Ecole Normale Supérieure de l'Enseignement Technique Mohammedia
Université Hassan II de Casablanca

ENSET

المدرسة العليا لأساتذة التعليم التقني المحمدية
جامعة الحسن الثاني بالدار البيضاء

Département Mathématiques et Informatique

Filière :

« Ingénierie Informatique : Big Data et Cloud Computing »

II-BDCC

Travail Pratique : Systèmes Distribués Asynchrones avec Kafka Brokers

Réalisé par :

GUEDDI Hamza

Année universitaire : 2020-2021

Table des matières

Enoncé

On souhaite créer une application basée sur l'envoi de messages et des objets en temps réel en utilisant les API de Kafka.

En premier lieu, on va créer une simple application Java et envoyer des messages du producer vers le consumer.

La deuxième application va être créée avec Spring, et donc on va utiliser Spring REST pour l'envoi des messages et un service qui va nous servir pour consommer ces messages.

Introduction

Apache Kafka est une plate-forme de diffusion (streaming) distribuée développée en Scala et Java.

- Une plate-forme de streaming possède trois fonctionnalités clés :

- Permettre aux applications clientes Kafka de publier et s'abonner à des flux d'enregistrements. Similaires à une file d'attente de messages ou à un système de messagerie d'entreprise comme les Brokers JMS (ActiveMQ) ou AMQP (RabbitMQ)

- Permet de stocker les flux d'enregistrements de manière durable et tolérante aux pannes.

- Permet de traiter les flux d'enregistrements au fur et à mesure qu'ils se produisent (Real Time Stream Processing)

Les API de Kafka

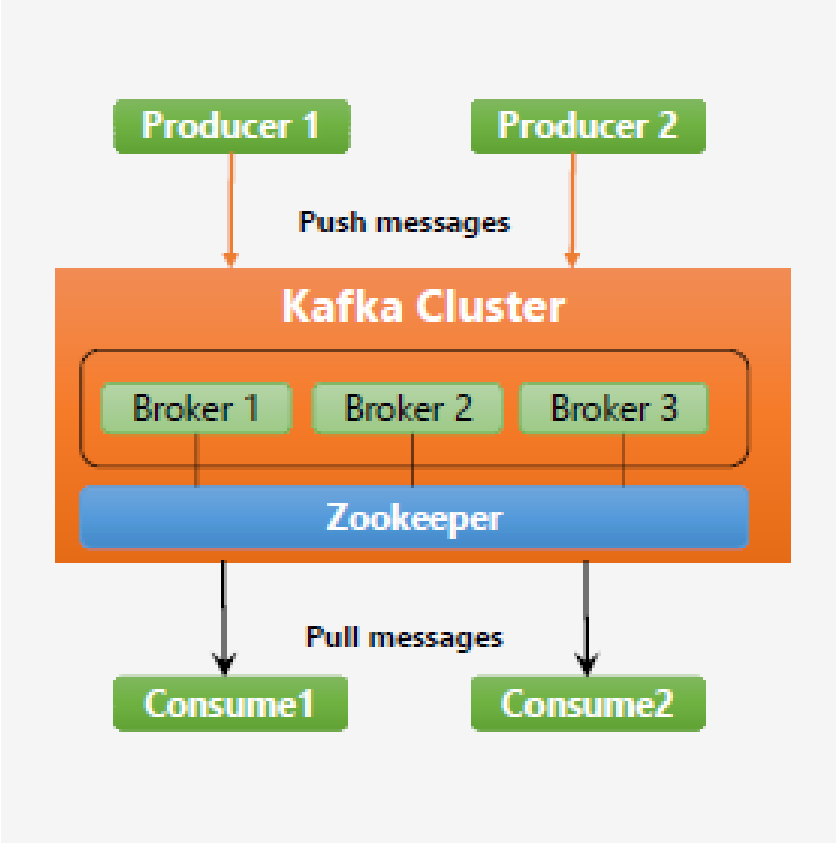
Kafka a quatre API principales :
• Producer API: Permet à une application de publier un flux d'enregistrements vers un ou plusieurs Topics (Sujets) Kafka.
• Consumer API: Permet à une application de s'abonner à un ou

plusieurs Topics et de traiter le flux d'enregistrements qui lui sont transmis.

• Streams API: Permet à une application d'agir en tant que processeur de flux, en
• Consommant un flux d'entrée provenant d'un ou plusieurs Topics
• Transformant efficacement les flux d'entrée en flux de sortie
• Produisant un flux de sortie vers un ou plusieurs Topics en sortie.

• Connector API: Permet de créer et d'exécuter des producteurs ou des consommateurs réutilisables qui connectent des topics Kafka à des applications ou des systèmes de données existants. Par exemple, un connecteur vers une base de données relationnelle peut capturer chaque modification apportée à une table.

• Kafka fournit des API clientes pour différents langages : Java, C++, Node JS, .Net, PHP, Python, etc…



Code (App1)

Les dépendances (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
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>org.bdcc</groupId>
    <artifactId>kafka-app</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
    </properties>

    <dependencies>
        <!--
https://mvnrepository.com/artifact/org.apache.kafka/kafka -->
        <dependency>
            <groupId>org.apache.kafka</groupId>
            <artifactId>kafka_2.13</artifactId>
            <version>2.7.0</version>
        </dependency>


        <!--
https://mvnrepository.com/artifact/org.apache.kafka/kafka-
clients -->
        <dependency>
            <groupId>org.apache.kafka</groupId>
            <artifactId>kafka-clients</artifactId>
            <version>2.7.0</version>
        </dependency>


        <!--
https://mvnrepository.com/artifact/org.apache.kafka/kafka-
streams -->
        <dependency>
            <groupId>org.apache.kafka</groupId>
            <artifactId>kafka-streams</artifactId>
```

```
            <version>2.7.0</version>
        </dependency>

    </dependencies>

</project>
```

ConsumerApp.java

```
import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.KafkaConsumer;
import
org.apache.kafka.common.serialization.StringDeserializer;

import java.time.Duration;
import java.util.Collections;
import java.util.Properties;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;

public class ConsumerApp {
    public static void main(String[] args) {
        Properties properties = new Properties();
        properties.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG
, "localhost:9092");

properties.put(ConsumerConfig.AUTO_COMMIT_INTERVAL_MS_CONFIG ,
"1000");

properties.put(ConsumerConfig.ENABLE_AUTO_COMMIT_CONFIG ,
"true");
        properties.put(ConsumerConfig.GROUP_ID_CONFIG , "test-
group-1");

properties.put(ConsumerConfig.SESSION_TIMEOUT_MS_CONFIG  ,
"30000");

properties.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG ,
StringDeserializer.class.getName());

properties.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG
, StringDeserializer.class.getName());


        KafkaConsumer<String , String> kafkaConsumer = new
KafkaConsumer<String, String>(properties);
```

```
kafkaConsumer.subscribe(Collections.singleton("test1"));



Executors.newScheduledThreadPool(1).scheduleAtFixedRate(() ->
{
            System.out.println("---------------");
            ConsumerRecords<String , String> consumerRecords =
kafkaConsumer.poll(Duration.ofMillis(1000));
            consumerRecords.forEach(consumerRecord -> {

System.out.println("Key=>"+consumerRecord.key()+"=>"+consumerR
ecord.value()+" "+consumerRecord.offset());

            });
        } , 1000 , 1000 , TimeUnit.MILLISECONDS);
    }
}
```

ProducerApp.java

```
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.ProducerConfig;
import org.apache.kafka.clients.producer.ProducerRecord;
import org.apache.kafka.common.serialization.StringSerializer;

import java.util.Properties;
import java.util.Random;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;

public class ProducerApp {

    private int counter;

    public ProducerApp() {
        Properties properties = new Properties();
        properties.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG
, "localhost:9092");
        properties.put(ProducerConfig.CLIENT_ID_CONFIG ,
"client-producer-1");

properties.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG ,
StringSerializer.class.getName());

properties.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG ,
StringSerializer.class.getName());
```

```
        KafkaProducer<String,String> kafkaProducer = new
KafkaProducer<String, String>(properties);
        Random random = new Random();


Executors.newScheduledThreadPool(1).scheduleAtFixedRate(()->{
            String key = String.valueOf(++counter);
            String value = String.valueOf(random.nextDouble()
* 999999);
            kafkaProducer.send(new
ProducerRecord<String,String>("test1" , key , value)
,(metadata , exception)->{
                System.out.println("Sending message =>
"+value+" Partition => "+metadata.partition()
                        + "=> "+metadata.offset());
            });

        }, 1000 , 1000 , TimeUnit.MILLISECONDS);


    }

    public static void main(String[] args) {
        new ProducerApp();
    }
}
```

Captures d'écran

Premièrement on va démarrer les serveurs Zookeeper et Kafka .

