Spring Boot : Kafka

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en programmation par contrainte (IA)
Ingénieur en génie logiciel

elmouelhi.achref@gmail.com

Plan

Kafka

Kafka

- Plateforme open-source de streaming de données développée par **Apache**.
- Permettant la gestion et le traitement de flux massifs de données en temps réel.
- Utilisant le modèle de publication/souscription (pub/sub) où les producteurs envoient des messages, et les consommateurs s'abonnent pour recevoir les messages.
- Conçu pour être hautement disponible et tolérant aux pannes, garantissant ainsi la continuité des opérations même en cas de défaillance d'un nœud.

Kafka

Kafka : composants

- **Broker**
  - responsables du stockage et de la gestion des données
  - reçoivent, stockent et distribuent les messages entre producteurs/consommateurs
- **Topic** : une catégorie ou un canal logique auquel les producteurs envoient des messages et à partir duquel les consommateurs les reçoivent.
- **Zookeeper** : utilisé pour la gestion des **brokers** et le suivi des **topics**
- **Producer**
  - des applications qui publient des messages sur les **topics**
  - envoient les données aux **brokers** pour les stocker et les distribuer aux consommateurs.
- **Consumer**
  - des applications qui s'abonnent à des topics spécifiques pour recevoir des messages.
  - reçoivent les données des **brokers** et les traitent selon leurs besoins

Kafka

Kafka : téléchargement et démarrage

- Allez à `https://kafka.apache.org/36/documentation.html`
- Choisissez une version récente puis téléchargez
- Décompressez le fichier téléchargé et choisissez un nom court pour le dossier décompressé (comme `kafka`)
- Placez le à la racine (`C:\` par exemple)
- Ouvrez **PowerShell** et déplacez vous dans `kafka`

Kafka

Ensuite

1. Démarrez `zookeeper`
2. Démarrer `kafka`

Kafka

Pour démarrer `zookeeper`

```
bin\windows\zookeeper-server-start.bat config\zookeeper.properties
```

Kafka

Pour démarrer `zookeeper`

```
bin\windows\zookeeper-server-start.bat config\zookeeper.properties
```

Sous `Windows`**, nous pourrons utiliser** `start` **pour démarrer** `zookeeper` **dans un autre terminal**

```
start bin\windows\zookeeper-server-start.bat
config\zookeeper.properties
```

Kafka

Pour démarrer `zookeeper`

```
bin\windows\zookeeper-server-start.bat config\zookeeper.properties
```

Sous `Windows`**, nous pourrons utiliser** `start` **pour démarrer** `zookeeper` **dans un autre terminal**

```
start bin\windows\zookeeper-server-start.bat
config\zookeeper.properties
```

Et sous `Linux` **et** `Mac`

```
bin\zookeeper-server-start.sh config\zookeeper.properties
```

Kafka

Pour démarrer `kafka`

```
bin\windows\kafka-server-start.bat config\server.properties
```

Kafka

Pour démarrer `kafka`

```
bin\windows\kafka-server-start.bat config\server.properties
```

Sous `Windows`**, nous pourrons utiliser** `start` **pour démarrer** `kafka` **dans un autre terminal**

```
start bin\windows\kafka-server-start.bat config\server.properties
```

Kafka

Pour démarrer `kafka`

```
bin\windows\kafka-server-start.bat config\server.properties
```

Sous `Windows`**, nous pourrons utiliser** `start` **pour démarrer** `kafka` **dans un autre terminal**

```
start bin\windows\kafka-server-start.bat config\server.properties
```

Et sous `Linux` **et** `Mac`

```
bin\kafka-server-start.sh config\server.properties
```

Kafka

Vérifiez que

1. `zookeeper` utilise le port `2181`
2. `kafka` utilise le port `9092`

Kafka

Pour créer un nouveau `topic` (appelé `mon-topic`)

```
bin\windows\kafka-topics.bat --create --topic mon-topic --bootstrap-server localhost:9092
```

