Cas pratique : Architecture Microservice

Loic Sanou : Ingénieur logiciel chez Capgemini

December 14, 2023

Abstract

Cette présentation pratique ne montre pas de bout en bout ce qui a été fait dans la démonstration de notre sujet portant sur les microservices. Nous allons mettre en exergue les grandes lignes que nous avons utilisées pour mettre en ouvre ce cas pratique.

Nous allons vous décrire en quoi consite l'exemple sur lequel nous nous sommes basés et détailler les aspects importants pour la compréhension de ce sujet.

Le TP est accéssible à l'adresse suivante : https://github.com/loicosquare/microservice-presentation

1 Introduction et contexte

1.1 Introdcution

Ces dernières années, les microservices ont connu une explosion remarquable en raison de leur efficacité et de leur facilité de maintenance. Contrairement aux architectures monolithiques, les microservices offrent une scalabilité accrue et sont mieux adaptés, de nos jours, à la conception d'applications cloud telles que celles utilisées par des géants comme **Netflix, Amazon**, et bien d'autres. Notre objectif n'est pas de vous donner un cours complet ici, mais vous pouvez trouver une présentation détaillée sur ce sujet en suivant ce lien : https://github.com/loicosquare/microservice-presentation.

1.2 Contexte

Notre application se concentre sur un réseau d'entreprises spécialisées dans la fabrication de jeux vidéo. Par exemple, Sony crée des titres comme FIFA 24 pour la console PS5. L'objectif principal est de recueillir l'appréciation ou les notes attribuées par les utilisateurs à ces produits (les jeux qu'ils ont achetés), sachant qu'un jeu (produit) appartient à une catégorie car tous les utilisateurs ont leur préférence. De plus, nous cherchons à évaluer les connaissances des utilisateurs concernant un jeu spécifique, une entreprise ou tout autre aspect via des quiz comportant diverses questions. Par exemple, une question pourrait porter sur l'année de sortie du premier jeu FIFA sur Playstation (quiz) avec des options de réponse comme 2005 ou 2000 (questions).

1.3 Différents microservices

Notre architecture se compose de plusieurs microservices, chacun représentant une entité spécifique. Voici un résumé détaillé de ces entités interconnectées :

Entreprise: Ce service représente une entreprise de fabrication de jeux vidéo, par exemple, Sony, EA Sports, etc.

Produit (Jeu): Chaque entreprise crée des produits tels que des jeux vidéo. Par exemple, FIFA 24 pour PS5 serait un produit spécifique associé à une entreprise.

Catégorie: Chaque catégorie contient une variété de jeux (Football, Aventures, etc...).

Utilisateur: Les utilisateurs interagissent avec l'application en effectuant des paiements pour acheter des produits (jeux) et en donnant leur avis sous forme de notes (ratings) sur les produits qu'ils ont achetés.

Paiement: Ce service gère les transactions financières entre les utilisateurs et les produits achetés.

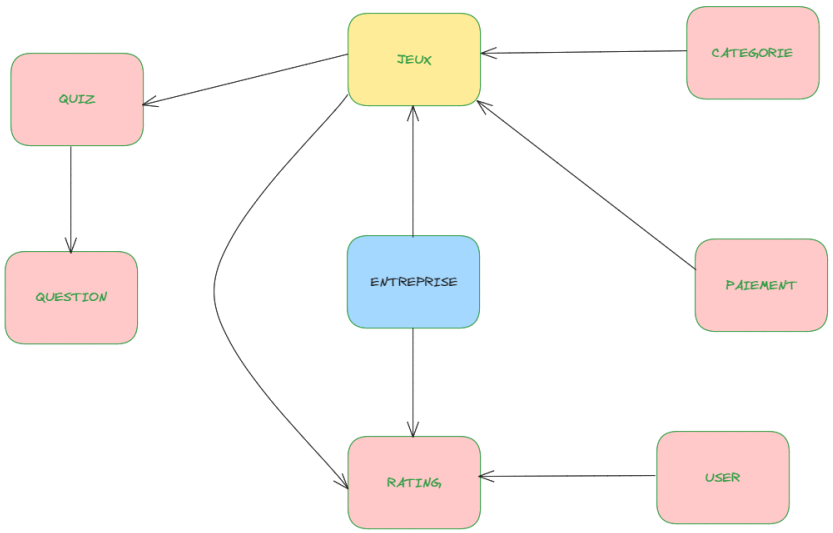

Figure 1: Architecture Microservice.

Note (Rating): Les utilisateurs peuvent attribuer des notes aux jeux qu'ils ont achetés, reflétant ainsi leur appréciation.

Quiz: Les utilisateurs ont la possibilité de répondre à des quiz comprenant des questions variées.

Question: Ces questions peuvent porter sur des détails spécifiques concernant un jeu ou une entreprise, permettant d'évaluer les connaissances des utilisateurs.

L'aspect crucial de notre architecture réside dans les dépendances fonctionnelles entre ces microservices. Cette interconnexion nous permet de mettre en œuvre des concepts essentiels tels que la tolérance aux fautes (tolerance to faults), les mécanismes de rupture de circuit (circuit breakers), etc. Ces fonctionnalités sont particulièrement utiles en cas d'indisponibilité d'un microservice, garantissant la continuité des opérations et la disponibilité des fonctionnalités de l'application.

En résumé, notre architecture orientée microservices comprend différents modules interdépendants pour gérer les entreprises, les produits, les utilisateurs, les paiements, les évaluations sous forme de notes et les quiz avec leurs questions. Ces éléments sont conçus pour fonctionner de manière cohérente tout en offrant la flexibilité et la fiabilité nécessaires pour une application robuste et performante.

1.4 How to add Lists

Pour faire cette première partie du TP nous avons recensé quelques étapes ...

1. créer normalement vos microservices (Vos applications monolithiques).
2. créer le services Registry. (Eureka Server, spring cloud) qui contiendra la liste de vos microservices.
3. Implémenter le discovery Client : renseigner nos différentes services pourqu'ils soient enregistrés dans le service Registry. (netflix eureka client et toujours mettre spring cloud starter), Toujours renseigner EurekaServer dans la classe main
4. dans chaque microservice concerné mettre dans le application.yml—proporties les instructions pour se connecter à Eureka server (Service Rgistry) et toujours renseigner Eureka client dans les classes main.
5. Appeler vos microservice si besoin en utilisant RestTemplate ou Feign Client (Que nous aborderons dans la pratique proprement dite).
6. Créer le API GATEWAY qui sert de routing, il est aussi un client (discovery client): il doit toujours être enregistré à Eureka server (Service registry).

7. Ajouter les routes des microservices dans le API GATEWAY en se servant de la doc : https://docs.spring.io/spring-cloud-gateway/reference/spring-cloud-gateway/configuring-route-predicate-factories-and-filter-factories.html

8. Une fois que le API gateway est fait on doit centraliser toutes les configurations pour connecter au server Eureka vers le config-server (stocké sur github); il doit aussi être enregistré en tant que discovery client au service registry.

9. Déplacer les configs de nos microservices vers un dépot centralisé le config-server en ligne.

10. Vu que les configurations sont stockées en ligne, rendre nos microservices comme config-client pour qu'ils aillent fetch les configs. En suite bien vérifier qu'au lancement de nos microservices ceux-ci chargent les ressources sur le config-server-local en local.

11. Pour la tolérances aux fautes en cas d'indisponibilité d'un microservice, penser aux circuits breakers, au retry, au ratelimiter pour éviter qu'un utilisateur lance un nombre de requêtes illimitées.

1.5 Conclusion

Dans cet extrait nous avons détaillé l'ensemble des étapes pour mettre sur pieds le projet dont nous avons décrit plus haut. Il peut sembler flou, ou incompréhensible mais sur le Github du dépot vous avez un code bien détaillée et chaque étape décrite plus haut est stocké sur une branche différence pour vous donner une visibilité et une tracabilité des modification faites à chaque fois.
[Goo23].

References

[Goo23] et autres pages webs consultées Google. Recherches sur les microservices. 14(12):2023, 2023.