27-05-2022

Phetsinorath Pierre ET Moindjié Mohamed

Rapport du module Introduction à l'événementiel

Context du projet

Dans ce projet en binôme, il était question de créer plusieurs services qui communiquent entre eux via des canaux bien particuliers. En partant d'une base existante, le projet est donc réalisé sous 4 jours avec une répartition de tâches qui permettaient de diminuer la complexité dans le but de répondre aux besoins fixés.

Réalisation du projet

On a utilisé le framework **Spring** avec les configurations de **boot** pour donner toutes les dépendances nécessaires. Ce framework fournit une injection de dépendance et une inversion de contrôle qui facilitent aux développeurs de se focaliser sur la logique. Quant à l'IDE, on a choisi IntelliJ qui offre des fonctionnalités et une appréhension de java. La réalisation du projet se focalise dans trois axes: la création d'une table via **les entités** et ses méthodes pour appeler des api, la création d'un "job master" en précisant ses tâches et celui d'un "worker".

Création d'une entité Lettre

Dans cette, on a défini une classe **Lettre** avec des méthodes getter, setter. Afin de préciser que la classe est une entité, on utilise l'annotation **@Entity**, par la suite on précise notre clé primaire de la table via **@Id**.
On a implémenté une classe **LettreController** qui appelle l'interface **JpaRepository** pour décrire les méthodes API grâce aux annotations **@RestController** et **@RequestMapping**

Implémentation de la méthode chunking

Pour faire communiquer deux serveurs Spring boot, nous avons implémenté la méthode chunking.
Dans cette méthode, nous répartissons le traitements des données en fonction du modèle suivant:

- les serveurs job-master: Il se charge de la lecture des données au sein du fichier CSV. Puis, il communique les données à l'aide d'un **chunkRequest** sur le canal Kafka qui lui est attribué.

- les serveurs worker: Il se charge du traitement et de l'écriture en base de données. Les données sont lues sur le canal Kafka, à la fin de son exécution le worker envoie un message sur son canal, indiquant la terminaison de son job.

Pour la lecture des données, nous avons implémenté la méthode **ItemReader**, à l'aide de **FlatFileItemReader** pour la lecture du fichier dans la classe job-master.

Pour l'écriture des données, nous avons implémenté la méthode ItemWriter au sein de la classe worker. Nous avons utilisé **RestTemplate** pour faire des appels API au **LettreController** située dans le domaine.

Les méthodes **itemReader** et **ItemWriter** sont ensuite injectées dans leur classe respective à l'aide des annotations **@EnableBinding** et **@Autowired**, qui se charge de l'injection de dépendance.

Pour permettre la communication via Kafka, nous avons instancié deux channel , une **DirectChannel** pour l'envoie de RequestChunk et une QueueChannel pour la réception des chunk. Puis nous les avons configurés à l'intérieur des méthodes **outBoundFlow** et **inboundFlow.** Il faut être vigilant à ce que les job-master et les workers lisent et répondent sur des canaux distincts, afin d'éviter des collisions lors de la réception ou l'envoie des chunk.

Conclusion

Durant ce projet, malgré quelques difficultés, comme lors de l'enregistrement des données avec l'appel API. Nous avons pu finir l'implémentation de la communication entre deux serveurs Spring à l'aide de la méthode chunking.

Une évolution de ce projet pourrait être l'hébergement sur des serveurs distants, terminer l'intégration de la communication entre plusieurs job-master et plusieurs worker ou encore l'implémentation de la deuxième méthode via les partitions des tâches aux différents worker