Architecture Logicielle: Construction

Equipe B: Guillaume Piccina, William d'Andrea, Nicolas Fernandez, Yann Brault

Sujet V6 : Pay as you pollute

Ce document traite de l'architecture et de son évolution chaque semaine.

Semaine 43: 25_10

Résumé du point avec le prof:

- clarification et aide sur comment envoyer la config aux utilisateurs
  -> Impossible de contacter un utilisateur en particulier dans RabbitMQ

  Solution envisagée: Délocaliser cette logique métier :

  - Création d'un service avant le bus.
  - Son fonctionnement:
    l'utilisateur lui envoie une requête avec juste la position gps et le service lui renvoie une fréquence de publication de message ainsi que le prix de la zone actuelle. Ainsi l'utilisateur sait avec quel intervalle de temps il doit envoyer sa position et il peut calculer la facture estimée.

Les modifications de l'architecture:

- Ajout d'un service TrackingPolicy avant le bus qui expose une route post pour le user et va pull sur le service externe de pollution les zones et les stocks dans une solution de cache
- Le user fait des requêtes sur le TrackingPolicy avant d'envoyer ses données dans le bus.

Les services en détails:

- **Car tracker :**
    - Consomme dans le bus un topic de tracking
    - Transmettre à grande vitesse les infos de tracking vers la BD (heure de pointes, bouchons)
    - Doit attendre le retour de zone pollution avant d'envoyer les data dans la base (y a un lock ici)
- **tracking shutdown :**
    - Consomme dans le bus un topic de shutdown
    - Récupère en db *Tracking_infos* les data du user
    - Déclenche le billing pour un user en envoyant les data
    - flush la db de ces data
- **Position checker**
    - Consomme dans le bus un topic de redémarrage
    - Communique avec le système externe de caméra pour surveillance
    - contact billing en cas de problème
- **Zones pollution :**
    - Doit tirer à intervalle régulier les infos de pollution et les mettre en cache (solution de stockage clé valeur avec très gros débit de lecture)
    - Doit trouver une correspondance entre la position gps et la zone pour renvoyer la zone à car tracker
- **Billing :**
    - Reçoit les infos de tracking depuis tracking shutdown
    - Calcul la facture finale
    - Fait l'interface avec la banque pour le paiement.

- **TrackingPolicy** :
    - expose une route pour les users
    - Pull les infos de zones de pollutions (stock dans une petite bd / cache qui peut supporter une très grosse charge en lecture)
    - Doit définir des sous zones géographiques pour chaque zone de pollution
    - Doit faire correspondre une position gps avec des zones
    - Retourne aux utilisateurs une politique de tracking (fréquent si proche d'une frontière de zones)
    - Comportement à définir : Renvoie aux utilisateurs le prix de la zone dans laquelle ils sont pour le calcul du prix estimé en real time

Scénario du flot de données:

- L'utilisateur fait une requête vers le service tracking policy, en lui envoyant simplement sa position gps, le service lui retourne une fréquence d'envoie de données dans le bus ainsi que le prix de la zone dans laquelle l'utilisateur se trouve afin de pouvoir calculer le coût estimé du trajet.
- L'utilisateur envoie son id, sa position gps et un timestamp dans le bus.
- Le service de tracking consomme les infos dispo dans le topic de tracking du bus et envoie la position gps au service de pollution qui lui renvoie une zone de pollution, puis le tracker envoie les 4 informations au stockage.
- Quand l'utilisateur finit son trajet, il envoie les mêmes informations que d'habitude sauf que cette fois-ci le service tracking shutdown sera le consommateur.
- Le service tracking shutdown, en consommant les infos de son topic, reçoit l'info que l'utilisateur s'est arrêté, alors il va chercher en DB les infos de tracking de l'utilisateur et les envoie vers le billing
- Le service de billing avec les infos qu'il a reçu va calculer la facture puis il va envoyer la facture au service de banque
- L'utilisateur recevra sa facture directement de sa banque

Choix de techno :

- Bus : RabbitMQ pour le moment (évolution possible en AL Evolution)
- BD principale : NoSQL
- BD zone pollution, tracking policy: SQL qui supporte une grosse charge de lecture
- Archi micro service : NestJS

Enfin voilà un schéma de l’architecture: