TP: Kafka Streams

Exercice1:

Collecte et Traitement des Données Météorologiques en Temps Réel avec Kafka Streams

Contexte

L'objectif de cet exercice est de développer une application Java utilisant Kafka Streams pour traiter en temps réel des données météorologiques issues d'un topic Kafka. L'application filtre les données, les transforme et calcule des statistiques agrégées par station météorologique.

Architecture

1. **Source des Données** : Les données sont consommées depuis un topic Kafka nommé `weather-data`.
    - Format attendu : `station_id,temperature_celsius,humidity`.
2. **Pipeline de Traitement** :
    - **Filtrage** : Les relevés où la température dépasse 30°C sont sélectionnés.
    - **Transformation** : La température est convertie en Fahrenheit, et les données sont réorganisées.
    - **Agrégation** : Calcul des moyennes de température et d'humidité par station.
3. **Résultat** : Les données agrégées sont publiées dans un topic Kafka nommé `station-averages`.

Code Source:

```java
KTable<String, String> aggregatedTable = groupedStream.aggregate(
        () -> "0.0,0.0,0",
        (station, newValue, aggregate) -> {
            try {
                String[] newValueParts = newValue.split( regex: ",");
                String[] aggregateParts = aggregate.split( regex: ",");

                if (newValueParts.length < 3 || aggregateParts.length < 3) {
                    throw new IllegalArgumentException("Invalid data format");
                }

                double newTemp = Double.parseDouble(newValueParts[1]);
                double newHumidity = Double.parseDouble(newValueParts[2]);
                double totalTemp = Double.parseDouble(aggregateParts[0]) + newTemp;
                double totalHumidity = Double.parseDouble(aggregateParts[1]) + newHumidity;
                int count = Integer.parseInt(aggregateParts[2]) + 1;

                return totalTemp + "," + totalHumidity + "," + count;
            } catch (Exception e) {
                System.err.println("Error aggregating record: " + newValue + " - " + e.getMessage());
                return aggregate; // Retourner l'agrégat inchangé en cas d'erreur
```

```java
KTable<String, String> aggregatedTable = groupedStream.aggregate(
        () -> "0.0,0.0,0",
        (station, newValue, aggregate) -> {
            try {
                String[] newValueParts = newValue.split( regex: ",");
                String[] aggregateParts = aggregate.split( regex: ",");

                if (newValueParts.length < 3 || aggregateParts.length < 3) {
                    throw new IllegalArgumentException("Invalid data format");
                }
```

```java
KGroupedStream<String, String> groupedStream = convertedStream.groupBy(
        (key, value) -> value.split( regex: ",")[0],
        Grouped.with(Serdes.String(),Serdes.String())
);
```

`Calcul des moyennes et conversion pour le topic de sortie` :

```java
KStream<String, String> resultStream = aggregatedTable.toStream().mapValues(aggregate -> {
    String[] parts = aggregate.split( regex: ",");
    double totalTemp = Double.parseDouble(parts[0]);
    double totalHumidity = Double.parseDouble(parts[1]);
    int count = Integer.parseInt(parts[2]);

    double avgTemp = totalTemp / count;
    double avgHumidity = totalHumidity / count;

    return "Température Moyenne = " + avgTemp + "F, Humidité Moyenne = " + avgHumidity + "%";
});
// Écriture des résultats dans le topic "station-averages"
resultStream.to( s: "station-averages",Produced.with(Serdes.String(), Serdes.String()));;

// Création et démarrage de l'application Kafka Streams
KafkaStreams streams = new KafkaStreams(builder.build(), props);
streams.start();

// Ajout d'un hook pour arrêter proprement l'application
Runtime.getRuntime().addShutdownHook(new Thread(streams::close));
```

Tests et Validation

```
kafka-console-producer --topic weather-data --bootstrap-serve
kafka-console-consumer --topic station-averages --bootstrap-s
```

- **Données d'entrée simulées** :

```
>^C[appuser@85243fb291cb ~]$ kafka-console-producer --topic weather-data --bootstrap-server localhost:9092
>Station1,25.3,60
>Station2,30.5,50
>Station1,35.0,70
>Station1,36.0,80
>Station1,25.3,60
>Station2,30.5,70
>
```

