ALOM - Tony Landschoot

TP 5 CHAT
DOCUMENTATION

Démarrage du projet

Le projet se compose en 4 microservices. Ce sont tous des projets java. Java 8 est requis car il propose une meilleure compatibilité avec les technologies utilisées (jersey, kafka, socket...).

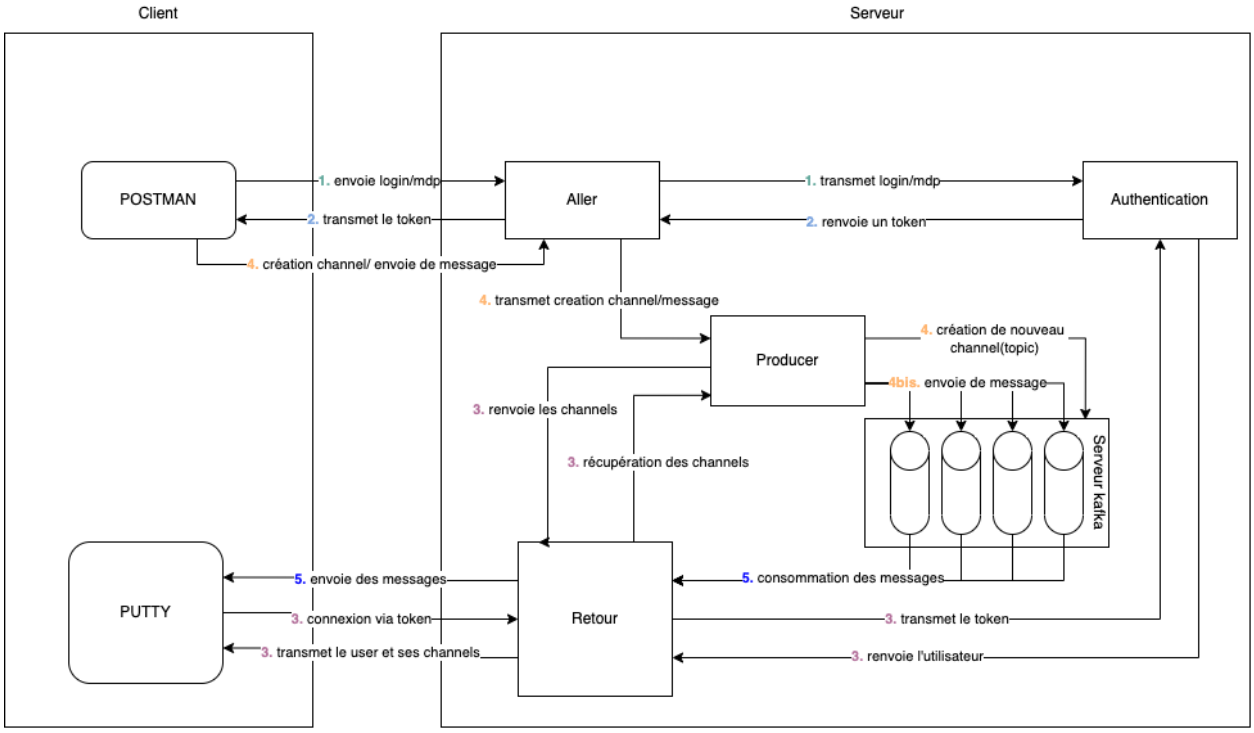
Ils peuvent tous être lancé indépendamment mais pour que l'ensemble du chat fonctionne il faut tous les lancer

Voici un détail de chaque microservice et de comment les lancer :

- Retour: Socket, Client jersey, Kafka consumer -> se lance via le main dans la classe App.java
- Aller: Serveur/Client jersey -> se lance via tomcat
- Authentication: Serveur jersey -> se lance via tomcat
- Producer: Serveur/Client jersey, Kafka producer -> se lance via tomcat

Architecture

Architecture logiciel



Zoom sur l'architecture

Un peu d'explication sur le schéma du dessus ne fera pas de mal. Il explique le scénario classique et les différentes communications entre les microservices. Déjà nous voyons que les clients ne communiquent qu'avec **aller** et **retour**.

Aller est une passerelle qui permet au client de regrouper ses requêtes sur un seul micro service et celui-ci transmet la requête au micro service dédié. Il n'y a donc aucun code métier à l'intérieur de ce microservice.

Concernant **retour**, il permet au client de recevoir les différentes informations concernant le user courant, ses channels ainsi que ses messages dédiés via la communication avec **authentication** et **producer** et la consommation du serveur kafka.

Le microservice **authentication** va gérer tout le métier en ce qui concerne l'authentification (connexion/inscription) ainsi que la génération de token. Il va être appelé par **aller** lors de l'inscription ou de la connexion d'un user pour la récupération du token, et par **retour** et **producer** qui permet de vérifier l'authenticité d'un token.

Producer est un microservice qui gère toute la partie channels et messages. Il est juste appelé par **aller** lors de la création d'un channel ou d'un envoi de message. Celui-ci va créer des topics dans le serveur kafka représentant les channels publiques ou privées, ainsi que produire des messages dans ces différents topics.

Architecture de code

Concernant l'architecture de code, pour respecter une homogénéité, un schéma est établi et se répète dans les 4 microservices.

Nous retrouvons un package **entities**, présent dans chaque microservices, qui permet de définir les différents objets qui transitent via jersey. À l'intérieur nous pouvons trouver un package **requests** qui définit les body de certaines requêtes.

Dans les microservices clientes jersey, nous retrouvons un package **config** qui permet de définir les différentes routes sur lesquelles effectuer des requêtes, ainsi qu'un package **clients** qui créer la connexion aux serveurs jersey via une webresource et de faire les différents appels.

Pour les serveurs jersey. Un package **services** va regrouper l'ensemble du code métier qui sera appelé par les contrôleurs qui eux définissent l'ensemble des routes. On peut trouver les contrôleurs dans les packages **resources**. Les serveurs jersey ont également un package **exceptions** qui va regrouper l'ensemble des exceptions renvoyer par le serveur.

Concernant la base de données ont retrouve un package database dans **authentication**, qui permet de stocker les informations utilisateurs ainsi que les tokens, et dans **producer** qui stock les différents channels privés ou publiques.

Scénario

Inscription / connexion

Pour créer un utilisateur ou se connecter et récupérer le token cela se passe sur postman. Il faut appeler le endpoint "/signin ou "/signup" avec un mot de passe et un nom d'utilisateur.

connexion putty

À l'aide du token récupéré précédemment, lors de la connexion sur putty, il est nécessaire de rentrer le token, à partir de la le nom d'utilisateur associé et la liste des channels sont affichés.

channel privé

Pour envoyer un message privé, il faut simplement appeler le endpoint "/message/private" en passant le username du receveur, notre token et le message à envoyer. À partir de là, un topic est créé ou non si il existe déjà un channel privé comprenant le couple sender/receiver. et le message est envoyé.

channel publique

Un channel pulique doit être au préalable créé par un utilisateur à l'aide de l'endpoint "/create/publicChannel", ensuite n'importe qui peut rejoindre le channel à l'aide du endpoint "/join/publicChannel". Les personnes du channel peuvent envoyer un message dans celui ci à l'aide du endpoint "/message"

La gestion des erreurs est bien prise en compte pour les différents scénarios, comme par exemple envoyer un message dans un channel ou on y est pas, rejoindre un channel dans lequel on est déjà, créer 2 channel avec le même nom, créer 2 utilisateurs avec le même nom.

Problème rencontré

Je n'ai pas rencontré de problème particulier. Le plus dur était d'imaginer la conception du système. Une fois la conception réalisée, il a été simple de mettre en œuvre les différents microservices avec les différentes technologies à l'aide des anciens TPs et des connaissances acquises.