```
C:\kafka_2.13-2.7.0>start bin\windows\zookeeper-server-start.bat config\zookeeper.properties
```

```
C:\kafka_2.13-2.7.0>start bin\windows\kafka-server-start.bat config\server.properties
```

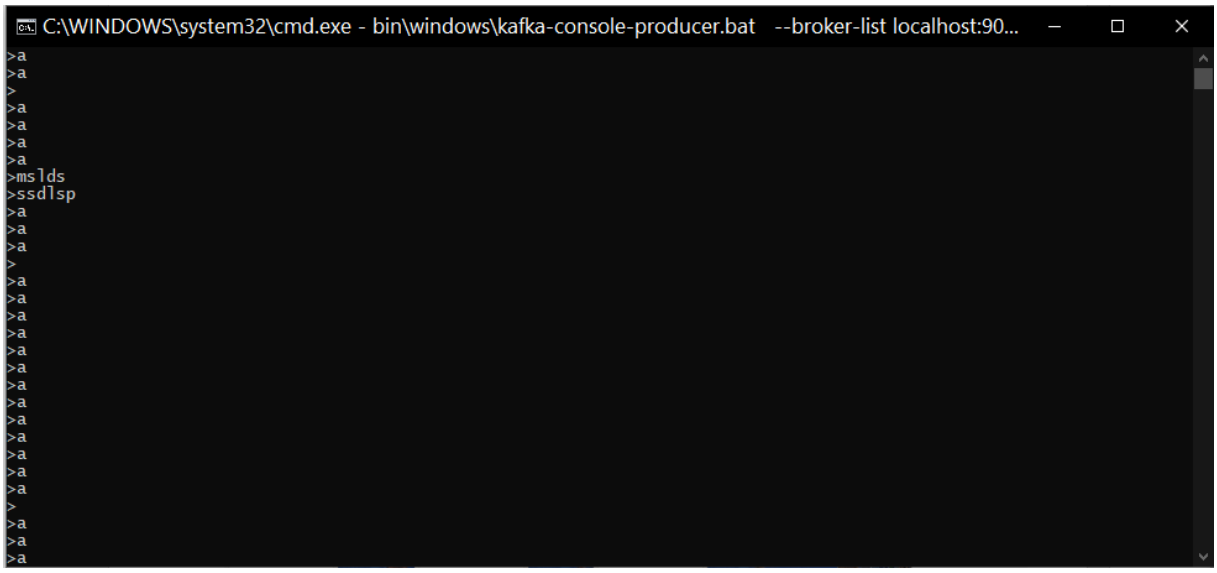
Maintenant , avant de passer à l'application Java , on va créer un producer et on va utiliser le topic « test3 » .

```
C:\kafka_2.13-2.7.0>start bin\windows\kafka-console-producer.bat --broker-list localhost:9092 --topic test3
```

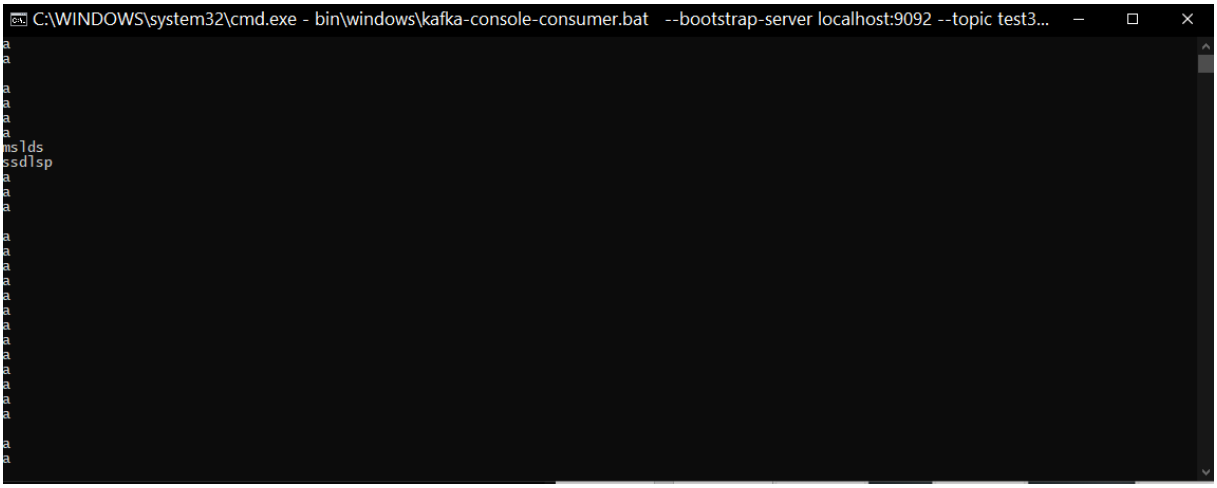
De même , on va créer un consumer et nous allons faire un petit test .

