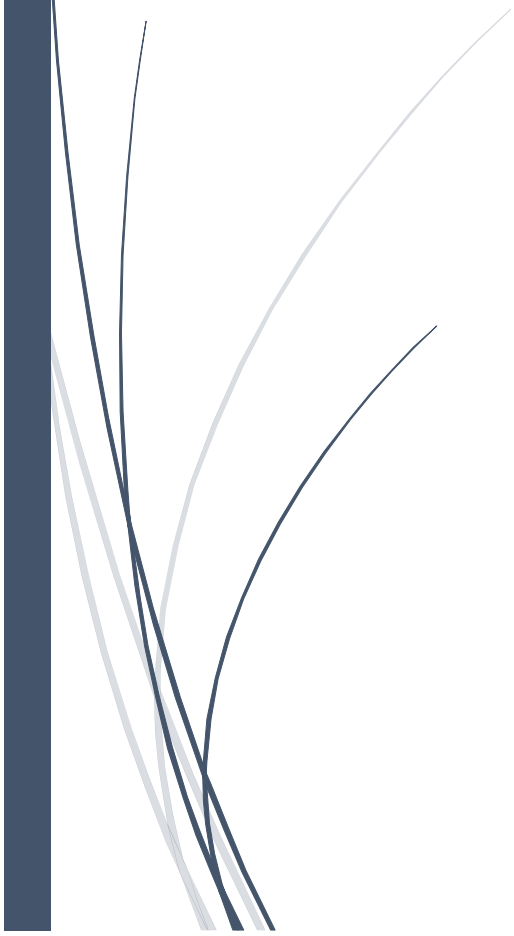
Introduction à Spring Cloud Stream Function

TP : Développement d'une application avec Spring Cloud Stream Function et Kafka



Architecture des composants d'entreprise

Table des matières

I. Objectif du TP

- ✓ Expliquer le modèle de programmation fourni par ***Spring Cloud Streams Function***.
- ✓ Consommer facilement des données d'un Topic moyennant la fonction ***Consumer*** (**input -> void**).
- ✓ Publier les données dans un Topic moyennant la fonction ***Supplier*** (**void -> output**).
- ✓ Consommer les données d'un Topic, les transformer et les publier dans un autre Topic moyennant la fonction ***Function*** (**input >output**).
- ✓ Consommer les données d'un Topic, faire des opérations avancées (filter, map, .) avec l'utilisation de l'api KAFKA STREAMS et publier ensuite ces données dans un autre Topic moyennant la fonction ***Function*** (**input >output**).

II. Objet du TP

Nous allons voir à travers cet exemple comment développer des fonctions de type Consumer, Supplier et Function.

Le modèle de données que nous allons utiliser est de type Notification. Nous simulons les données envoyées par des radars en cas de dépassement de vitesse autorisée. Chaque notification est composée des éléments suivants :

- ✓ ***code :*** le code de la notification ;
- ✓ ***date :*** la date de la notification ;
- ✓ ***authorizedSpeed :*** la vitesse autorisée ;
- ✓ ***currentSpeed :*** la vitesse enregistrée ;
- ✓ ***registrationNumber :*** le matricule de l'engin.