- Sortie:

```
[appuser@85243fb291cb ~]$ kafka-console-consumer --topic station-averages --bootstrap-server localhost:9092 --from-begin
ning
Température Moyenne = 90.275F, Humidité Moyenne = 55.0%
Température Moyenne = 96.08000000000001F, Humidité Moyenne = 76.0%
Température Moyenne = 89.6F, Humidité Moyenne = 58.0%
```

Conclusion

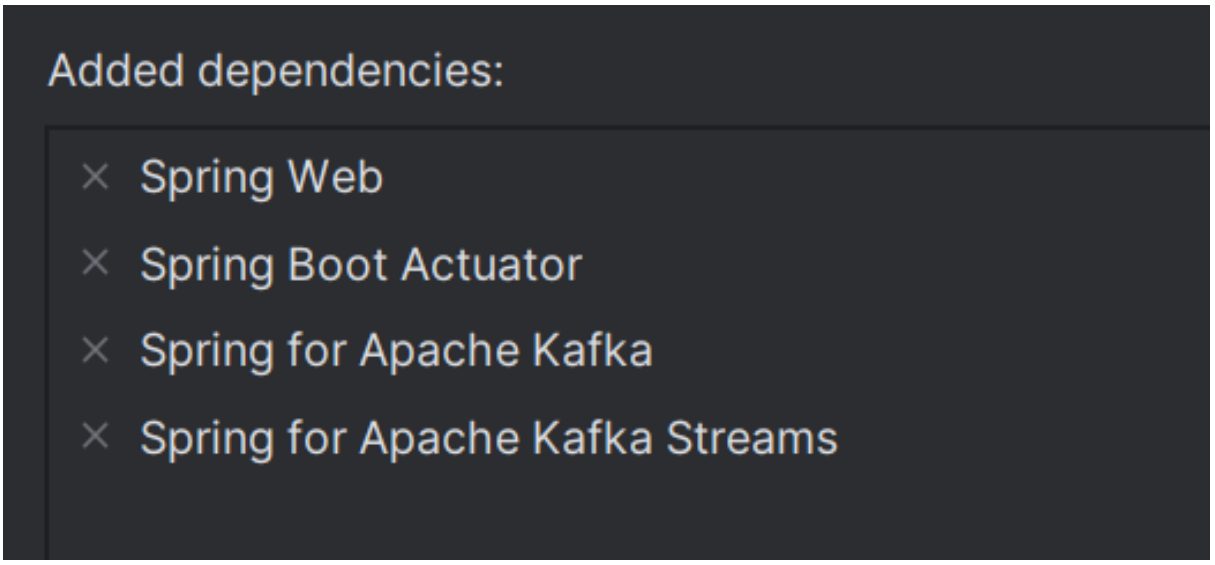
Cet exercice a permis de construire un pipeline Kafka Streams robuste pour traiter et analyser les données météorologiques en temps réel. Les validations et la gestion des erreurs garantissent la stabilité du traitement.

Exercice 2:

- **Création des topics**:

```
sh-4.4$ kafka-topics --bootstrap-server localhost:9092 --create --topic clicks --partitions 1 --replication-factor 1
Created topic clicks.
sh-4.4$ kafka-topics --bootstrap-server localhost:9092 --create --topic click-counts --partitions 1 --replication-factor 1
Created topic click-counts.
```

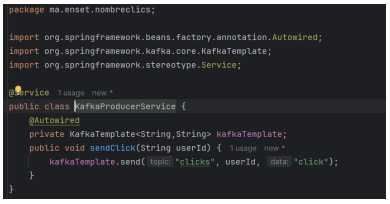
- **Dépendances :**



Configuration de l'application :

```
spring.application.name=NombreClics
spring.kafka.bootstrap-servers=localhost:9092
spring.kafka.producer.key-serializer=org.apache.kafka.common.serialization.StringSerializer
spring.kafka.producer.value-serializer=org.apache.kafka.common.serialization.StringSerializer
```

Implémentation du Producteur Kafka:



```
package ma.enset.nombreclics;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.kafka.core.KafkaTemplate;
import org.springframework.stereotype.Service;

@Service  1 usage  new *
public class KafkaProducerService {
    @Autowired
    private KafkaTemplate<String,String> kafkaTemplate;
    public void sendClick(String userId) {  1 usage  new *
        kafkaTemplate.send( topic: "clicks", userId,  data: "click");
    }
}
```

