Event Driven Distributed Processing :

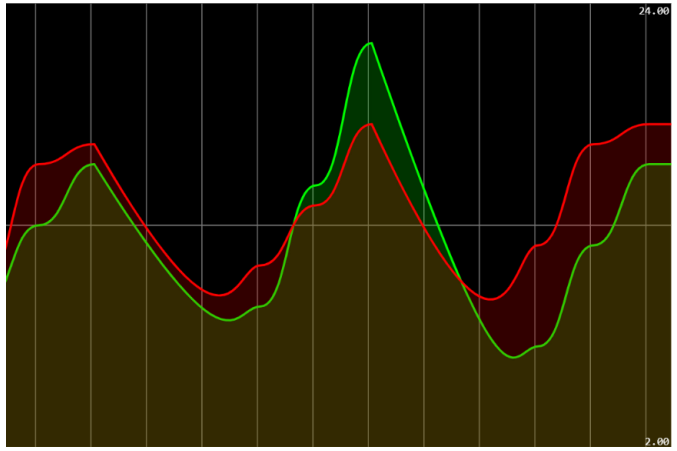
- JMS, ActiveMQ
- KAFKA, KAFKA STREAMS
- Spring Cloud Stream Functions

JMS

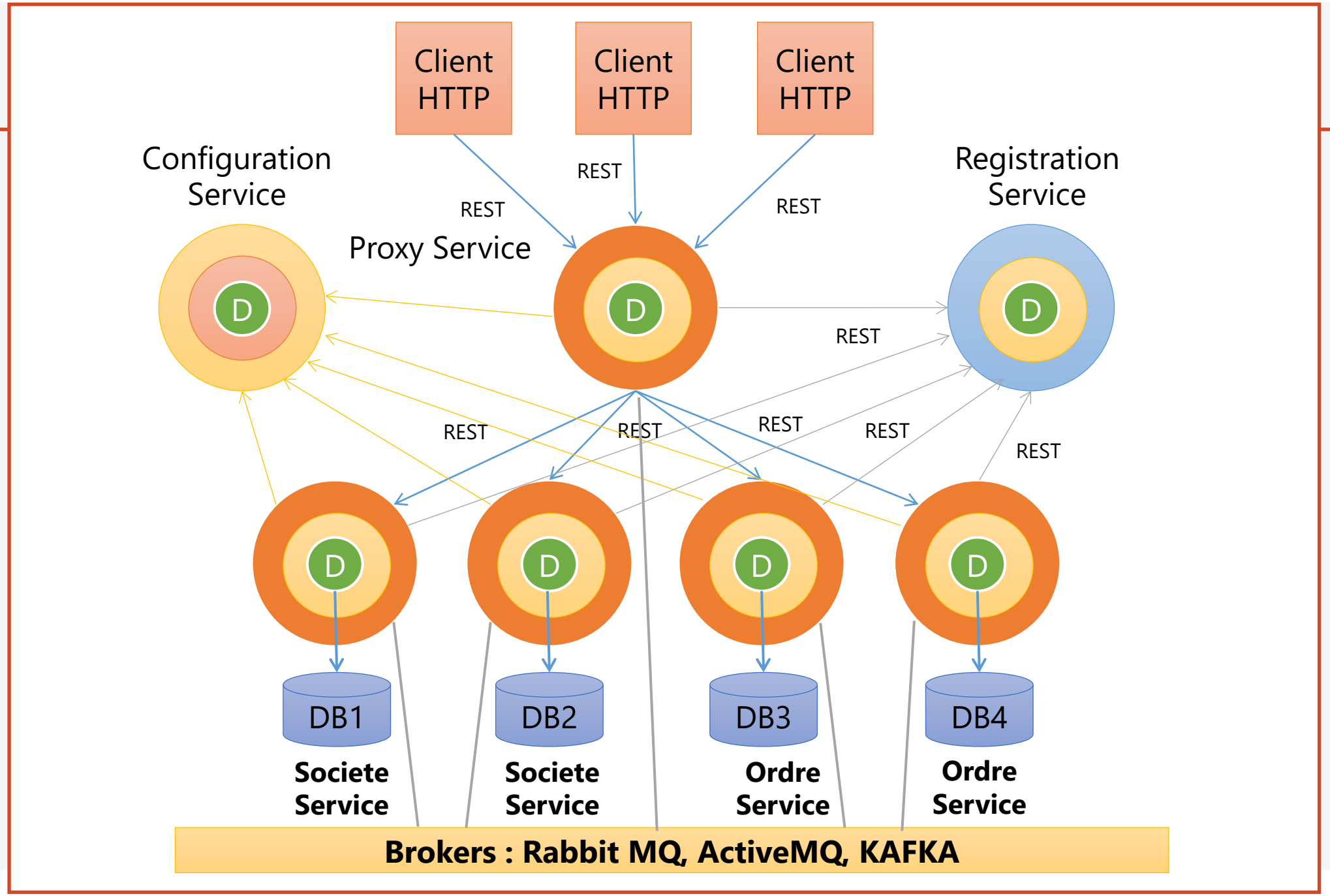
Apache ACTIVEMQ

RabbitMQ

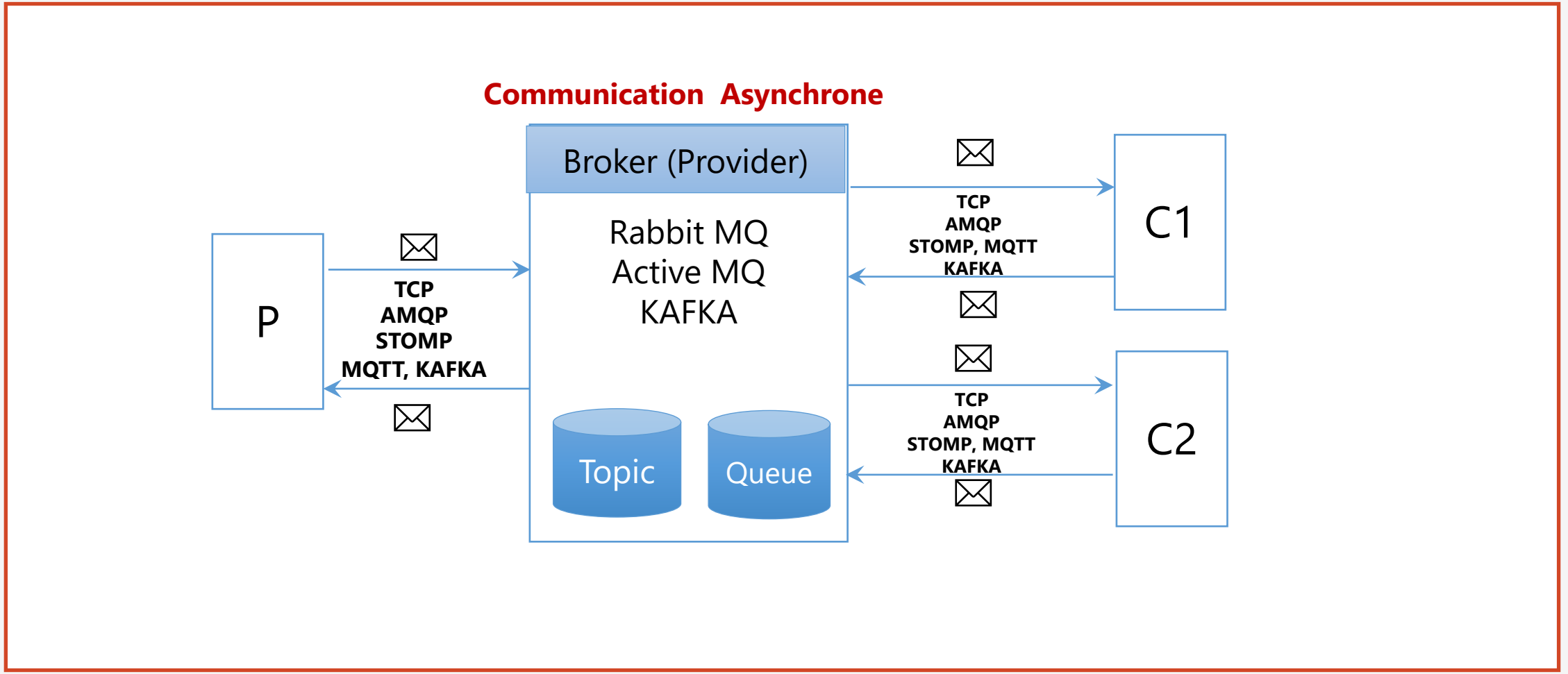
KAFKA



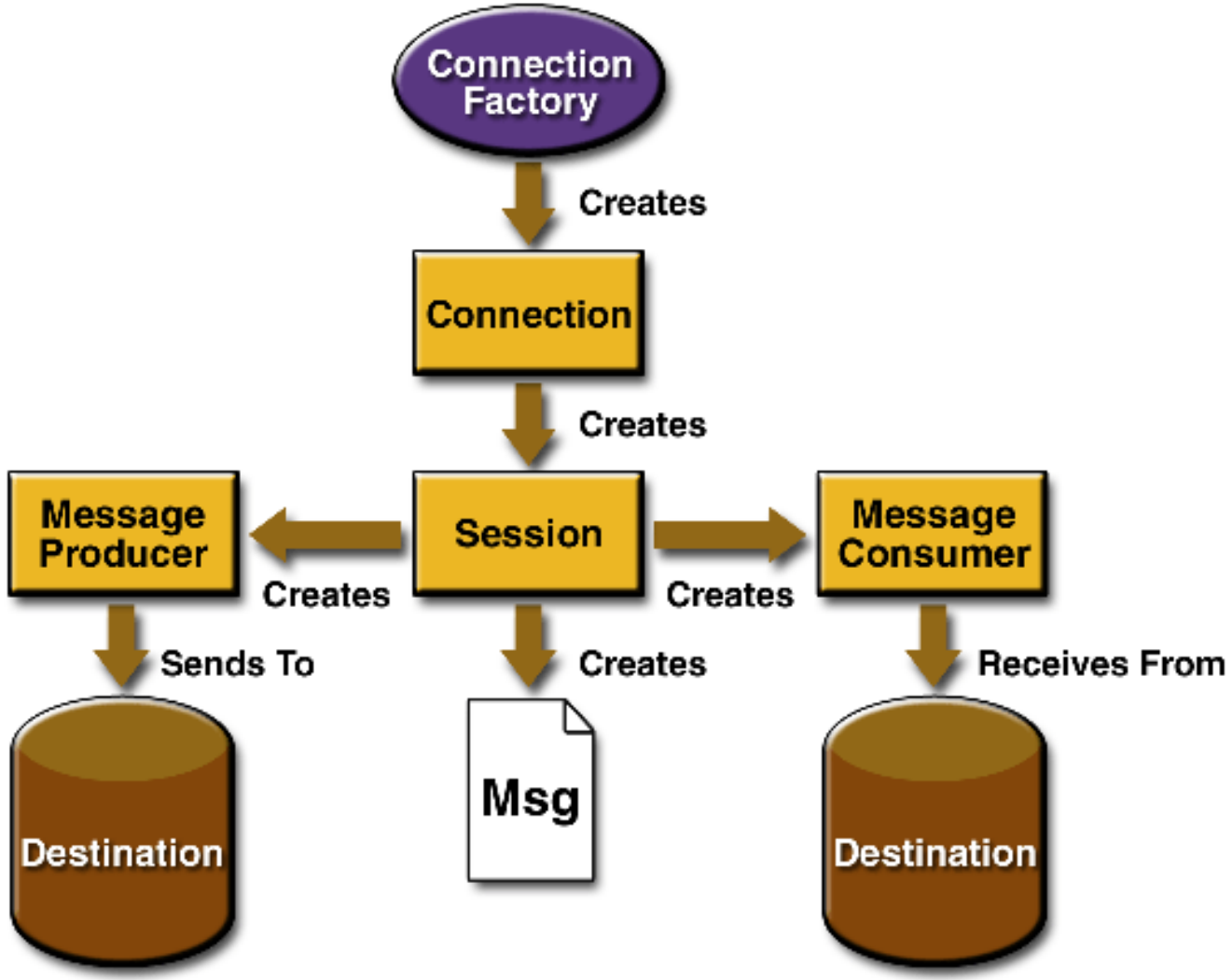
Mohamed Youssfi
Laboratoire Signaux Systèmes Distribués et Intelligence Artificielle (SSDIA)
ENSET, Université Hassan II Casablanca, Maroc
Email : med@youssfi.net
Supports de cours : http://fr.slideshare.net/mohamedyoussfi9
Chaîne vidéo : http://youtube.com/mohamedYoussfi
Recherche : http://www.researchgate.net/profile/Youssfi_Mohamed/publications


Client HTTP
Client HTTP
Client HTTP
Configuration Service
Proxy Service
Registration Service
REST
REST
REST
REST
REST
REST
REST
REST
REST
D
DB1
DB2
DB3
DB4
Societe Service
Societe Service
Ordre Service
Ordre Service
Brokers : Rabbit MQ, ActiveMQ, KAFKA

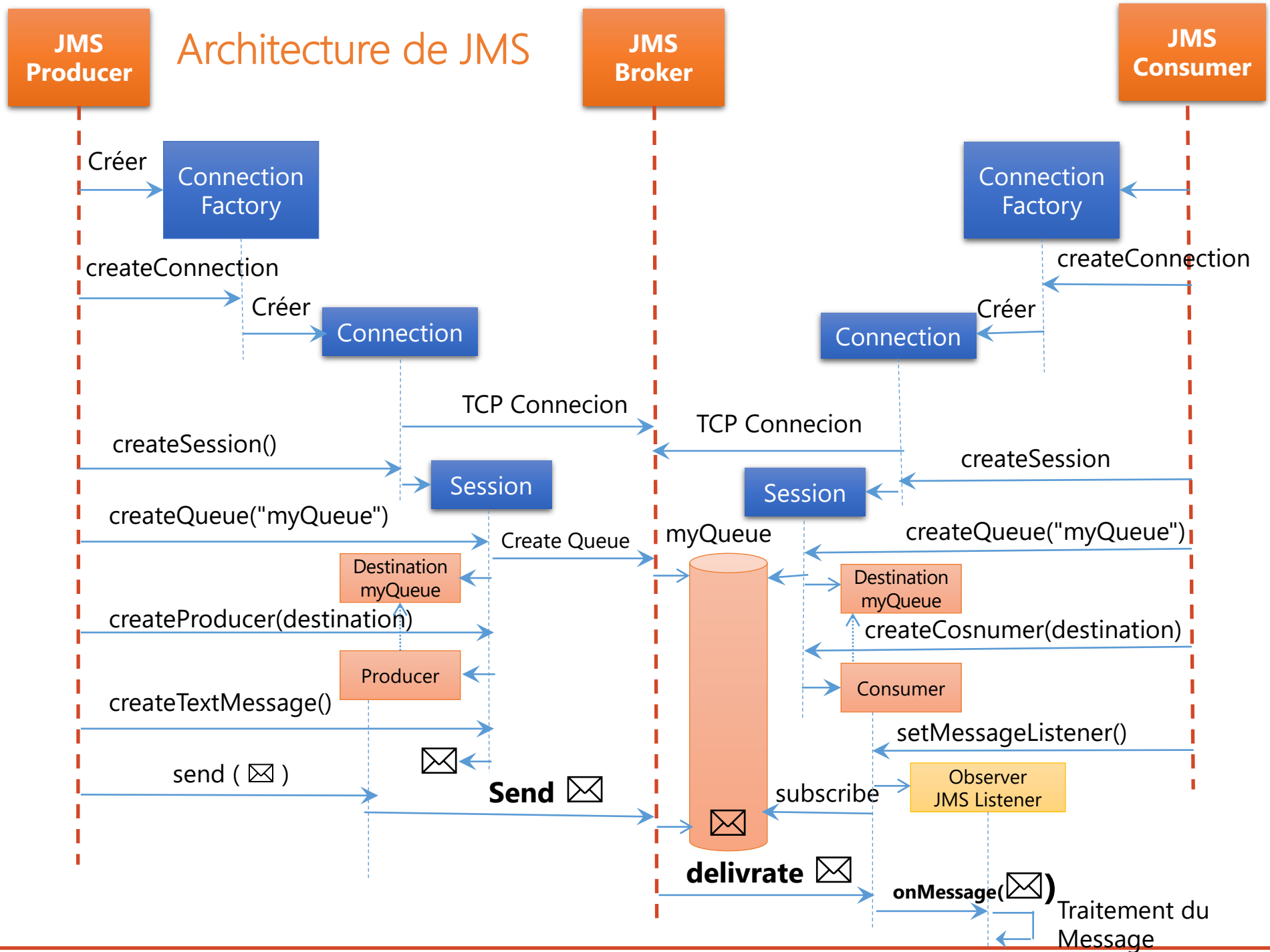
Modèle de systèmes distribués asynchrone



