

# Kafka Stream

---

- Concepts clés
  - KStream : Représente un flux de données en temps réel.
  - StreamsBuilder : Permet de construire le traitement des flux.
  - Serdes (Serializer/Deserializer) : Utilisé pour la sérialisation et désérialisation des messages.
  - Produced : Définit la manière de produire les messages vers les topics.
  - Consumed : Indique comment consommer les messages depuis les topics.

## 1 Méthode kStream - Transformation simple des messages

```
@Bean
public KStream<String, String> kStream(StreamsBuilder streamsBuilder) {
    System.out.println("🚀 KStream method called!");

    // Consomme le topic "demo-stream-topic"
    KStream<String, String> kStream = streamsBuilder.stream("demo-stream-topic", Consumed.with(Serdes.String(), Serdes.String()));

    // Transforme chaque valeur en majuscules et l'encadre de crochets
    KStream<String, String> processedStream = kStream
        .mapValues(value -> "[" + value.toUpperCase() + "]")
        .peek((key, value) -> System.out.println("🔄 Transformed message: key=" + key + ", value=" + value));

    // Envoie le flux transformé vers le topic "demo-output-stream-topic"
    processedStream.to("demo-output-stream-topic", Produced.with(Serdes.String(), Serdes.String()));

    return processedStream;
}
```

✓ Ce que ça fait :

- Prend des messages du topic "demo-stream-topic".
- Met les valeurs en majuscules et ajoute des crochets ([HELLO]).
- Affiche les messages transformés dans la console.
- Publie les messages transformés dans "demo-output-stream-topic".


## 2 Méthode filterStream - Filtrage des messages

```
@Bean
public KStream<String, String> filterStream(StreamsBuilder streamsBuilder) {
    KStream<String, String> kStream = streamsBuilder.stream("demo-stream-filter-topic", Consumed.with(Serdes.String(), Serdes.String()));

    // Filtre les messages contenant "important"
    KStream<String, String> filteredStream = kStream
        .filter((key, value) -> value != null && value.contains("important"))
        .peek((key, value) -> System.out.println("✉ Filtered message: " + value));

    // Envoie les messages filtrés dans le topic de sortie
    filteredStream.to("demo-output-filter-topic", Produced.with(Serdes.String(), Serdes.String()));

    return filteredStream;
}
```

-  Ce que ça fait :
  - Consomme le topic "demo-stream-filter-topic".
  - Filtre les messages contenant le mot "important".
  - Affiche les messages filtrés.
  - Publie les messages filtrés dans "demo-output-filter-topic".

### 3 Méthode mapValuesStream - Transformation des valeurs

```
@Bean
public KStream<String, String> mapValuesStream(StreamsBuilder streamsBuilder) {
    KStream<String, String> kStream = streamsBuilder.stream("demo-stream-map-topic", Consumed.with(Serdes.String(), Serdes.String()));

    // Transforme les valeurs en majuscules
    KStream<String, String> mappedStream = kStream
        .mapValues(value -> value.toUpperCase())
        .peek((key, value) -> System.out.println("🔄 Transformed message: " + value));

    mappedStream.to("demo-output-map-topic", Produced.with(Serdes.String(), Serdes.String()));

    return mappedStream;
}
```

-  Ce que ça fait :
  - Prend des messages du topic "demo-stream-map-topic".
  - Convertit les valeurs en majuscules.
  - Affiche les messages transformés.
  - Envoie le résultat dans "demo-output-map-topic".




## 4 Méthode branchStream - Division du flux (branching)

```
@Bean
public KStream<String, String> branchStream(StreamsBuilder streamsBuilder) {
    KStream<String, String> kStream = streamsBuilder.stream("demo-stream-branch-topic", Consumed.with(Serdes.String(), Serdes.String()));

    // Crée deux branches : une pour les messages contenant "error", une autre pour ceux contenant "info"
    KStream<String, String>[] branches = kStream.branch(
        (key, value) -> value.contains("error"),
        (key, value) -> value.contains("info")
    );

    // Envoie chaque branche vers un topic spécifique
    branches[0].to("demo-output-branch-topic-1", Produced.with(Serdes.String(), Serdes.String())); // Pour les erreurs
    branches[1].to("demo-output-branch-topic-2", Produced.with(Serdes.String(), Serdes.String())); // Pour les infos

    return kStream; // Retourne le flux d'origine (optionnel)
}
```

-  Ce que ça fait :
  - Divise le flux en deux branches :
  - Branche 1 : Messages contenant "error".
  - Branche 2 : Messages contenant "info".
  - Envoie les messages vers des topics distincts.


## 5 Méthode `aggregateStream` - Agrégation et comptage des messages

```
@Bean
public KStream<String, Long> aggregateStream(StreamsBuilder streamsBuilder) {
    KStream<String, String> kStream = streamsBuilder.stream("demo-stream-aggregate-topic", Consumed.with(Serdes.String(), Serdes.String()));

    // Regroupe les messages par clé, puis compte leur nombre
    KStream<String, Long> aggregatedStream = kStream
        .groupByKey(Grouped.with(Serdes.String(), Serdes.String()))
        .count() // Renvoie un KTable
        .toStream() // Convertit en KStream
        .peek((key, value) -> System.out.println("🇫🇷 Aggregated key: " + key + ", count: " + value));

    aggregatedStream.to("demo-output-aggregation-topic", Produced.with(Serdes.String(), Serdes.Long()));

    return aggregatedStream;
}
```

-  Ce que ça fait :
  - Groupe les messages par clé.
  - Compte le nombre de messages pour chaque clé.
  - Affiche les résultats (clé + nombre d'occurrences).
  - Publie le résultat dans "demo-output-aggregation-topic".


## 6 Méthode mergeStreams - Fusion de deux flux

```
@Bean
public KStream<String, String> mergeStreams(StreamsBuilder streamsBuilder) {
    KStream<String, String> kStream1 = streamsBuilder.stream("demo-stream-merge1-topic", Consumed.with(Serdes.String(), Serdes.String()));
    KStream<String, String> kStream2 = streamsBuilder.stream("demo-stream-merge2-topic", Consumed.with(Serdes.String(), Serdes.String()));

    // Fusionne les deux flux
    KStream<String, String> mergedStream = kStream1.merge(kStream2)
        .peek((key, value) -> System.out.println("🔥 Merged message: " + value));

    mergedStream.to("demo-output-merge-topic", Produced.with(Serdes.String(), Serdes.String()));

    return mergedStream;
}
```

-  Ce que ça fait :
  - Consomme deux topics distincts.
  - Fusionne les deux flux.
  - Affiche les messages fusionnés.
  - Envoie le flux fusionné dans "demo-output-merge-topic".



## Conclusion :

Ce code montre des opérations de base sur les flux avec Kafka Streams : transformation, filtrage, branchement, agrégation et fusion. Chaque méthode illustre une fonctionnalité utile pour manipuler des données en temps réel sur Kafka avec Spring Boot.