Kafka

Pour créer un nouveau `topic` (appelé `mon-topic`)

```
bin\windows\kafka-topics.bat --create --topic mon-topic --bootstrap-server localhost:9092
```

Résultat

```
Created topic mon-topic.
```

Kafka

Pour créer un nouveau `topic` (appelé `mon-topic`)

```
bin\windows\kafka-topics.bat --create --topic mon-topic --bootstrap-server localhost:9092
```

Résultat

```
Created topic mon-topic.
```

Explication

- `bin\windows\kafka-topics.bat` : script exécutable sous **Windows** pour interagir avec les topics **Kafka**.
- `--create` : demande de créer un nouveau `topic`.
- `--topic mon-topic` : spécifie le nom du `topic`.
- `--bootstrap-server localhost:9092` : spécifie l'adresse et le port `9092`) du serveur **Kafka** à laquelle se connecter pour créer le topic.

Kafka

Pour écrire dans le `topic`, exécutez depuis un nouveau terminal

```
bin\windows\kafka-console-producer.bat --topic mon-topic
--bootstrap-server localhost:9092
```

Kafka

Pour écrire dans le `topic`, exécutez depuis un nouveau terminal

```
bin\windows\kafka-console-producer.bat --topic mon-topic
--bootstrap-server localhost:9092
```

Envoyez au mois un message (`hello` par exemple)

```
>hello
```

Kafka

Pour lire depuis le `topic`, exécutez depuis un nouveau terminal

```
bin/windows/kafka-console-consumer.bat --topic mon-topic
--bootstrap-server localhost:9092
```

Kafka

Pour lire depuis le `topic`, exécutez depuis un nouveau terminal

```
bin/windows/kafka-console-consumer.bat --topic mon-topic
--bootstrap-server localhost:9092
```

Pour lire tous les messages qui ont été envoyés depuis le début

```
bin/windows/kafka-console-consumer.bat --topic mon-topic
--from-beginning --bootstrap-server localhost:9092
```

Kafka

Pour lire depuis le `topic`, exécutez depuis un nouveau terminal

```
bin/windows/kafka-console-consumer.bat --topic mon-topic
--bootstrap-server localhost:9092
```

Pour lire tous les messages qui ont été envoyés depuis le début

```
bin/windows/kafka-console-consumer.bat --topic mon-topic
--from-beginning --bootstrap-server localhost:9092
```


Remarques

- En exécutant la deuxième commande, vérifiez que `hello` a été lu par le `consumer`.
- Envoyez des nouveaux messages depuis le `producer` et vérifiez qu'ils ont été reçus par le `consumer`

Kafka

Création de projet Spring Boot

- Aller dans `File > New > Other`
- Chercher `Spring`, dans `Spring Boot` sélectionner `Spring Starter Project` et cliquer sur `Next >`
- Saisir
  - `spring-kafka` dans `Name`,
  - `com.example` dans `Group`,
  - `spring-kafka` dans `Artifact`
  - `com.example.demo` dans `Package`
- Cliquer sur `Next`
- Chercher et cocher les cases correspondantes aux `Spring for Apache Kafka`, `Spring for Apache Kafka Streams`, `Cloud Stream`, `Lombok` et `Spring Web`
- Cliquer sur `Next` puis sur `Finish`

Spring Boot & REST

Créons une entité `PersonneEvent` **dans** `com.example.demo.model`

```
package com.example.demo.model;

import java.time.LocalDateTime;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.NonNull;
import lombok.RequiredArgsConstructor;

@NoArgsConstructor
@AllArgsConstructor
@Data
@RequiredArgsConstructor
public class PersonneEvent {
    private Long id;
    @NonNull
    private String nom;
    @NonNull
    private String prenom;
    @NonNull
    private LocalDateTime dateNaissance;
}
```