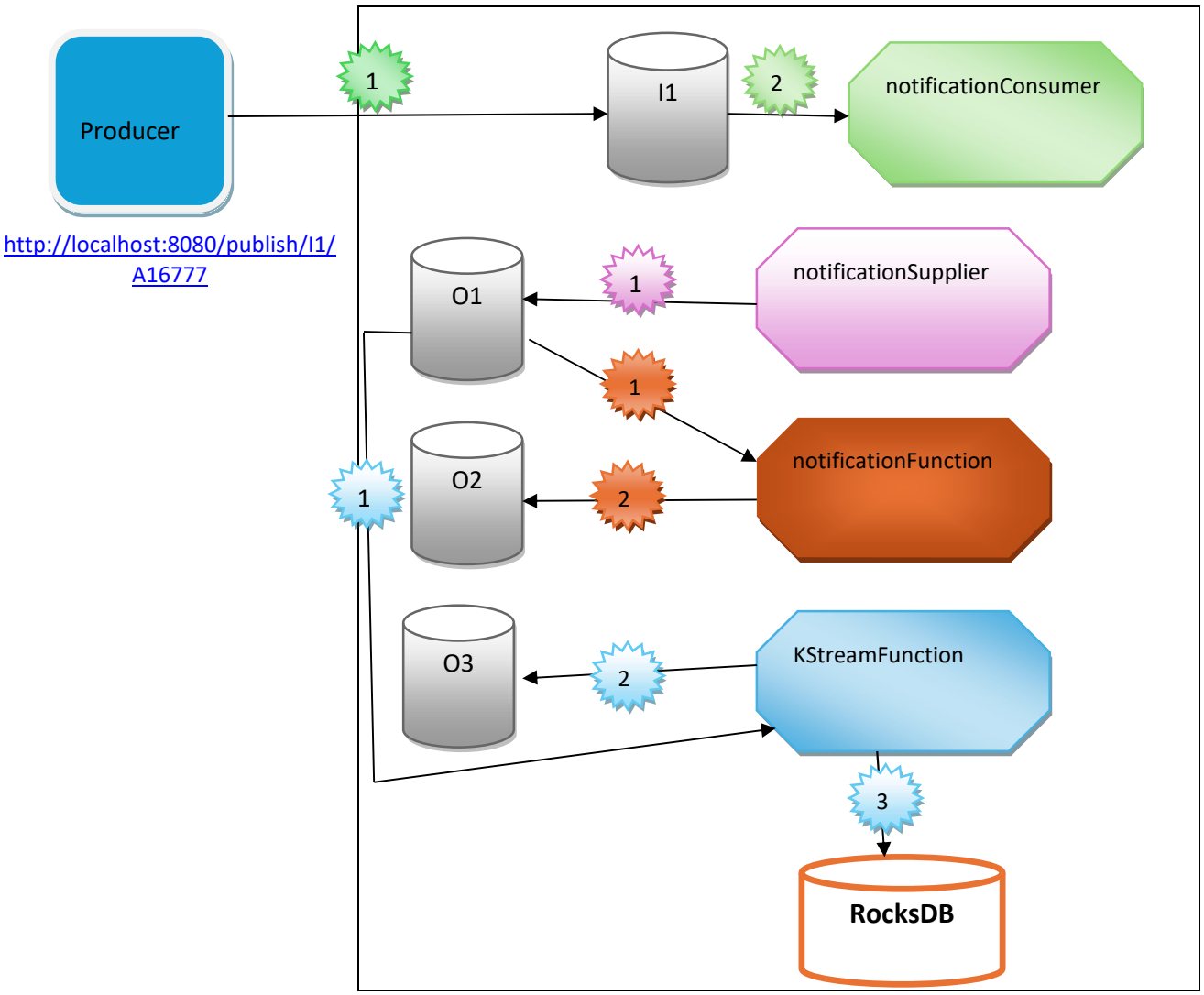
La fonction ***notificationConsumer*** **:** permet de consommer des messages, de type Notification, publiés dans le topic I1 en affichant l'objet consommé en format JSON au niveau de la console : ***System.out***.

La fonction ***notificationSupplier :*** permet de publier des messages aléatoires de type Notification au niveau du topic O1.

La fonction ***notificationFunction :*** permet de consommer les messages se trouvant dans le topic O1, transformer chaque objet de type Notification à une MAP et publier le résultat dans le topic O2.

La fonction ***kStreamFunction :*** permet de consommer les messages se trouvant dans le topic O1, filtrer uniquement les notifications concernant le matricule « A-2 5643 », grouper les données par la clé du message, calculer le nombre de notification et ensuite enregistrer au niveau du store du Kafka le résultat.

