Projet Microservice

Auteurs
Ibrahim Traoré, Ilyass Koné

Université Paris Dauphine

Juin 2025

Table des matières

Table des figures

1 Introduction

La transition vers des architectures **microservices** représente aujourd'hui une solution privilégiée dans la conception d'applications complexes, distribuées et évolutives. Contrairement aux architectures monolithiques, les microservices offrent une meilleure **modularité**, une **scalabilité accrue** et facilitent le **déploiement indépendant** de chaque composant métier.

Dans le cadre de ce projet réalisé en Master 2 MIAGE IF, nous avons conçu une application de **planification de voyages touristiques**, reposant sur une architecture distribuée. Celle-ci est composée de cinq microservices principaux :

- **city-service** : gère les villes et leurs informations associées,
- **activity-service** : gère les activités disponibles par ville,
- **accommodation-service** : gère les hébergements,
- **trip-service** : centralise la création et la planification de voyages incluant des étapes,
- **gateway-service** : joue le rôle de point d'entrée unique vers les services, avec routage via Spring Cloud Gateway.

Chaque service fonctionne de manière autonome, expose des API REST et utilise une base de données adaptée à ses besoins (MongoDB ou PostgreSQL). Ces services communiquent entre eux via le **Gateway**, qui simplifie l'interaction globale du système.

Une interface front-end a également été amorcée avec **ReactJS**, dans le but d'illustrer l'usage concret des API. Cependant, en raison de certaines contraintes techniques et de temps, cette interface n'a pas pu être entièrement finalisée ni stabilisée à ce stade. Néanmoins, les tests d'intégration via Postman confirment le bon fonctionnement de bout en bout du système.

2 Méthodologie : Les bases de données utilisées

2.1 PostgreSQL

PostgreSQL est un Système de Gestion de Base de Données Relationnelle (SGBDR).

Il a été initialement développé à l'Université de Berkeley sous le nom de Ingres (1977-1985). Il a été amélioré régulièrement au cours de ses premières années par les sociétés Rational Technologies et Ingres Corp.. Ces sociétés ont été rachetées par la société nommée maintenant Informix. Le projet a ensuite continué indépendamment à Berkeley sous le nom de Postgres (1986-1994). Après l'apparition des premières normes du langage SQL, le langage de requête de Postgres a été remplacé par le langage SQL. En 1995, le projet a ensuite été repris par des développeurs indépendants de l'Université de Berkeley et renommé Postgres95. Là, le projet s'est étoffé et transformé à partir de 1997 en PostgreSQL lorsque l'ensemble des fonctionnalités SQL a été intégré au serveur.

2.1.1 Avantages et Inconvénients

Il possède donc tous les avantages qui en découlent, entre autres :

- La gratuité ;
- Le nombre de déploiements illimité ;
- La communauté de développement active et réactive ;
- Les possibilité d'extension à volonté (le code source étant disponible gratuitement, il est possible de modifier ou d'étendre les fonctionnalités de PostgreSQL de façon autonome).

PostgreSQL respecte la quasi-totalité de la norme SQL et propose la plupart des fonctionnalités présentes dans les autres "grands" SGBD (gestion des transactions, procédures stockées, gestion des droits d'accès aux données, ...).

PostgreSQL, s'il est connu pour sa robustesse, a également la réputation d'être relativement lent (même si chaque nouvelle version amène des progrès notables). Cela est dû au grand nombre de mécanismes de préservation et de protection des données qui sont activés par défaut mais qui peuvent pour la plupart être désactivés.

2.2 MongoDB

MongoDB mongoDB Inc. possède une architecture de schéma dynamique qui fonctionne avec des données et un stockage non structurés. Les données étant stockées dans des documents flexibles de type JSON, le schéma de base de données n'a pas besoin d'être prédéfini et peut être modifié de manière dynamique sans entraîner d'arrêt.