Le modèle de programmation JMS



Architecture de JMS



Exemple de Code pour un producer JMS

```
// Create a ConnectionFactory
 ConnectionFactory connectionFactory = new ActiveMQConnectionFactory("tcp://localhost:61616");
// Create a Connection
 Connection connection = connectionFactory.createConnection();
 connection.start();
 // Create a Session
 Session session = connection.createSession(false,Session.AUTO_ACKNOWLEDGE);
 // Create the destination (Topic or Queue)
Destination destination = session.createTopic("DTopic");
  //Destination destination = session.createQueue("DQueue");
// Create a MessageProducer from the Session to the Topic or Queue
MessageProducer producer = session.createProducer(destination);
producer.setDeliveryMode(DeliveryMode.NON_PERSISTENT);
 // Create a messages
   String text = "Hello world! From: ";
   TextMessage message = session.createTextMessage(text);
 // Tell the producer to send the message
   producer.send(message);
 // Clean up
 session.close();
 connection.close();
```

Protocoles Supportés par ActiveMQ :

- TCP
- AMQP
- STOMP
- MQTT

Différent Types de Messages JMS :

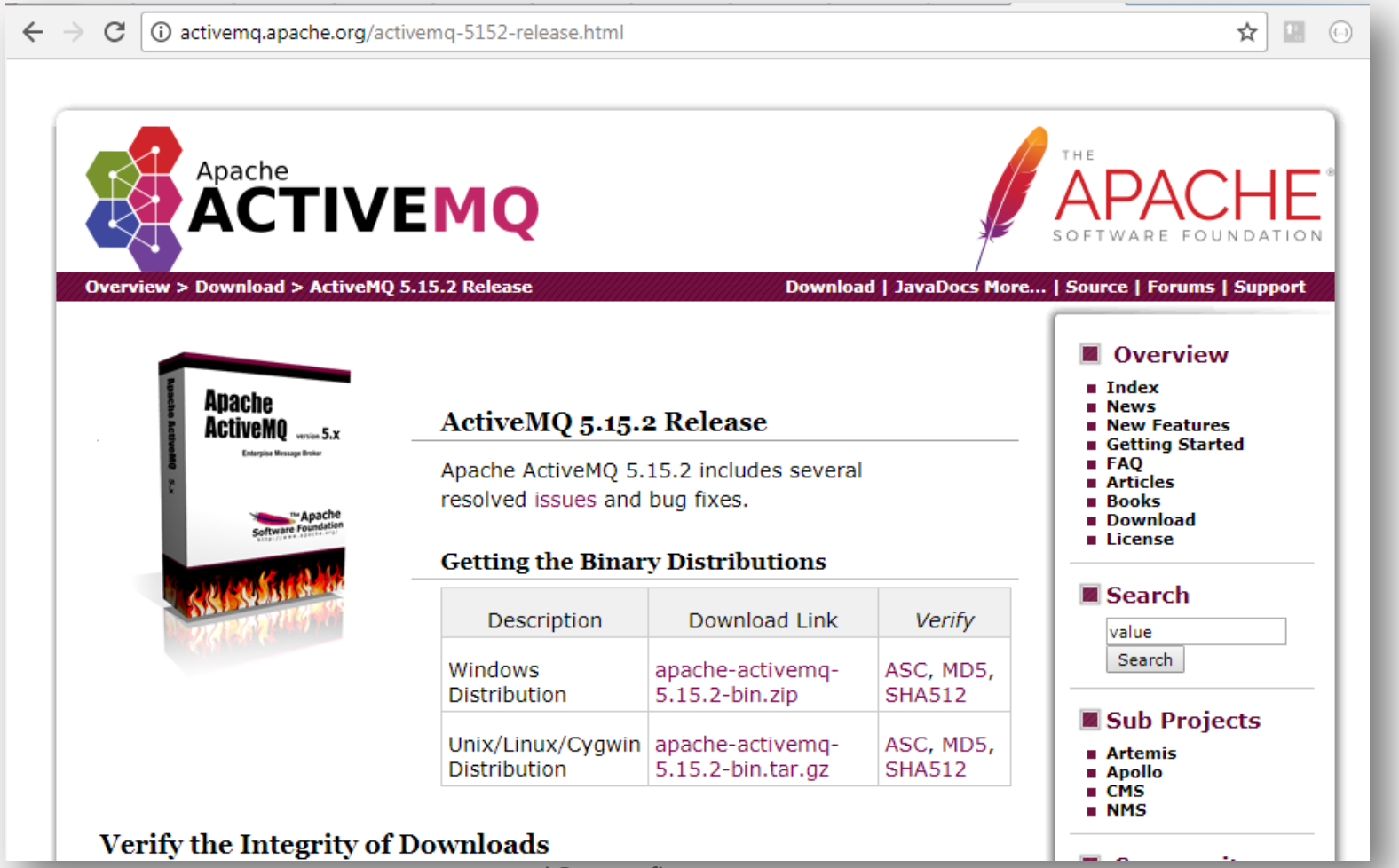
- TextMessage
- BytesMessage
- MapMessage
- ObjectMessage
- StreamMessage

Exemple de Code pour un consumer JMS

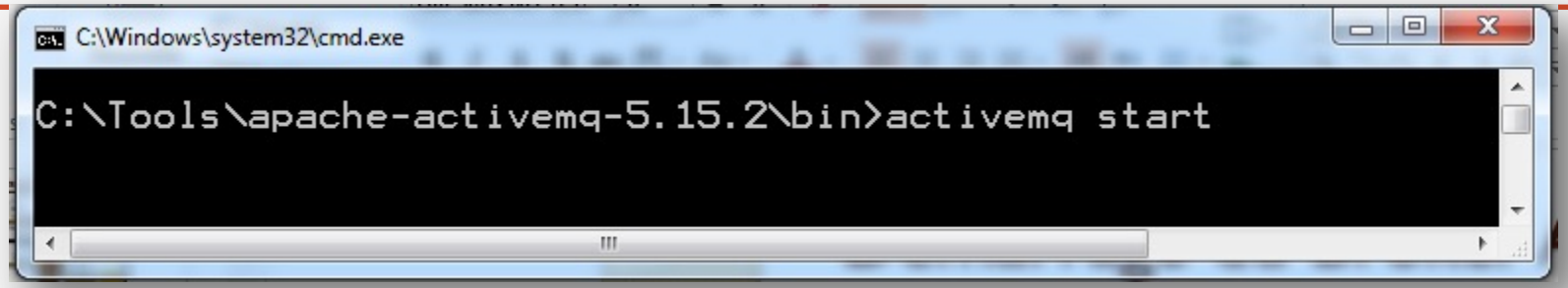
```
 // Create a ConnectionFactory
      ConnectionFactory connectionFactory = new ActiveMQConnectionFactory("tcp://192.168.43.8:61616");
// Create a Connection
      Connection connection = connectionFactory.createConnection();
// Create a Session
      Session session = connection.createSession(false,Session.AUTO_ACKNOWLEDGE);
// Create the destination (Topic or Queue)
     Destination destination = session.createTopic("DTopic");
 //Destination destination = session.createQueue("DQueue");
// Create a MessageConsumer from the Session to the Topic or Queue
      MessageConsumer consumer = session.createConsumer(destination);
// Start Connection
      connection.start();
// Create JMS Listener form messages
consumer.setMessageListener(new MessageListener() {
        @Override
        public void onMessage(Message message) {
          try {
                  if (message instanceof TextMessage) {
                     TextMessage textMessage = (TextMessage) message;
                     String text = textMessage.getText(); System.out.println("Received: " + text);
                  } else {
                     System.out.println("Received: " + message);
                  }
        } catch (JMSException e) { e.printStackTrace(); } }});
```