Voir le schéma suivant pour plus de précisions :



KAFKA

III. Prérequis

- IntelliJ IDEA ;
- JDK version 17 ;
- Une connexion Internet pour permettre à Maven de télécharger les librairies.

NB : *Ce TP a été réalisé avec IntelliJ IDEA 2023.2.3 (Ultimate Edition).*

IV. Spring Cloud Stream Function

a. C'est quoi Spring Cloud Stream Function ?

- ***Spring Cloud Stream Function*** est un module de Spring Cloud qui simplifie le développement d'applications basées sur des événements, en utilisant les fonctions Java comme Consumer, Supplier et Function.
- Il est basé qur le Framework ***Spring Cloud Function*** qui est une couche de programmation générale centrée sur les fonctions.

b. Les fonctionnalités de Spring Cloud Stream Function

1. Prise en charge de la programmation fonctionnelle

- Vous pouvez définir des fonctions Java simples (Supplier, Function, et Consumer) pour gérer la production, la transformation et la consommation des événements.
- Ces fonctions peuvent être automatiquement connectées à des flux de messages, simplifiant le développement d'applications réactives et pilotées par des événements.

2. Abstraction du middleware

- Spring Cloud Stream Function abstrait les détails spécifiques des systèmes de messagerie sous-jacents (comme ActiveMQ, Kafka, RabbitMQ, etc.).
- Les fonctions peuvent être facilement réutilisées avec différents systèmes en changeant simplement la Configuration (le binder).

3. Routage dynamique des fonctions

- Spring Cloud Stream peut router dynamiquement les messages vers différentes fonctions en fonction des besoins de l'application.
- Les fonctions peuvent être sélectionnées via des configurations basées sur des propriétés (par exemple, spring.cloud.function.definition) ou dynamiquement au moment de l'exécution.

4. Configuration simplifiée

- L'intégration avec des systèmes de messagerie (comme Apache Kafka ou RabbitMQ) est simple grâce à des propriétés de configuration Spring Boot (application.yml ou application.properties).
- Il est possible de configurer les bindings entre les canaux de messages et les fonctions via des noms clairs et explicites.

5. Facilité d'intégration avec Spring Cloud Stream

- Spring Cloud Stream Function s'intègre de manière transparente avec Spring Cloud Stream pour prendre en charge les canaux, les bindings et les converters.
- On peut utiliser les annotations comme ***@EnableBinding*** ou profiter de ***la configuration automatique avec Spring Boot.***

6. Traitement des messages réactifs

- Spring Cloud Stream Function prend en charge la programmation réactive avec Reactor avec l'utilisation des classes Flux et Mono.
- Cela permet de construire des pipelines de traitement réactifs pour une meilleure gestion des flux asynchrones.

7. Support des fonctions multiples

- Il est possible de déclarer plusieurs fonctions dans une même application, chacune pouvant être configurée indépendamment.
- Cela facilite la gestion d'applications complexes qui doivent traiter différents types de données ou messages.

8. Extensibilité et flexibilité

- On peut étendre et personnaliser la logique via des mécanismes comme des intercepteurs, des convertisseurs personnalisés, et des serdes (sérialisation/désérialisation).
- Les fonctions peuvent également être composées pour créer des pipelines de traitement plus complexes.