Avec le format de données BSON de MongoDB, les objets d'une même collection peuvent avoir différents ensembles de champs, et presque tous les types de structures de données peuvent être modélisés et manipulés. Le modèle de base de données flexible de MongoDB s'avère ainsi particulièrement avantageux lorsque les besoins de l'entreprise et les exigences en matière de données évoluent.

2.3 Neo4j

Neo4j D-Booker [2020] est un système de gestion de bases de données orienté graphe. À la différence des systèmes classiques, son approche n'est pas fondée sur l'algèbre relationnelle mais sur la théorie des graphes. Les données sont stockées de manière assez libre (sans modèle prédéterminé) dans des nœuds reliés entre eux par des relations (des arcs porteurs d'une sémantique forte). Neo4j utilise un langage d'interrogation spécifique CYPHER, conçu pour être assez intuitif.

Les bases de données Neo4j, et d'une manière plus générale les bases de données orientées graphes, sont particulièrement adaptées dans des contextes où les données sont fortement connectées et organisées selon des modèles complexes. La structure de l'entité (noeud ou relation) y est définie au moment du stockage de la donnée (et non préalablement comme les tables d'une base de données relationnelle), ce qui lui confère une très grande flexibilité. En outre, elles présentent des performances exceptionnelles en termes de lecture et de parcours de données dans le graphe.

3 Indications pour compiler et exécuter le projet

Le projet est structuré en plusieurs microservices indépendants, chacun développé avec **Spring Boot** et **Maven**. Voici les étapes nécessaires pour compiler et exécuter l'ensemble de la solution.

Prérequis

- Java JDK 17 installé
- Maven installé (`mvn -v` pour vérifier)
- MongoDB local (par défaut sur `localhost:27017`)
- Docker et Minikube (optionnel pour le déploiement conteneurisé)
- Postman pour tester les requêtes API REST

Structure des services

Chaque microservice est un projet Maven autonome :

- `city-service`
- `activity-service`
- `accommodation-service`
- `trip-service`
- `gateway-service`

Lancement d'un service

1. Ouvrir un terminal dans le dossier du service souhaité (exemple : `trip-service`)

2. Lancer la commande suivante :

   ```
   mvn spring-boot:run
   ```

3. Le service démarre sur le port défini dans `application.properties`

Test des endpoints

Les services peuvent être testés à l'aide de Postman. Exemple pour `trip-service` :

```
GET http://localhost:8080/api/trips
```

Accès via le Gateway

Une fois le `gateway-service` lancé, les services sont disponibles via le port 8084 :

```
http://localhost:8084/api/trips
http://localhost:8084/api/cities
http://localhost:8084/api/activities
http://localhost:8084/api/accommodations
```

Remarque importante

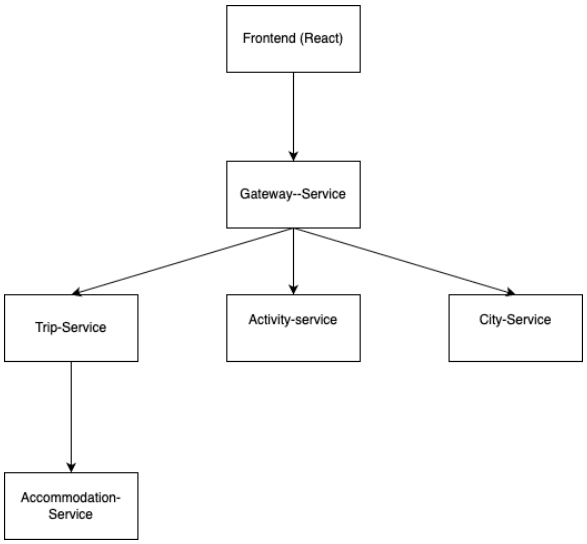
La partie frontend développée avec `React.js` a bien été initialisée et connectée, mais n'a pas pu être finalisée en raison de problèmes techniques liés aux politiques CORS. Une partie des appels API échoue encore, ce qui empêche le bon affichage des données. Cette partie reste à corriger ou à améliorer.

