Rapport sur l'Architecture Microservices de l'Online Library

Auteur

`Mohamed Djawad Abi Ayad`

Details de la remise

`Projet 3`

`MGL7361` : `PRINCIPES ET APPLICATIONS DE LA CONCEPT.DE LOGICIELS`

`Professeur` : `Boulgoudan, Moustapha`

Sommaire

Introduction

L'Online Library est une application basée sur une architecture microservices.
Chaque fonctionnalité majeure est décomposée en un service autonome, interconnecté via un réseau Docker.
L'objectif : offrir scalabilité, résilience et souplesse d'évolution.

Deux approches de coordination sont adoptées :

- **Orchestration** : utilisée pour les workflows simples nécessitant un contrôle centralisé.
- **Chorégraphie** : employée pour les workflows complexes impliquant plusieurs services interagissant directement.

Migration depuis un monolithe modulaire

L'application était initialement conçue sous forme de **monolithe modulaire**.
Dans la nouvelle architecture, nous avons **extrait le service Cart et toute la logique associée au panier du service Order**.
Les motivations :

- **Réduire la responsabilité du service Order** pour améliorer sa maintenabilité.
- **Permettre la mise à l'échelle indépendante** : les opérations sur le panier (très fréquentes) peuvent maintenant être gérées séparément des commandes (moins fréquentes).
- **Accroître la résilience** : un grand nombre d'utilisateurs manipulent le panier, tandis que seul un petit nombre déclenche un achat, justifiant cette séparation.

De plus, il est envisagé **d'extraire les services Notification et Delivery des services Payment et Order respectivement**.
Cette décision est motivée par leur **nature asynchrone**, qui nécessite une gestion indépendante pour améliorer la réactivité et faciliter la montée en charge.

Configuration et Déploiement

Conteneurisation

Tous les microservices tournent dans des conteneurs Docker orchestrés via Docker Compose.
Ce choix garantit :

- isolation des environnements ;
- portabilité sur Windows, Linux et WSL 2 ;
- déploiement automatisé grâce à des scripts (.ps1 et .sh).

Services déployés

- **Caddy (proxy)** : point d'entrée unique exposant l'API (:80).
- **Webapp** : interface utilisateur (:5173).
- **Orchestre** : service central coordonnant les workflows simples (:8080).
- **Profil, Inventary, Cart, Payment, Order** : services métier autonomes, chacun relié à une base PostgreSQL.

Déploiement

Windows :

```
docker-compose up -d
ou
./FullClean-Deploy-Docker.ps1
```

Linux :

```
docker-compose up -d
ou
./FullClean-Deploy-Docker.sh
```

Scripts complémentaires :

- `Display-Logs.ps1` / `Display-Logs.sh` : logs des conteneurs.
- `Display-DBTables.ps1` / `Display-Logs.sh` : inspection des bases PostgreSQL.

Documentation des API REST

Chaque microservice expose et documente ses API REST via **Swagger**.
Après le démarrage de l'application, la documentation est accessible à :

```
http://localhost:{PORT_DU_SERVICE}/swagger-ui/index.html#/
```

Exemples :

- Profil Service : http://localhost:8081/swagger-ui/index.html#/
- Inventary Service : http://localhost:8082/swagger-ui/index.html#/
- Cart Service : http://localhost:8085/swagger-ui/index.html#/
- Payment Service : http://localhost:8083/swagger-ui/index.html#/
- Order Service : http://localhost:8084/swagger-ui/index.html#/
- Orchestre Service : http://localhost:8080/swagger-ui/index.html#/

Workflows et Coordination

1. **Authentification utilisateur (orchestration)** :
   - User → Caddy → Orchestre → Profil : création/vérification du compte.
   - Réponse renvoyée à l'utilisateur via Orchestre et Caddy.
2. **Recherche de livres (orchestration)** :
   - User → Caddy → Orchestre → Inventary : recherche dans l'inventaire.
   - Résultat agrégé et renvoyé à l'utilisateur.
3. **Ajout au panier (orchestration)** :
   - User → Caddy → Orchestre → Cart : ajout des articles.
   - Confirmation renvoyée à l'utilisateur.

4. **Paiement et commande (chorégraphie)** :
   Workflow complexe où plusieurs services interagissent directement :
   - User → Caddy → Orchestre → Payment : initie le paiement.
   - **Payment** :
     - Récupère le montant total auprès de `Cart`.
     - Génère la facture(processus interne).
     - Déclenche la création d'une commande via `Order`.
     - Récupère l'email utilisateur via `Profil`.
     - Notifie l'utilisateur et confirme la commande(processus interne).
   - **Order** :
     - Récupère l'adresse auprès de `Profil`.
     - Récupère et déclenche paar la suite le vidange du panier dans `Cart`.
     - Intérroge `Inventary` pour la mise a jour du stock.
     - Déclenche la livraison (processus interne).
   - **Orchestre** renvoie la confirmation globale.

Conclusion

- **Orchestration** : pour les processus simples (authentification, recherche, panier), garantissant un contrôle centralisé.
- **Chorégraphie** : pour le processus complexe de paiement, favorisant l'autonomie et le découplage des services.
- **Conteneurisation Docker** : déploiement fiable et portable sur Windows, Linux et WSL2.

Cette combinaison permet d'atteindre un équilibre entre simplicité, évolutivité et résilience.