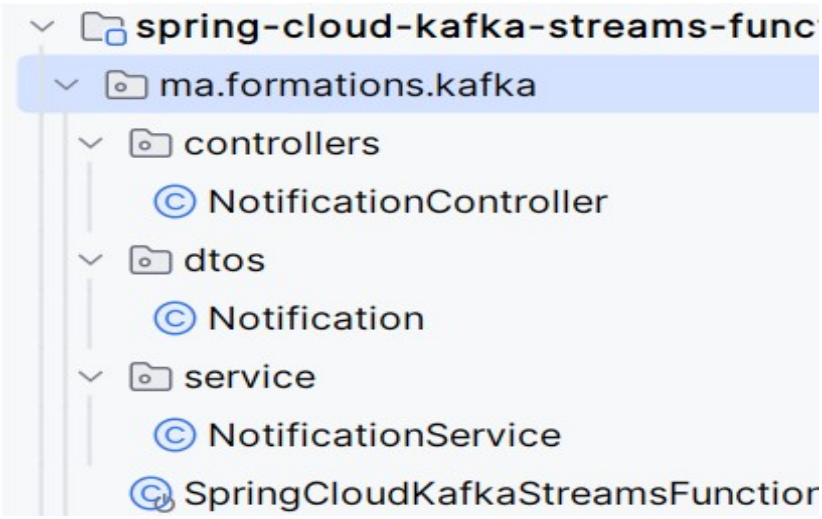
9. Interopérabilité avec Spring Cloud Function

- Spring Cloud Stream Function est basé sur Spring Cloud Function, ce qui permet une interopérabilité avec d'autres plateformes et frameworks prenant en charge Spring Cloud Function.
- Cela inclut des environnements serverless comme par exemple :
    - ✓ **AWS Lambda (Amazon Web Services) :**
        - Permet d'exécuter du code en réponse à des événements comme des requêtes HTTP (via API Gateway), des fichiers ajoutés dans S3, etc.
    - ✓ **Google Cloud Functions :**
        - Une plateforme similaire à AWS Lambda pour exécuter des fonctions en réponse à des événements dans Google Cloud Platform.
    - ✓ **Azure Functions :**
        - Service proposé par Microsoft pour exécuter des fonctions déclenchées par divers événements (HTTP, files d'attente, etc.).
    - ✓ **Cloudflare Workers :**
        - Permet de déployer du code JavaScript sur des serveurs de périphérie (edge) pour des réponses ultra-rapides.
    - ✓ **OpenFaaS, Knative (open source) :**
        - Plateformes serverless open source pour des environnements Kubernetes.

V. Développement d'une application avec Spring Cloud Stream Function et Kafka

a. Arborescence du projet

- Créer un projet Spring Boot (par exemple spring-cloud-kafka-streams-function).
- Créer l'arborescence suivante :



b. Le fichier pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>3.4.1</version>
    <relativePath/>
  </parent>
  <groupId>ma.formations.kafka</groupId>
  <artifactId>spring-cloud-kafka-streams-function</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>spring-cloud-kafka-streams-function</name>
  <description>spring-cloud-kafka-streams-function</description>

  <properties>
    <java.version>17</java.version>
    <spring-cloud.version>2024.0.0</spring-cloud.version>
  </properties>
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
      <groupId>org.apache.kafka</groupId>
      <artifactId>kafka-streams</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-stream</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-stream-binder-kafka</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-stream-binder-kafka-streams</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.kafka</groupId>
      <artifactId>spring-kafka</artifactId>
    </dependency>

    <dependency>
      <groupId>org.projectlombok</groupId>
      <artifactId>lombok</artifactId>
      <optional>true</optional>
    </dependency>
  </dependencies>
  <dependencyManagement>
    <dependencies>
      <dependency>
```

```
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-dependencies</artifactId>
            <version>${spring-cloud.version}</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>

<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
            <configuration>
                <excludes>
                    <exclude>
                        <groupId>org.projectlombok</groupId>
                        <artifactId>lombok</artifactId>
                    </exclude>
                </excludes>
            </configuration>
        </plugin>
    </plugins>
</build>

</project>
```

c. Le fichier application.properties

```
spring.application.name=spring-cloud-kafka-streams-function
spring.cloud.function.definition=notificationConsumer;notificationSupplier;notificationFunction;kStreamFunction
spring.cloud.stream.bindings.notificationConsumer-in-0.destination=I1
spring.cloud.stream.bindings.notificationSupplier-out-0.destination=O1
spring.cloud.stream.bindings.notificationFunction-in-0.destination=O1
spring.cloud.stream.bindings.notificationFunction-out-0.destination=O2
spring.cloud.stream.bindings.kStreamFunction-in-0.destination=O1
spring.cloud.stream.bindings.kStreamFunction-out-0.destination=O3
spring.cloud.stream.poller.fixed-delay=1000
```

- Par défaut, Spring Cloud Stream Function configure pour une fonction nommée NOM-DE-LA-FONCTION :
  - Le topic « NOM-DE-LA-FONCTION-in-0 » comme un topic d'entrée.
  - Le tipic « NOM-DE-LA-FONCTION-out-0 » comme un topic de sortie.

- Pour personnaliser le nom du topic en entrée, il faut rajouter :

```
spring.cloud.stream.bindings.notificationConsumer-in-0.destination= « vous mettez ici le nom de votre topic »
```

- Pour personnaliser le nom du topic en sortie, il faut rajouter :

```
spring.cloud.stream.bindings.notificationConsumer-out-0.destination= « vous mettez ici le nom de votre topic »
```

- La propriété ***spring.cloud.function.definition*** permet de définir les fonctions qui seront utilisées.

- La propriété ***spring.cloud.stream.poller.fixed-delay*** est utilisée dans Spring Cloud Stream pour configurer le délai fixe entre deux exécutions successives d'un poller dans un scénario où une application consomme des données à partir d'une source qui n'est pas basée sur des événements, comme une base de données ou un fichier.

d. La classe Notification (le modèle)

```
package ma.formations.kafka.dtos;

import lombok.*;

import java.util.Date;

@Getter @Setter @NoArgsConstructor @AllArgsConstructor @Builder @ToString
public class Notification {
   private String code;
   private Date date;
   private Double authorizedSpeed;
   private Double currentSpeed;
   private String registrationNumber;
}
```

e. La classe NotificationService

```
package ma.formations.kafka.service;

import ma.formations.kafka.dtos.Notification;
import org.apache.kafka.common.serialization.Serdes;
import org.apache.kafka.streams.KeyValue;
import org.apache.kafka.streams.kstream.Grouped;
import org.apache.kafka.streams.kstream.KStream;
import org.apache.kafka.streams.kstream.Materialized;
import org.apache.kafka.streams.kstream.TimeWindows;
import org.springframework.context.annotation.Bean;
import org.springframework.stereotype.Service;

import java.text.SimpleDateFormat;
import java.time.Duration;
import java.util.*;
import java.util.function.Consumer;
import java.util.function.Function;
import java.util.function.Supplier;

@Service
public class NotificationService {

  @Bean
   public Consumer<Notification> notificationConsumer() {
      return (notification -> {
         System.out.println("**********************");
```

```
        System.out.println(notification);
        System.out.println("**********************");
      });
    }

    @Bean
    public Supplier<Notification> notificationSupplier(){

      List<String> matricules = List.of("A-2 5643", "A-6 9876", "A-45 6549", "A-3 785","A-2 5643");
      return ()-> Notification.builder().
            registrationNumber(matricules.get(new Random().nextInt(matricules.size()))).
            date(new Date()).
            code(UUID.randomUUID().toString()).
            authorizedSpeed(Math.random()>60?60:Math.random()).
            currentSpeed(Math.random()>70?Math.random():100).
            build();

    }

    @Bean
    public Function<Notification, Map<String,String>> notificationFunction(){
      return input-> {
        HashMap<String,String> map=new HashMap<>();
        map.put(input.getRegistrationNumber(),"CS="+input.getCurrentSpeed()+", AS="+input.getAuthorizedSpeed() + " at
"+
                    new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").format(input.getDate())
                );
        return map;
      };
    }

    @Bean
    public Function<KStream<String, Notification>, KStream<String, Long>> kStreamFunction() {
      return (input) -> {
        return input
              .filter((k, v) -> v.getRegistrationNumber().equals("A-2 5643"))
              .map((k, v) -> new KeyValue<>(v.getRegistrationNumber(), 0L))
              .groupBy((k, v) -> k, Grouped.with(Serdes.String(), Serdes.Long()))
              .windowedBy(TimeWindows.of(Duration.ofSeconds(5)))
              .count(Materialized.as("counts"))
              .toStream()
              .map((k, v) -> new KeyValue<>("=>" + k.window().startTime() + k.window().endTime() + ":" + k.key(), v));
      };
    }
}
```

- La fonction ***kStreamFunction ci-dessus effectue les opérations suivantes :***
    1. Filtrer uniquement les notifications concernant l'engin dont le matricule est A-2 5643
    2. Transforme chaque objet de notification à un objet de type KeyValue dont la clé est le numérode matricule et la valeur est 0.

3. Groupe les données par le numéro de matricule.
4. Les messages sont divisés en fenêtres temporelles de 5 secondes. Cela signifie que le traitement des messages est limité à des blocs de 5 secondes.
5. Calculer le nombre de messages dans chaque fenêtre (par clé). Les résultats sont matérialisés dans un magasin d'état Kafka Streams nommé "counts".
6. Les résultats de comptage sont convertis en un KStream.

f.La classe NotificationController

```
package ma.formations.kafka.controllers;

import ma.formations.kafka.dtos.Notification;
import org.apache.kafka.streams.KeyValue;
import org.apache.kafka.streams.kstream.Windowed;
import org.apache.kafka.streams.state.KeyValueIterator;
import org.apache.kafka.streams.state.QueryableStoreTypes;
import org.apache.kafka.streams.state.ReadOnlyWindowStore;
import org.springframework.cloud.stream.binder.kafka.streams.InteractiveQueryService;
import org.springframework.cloud.stream.function.StreamBridge;
import org.springframework.http.MediaType;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RestController;
import reactor.core.publisher.Flux;

import java.time.Duration;
import java.time.Instant;
import java.util.Date;
import java.util.HashMap;
import java.util.Map;
import java.util.UUID;

@RestController
public class NotificationController {
  private StreamBridge streamBridge;
  private InteractiveQueryService interactiveQueryService;

  public NotificationController(StreamBridge streamBridge, InteractiveQueryService interactiveQueryService) {
    this.streamBridge = streamBridge;
    this.interactiveQueryService = interactiveQueryService;
  }
  @GetMapping("/publish/{topic}/{registrationNumber}")
  public Notification publish(@PathVariable String topic,@PathVariable String registrationNumber) {
    Notification notification=Notification.builder().
        registrationNumber(registrationNumber).
        date(new Date()).
        code(UUID.randomUUID().toString()).
        authorizedSpeed(60d).
        currentSpeed(Math.random()<=60?70:Math.random()).
        build();
    streamBridge.send(topic,notification);
    return notification;
```

```
    }

    @GetMapping(path = "/analytics",produces = MediaType.TEXT_EVENT_STREAM_VALUE)
    public Flux<Map<String, Long>> analytics(){
        return Flux.interval(Duration.ofSeconds(1))
                .map(sequence->{
                    Map<String,Long> stringLongMap=new HashMap<>();
                    ReadOnlyWindowStore<String, Long> windowStore = interactiveQueryService.getQueryableStore("counts",
QueryableStoreTypes.windowStore());
                    Instant now=Instant.now();
                    Instant from=now.minusMillis(5000);
                    KeyValueIterator<Windowed<String>, Long> fetchAll = windowStore.fetchAll(from, now);
                    while (fetchAll.hasNext()){
                        KeyValue<Windowed<String>, Long> next = fetchAll.next();
                        stringLongMap.put(next.key.key(),next.value);
                    }
                    return stringLongMap;
                }).share();
    }
}
```

```
private StreamBridge st
```

Le rôle de StreamBridge est d'agir comme un pont (bridge) entre l'application et les destinations Kafka, RabbitMQ, ou d'autres systèmes de messagerie configurés, sans avoir à définir des **bindings** statiques dans le fichier de configuration.