4 Documentation technique

4.1 Schéma d'architecture globale

Le projet repose sur une architecture de type **microservices**, où chaque entité métier (ville, hébergement, activité, voyage) est gérée par un service indépendant :

- **Gateway-Service** : point d'entrée unique, permet de router les requêtes.
- **City-Service**, **Accommodation-Service**, **Activity-Service** : chaque service expose une API REST et utilise une base NoSQL (MongoDB, Neo4j, PostgreSQL).
- **Trip-Service** : centralise la logique de planification d'un voyage en orchestrant les autres services.

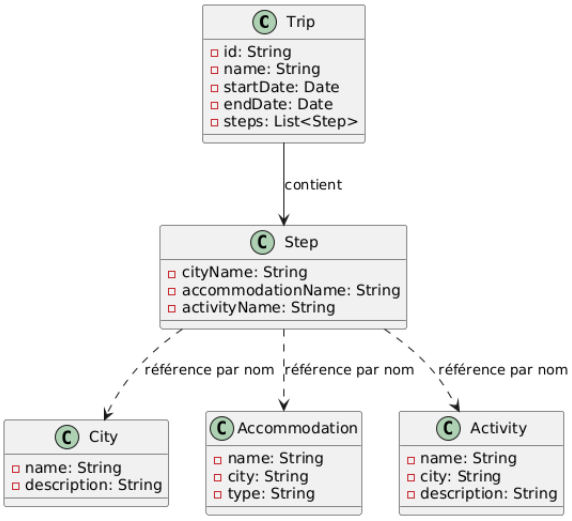


4.2 Choix techniques

- **Backend** : Spring Boot 3.x pour chaque microservice, Maven pour le build.
- **Base de données** :
  - MongoDB pour les services City et Trip.
  - PostgreSQL pour Accommodation.
  - Neo4j pour Activity (graphes).
- **Communication** : API REST via Spring Web, conteneurisation Docker, routage avec Spring Cloud Gateway.
- **Déploiement local** : via Minikube (Kubernetes local).

4.3 Diagramme de classes métier

Voici le diagramme de classes représentant les entités centrales manipulées dans les services.



Ce diagramme montre les entités suivantes :

- **Trip** contient une liste de **Step**.
- Chaque **Step** référence des entités par nom :
  - **City**,
  - **Accommodation**,
  - **Activity**.

Remarque : Il ne s'agit pas d'un diagramme d'implémentation. Les méthodes (`getName()`, etc.) ne sont donc pas représentées ici.

5 Conclusion

Ce projet nous a permis d'approfondir concrètement les notions vues en cours autour des architectures microservices, de la communication entre services REST, de l'intégration avec MongoDB et de la conteneurisation via Docker. Nous avons apprécié l'approche modulaire qui permet de développer, tester et déboguer chaque composant indépendamment.

Nous avons particulièrement aimé la clarté offerte par l'organisation en services, ainsi que la montée en compétence sur la configuration des routes via un `gateway-service` central. La mise en place des tests de bout-en-bout à travers Postman a aussi été un point fort qui a validé le bon fonctionnement global du système.

Du côté des difficultés, la configuration du `CORS` pour l'interface web a été un point bloquant que nous n'avons pas pu résoudre entièrement dans les délais. Cela a limité la finalisation de la partie `frontend` même si les données étaient bien créées et récupérées côté serveur.

En résumé, ce projet a été très formateur sur le plan technique, mais aussi en termes de gestion de projet. Il nous a permis d'expérimenter un écosystème réaliste, proche des pratiques actuelles dans l'industrie logicielle.

Références

mongoDB Inc. Qu'est-ce que mongodb et comment fonctionne-t-elle ? URL https://www.purestorage.com/fr/knowledge/what-is-mongodb.html. Guide complet sur MongoDB, base de données NoSQL orientée document.

D-Booker. Neo4j : Une autre façon de manipuler les données, 2020. URL https://www.d-booker.fr/content/68-introduction-bases-de-donnees-neo4j. Introduction aux bases de données Neo4j.