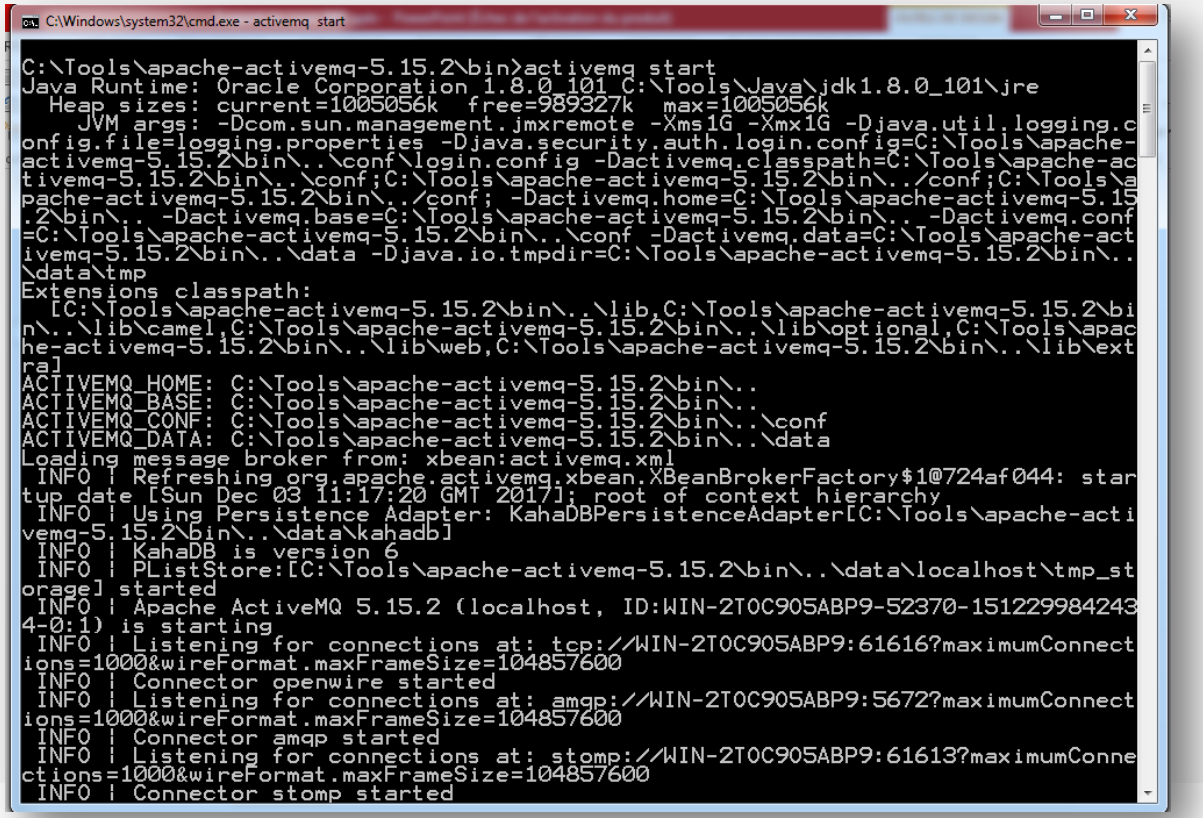
Broker ActiveMQ

Télécharger ActiveMQ : http://activemq.apache.org/activemq-5152-release.html

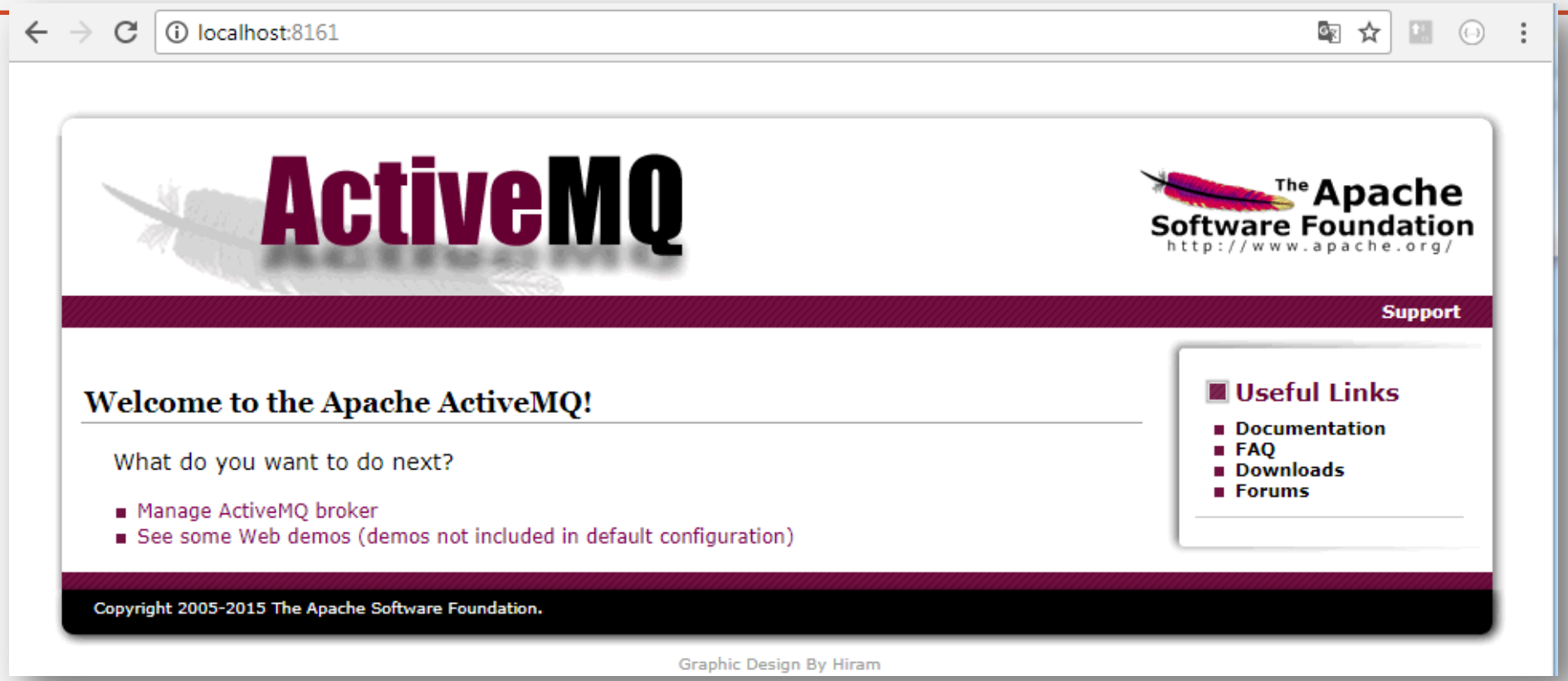


Démarrage du Broker Active MQ

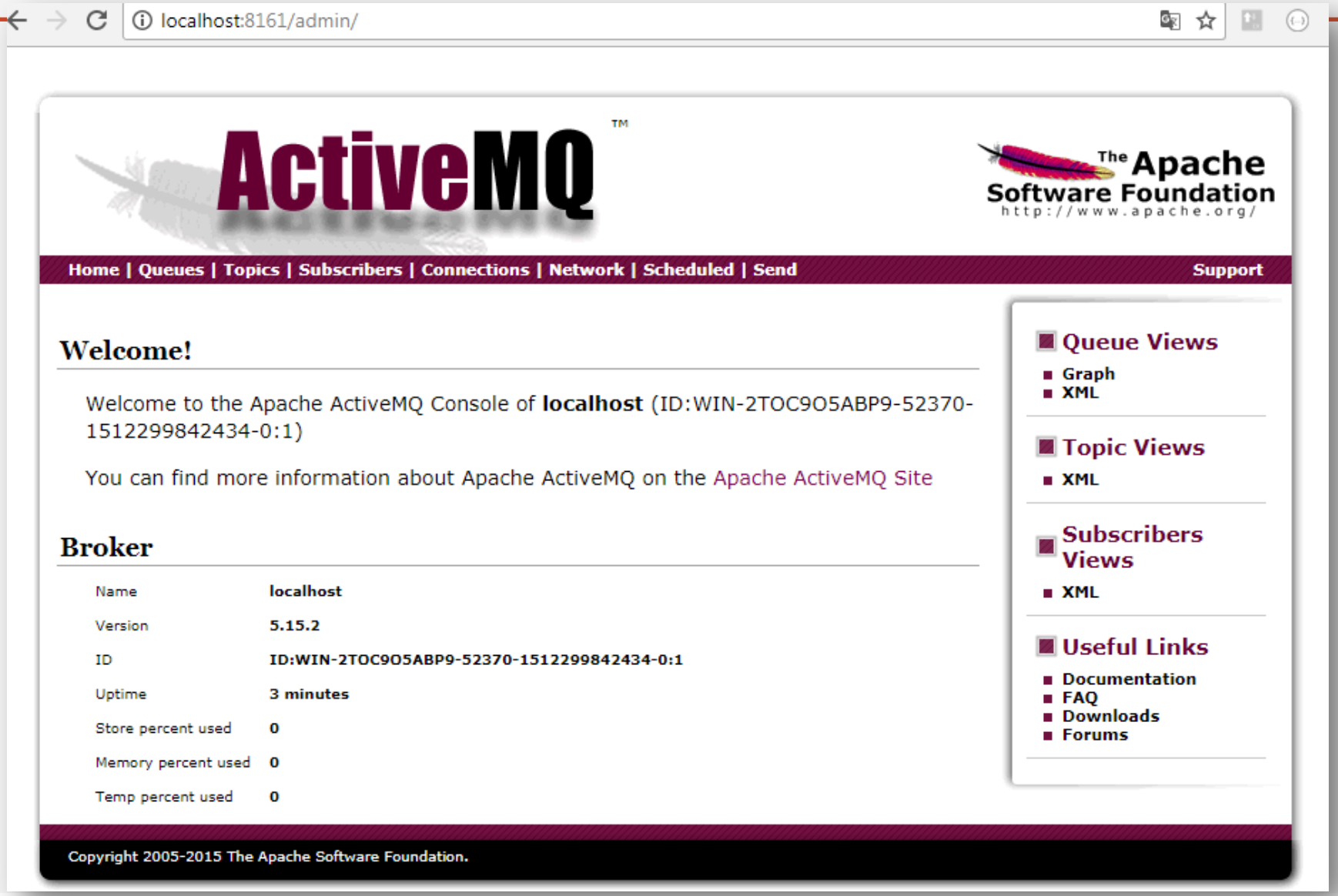




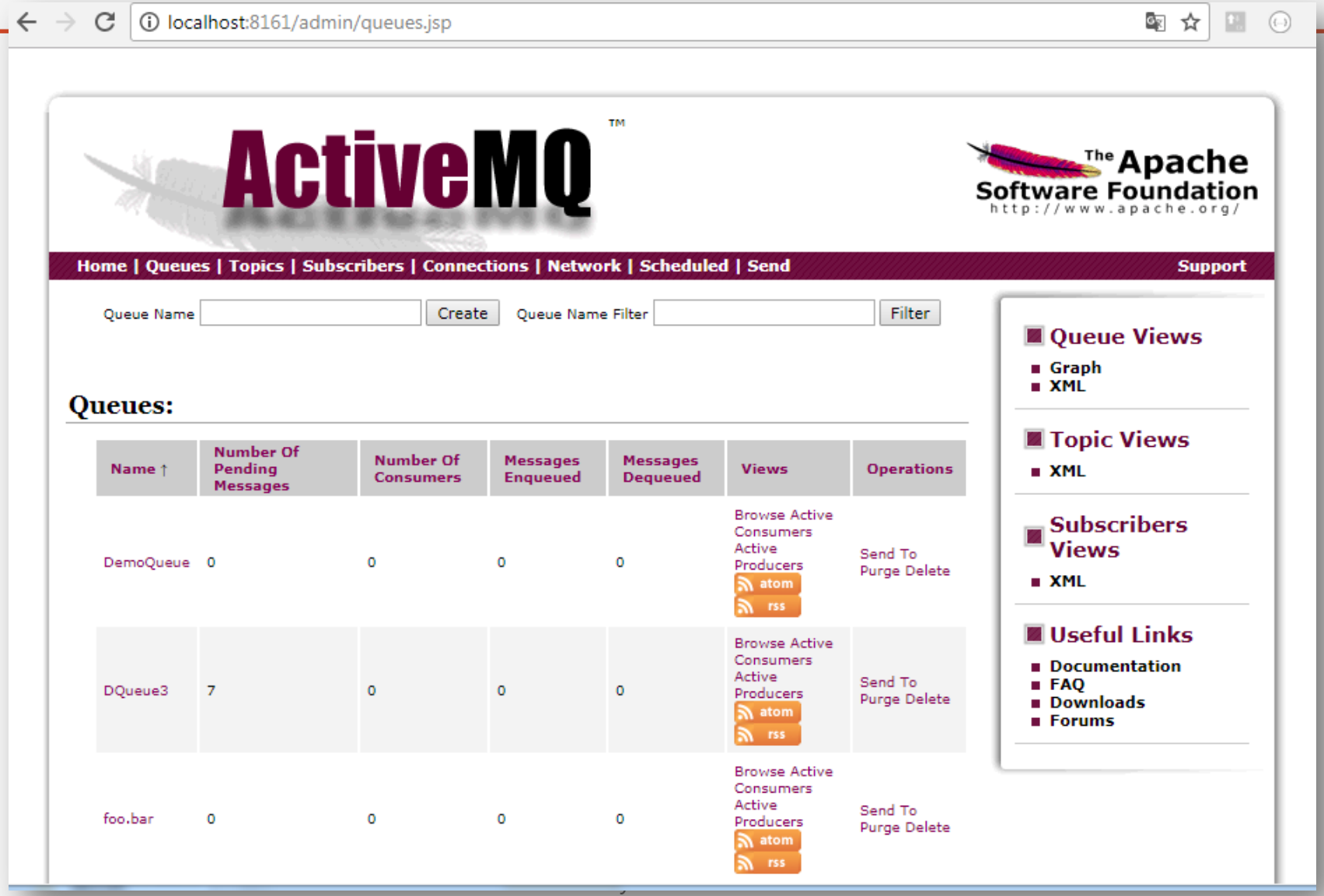
Console d'administration de ActiveMQ



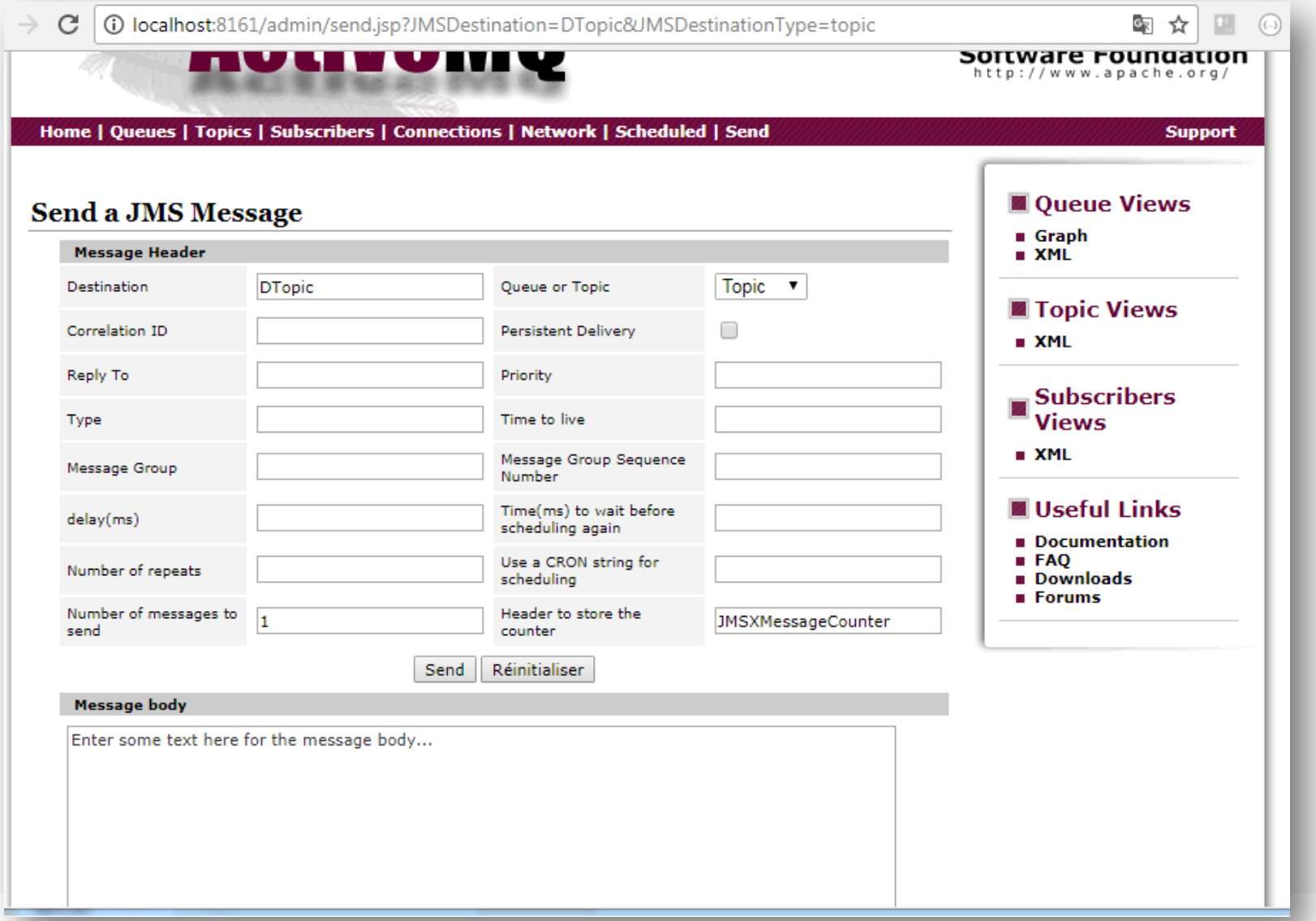
Console d'administration de ActiveMQ



Console d'administration de ActiveMQ



Console d'administration de ActiveMQ



Démarrer ActiveMQ à partir d'une application java : Embeded ActiveMQ

```java
package jms;
import org.apache.activemq.broker.BrokerService;
public class ActiveMQBroker {
public static void main(String[] args) {
try {
  BrokerService broker = new BrokerService();
  // configure the broker
  //broker.setPersistent(false);
  broker.addConnector("tcp://0.0.0.0:61616");
  broker.start();
} catch (Exception e) { e.printStackTrace(); }}}
```

```xml
<dependency>
    <groupId>org.apache.activemq</groupId>
    <artifactId>activemq-broker</artifactId>
    <version>5.15.2</version>
</dependency>
<dependency>
    <groupId>org.apache.activemq</groupId>
    <artifactId>activemq-kahadb-store</artifactId>
    <version>5.8.0</version>
    <scope>runtime</scope>
</dependency>
```

Listener JMS : Coté Application Spring

```
package org.sid.web;
import javax.jms.JMSException; import javax.jms.Message; import javax.jms.TextMessage;
import org.sid.Student; import org.sid.StudentRepository; import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jms.annotation.JmsListener; import org.springframework.stereotype.Component;
@Component
public class MyJMSListeners {
@Autowired
private StudentRepository studentRepository;
@JmsListener(destination="sample.queue")
public void receive(byte[] message) throws Exception {
        String strMessage=new String(message,"UTF-8");
        System.out.println(strMessage);
        ObjectMapper objectMapper=new ObjectMapper();
        Etudiant et=objectMapper.readValue(message, Etudiant.class);
        System.out.println("Nom:"+et.getNom());
        System.out.println("Prénom:"+et.getPrenom());
        etudiantRepositoy.save(et);
   }
}
```

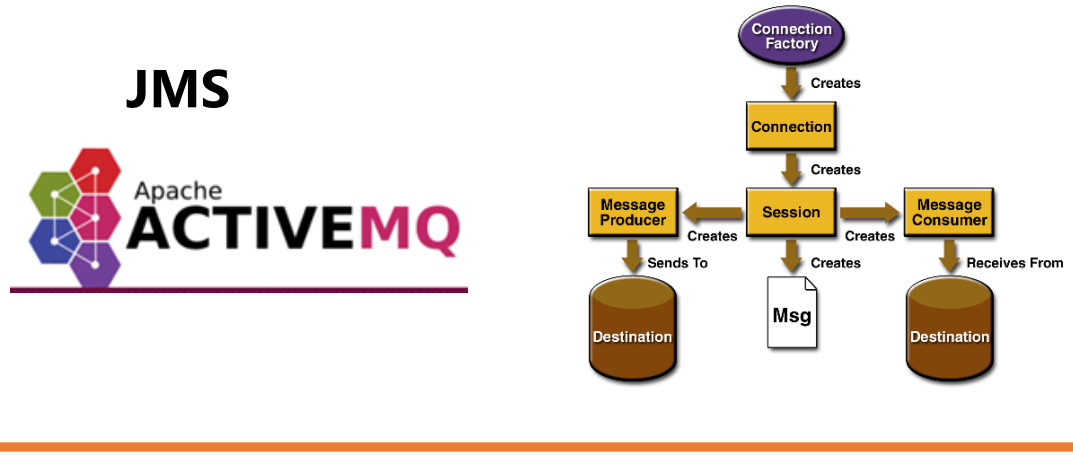
Envoie d'un message JMS : Spring

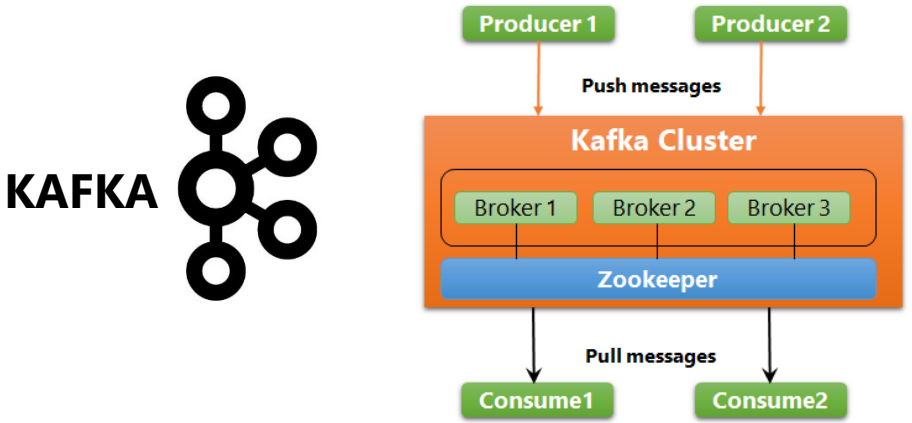
```java
@RestController
public class MyRestController {
  @Autowired
  private JmsTemplate jmsTemplate;
  @Autowired
  private Topic messagesTopic;
  @GetMapping("/sendMessage")
  public void send(@RequestParam(name="message")String message) {
        System.out.println(message+".............");
        jmsTemplate.convertAndSend(messagesTopic, message);
  }
}
```

Déclaration des files d'attentes : Spring

```java
@Configuration
public class MyConfig {
    @Bean
    public Queue queue() {
        return new ActiveMQQueue("scolarite.queue");
    }
    @Bean
    public Topic topic() {
        return new ActiveMQTopic("messages.topic");
    }
}
```

Event Drinen Architecture: JMS, KAFKA Brokers

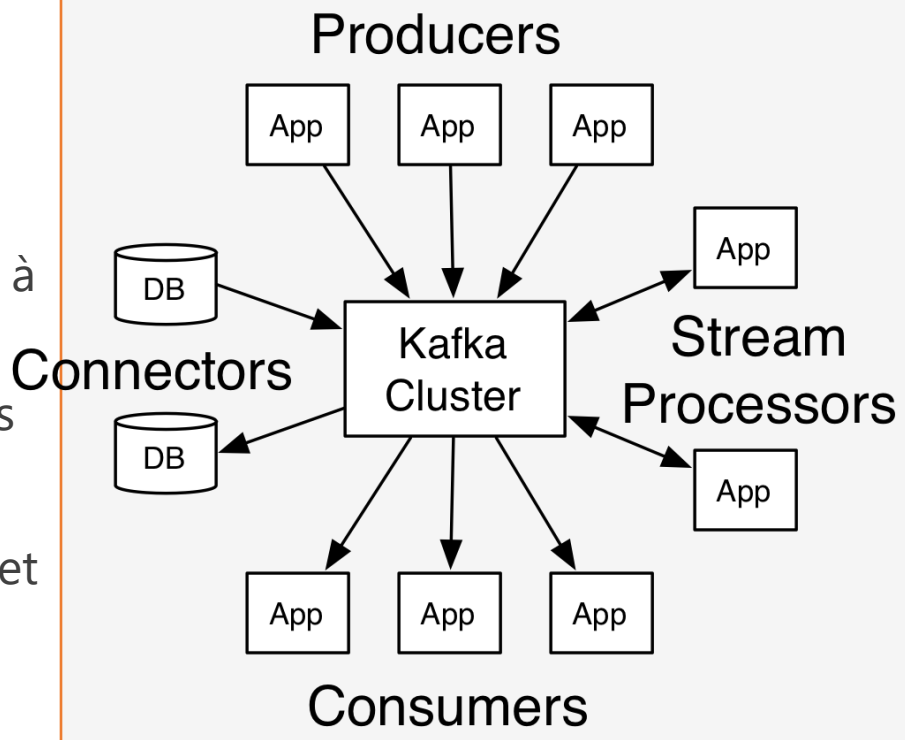


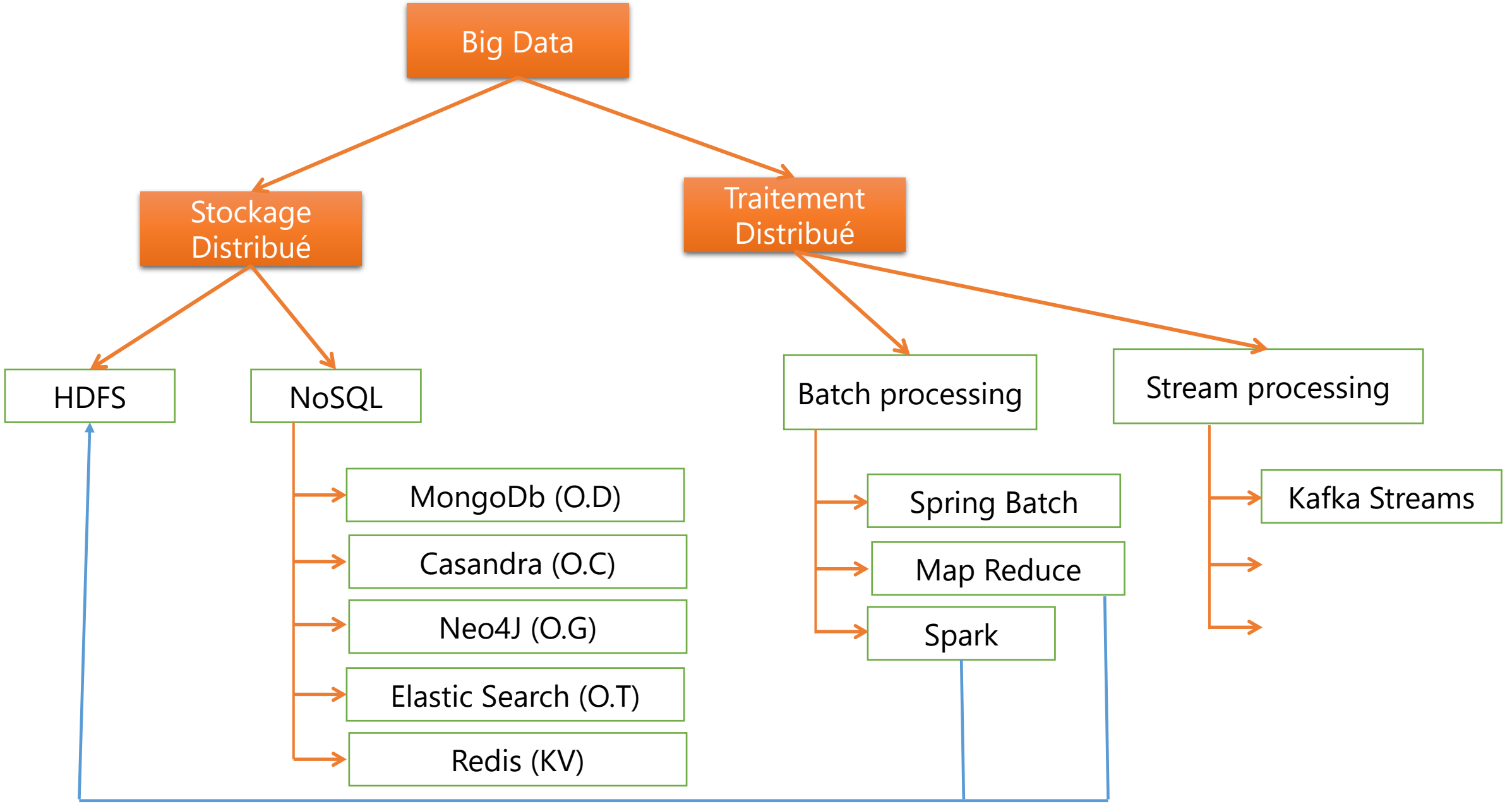


Mohamed Youssfi
Laboratoire Signaux Systèmes Distribués et Intelligence Artificielle (SSDIA)
ENSET, Université Hassan II Casablanca, Maroc
Email : med@youssfi.net
Supports de cours : http://fr.slideshare.net/mohamedyoussfi9
Chaîne vidéo : http://youtube.com/mohamedYoussfi
Recherche : http://www.researchgate.net/profile/Youssfi_Mohamed/publications

KAFKA

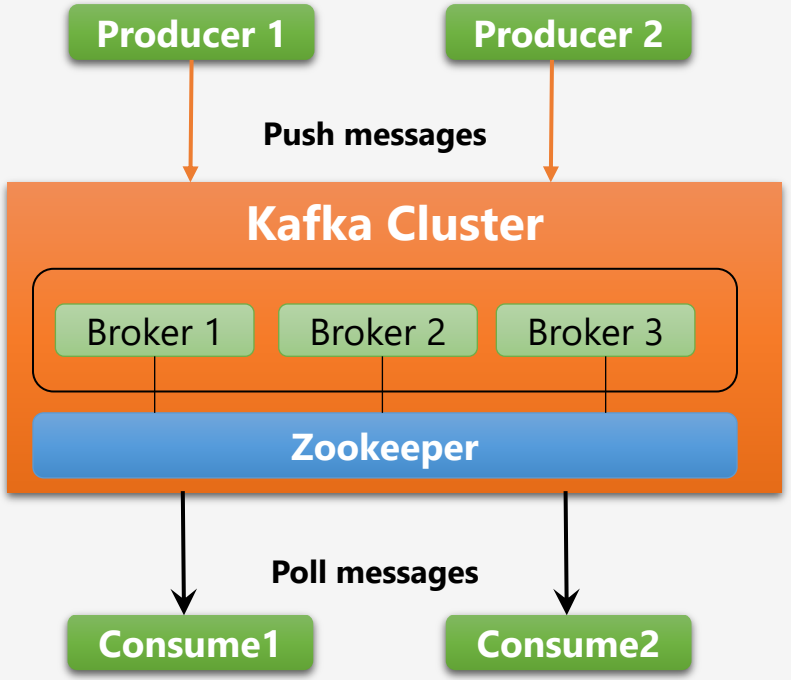
- Apache Kafka est une plate-forme de diffusion (streaming) distribuée développée en Scala et Java.
- Une plate-forme de streaming possède trois fonctionnalités clés:
  - Permettre aux applications clientes Kafka de publiez et s'abonner à des flux d'enregistrements. Similaires à une file d'attente de 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)




Big Data
Stockage Distribué
Traitement Distribué
HDFS
NoSQL
Batch processing
Stream processing
MongoDb (O.D)
Casandra (O.C)
Neo4J (O.G)
Elastic Search (O.T)
Redis (KV)
Spring Batch
Map Reduce
Spark
Kafka Streams

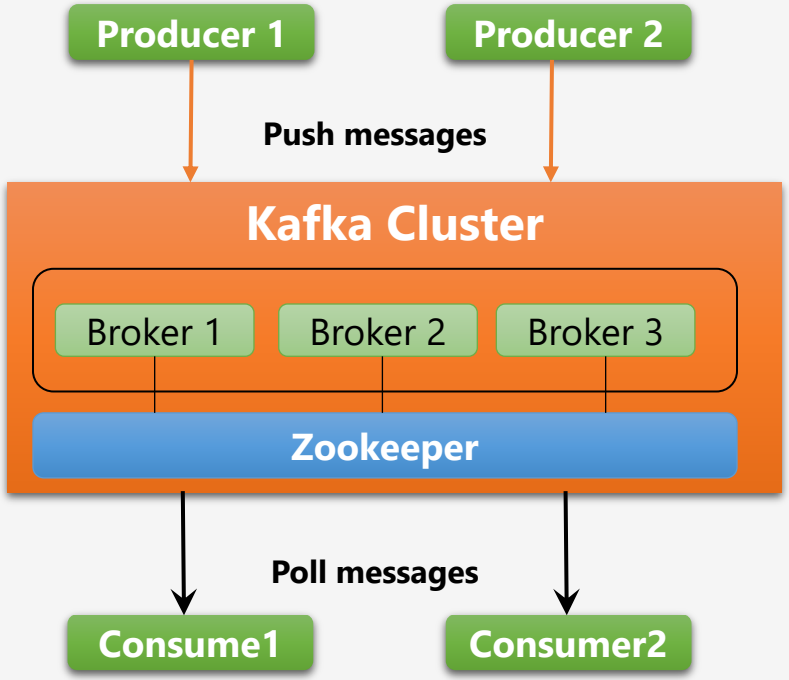
Les API de Kafka

- 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 fourni des API clientes pour différents langages : Java, C++, Node JS, .Net, PHP, Python, etc…



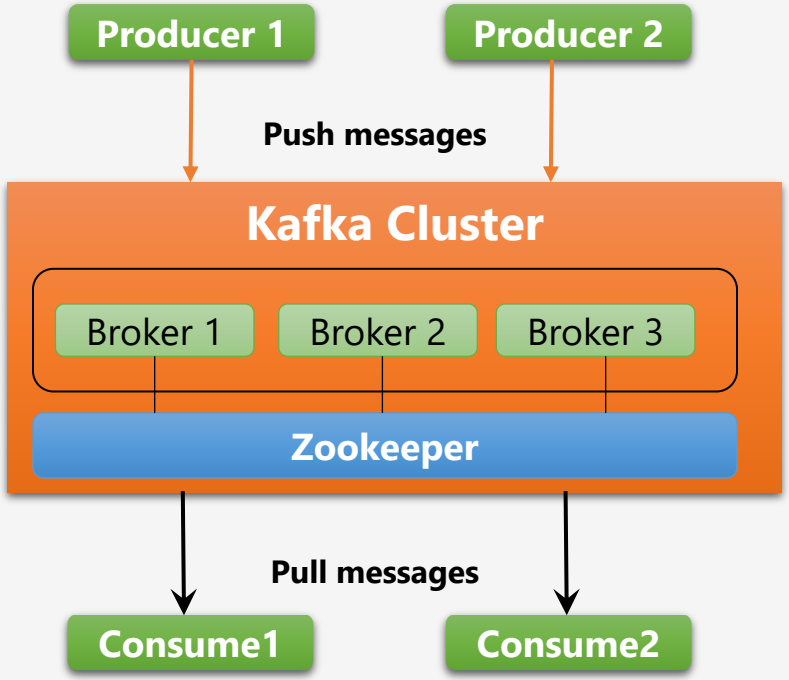
Architecture de KAFKA

- **Kafka Brokers**
  - Pour maintenir l'équilibre de la charge, le cluster Kafka est généralement composé de plusieurs Brokers dont un est élu comme Leader.
  - Cependant, ils sont sans état, c'est pourquoi ils utilisent **ZooKeeper** pour conserver l'état du cluster.
  - Une instance de Kafka Broker peut gérer des centaines de milliers de lectures et d'écritures par seconde.
  - Chaque Broker peut gérer des To de messages.



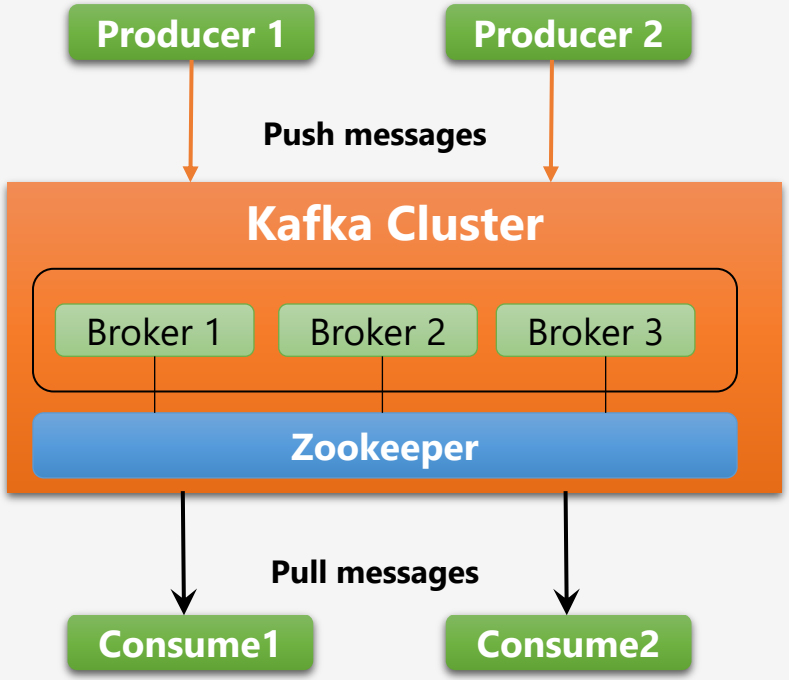
Architecture de KAFKA

- **ZooKeeper :**
  - Kafka broker utilise ZooKeeper pour la gestion et la coordination.
  - Il l'utilise également pour informer le producteur et le consommateur de la présence d'un nouveau broker dans le système Kafka ou d'une défaillance du broker dans le système Kafka.



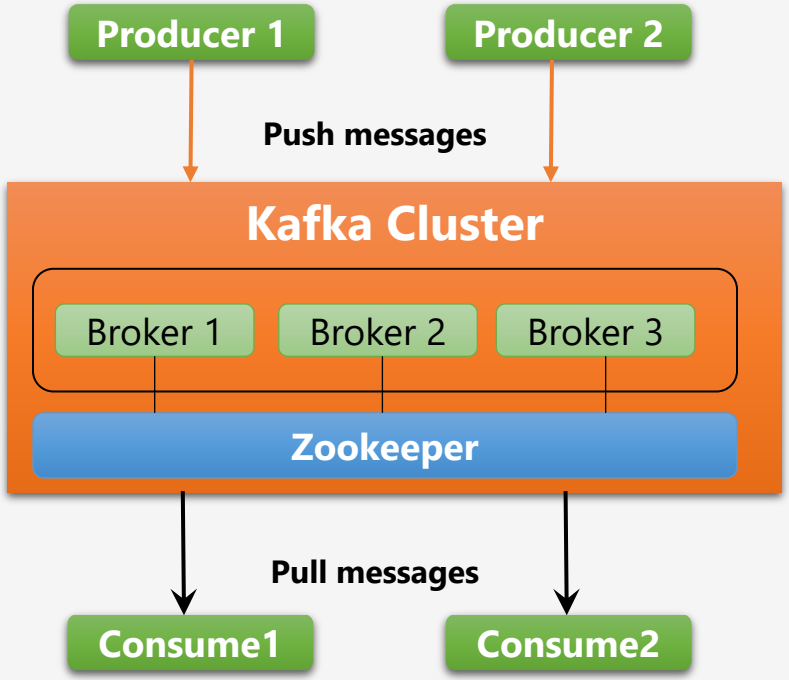
Architecture de KAFKA

- **Producers :**
  - les producteurs de Kafka transmettent les données aux Brokers.
  - Ces données sont à destination des Topics réparties en partitions dans les brokers du cluster
  - Le producteur est responsable du choix de l'enregistrement à affecter à quelle partition du cluster.



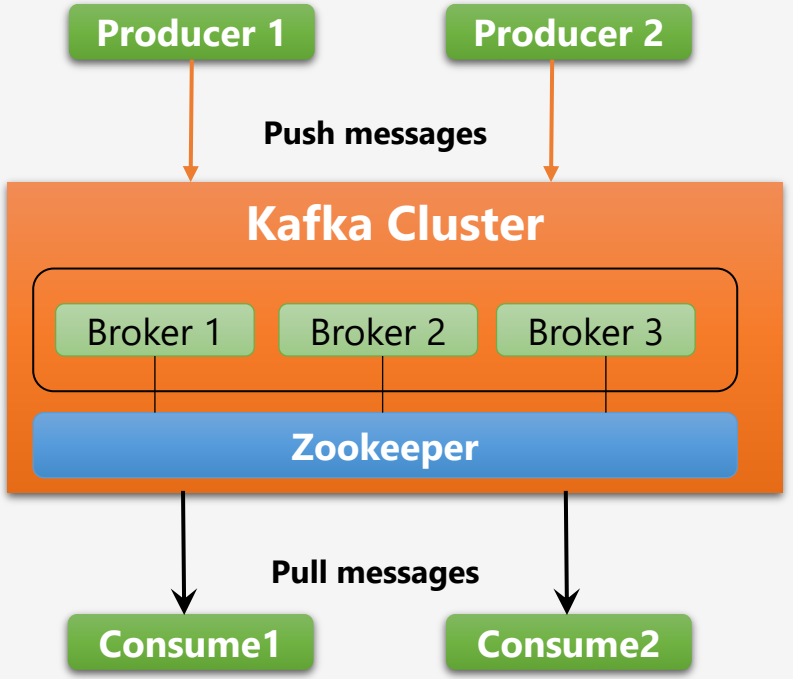
Architecture de KAFKA

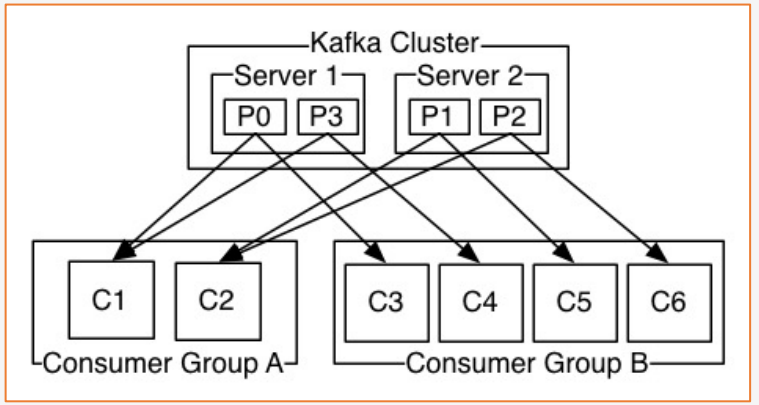
- **Consumers :**
  - Consomme les messages à partir des Topics.
  - En utilisant l'offset de la partitions de Kafka Broker, le consommateur Kafka retient combien de messages ont été consommés parce que les Brokers Kafka sont sans état.
  - Les utilisateurs peuvent rembobiner ou passer à n'importe quel point d'une partition.
  - Chaque enregistrement publié dans un Topic est remis à une instance de consommateur au sein de chaque groupe de consommateurs abonné.



Architecture de KAFKA

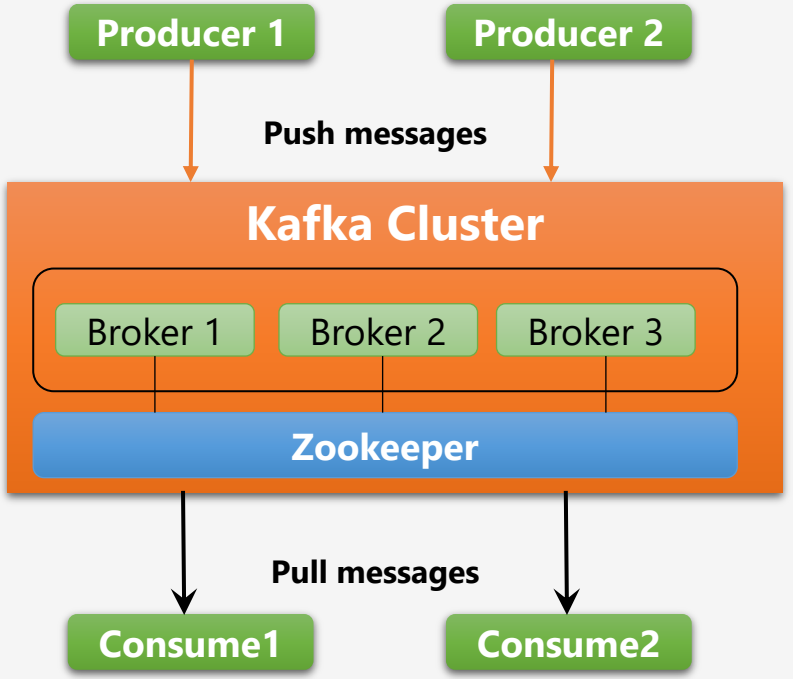
- **Consumers :**
  - Les instances de consommateur peuvent être dans des processus séparés ou sur des machines séparées.
  - Si toutes les instances de consommateur ont le même groupe de consommateurs, la charge des enregistrements sera effectivement équilibrée sur les instances de consommateur.
  - Si toutes les instances de consommateur ont des groupes de consommateurs différents, chaque enregistrement est alors diffusé vers tous les processus de consommation.

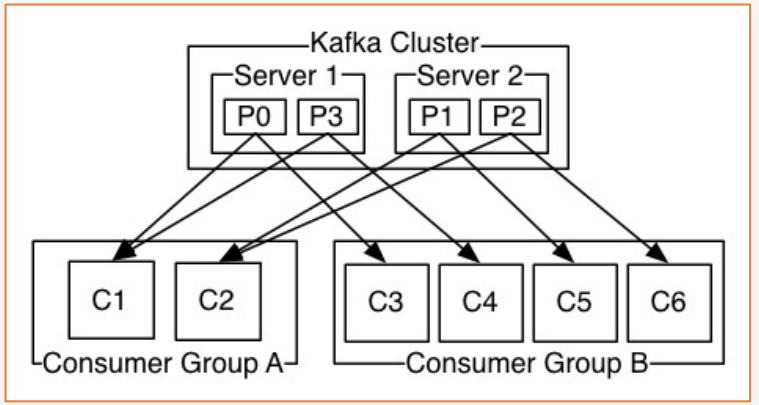




Architecture de KAFKA

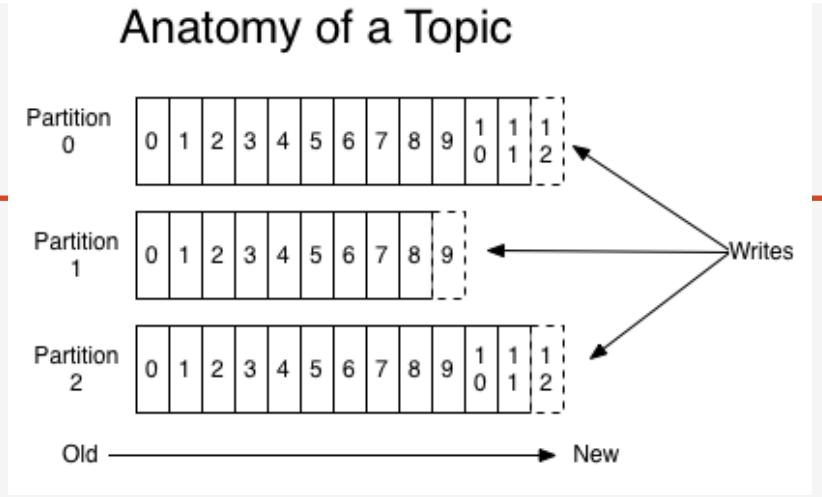
- **Consumer-Group :**
  - Les consommateurs peuvent être organisés en groupes.
  - Des partitions de Topics sont assignées pour équilibrer les assignations entre tous les consommateurs du groupe.
  - Chaque message sera vu par un consommateur du groupe.
  - Si un consommateur s'en va, la partition est assignée à un autre consommateur du groupe.
  - S'il y a plus de consommateurs dans un groupe que de partitions, certains consommateurs resteront inactifs.
  - S'il y a moins de consommateurs dans un groupe que de partitions, certains consommateurs consommeront des messages provenant de plus d'une partition.

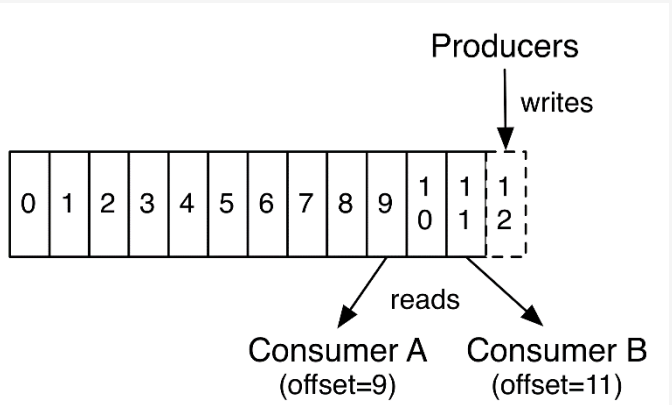


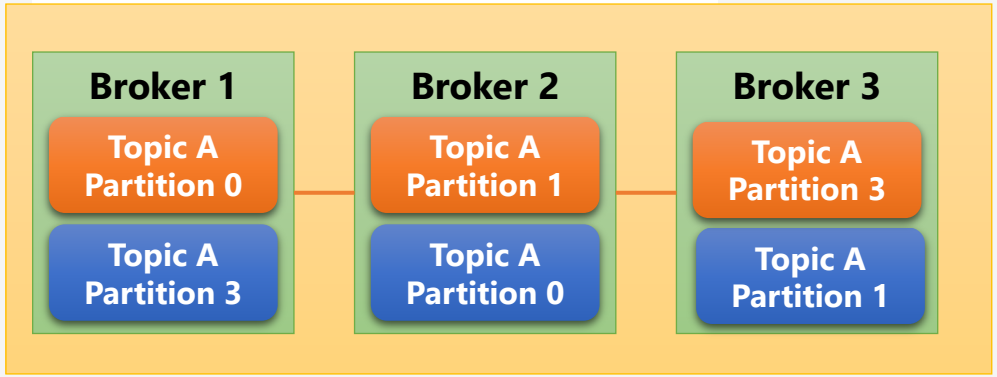


Topics

- Un Topic est une sorte de boite à lettre vers laquelle les messages (enregistrements) sont publiés
- Chaque Topic peut avoir 0 à plusieurs abonnés consommateurs
- Pour chaque topic, le cluster Kafka maintient un journal (Log) partitionné.
- Chaque partition est une séquence d'enregistrements ordonnée et immutable
- Un numéro d'identification séquentiel, appelé offset, est attribué aux enregistrements des partitions. Il identifie de manière unique chaque enregistrement de la partition.
- Le cluster Kafka conserve durablement tous les enregistrements publiés, qu'ils aient été consommés ou non, en utilisant une période de conservation configurable.
- Les performances de Kafka sont en réalité constantes en ce qui concerne la taille des données, de sorte que leur stockage pendant longtemps ne pose pas de problème.

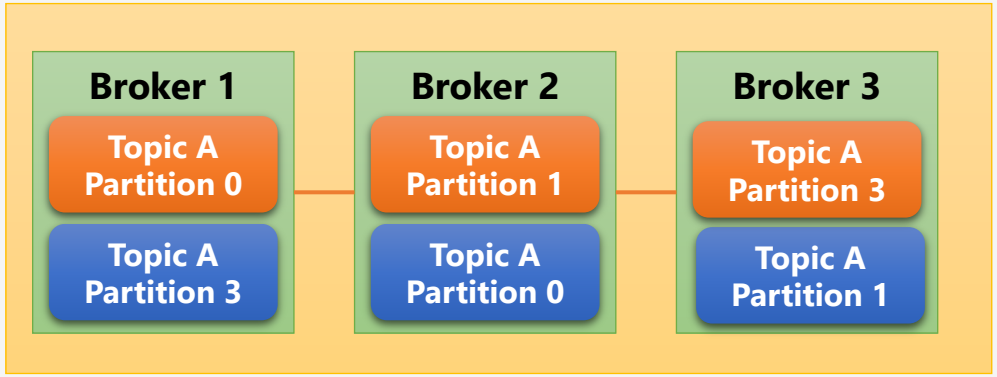






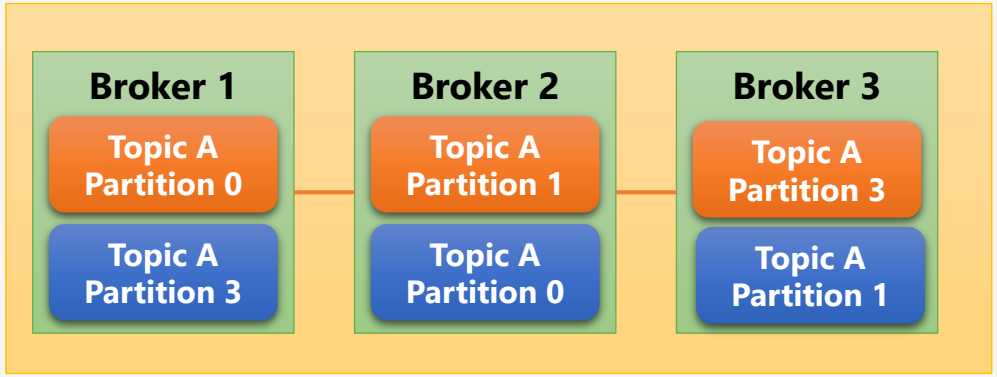
Distribution

- Les partitions du log sont distribuées sur les serveurs du cluster Kafka
- Chaque partition est répliquée sur un nombre configurable de serveurs pour la tolérance aux pannes.
- Chaque partition a un serveur qui joue le rôle de "leader" et zéro ou plusieurs serveurs qui servent de « Followers ».
- Le leader gère toutes les demandes de lecture et d'écriture pour la partition, tandis que les Followers répliquent passivement le leader.
- Si le leader échoue, l'un des Followers deviendra automatiquement le nouveau leader.
- Chaque serveur joue le rôle Leader pour certaines de ses partitions et Follower pour d'autres, de sorte que la charge soit bien équilibrée au sein du cluster.



Geo-Replication

- Kafka MirrorMaker fournit une prise en charge de la réplication géographique pour vos clusters.
- Avec MirrorMaker, les messages sont répliqués dans plusieurs Data centers ou régions de cloud.
- Vous pouvez l'utiliser dans des scénarios
  - Actifs / passifs pour la sauvegarde et la récupération.
  - ou dans des scénarios actifs / actifs pour rapprocher les données de vos utilisateurs ou pour répondre aux exigences de localisation des données.



Installation de Kafka

- Téléchargement et décompression

```
$ wget http://miroir.univ-lorraine.fr/apache/kafka/2.3.0/kafka_2.12-2.3.0.tgz

$ tar -xzf kafka_2.12-2.3.0.tgz
```

Démarrage de KAFKA Server

- Start zookeeper

```
$ cd kafka_2.12-2.3.0
$ bin/zookeeper-server-start.sh config/zookeeper.properties

[2019-09-08 11:43:36,135] INFO binding to port 0.0.0.0/0.0.0.0:2181
(org.apache.zookeeper.server.NIOServerCnxnFactory)
```

Démarrage de KAFKA Server

- Start kafka server

```
$ cd kafka_2.12-2.3.0
$ bin/kafka-server-start.sh config/server.properties
...
[2019-09-08 11:44:54,652] INFO Initiating client connection, connectString=localhost:2181 sessionTimeout=6000
watcher=kafka.zookeeper.ZooKeeperClient$ZooKeeperClientWatcher$@48ae9b55 (org.apache.zookeeper.ZooKeeper)
[2019-09-08 11:44:54,692] INFO [ZooKeeperClient Kafka server] Waiting until connected.
(kafka.zookeeper.ZooKeeperClient)
[2019-09-08 11:44:54,694] INFO Opening socket connection to server localhost/127.0.0.1:2181. Will not attempt to
authenticate using SASL (unknown error) (org.apache.zookeeper.ClientCnxn)
[2019-09-08 11:44:54,722] INFO Socket connection established to localhost/127.0.0.1:2181, initiating session
(org.apache.zookeeper.ClientCnxn)
[2019-09-08 11:44:54,805] INFO Session establishment complete on server localhost/127.0.0.1:2181, sessionid =
0x100000219690000, negotiated timeout = 6000 (org.apache.zookeeper.ClientCnxn)
[2019-09-08 11:44:54,811] INFO [ZooKeeperClient Kafka server] Connected. (kafka.zookeeper.ZooKeeperClient)
...
[2019-09-08 11:44:56,489] INFO Awaiting socket connections on 0.0.0.0:9092. (kafka.network.Acceptor)
...
[2019-09-08 11:44:57,477] INFO [KafkaServer id=0] started (kafka.server.KafkaServer)
```

Création d'un Topic

- Création d'un Topic

```
$ bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1
--partitions 1 --topic test
Created topic test.
$ bin/kafka-topics.sh --list --zookeeper localhost 2181
test
```

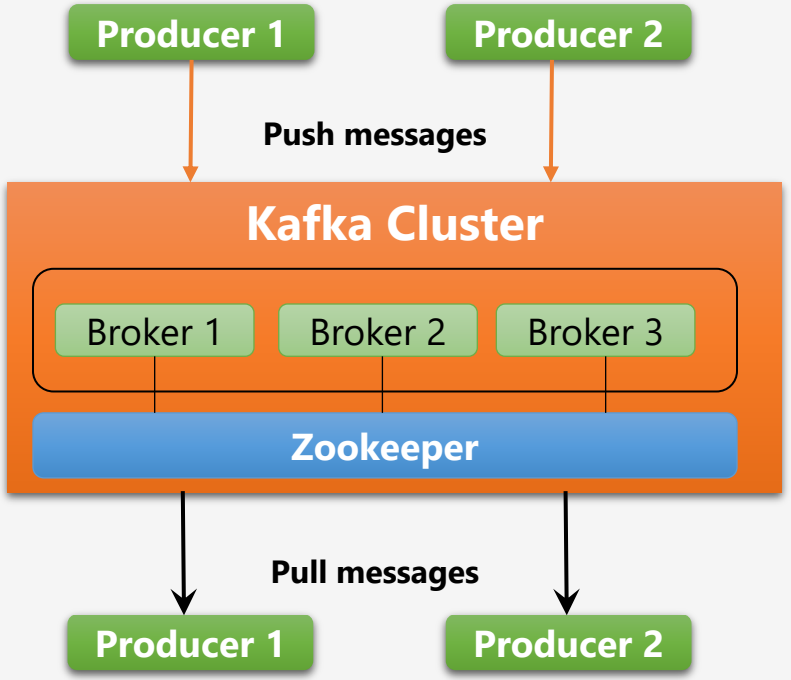
S'abonner à un Topic pour consommer des messages

- S'abonner au topic test pour consommer les messages

```
$ bin/kafka-console-consumer.sh --bootstrap-server
localhost:9092 --topic test --from-beginning
    test
    azerty
    me
    you
    and me
```

- Produire des messages vers le topic

```
$ bin/kafka-console-producer.sh --broker-list
localhost:9092 --topic test
    >test
    >azerty
    >me
    >you
    >and me
    >
```



Produire des messages vers le topic test

```
$ bin/kafka-console-producer.sh --broker-list localhost:9092 --topic test
>test
>azerty
>me
>you
>and me
>
```

Démarrage d'un Cluster

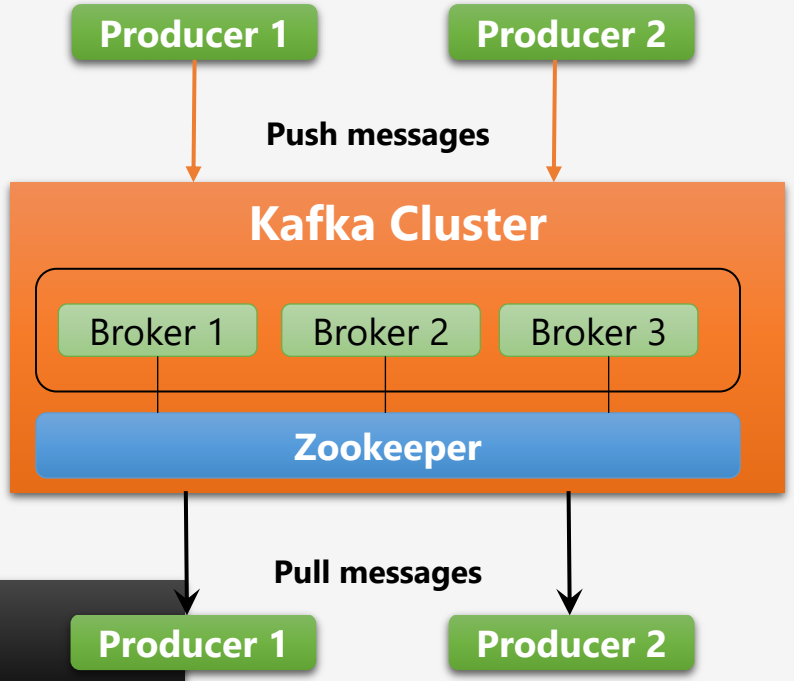
```
$ cp config/server.properties config/server-1.properties
$ cp config/server.properties config/server-2.properties
```

config/server-1.properties:

```
broker.id=1
listeners=PLAINTEXT://:9093
log.dirs=/tmp/kafka-logs-1
```

config/server-2.properties:

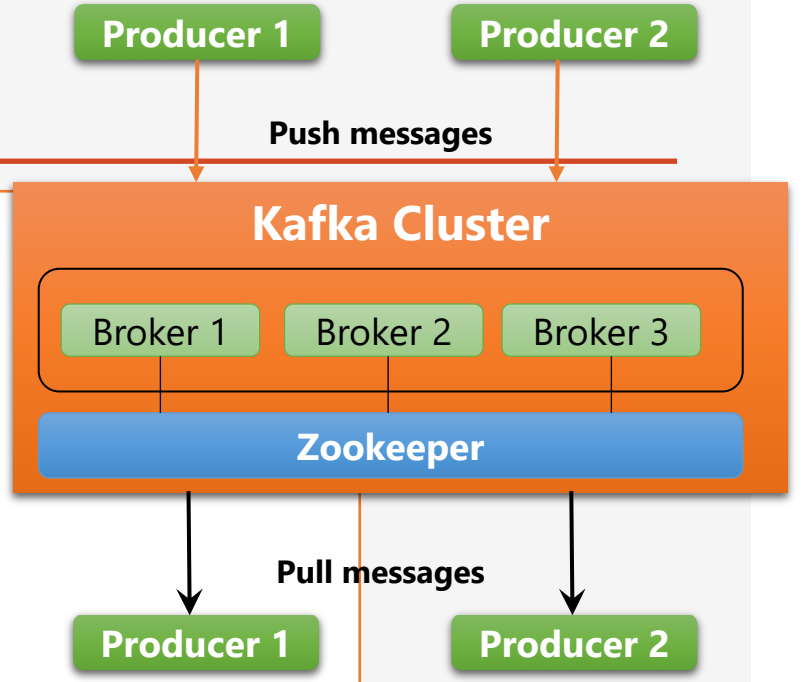
```
broker.id=2
listeners=PLAINTEXT://:9094
log.dirs=/tmp/kafka-logs-2
```




```
$ bin/kafka-server-start.sh config/server-1.properties
...
$ bin/kafka-server-start.sh config/server-2.properties
...
```

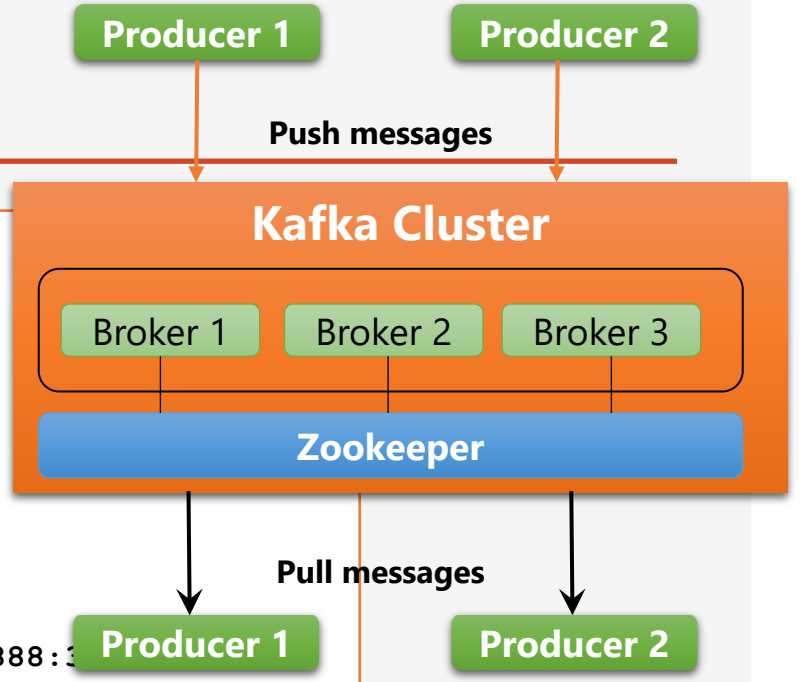
Démarrage d'un Cluster avec Docker-compose

```
version: '2'
services:
  zookeeper-1:
    image: confluentinc/cp-zookeeper:latest
    hostname: zookeeper-1
    ports:
      - "12181:12181"
    environment:
      ZOOKEEPER_SERVER_ID: 1
      ZOOKEEPER_CLIENT_PORT: 12181
      ZOOKEEPER_TICK_TIME: 2000
      ZOOKEEPER_INIT_LIMIT: 5
      ZOOKEEPER_SYNC_LIMIT: 2
      ZOOKEEPER_SERVERS: zookeeper-1:12888:13888;zookeeper-2:22888:23888;zookeeper-3:32888:33888

  zookeeper-2:
    image: confluentinc/cp-zookeeper:latest
    hostname: zookeeper-2
    ports:
      - "22181:22181"
    environment:
      ZOOKEEPER_SERVER_ID: 2
      ZOOKEEPER_CLIENT_PORT: 22181
      ZOOKEEPER_TICK_TIME: 2000
      ZOOKEEPER_INIT_LIMIT: 5
      ZOOKEEPER_SYNC_LIMIT: 2
      ZOOKEEPER_SERVERS: zookeeper-1:12888:13888;zookeeper-2:22888:23888;zookeeper-3:32888:33888
```



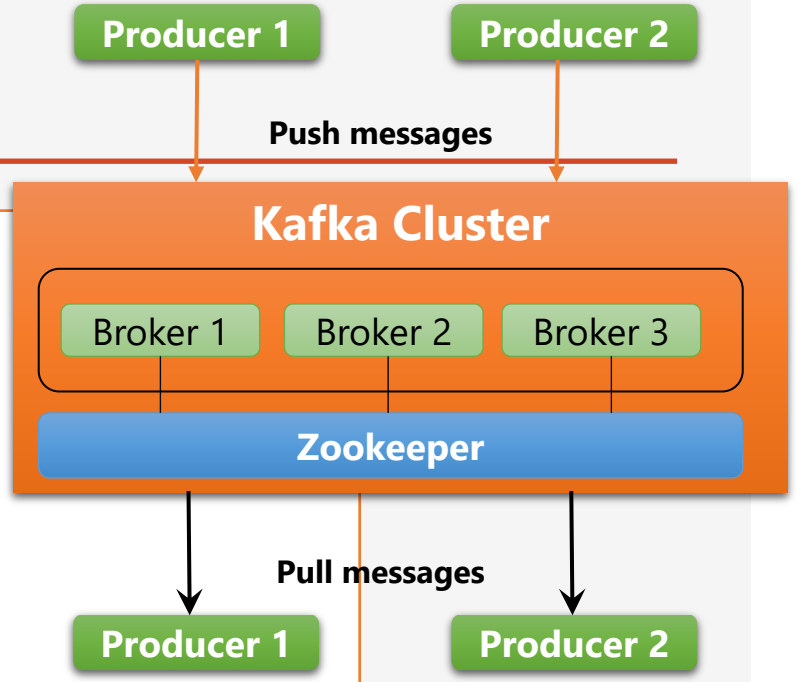
Démarrage d'un Cluster avec Docker-compose

```
zookeeper-3:
    image: confluentinc/cp-zookeeper:latest
    hostname: zookeeper-3
    ports:
      - "32181:32181"
    environment:
      ZOOKEEPER_SERVER_ID: 3
      ZOOKEEPER_CLIENT_PORT: 32181
      ZOOKEEPER_TICK_TIME: 2000
      ZOOKEEPER_INIT_LIMIT: 5
      ZOOKEEPER_SYNC_LIMIT: 2
      ZOOKEEPER_SERVERS: zookeeper-1:12888:13888;zookeeper-2:22888:23888;zookeeper-3:32888:3

  kafka-1:
    image: confluentinc/cp-kafka:latest
    hostname: kafka-1
    ports:
      - "19092:19092"
    depends_on:
      - zookeeper-1
      - zookeeper-2
      - zookeeper-3
    environment:
      KAFKA_BROKER_ID: 1
      KAFKA_ZOOKEEPER_CONNECT: zookeeper-1:12181,zookeeper-2:12181,zookeeper-3:12181
      KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://kafka-1:19092
```



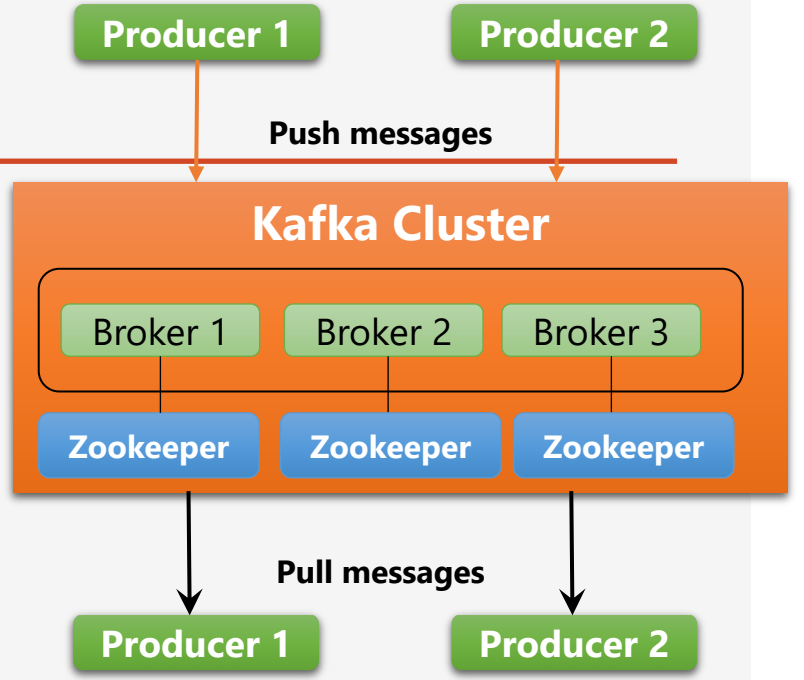
Démarrage d'un Cluster avec Docker-compose

```
kafka-2:
    image: confluentinc/cp-kafka:latest
    hostname: kafka-2
    ports:
      - "29092:29092"
    depends_on:
      - zookeeper-1
      - zookeeper-2
      - zookeeper-3
    environment:
      KAFKA_BROKER_ID: 2
      KAFKA_ZOOKEEPER_CONNECT: zookeeper-1:12181,zookeeper-2:12181,zookeeper-3:12181
      KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://kafka-2:29092

  kafka-3:
    image: confluentinc/cp-kafka:latest
    hostname: kafka-3
    ports:
      - "39092:39092"
    depends_on:
      - zookeeper-1
      - zookeeper-2
      - zookeeper-3
    environment:
      KAFKA_BROKER_ID: 3
      KAFKA_ZOOKEEPER_CONNECT: zookeeper-1:12181,zookeeper-2:12181,zookeeper-3:12181
      KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://kafka-3:39092
```



Démarrage d'un Cluster avec Docker-compose

```
C:\tpdockcompose\kafka_cluster>dir
 Le volume dans le lecteur C s'appelle OS
 Le numéro de série du volume est CE70-C12F

 Répertoire de C:\tpdockcompose\kafka_cluster

15/10/2019  11:38    <DIR>          .
15/10/2019  11:38    <DIR>          ..
15/10/2019  11:38             2.381 docker-compose.yml
               1 fichier(s)              2.381 octets
               2 Rép(s)  36.303.675.392 octets libres

C:\tpdockcompose\kafka_cluster>docker-compose up
Creating network "kafka_cluster_default" with the default driver
Creating kafka_cluster_zookeeper-1_1 ... done
Creating kafka_cluster_zookeeper-3_1 ... done
Creating kafka_cluster_zookeeper-2_1 ... done
Creating kafka_cluster_kafka-1_1     ... done
Creating kafka_cluster_kafka-2_1     ... done
Creating kafka_cluster_kafka-3_1     ...
```



Tester la tolérance aux panes

- Now let's test out fault-tolerance. Broker 1 was acting as the leader so let's kill it:

```
$ ps aux | grep server-1.properties
youssfi   8179 13.1 16.2 3633872 331472 pts/3  Sl+  16:48   0:06 java -Xmx1G -Xms1G -server -
XX:+UseG1GC -XX:MaxGCPauseMillis=20 -XX:InitiatingHeapOccupancyPercent=35 -
XX:+ExplicitGCInvokesConcurrent -Djava.a>
$ kill -9 8179
On Windows use:
> wmic process where "caption = 'java.exe' and commandline like '%server-1.properties%'" get processid
ProcessId
6016
> taskkill /pid 6016 /f
```

- Leadership has switched to one of the followers and node 1 is no longer in the in-sync replica set, But the messages are still available for consumption even though the leader

```
$ bin/kafka-topics.sh --describe --bootstrap-server localhost:9092 --topic my-replicated-topic
Topic:my-replicated-topic   PartitionCount:1    ReplicationFactor:3 Configs:
    Topic: my-replicated-topic  Partition: 0    Leader: 2   Replicas: 1,2,0 Isr: 2,0
```

Exemple d'application Java : Producer/Consumer

```
<dependencies>
    <!-- https://mvnrepository.com/artifact/org.apache.kafka/kafka-clients -->
    <dependency>
        <groupId>org.apache.kafka</groupId>
        <artifactId>kafka-clients</artifactId>
        <version>2.3.0</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/org.apache.kafka/kafka -->
    <dependency>
        <groupId>org.apache.kafka</groupId>
        <artifactId>kafka_2.12</artifactId>
        <version>2.3.0</version>
    </dependency>
    <dependency>
        <groupId>org.apache.kafka</groupId>
        <artifactId>kafka-streams</artifactId>
        <version>2.3.0</version>
    </dependency>

</dependencies>
```

Exemple Consumer Java

```
import org.apache.kafka.clients.consumer.ConsumerConfig; import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.KafkaConsumer; import java.time.Duration; import java.util.Collections;
import java.util.Properties; import java.util.concurrent.Executors;import java.util.concurrent.TimeUnit;
public class StreamConsumer {
    private String KAFKA_BROKER_URL="192.168.43.22:9092";    private String TOPIC_NAME="testTopic";
    public static void main(String[] args) {
        new StreamConsumer();
    }
    public StreamConsumer() {
        Properties properties=new Properties();
        properties.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG,KAFKA_BROKER_URL);
        properties.put(ConsumerConfig.GROUP_ID_CONFIG,"sample-group-test");
        properties.put(ConsumerConfig.ENABLE_AUTO_COMMIT_CONFIG,"true");
        properties.put(ConsumerConfig.AUTO_COMMIT_INTERVAL_MS_CONFIG,"1000");
        properties.put(ConsumerConfig.SESSION_TIMEOUT_MS_CONFIG,"30000");
        properties.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, "org.apache.kafka.common.serialization.IntegerDeserializer");
        properties.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, "org.apache.kafka.common.serialization.StringDeserializer");

        KafkaConsumer<Integer, String> kafkaConsumer = new KafkaConsumer<Integer, String>(properties);
        kafkaConsumer.subscribe(Collections.singletonList(TOPIC_NAME));
        Executors.newScheduledThreadPool(1).scheduleAtFixedRate(()->{
            System.out.println("----------------------------------------");
            ConsumerRecords<Integer,String> consumerRecords=kafkaConsumer.poll(Duration.ofMillis(10));
            consumerRecords.forEach(cr->{
                System.out.println("Key=>"+cr.key()+", Value=>"+cr.value()+", offset=>"+cr.offset());
            });
        },1000,1000, TimeUnit.MILLISECONDS);

    }
}
```

Exemple Consumer Java

```
Console App :
Key=>null, Value=>A, offset=>9
Key=>null, Value=>B, offset=>10
Key=>null, Value=>C, offset=>11
Key=>null, Value=>X, offset=>12
Key=>null, Value=>B, offset=>13
Key=>null, Value=>X, offset=>14
Key=>null, Value=>P, offset=>15
```

```
$ bin/kafka-console-producer.sh --broker-list localhost:9092 --
topic testTopic
>A
>B
>C
>X
>B
>X
>P
```

Exemple Producer Kafka Java

```
import org.apache.kafka.clients.producer.KafkaProducer; import org.apache.kafka.clients.producer.ProducerRecord;
import java.util.Properties; import java.util.concurrent.Executors; import java.util.concurrent.TimeUnit;
public class StreamProducer {
    private int counter;
     private String KAFKA_BROKER_URL="192.168.43.22:9092";
     private String TOPIC_NAME="testTopic";    private String clientID="client_prod_1";
     public static void main(String[] args) {
         new StreamProducer();
     }
     public StreamProducer() {
         Properties properties = new Properties();
         properties.put("bootstrap.servers", KAFKA_BROKER_URL);
         properties.put("client.id", clientID);
         properties.put("key.serializer", "org.apache.kafka.common.serialization.IntegerSerializer");
         properties.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");
         KafkaProducer<Integer, String> producer = new KafkaProducer<>(properties);
         Executors.newScheduledThreadPool(1).scheduleAtFixedRate(()->{
             ++counter;String msg=String.valueOf(Math.random()*1000);
             producer.send(new ProducerRecord<Integer, String>(TOPIC_NAME,++counter,msg),
                     (metadata,ex)->{
                         System.out.println("Sending Message key=>"+counter+" Value =>"+msg);
                         System.out.println("Partition => "+metadata.partition()+" Offset=>"+metadata.offset());
                     });
         },1000,1000, TimeUnit.MILLISECONDS);
     }
}
```

Exemple Producer Kafka Java

Consumer

```
Key=>32, Value=>609.6765398053385, offset=>3191
Key=>34, Value=>767.7805809027903, offset=>3192
Key=>36, Value=>787.1553570236956, offset=>3193
Key=>38, Value=>445.8611979566287, offset=>3194
Key=>40, Value=>497.64168259174227, offset=>3195
Key=>42, Value=>757.0605546528924, offset=>3196
Key=>44, Value=>547.2387037943823, offset=>3197
Key=>46, Value=>805.6218738236041, offset=>3198
Key=>48, Value=>199.4860895170022, offset=>3199
...
```

Producer en mode asynchrone

```
Sending Message key=>180 Value =>573.491814792023
Partition => 0 Offset=>3265
Sending Message key=>182 Value =>470.02504583700664
Partition => 0 Offset=>3266
Sending Message key=>184 Value =>988.7152347439281
Partition => 0 Offset=>3267
...
```

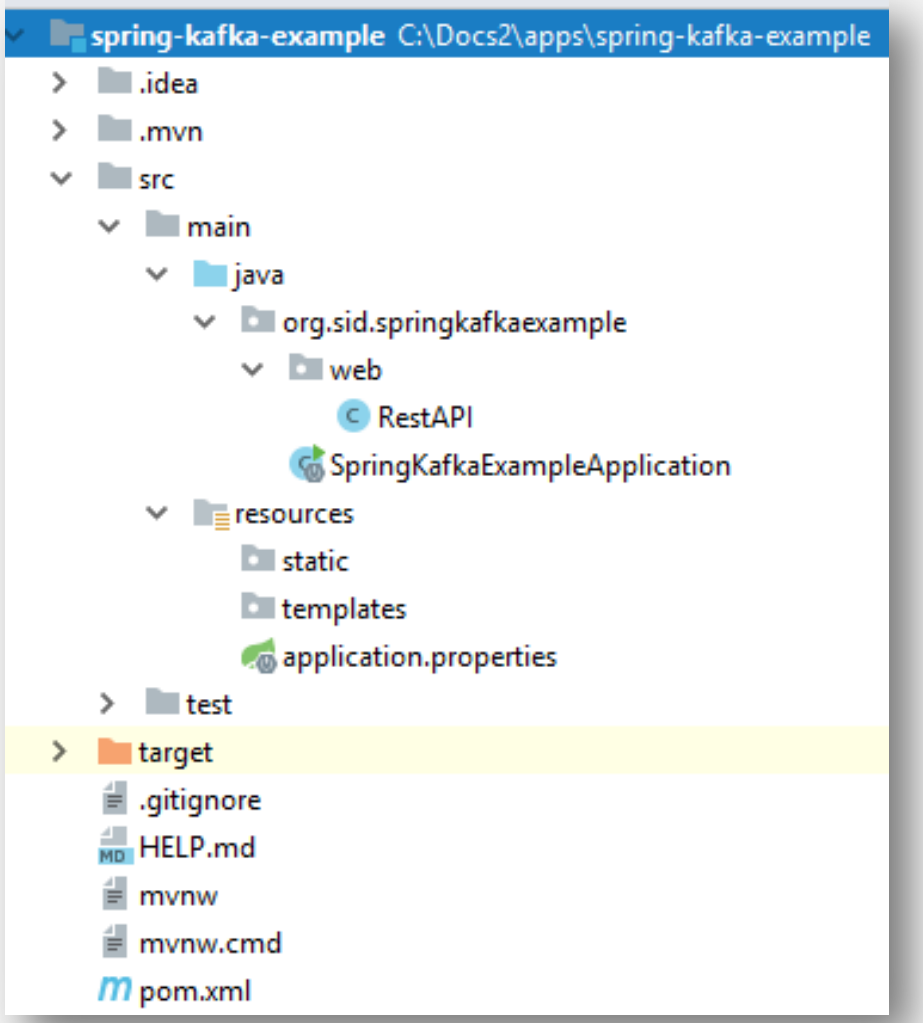
Application Web : Spring et Kafka

Maven 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>
```

appliaction.properties

```
server.port=8083
spring.kafka.producer.bootstrap-servers=192.168.57.3:9092
```

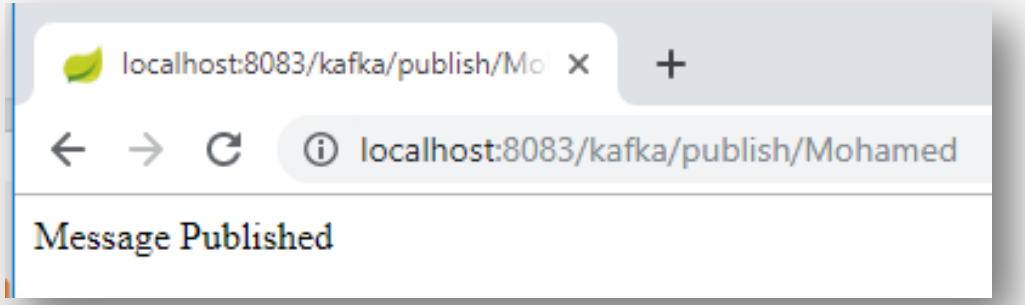


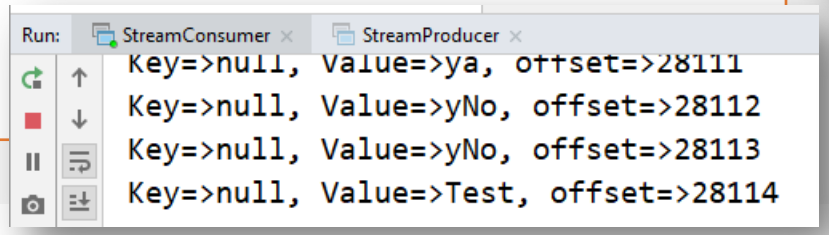
Application Web : Spring et Kafka (Producer, Consumer)

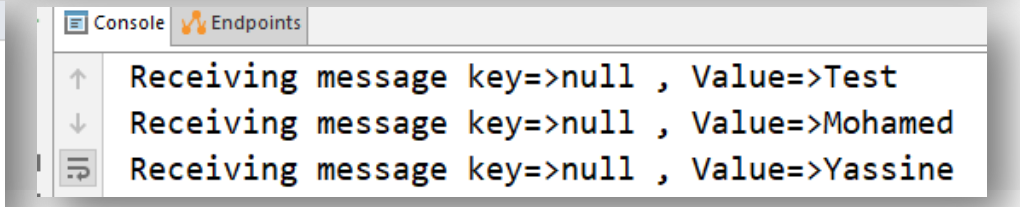
Maven dependencies

```
package org.sid.springkafkaexample.web;
import org.springframework.beans.factory.annotation.Autowired
import org.springframework.kafka.core.KafkaTemplate;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping
import org.springframework.web.bind.annotation.RestController

@RestController
@RequestMapping("kafka")
public class RestAPI {
    @Autowired
    private KafkaTemplate<Integer, String>
kafkaTemplate;
    private String topic="testTopic";
    @GetMapping("/publish/{message}")
    public String publishMessage(@PathVariable String
message){
        kafkaTemplate.send(topic,message);
        return "Message Published";
    }
}
```

```
package org.sid.demokafka;
import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.springframework.kafka.annotation.KafkaListener;
import org.springframework.stereotype.Service;
@Service
public class KafkaConsumer {
    @KafkaListener(topics = "testTopic",groupId = "sample_consumer")
    public void onMessage(ConsumerRecord message){
        System.out.println("Receiving message key=>"+message.key()+"" +
                " , Value=>"+message.value());
    }
}
```







Application Web : Spring et Kafka

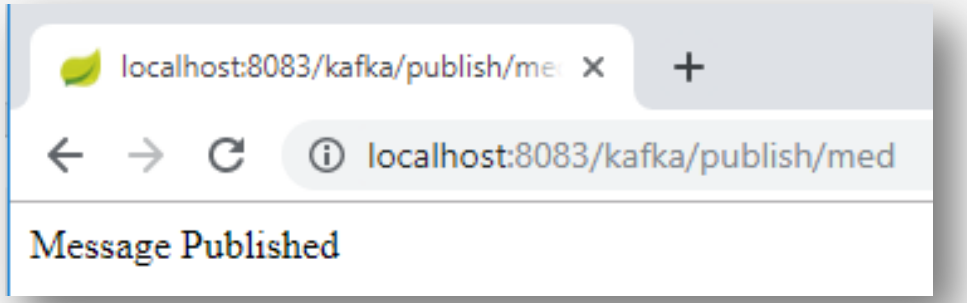
Maven dependencies

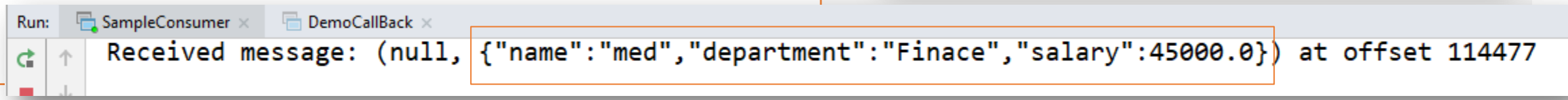
```
package org.sid.springkafkaexample.web;
import org.sid.springkafkaexample.model.Employee;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.kafka.core.KafkaTemplate;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("kafka")
public class RestAPI {
     @Autowired
    private KafkaTemplate<Integer, Employee> kafkaTemplate;
    private String topic="testTopic";
    @GetMapping("/publish/{name}")
    public String publishMessage(@PathVariable String name){
        kafkaTemplate.send(topic,new
Employee(name,"Finace",45000.0));
        return "Message Published";
    }
}
```

```
package org.sid.springkafkaexample.model;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.ToString;

@Data @NoArgsConstructor @AllArgsConstructor
@ToString
public class Employee {
    private String name;
    private String department;
    private double salary;
}
```

appliaction.properties

```
server.port=8083
spring.kafka.producer.bootstrap-servers=192.168.57.3:9092
spring.kafka.producer.key-serializer=org.apache.kafka.common.serialization.StringSerializer
spring.kafka.producer.value-serializer=org.springframework.kafka.support.serializer.JsonSerializer
```





Application Web : Spring et Kafka

En Utilisant une classe de configuration au lieu de application.properties

```
package org.sid.springkafkaexample.config;
import org.apache.kafka.clients.producer.ProducerConfig;
import org.apache.kafka.common.serialization.StringSerializer;
import org.sid.springkafkaexample.model.Employee; 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, Employee> producerFactory(){
        Map<String,Object> config=new HashMap<>();
        config.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,"192.168.57.3:9092");
        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,Employee> kafkaTemplate(){
        return new KafkaTemplate<>(producerFactory());
    }
}
```

appliaction.properties

```
server.port=8083
#spring.kafka.producer.bootstrap-servers=192.168.57.3:9092
#spring.kafka.producer.key-serializer=org.apache.kafka.common.serialization.StringSerializer
#spring.kafka.producer.value-serializer=org.springframework.kafka.support.serializer.JsonSerializer
```

Application Web : Spring Kafka Consumer

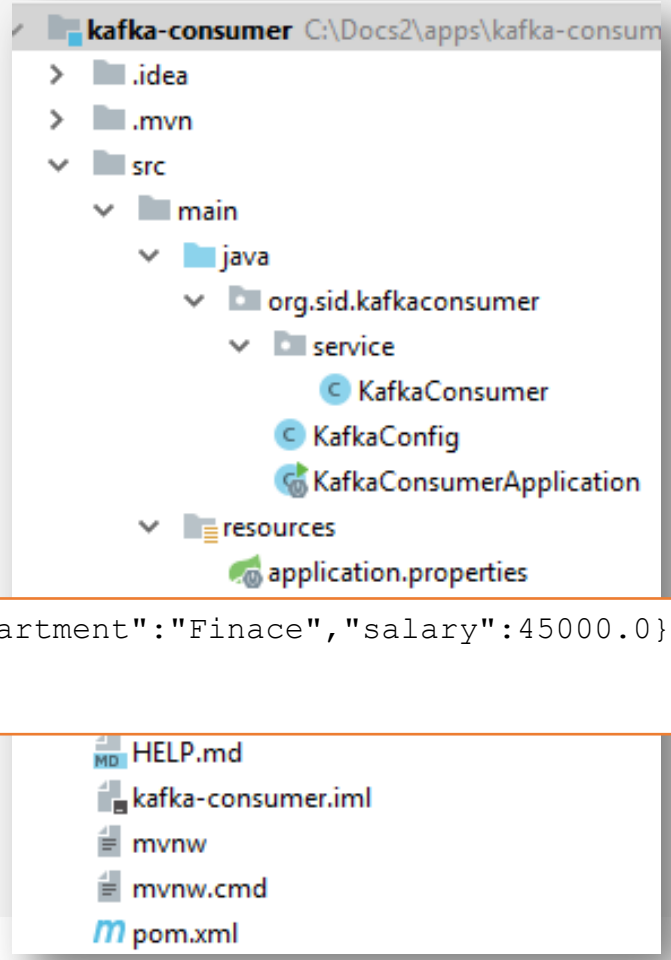
En Utilisant une classe de configuration au lieu de application.properties

```xml
<dependency>
   <groupId>org.springframework.kafka</groupId>
    <artifactId>spring-kafka</artifactId>
</dependency>
```

```
spring.kafka.consumer.bootstrap-servers=192.168.57.3:9092
spring.kafka.consumer.group-id=enset_uh2c
```

```java
package org.sid.kafkaconsumer.service; import org.springframework.kafka.annotation.KafkaListener;
import org.springframework.stereotype.Service;
@Service
public class KafkaConsumer {
    @KafkaListener(topics = {"testTopic"},groupId = "enset_uh2c")
    public void onMessage(String message){
        System.out.println("Consume ====>:"+message);
    }
}
```

```java
package org.sid.kafkaconsumer; import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.kafka.annotation.EnableKafka;
@SpringBootApplication
@EnableKafka
public class KafkaConsumerApplication {
    public static void main(String[] args) {
        SpringApplication.run(KafkaConsumerApplication.class, args);
    }
}
```




```
Consume ====>:{"name":"med","department":"Finace","salary":45000.0}
Consume ====>:f
Consume ====>:Simple message
```

Application Web : Spring Kafka Consumer avec Classe de configuration

En Utilisant une classe de configuration au lieu de application.properties

```
package org.sid.kafkaconsumer;
import org.apache.kafka.clients.consumer.ConsumerConfig;import
org.apache.kafka.common.serialization.StringDeserializer;
import org.springframework.context.annotation.Bean; import
org.springframework.context.annotation.Configuration;
import org.springframework.kafka.config.ConcurrentKafkaListenerContainerFactory;
import org.springframework.kafka.config.KafkaListenerContainerFactory;
import org.springframework.kafka.core.ConsumerFactory; import
org.springframework.kafka.core.DefaultKafkaConsumerFactory;
import java.util.HashMap; import java.util.Map;
@Configuration
public class KafkaConfig {
    @Bean
    ConsumerFactory<String,String> consumerFactory(){
        Map<String, Object> config=new HashMap<>();
        config.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG,"192.168.57.3:9092");
        config.put(ConsumerConfig.GROUP_ID_CONFIG,"enset_uh2c");
        config.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class);
        config.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,StringDeserializer.class);
        return new DefaultKafkaConsumerFactory<>(config);
    }
    @Bean
    ConcurrentKafkaListenerContainerFactory<String,String> kafkaListenerContainerFactory(){
        ConcurrentKafkaListenerContainerFactory factory=new
ConcurrentKafkaListenerContainerFactory();
        factory.setConsumerFactory(consumerFactory());
        return factory;
    }