```
private InteractiveQueryService intera
```

- Cette classe permet d'interagir avec les données de l'application en temps réel, sans interroger directement le cluster Kafka. Elle facilite la récupération des données locales ou des informations sur d'autres partitions Kafka Streams distribuées dans un cluster.

g. Tests

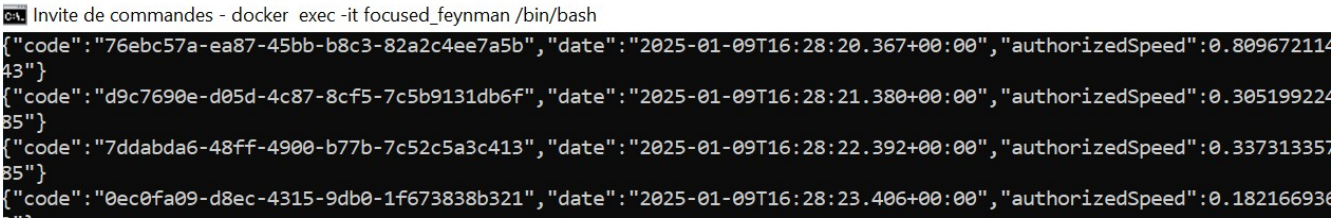
- Démarrer Kafka (voir TP 7).
- Créer les Topic : I1, O1, O2 et O3 (voir TP 7).
- Lancer la méthode main.
- Exécuter lien : localhost:8080/publish/I1/A16777 et vérifier que la fonction notificationConsumer a été bien exécutée :

```
**********************
Notification(code=fcb8027c-f059-4de3-8007-29aaf34b2488, date=Thu Jan 09 17:25:15 WEST 2025, au
**********************
```

- Dans un terminal Docker en mode interactif, lancer la commande suivante :

  sh kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic O1

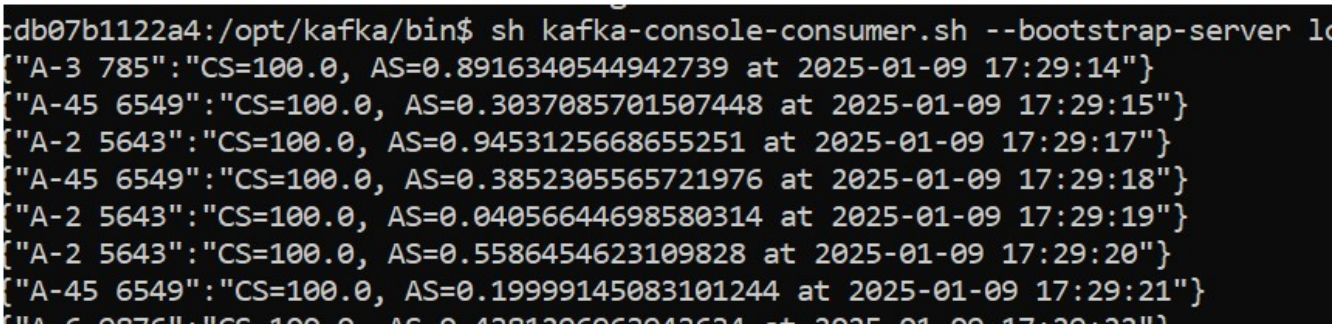
  et vérifier que la fonction notificationSupplier a été bien exécutée :



- Exécuter la commande suivante :

  sh kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic O2

  et vérifier que la foonction notificationFunction a été bien exécutée :



- Exécuter le lien suivant : localhost:8080/analytics et vérifier que la foonction kStreamFunction a été bien exécutée :

- Lancer la commande suivante pour consommer les données du Topic O3 :

```
sh kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic O3 --property print.key=true --property print.value=true --property key.deserializer=org.apache.kafka.common.serialization.StringDeserializer --property value.deserializer=org.apache.kafka.common.serialization.LongDeserializer
```