Implémentation du consumer Kafka:

Création du Contrôleur :

```java
@RestController  new *
@RequestMapping("/click")
@CrossOrigin(origins = "*")

public class ClickController {
    @Autowired
    private KafkaProducerService producerService;
    @Autowired
    private KafkaConsumerService consumerService;
    @GetMapping("/{userId}")  new *
    public ResponseEntity<String> registerClick(@PathVariable String userId) {
        producerService.sendClick(userId);
        return ResponseEntity.ok( body: "Click enregistre pour l'utilisateur" + userId);
    }
    @GetMapping("/count")  new *
    public ResponseEntity<Map<String, Long>> getClickCounts() {
        return ResponseEntity.ok(consumerService.getClickCounts());
    }

}
```

Le stream processing:

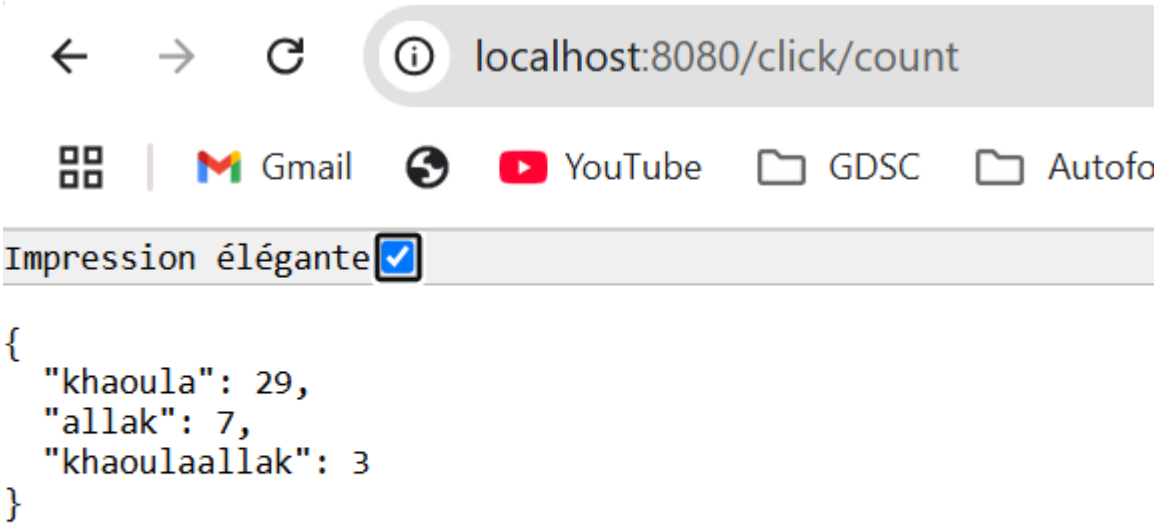
```java
@Configuration  new *
@EnableKafkaStreams
public class ClickStreamProcessor {
    @Bean  new *
    public KStream<String, String> processClicks(StreamsBuilder builder) {
        Consumed<String, String> consumed = Consumed.with(Serdes.String(), Serdes.String());
        Produced<String, Long> produced = Produced.with(Serdes.String(), Serdes.Long());

        KStream<String, String> clicksStream = builder.stream( topic: "clicks", consumed);

        clicksStream
                .groupByKey() KGroupedStream<String, String>
                .count(Materialized.as( storeName: "clicks-count-store")) KTable<String, Long>
                .toStream() KStream<String, Long>
                .to( s: "click-counts", produced);

        return clicksStream;
    }
}
```

Résultats:

- Sur le terminal

```
[appuser@85243fb291cb ~]$ kafka-console-consumer --topic click-counts --bootstrap-server localhost:9092 --from-beginning --formatter kafka.tools.DefaultMessageFormatter --p
roperty print.key=true --property print.value=true --property key.deserializer=org.apache.kafka.common.serialization.StringDeserializer --property value.deserializer=org.ap
ache.kafka.common.serialization.LongDeserializer
khaoula 21
mohamed 10
douaa   1
khoukha 1
KHAOULA 6
KHAOULA 9
KHAOULA 15
KHAOULA 16
KHAOULA 17
allak   3
khaoulaallak    3
khaoula 29
allak   7
```

- A travers le Rest API:



- Dans l'application: