

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

5.1 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

5.2 Fault-tolerance

5.3 : 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

5.4 Réception de message

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

- Ajouter l'extension pour Kafka
- Définir un channel pointant 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

5.5 Réception de message et SSE

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

Visualisation des endpoints de health.

Ecriture d'un indicateur

6.2 Open Tracing

Mise en place d'OpenTracing sur delivery-service et notification-service

6.2 MP Metrics

Atelier 7 : Sécurité

Objectifs

7.1 Authentification HTTP

Sur le projet *order-service*, ajouter l'extension : *elytron-security-properties-file*

Mettre en place un *IdentityProvider* basé sur un fichier *.properties* comme suit :

```
%dev.quarkus.security.users.embedded.enabled=true
%dev.quarkus.security.users.embedded.plain-text=true
%dev.quarkus.security.users.embedded.users.scott=reader
%dev.quarkus.security.users.embedded.users.stuart=writer
%dev.quarkus.security.users.embedded.roles.scott=READER
%dev.quarkus.security.users.embedded.roles.stuart=READER,WRITER
```

Utiliser les annotations afin que les endpoints du controller GET soient accessibles par le rôle READER et les autres endpoints par le rôle WRITER

7.2 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 l'extension *oidc*

Ajouter des contrôles d'accès :

- Rôle *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

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

7.3 RestClient

Objectif : utiliser l'extension *quarkus-oidc-client* afin que l'appel REST de *delivery-service* vers *notification-service* fournisse un jeton Bearer. Le service notification peut alors être protégée

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

Atelier 8: Déploiement

Objectifs

- Construction d'image
- Déploiement vers un cluster
- Remote developmennt / Debug developmenet
- Variables d'environnement et ConfigMap

8.0 Installation Kubernetes

- minikube
- kind

+ Helm

8.1 Construction d'image

Créer un nouveau projet :

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

Ajouter l'extension **container-image-jib**

Construire l'image en activant le profil de *dev* et en autorisant éventuellement contrôler le nom de l'image avec **quarkus.container-image.name**

Push vers DockerHub (Nécessite un compte DockerHub)

Exécuter ensuite l'image dans l'environnement local et essayer d'accéder à l'application

8.2 Kubernetes

Ajouter l'extension Kubernetes

Effectuer un build et visualiser les ressources Kubernetes créées

Assurer vous que votre cluster Kubernetes est démarré et que votre client kubectl est effectif

Déployer les ressources sur votre cluster avec *kubectl*

Vérifier le démarrage du pod et la présence du service

Effectuer un port-forward afin de pouvoir accéder au service via un port local :

```
kubectl port-forward service/greeting-service 8080:80
```

Ajouter la propriété **quarkus.kubernetes.deploy=true**

Définir également le namespace default

8.2.2 Development mode

Ajouter la configuration suivante :

```
quarkus.kubernetes.service-type=node-port
quarkus.package.type=mutable-jar
quarkus.kubernetes.env.vars.quarkus-launch-devmode=true
quarkus.live-reload.password=secret
```

Effectuer un déploiement, vérifier le remplacement du container (kubectl get -w pod)
Lors la nouvelle version est déployée, vérifier les logs (kubectl logs <pod-id> afin d'y voir :
(Quarkus Main Thread) Profile dev activated. Live Coding activated.

Récupérer le host de votre cluster :

```
minkube: minikube ip
```

```
kind: kubectl get po -o jsonpath='{.items[0].status.hostIP}'
```

Récupérer également le NodePort de greeting-service

Accéder à l'URL `http://<cluster-host>:<NodePort>`

Vous devez voir la page de Welcome de Quarkus

indiquer cette URL dans la propriété :

```
quarkus.live-reload.url
```

Lancer ensuite `mvn quarkus:remote-dev`

Vous devez voir : Connected to remote server

Modifier le source et faire reload dans le navigateur

Essayer sur un autre projet