```
C:\kafka_2.13-2.7.0>start bin\windows\kafka-console-consumer.bat --bootstrap-server localhost:9092 --topic test3 --from-beginning
```

Producer :



Consumer :



Donc on voit déjà que l'envoi de messages se passe sans problème et en temps réel.
Maintenant, on bascule vers l'application Java.

Donc après avoir exécuté les deux classe Producer et Consumer, l'output de la classe Producer :

```
Sending message => 123467.16125093905 Partition => 0=> 4203
Sending message => 797397.2045665012 Partition => 0=> 4204
Sending message => 113695.2110236009 Partition => 0=> 4205
Sending message => 522237.31860081205 Partition => 0=> 4206
Sending message => 686828.2918210567 Partition => 0=> 4207
Sending message => 761404.9051893041 Partition => 0=> 4208
Sending message => 855592.4157368928 Partition => 0=> 4209
Sending message => 637078.9645463488 Partition => 0=> 4210
Sending message => 831569.161513195 Partition => 0=> 4211
Sending message => 391136.38405033835 Partition => 0=> 4212
Sending message => 231830.47821792992 Partition => 0=> 4213
Sending message => 130384.82671570634 Partition => 0=> 4214
Sending message => 298903.45342605637 Partition => 0=> 4215
Sending message => 700788.8096234788 Partition => 0=> 4216
Sending message => 237003.4820565993 Partition => 0=> 4217
Sending message => 494881.09983220464 Partition => 0=> 4218
Sending message => 320494.79350946617 Partition => 0=> 4219
Sending message => 857102.3670788147 Partition => 0=> 4220
Sending message => 846946.8892024787 Partition => 0=> 4221
Sending message => 419602.13572026155 Partition => 0=> 4222
Sending message => 584719.9138577001 Partition => 0=> 4223
Sending message => 950356.0362523922 Partition => 0=> 4224
Sending message => 917259.7821982641 Partition => 0=> 4225
Sending message => 41506.641438636725 Partition => 0=> 4226
```

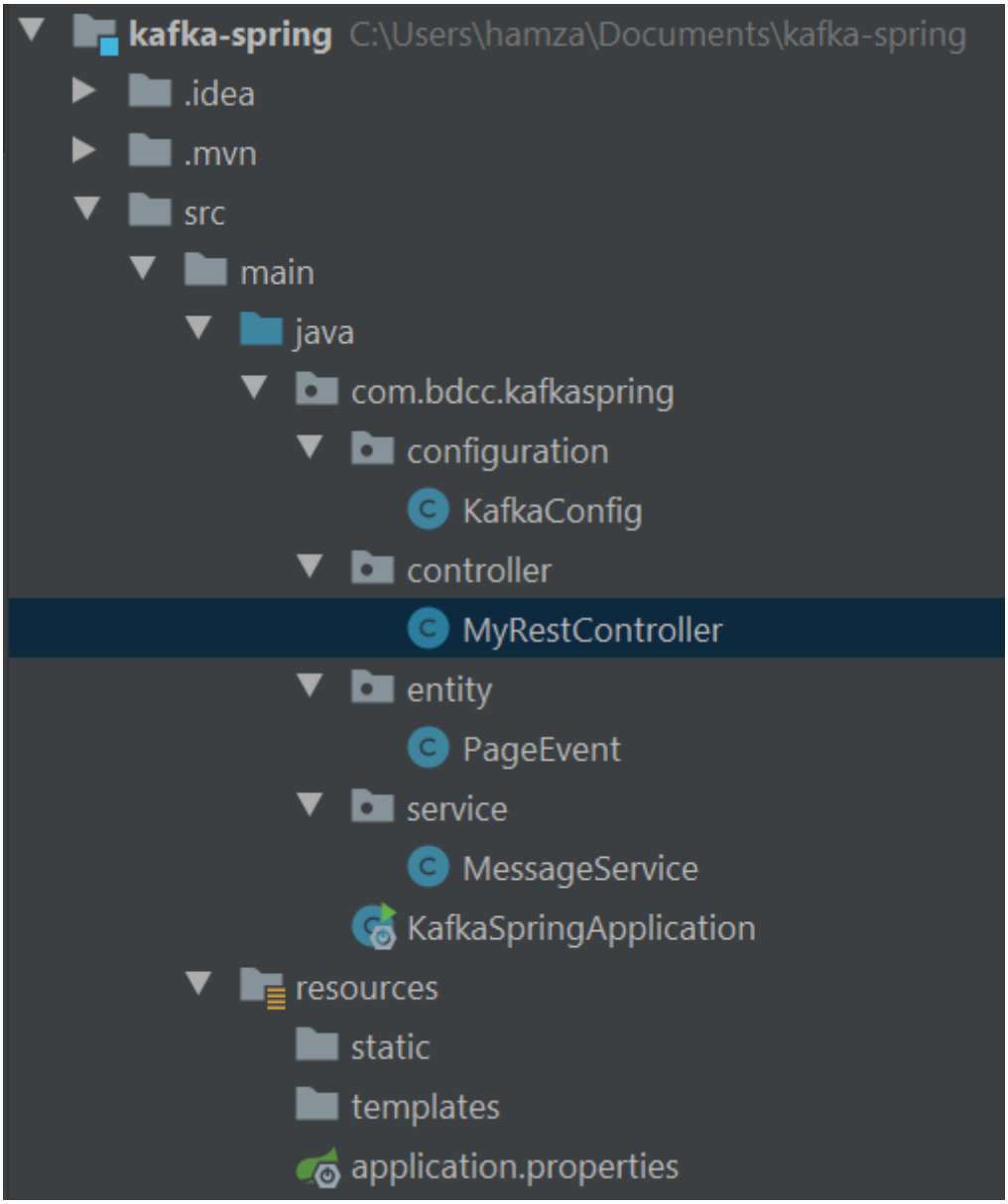
Output de la classe Consumer :