Kafka

Créons le contrôleur REST suivant

```
package com.example.demo;

import java.time.LocalDateTime;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

import com.example.demo.model.PersonneEvent;

@RestController
public class PersonneEventRestController {

    @GetMapping("/send/{topic}")
    public PersonneEvent sendTopic(@PathVariable String topic, @RequestParam String nom,
       @RequestParam String prenom) {
        PersonneEvent personneEvent = new PersonneEvent(nom, prenom, LocalDateTime.now());
        return personneEvent;
    }
}
```

Spring Boot & REST

En allant à `http://localhost:8080/send/mon-topic?nom=Wick&prenom=John`, vérifiez que le résultat a la structure suivante

```
{
    "id": null,
    "nom": "Wick",
    "prenom": "John",
    "dateNaissance": "2022-03-17T22:22:16.555011"
}
```

Kafka

Pour produire des messages, injectons `StremBridge`

```
package com.example.demo;

import java.time.LocalDateTime;

import org.springframework.cloud.stream.function.StreamBridge;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

import com.example.demo.model.PersonneEvent;

import lombok.AllArgsConstructor;

@RestController
@AllArgsConstructor
public class PersonneEventRestController {

    private StreamBridge streamBridge;
    @GetMapping("/send/{topic}")
    public PersonneEvent sendTopic(@PathVariable String topic, @RequestParam String nom,
       @RequestParam String prenom) {
        PersonneEvent personneEvent = new PersonneEvent(nom, prenom, LocalDateTime.now());
        return personneEvent;
    }
}
```

Kafka

Utilisons `StremBridge` pour envoyer des messages

```
package com.example.demo;

import java.time.LocalDateTime;

import org.springframework.cloud.stream.function.StreamBridge;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

import com.example.demo.model.PersonneEvent;

import lombok.AllArgsConstructor;

@RestController
@AllArgsConstructor
public class PersonneEventRestController {

    private StreamBridge streamBridge;
    @GetMapping("/send/{topic}")
    public PersonneEvent sendTopic(@PathVariable String topic, @RequestParam String nom,
       @RequestParam String prenom) {
        PersonneEvent personneEvent = new PersonneEvent(nom, prenom, LocalDateTime.now());
        streamBridge.send(topic, personneEvent);
        return personneEvent;
    }
}
```

Kafka

Pour tester

- allez à `http://localhost:8080/send/mon-topic?nom=Wick&prenom=John`
- vérifier dans `kafka-console-consumer` la réception du message

Kafka

Pour consommer les messages, développons un bean `Consumer<PersonneEvent>` dans une classe annotée par `@Service`

```
package com.example.demo.service;

import java.util.function.Consumer;

import org.springframework.context.annotation.Bean;
import org.springframework.stereotype.Service;

import com.example.demo.model.PersonneEvent;

@Service
public class PersonneEventService {

    @Bean
    Consumer<PersonneEvent> personneEventConsumer() {
        return (data) -> {
            System.out.println("Consumed data: " + data.toString());
        };
    }
}
```

Spring Boot & REST

Mettons à jour `application.properties`

```
spring.cloud.stream.bindings.personneEventConsumer-in-0.destination=mon-topic
spring.cloud.function.definition=personneEventConsumer
```

Spring Boot & REST

Mettons à jour `application.properties`

```
spring.cloud.stream.bindings.personneEventConsumer-in-0.destination=mon-topic
spring.cloud.function.definition=personneEventConsumer
```

Explication

- **Spring Cloud Stream** essaiera d'utiliser une destination par défaut dérivée du nom du binding : dans notre cas `personneEventConsumer-in-0`
  - `in` signifie que le canal est utilisé pour consommer des messages.
  - `0` : index qui permet de distinguer plusieurs canaux du même type (entrée ou sortie) au sein d'une même application.
- La première propriété indique que la destination des messages entrants pour le canal `personneEventConsumer-in-0` est `mon-topic`.
- La deuxième propriété est utilisée pour définir la fonction **Spring Cloud Function** qui sera utilisée pour traiter les messages entrants.