :

```
<dependency>
  <groupId>io.quarkus</groupId>
  <artifactId>quarkus-smallrye-opentracing</artifactId>
</dependency>
```

sur les projets ***delivery-service*** et ***notification-service***

Également leur ajouter la configuration suivante :

```
quarkus.jaeger.service-name=formation
quarkus.jaeger.sampler-type=const
quarkus.jaeger.sampler-param=1
quarkus.log.console.format=%d{HH:mm:ss} %-5p traceId=%X{traceId},
parentId=%X{parentId}, spanId=%X{spanId}, sampled=%X{sampled}
[%c{2.}] (%t) %s%e%n
```

Démarrer le serveur Jaeger :

```
docker run -p 5775:5775/udp -p 6831:6831/udp -p 6832:6832/udp -p
5778:5778 -p 16686:16686 -p 14268:14268 jaegertracing/all-in-one:latest
```

Démarrer les 2 micro-services et créer des Livraison.

Visitez l'interface jaeger à *http://localhost:16686/*

6.2 MP Metrics

Sur delivery-service ajouter la dépendance :

```
<dependency>
  <groupId>io.quarkus</groupId>
```

```
    <artifactId>quarkus-smallrye-metrics</artifactId>
  </dependency>
```

Démarrer l'application et accéder à http://localhost:8000/q/metrics

Ajouter des annotations sur le contôleur afin de compter, mesurer, etc …

Optionnel : Prometheus et Grafana

Visualiser le fichier de configuration Prometheus fourni et vérifier qu'il correspond à votre configuration de ***delivery-service***

Démarrer un serveur Prometheus :

```
docker run -d --name prometheus -p 9090:9090 -v
<PATH_TO_PROMETHEUS_YML_FILE>/prometheus.yml:/etc/prometheus/prome
theus.yml prom/prometheus
–config.file=/etc/prometheus/prometheus.yml
```

Accéder à http://localhost:9090/ et visualiser les métriques récupérés

Démarrer ensuite un serveur Grafana :

```
docker run -d --name grafana -p 3000:3000 grafana/grafana
```

Dans le menu DataSource ajouter le serveur Prometheus

Il ne reste plus qu'à créer un Dashboard Grafana

Atelier 7 : Sécurité

Objectifs

7.1 Authentification OpenId et Protection via Bearer Token

Objectif : utiliser l'extension Quarkus OpenID Connect (OIDC) pour protéger les endpoints de ***delivery-service*** à l'aide de l'autorisation d'un jeton Bearer émis par OpenID Connect et des serveurs d'autorisation conformes à OAuth 2.0 tels que Keycloak.

Ajouter la dépendance suivante :

```
<dependency>
  <groupId>io.quarkus</groupId>
  <artifactId>quarkus-oidc</artifactId>
</dependency>
```

Ajouter des contrôles d'accès :

- Role ***user*** pour les requêtes GET
- Rôle ***admin*** pour les autres requêtes

Démarrer l'application et accéder à la DevUI puis au lien OpenIDConnect
Se logger avec ***bob/bob***
Observer le token d'accès et vérifier sa conformité avec MP-JWT

Accéder à une URL GET (vous pouvez utiliser le lien swagger)
Essayer une URL POST
Observer les requêtes envoyées avec F12
Se logger avec ***alice/alice*** et essayer une URL POST

Implémenter un endpoint qui affiche le jeton

7.2 RestClient (Optionnel)

Ajouter OIDC à notification-service et protéger les endpoints via le rôle ***admin***

Ajouter l'extension ***quarkus-oidc-token-propagation-reactive*** au projet ***delivery-service***
Profiter de la configuration par défaut
Application du filtre ***AccessTokenRequestReactiveFilter*** sur REST Client accédant à notification-service