```
---------------
Key=>10=>391136.38405033835 4212
---------------
Key=>11=>231830.47821792992 4213
---------------
Key=>12=>130384.82671570634 4214
---------------
Key=>13=>298903.45342605637 4215
---------------
Key=>14=>700788.8096234788 4216
---------------
Key=>15=>237003.4820565993 4217
---------------
Key=>16=>494881.09983220464 4218
---------------
Key=>17=>320494.79350946617 4219
---------------
Key=>18=>857102.3670788147 4220
---------------
Key=>19=>846946.8892024787 4221
---------------
Key=>20=>419602.13572026155 4222
---------------
Key=>21=>584719.9138577001 4223
---------------
Key=>22=>950356.0362523922 4224
---------------
Key=>23=>917259.7821982641 4225
---------------
Key=>24=>41506.641438636725 4226
```

Maintenant, on passe à la 2ème application où on utilise Spring avec Kafka.

Code (Spring kafka)

Architecture



Les dépendances (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>2.4.1</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.bdcc</groupId>
    <artifactId>kafka-spring</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>kafka-spring</name>
    <description>Kafka Spring Application</description>

    <properties>
        <java.version>1.8</java.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</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>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.kafka</groupId>
            <artifactId>spring-kafka-test</artifactId>
            <scope>test</scope>
```

```
        </dependency>
    </dependencies>

    <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>
```

KafkaSpringApplication.java

```
package com.bdcc.kafkaspring;

import org.springframework.boot.SpringApplication;
import
org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.kafka.annotation.EnableKafka;

@SpringBootApplication
@EnableKafka
public class KafkaSpringApplication {

    public static void main(String[] args) {
        SpringApplication.run(KafkaSpringApplication.class,
args);
    }

}
```

PageEvent.java

```
package com.bdcc.kafkaspring.entity;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.util.Date;


@Data
@NoArgsConstructor @AllArgsConstructor
public class PageEvent {
    private String page;
    private Date date;
    private int duration;
}
```

KafkaConfig.java

```
package com.bdcc.kafkaspring.configuration;


import com.bdcc.kafkaspring.entity.PageEvent;
import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.kafka.clients.producer.ProducerConfig;
import org.apache.kafka.common.serialization.StringSerializer;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import
org.springframework.kafka.core.DefaultKafkaProducerFactory;
import org.springframework.kafka.core.KafkaTemplate;
import org.springframework.kafka.core.ProducerFactory;
import
org.springframework.kafka.support.serializer.JsonSerializer;

import java.util.HashMap;
import java.util.Map;

@Configuration
public class KafkaConfig {

    @Bean
    ProducerFactory<String , PageEvent> producerFactory(){
        Map<String,Object> config = new HashMap<>();
```

```
        config.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG ,
"localhost:9092");
        config.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG
, StringSerializer.class);

config.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG ,
JsonSerializer.class);
        return new DefaultKafkaProducerFactory<>(config);

    }

    @Bean
    KafkaTemplate<String , PageEvent> kafkaTemplate(){

        return new KafkaTemplate<>(producerFactory());
    }
}
```

MyRestController.java

```
package com.bdcc.kafkaspring.controller;

import com.bdcc.kafkaspring.entity.PageEvent;
import org.springframework.kafka.core.KafkaTemplate;
import
org.springframework.kafka.support.serializer.JsonDeserializer;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RestController;

import java.util.Date;
import java.util.Random;

@RestController
public class MyRestController {

    private KafkaTemplate<String, PageEvent> kafkaTemplate;
    JsonDeserializer jsonDeserializer;

    public MyRestController(KafkaTemplate<String, PageEvent>
kafkaTemplate) {
        this.kafkaTemplate = kafkaTemplate;
    }

    @GetMapping("/send/{page}/{topic}")
    public String send(@PathVariable String page,
                       @PathVariable String topic){
        PageEvent pageEvent = new PageEvent(page , new Date()
```

```
, new Random().nextInt(1000));
        kafkaTemplate.send(topic ,"key"+ pageEvent.getPage() ,
pageEvent);
        return "Message sent ...." ;

    }
}
```

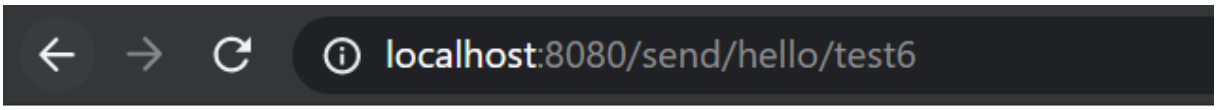
MessageService.java

```
package com.bdcc.kafkaspring.service;

import com.bdcc.kafkaspring.entity.PageEvent;
import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.json.JsonMapper;
import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.springframework.kafka.annotation.KafkaListener;
import
org.springframework.kafka.support.serializer.ErrorHandlingDese
rializer;
import org.springframework.stereotype.Service;

@Service
public class MessageService {

    @KafkaListener(topics = "test6", groupId = "group-ms")
    public void onMessage(ConsumerRecord<String, String>
consumerRecord) throws JsonProcessingException {
        System.out.println("**********************");
        PageEvent pageEvent =
pageEvent(consumerRecord.value());
        System.out.println(consumerRecord.key());
        System.out.println(pageEvent.getPage() + "," +
pageEvent.getDate() + "," + pageEvent.getDuration());
        System.out.println("**********************");
    }

    private PageEvent pageEvent(String jsonPageEvent) throws
JsonProcessingException {
        JsonMapper jsonMapper = new JsonMapper();
        PageEvent pageEvent =
jsonMapper.readValue(jsonPageEvent, PageEvent.class);
        return pageEvent;

    }

}
```

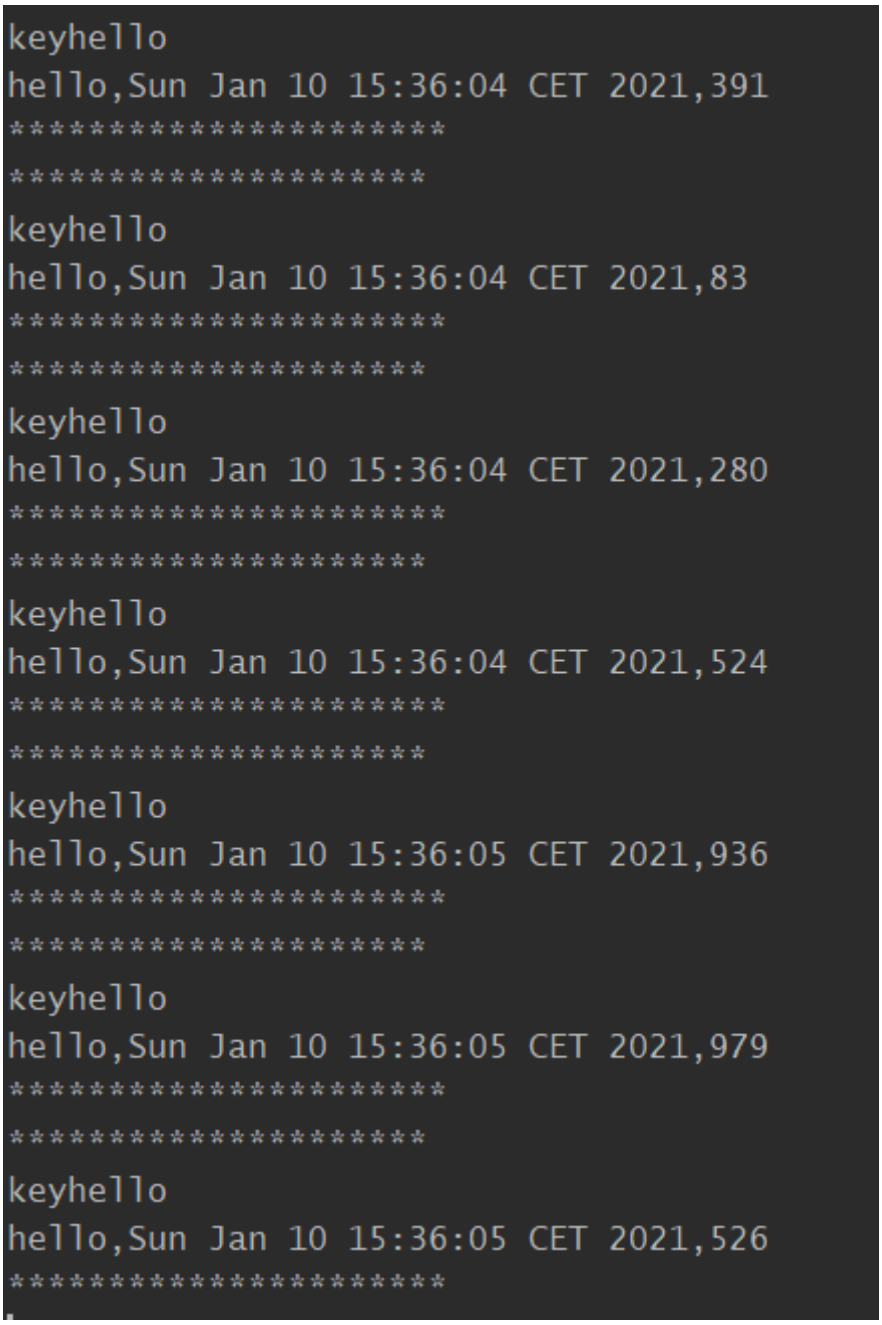
Captures d’écran (Spring Kafka)

On va essayer maintenant à travers notre REST Controller d’envoyer un objet JSON sérialisé de la classe PageEvent. Cet objet va être envoyé par le producer vers le topic « test6 » .

Du coup notre service va consommer cet objet et va le désérialiser.



Message sent ....



Donc on voit ici à travers l'output au-dessus que le service a réussi à consommer les objets du topic « test6 »

Conclusion

Ce TP a été bénéfique car on a vu un autre modèle pour envoyer des objets de façon asynchrone. On a également pu voir la performance de Kafka, et qu'il est très pratique de l'utiliser pour faire des traitements en temps réel surtout s'il s'agit du contexte du Big Data.