}
```

```
Consume ====>:{"name":"med","department":"Finace","salary":45000.0}
Consume ====>:f
Consume ====>:Simple message
```

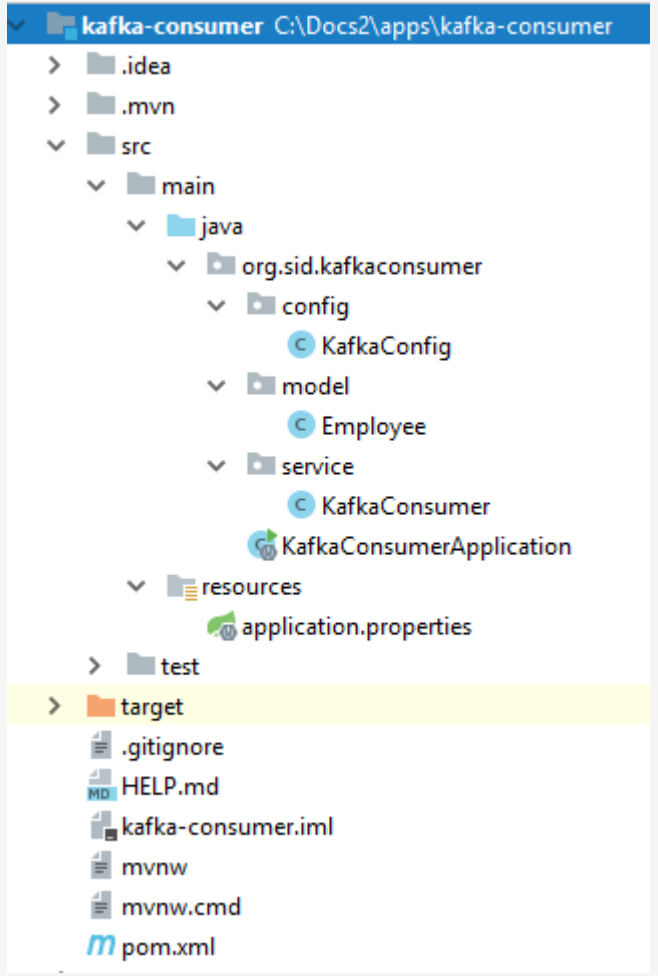
appliaction.properties

```
#spring.kafka.consumer.bootstrap-servers=192.168.57.3:9092
#spring.kafka.consumer.group-id=enset_uh2c
```

Application Web : Spring Kafka Consumer

Désérialisation des données structuré au format JSON en Objet Java Employee

```xml
<dependency>
    <groupId>org.springframework.kafka</groupId>
    <artifactId>spring-kafka</artifactId>
</dependency>
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.9.6</version>
</dependency>
```



Employee.java

```java
package org.sid.kafkaconsumer.model;
import lombok.AllArgsConstructor; import lombok.Data;
import lombok.NoArgsConstructor; import lombok.ToString;
@Data @NoArgsConstructor @AllArgsConstructor @ToString
public class Employee {
    private String name;
    private String department;
    private double salary;
}
```

Application Web : Spring Kafka Consumer

KafkaConfig.java

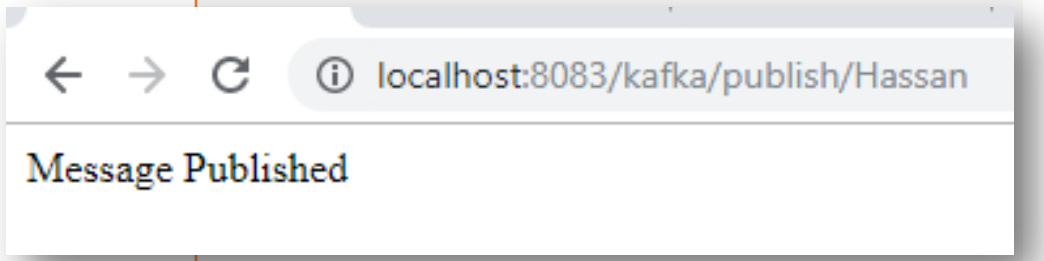
```
package org.sid.kafkaconsumer.config;
import org.apache.kafka.clients.consumer.ConsumerConfig; import org.apache.kafka.common.serialization.StringDeserializer;
import org.sid.kafkaconsumer.model.Employee; import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration; import org.springframework.kafka.config.ConcurrentKafkaListenerContainerFactory;
import org.springframework.kafka.core.ConsumerFactory; import org.springframework.kafka.core.DefaultKafkaConsumerFactory;
import org.springframework.kafka.support.serializer.JsonDeserializer;
import java.util.HashMap; import java.util.Map;
@Configuration
public class KafkaConfig {
    @Bean
    ConsumerFactory<String, Employee> consumerFactory(){
        Map<String, Object> config=new HashMap<>();
        config.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG,"192.168.57.3:9092");
        config.put(ConsumerConfig.GROUP_ID_CONFIG,"enset_uh2c");
        JsonDeserializer<Employee> jsonDeserializer=new JsonDeserializer<>(Employee.class);
        jsonDeserializer.addTrustedPackages("*");
        jsonDeserializer.setUseTypeHeaders(false);
        return new DefaultKafkaConsumerFactory(config,new StringDeserializer(),jsonDeserializer);
    }
    @Bean
    ConcurrentKafkaListenerContainerFactory<String, Employee> kafkaListenerContainerFactory(){
        ConcurrentKafkaListenerContainerFactory<String, Employee> factory=new ConcurrentKafkaListenerContainerFactory<>();
        factory.setConsumerFactory(consumerFactory());
        return factory;
    }

}
```

Application Web : Spring Kafka Consumer

KafkaConsumer.java

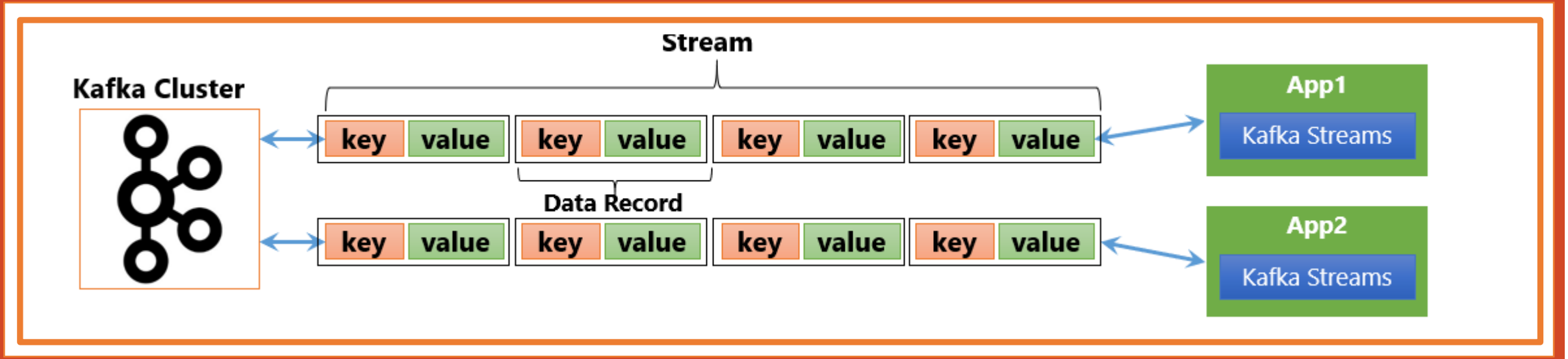
```
Received:=> Employee(name=med, department=Finace, salary=45000.0)
Received:=> Employee(name=Hassan, department=Finace, salary=45000.0)
```

```java
package org.sid.kafkaconsumer.service;

import org.sid.kafkaconsumer.model.Employee;
import org.springframework.kafka.annotation.KafkaListener;
import org.springframework.stereotype.Service;

@Service
public class KafkaConsumer {

    @KafkaListener(topics = {"testTopic"},groupId = "enset_uh2c")
    public void onMessage(Employee employee){
        System.out.println("Received:=>"+employee.toString());

    }
}
```




```
$ bin/kafka-console-producer.sh --broker-list localhost:9092 --topic testTopic
>{"name":"med","department":"info","salary":7000}
>
```

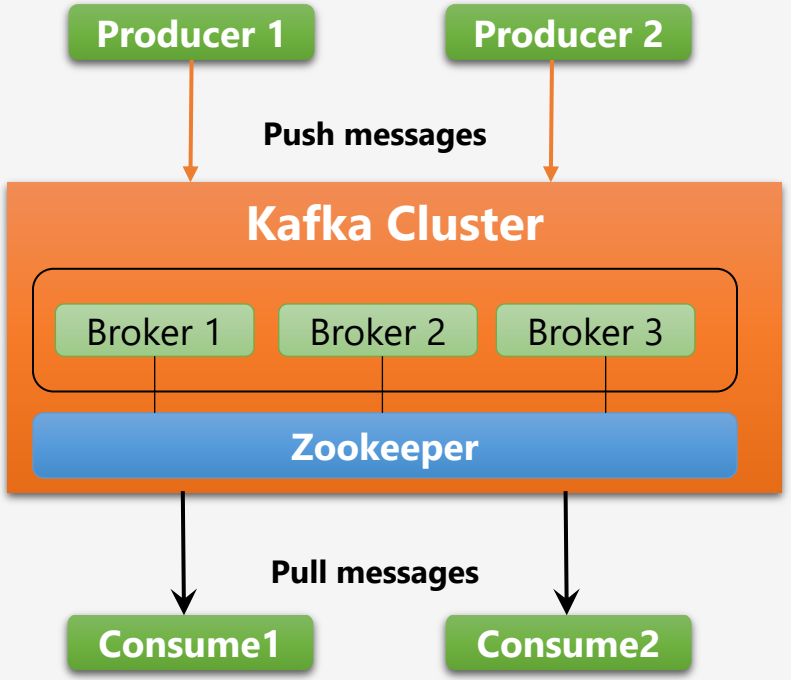
Big Data Real Time Stream Processing avec KAFKA Streams



Mohamed Youssfi
Laboratoire Signaux Systèmes Distribués et Intelligence Artificielle (SSDIA)
ENSET, Université Hassan II Casablanca, Maroc
Email : med@youssfi.net
Supports de cours : http://fr.slideshare.net/mohamedyoussfi9
Chaîne vidéo : http://youtube.com/mohamedYoussfi
Recherche : http://www.researchgate.net/profile/Youssfi_Mohamed/publications

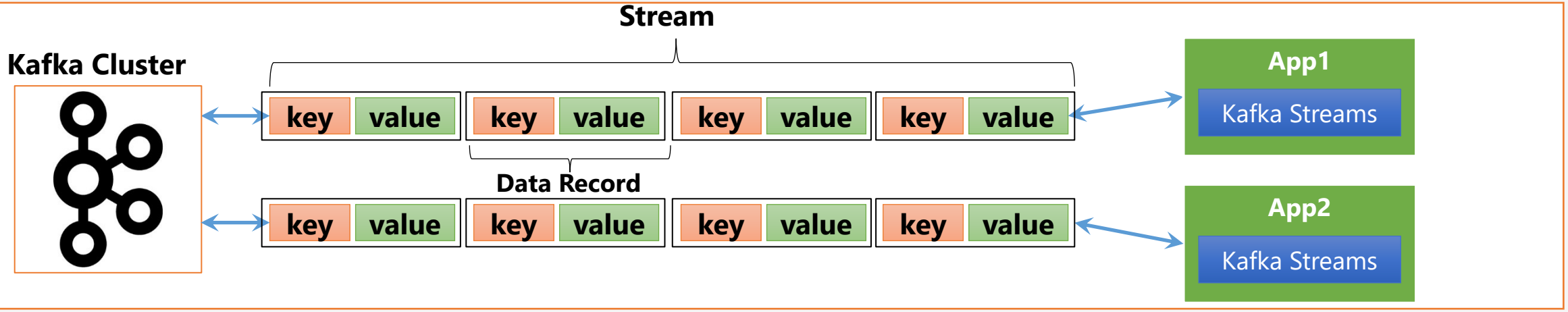
Les API de Kafka

- 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 fourni des API clientes pour différents langages : Java, C++, Node JS, .Net, PHP, Python, etc…



Kafka Stream API

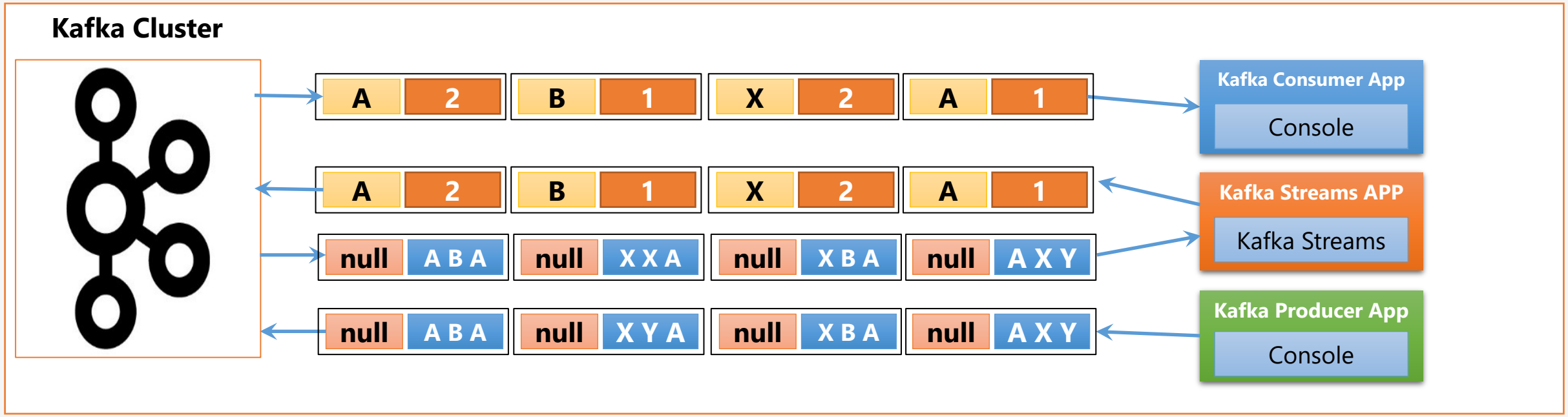
- Un Stream est un flux continu d'enregistrements de données en temps réel.
- Le consommateur reçoie continuellement le flux de données sans avoir besoin de les demander.
- Les Streams est une série de données de type paire clé, valeur



- L'API Kafka Streams transforme et enrichie les données des streams
- Supporte le traitement des streams par enregistrements avec **une latence faible de l'ordre de millisecondes**
- Supporte les **transformation statless**, les **transformations statfull**, et **les opérations fenêtrées**
- Permet d'écrire des applications avec différents langages sous forme de micro-services pour traiter les données en temps réel.
- Vous n'avez **pas besoin d'un cluster séparé pour les traitements**.
- Les traitements se déroulent au sein de l'application elle-même.
- Kafka streams est **scalable** et **tolèrent aux panes** et interagit avec le cluster de Kafka pour récupérer et publier les données et les résultats pour d'autres applications

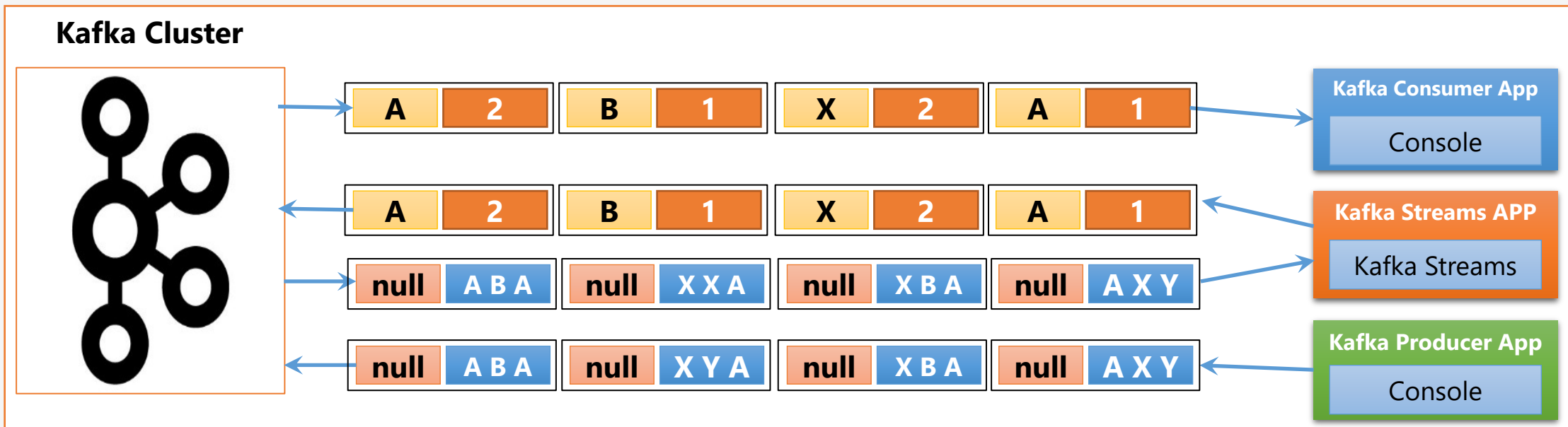
Kafka Stream API

```
Properties props = new Properties();
props.put(StreamsConfig.APPLICATION_ID_CONFIG, "wordcount-application");
props.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "192.168.57.3:9092");
props.put(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONFIG, Serdes.String().getClass());
props.put(StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG, Serdes.String().getClass());
```

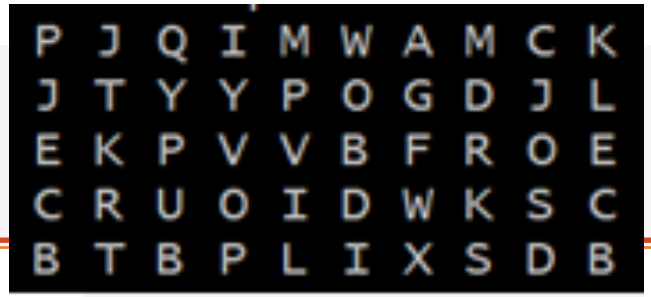


Kafka Stream API

```
StreamsBuilder builder = new StreamsBuilder();
// Serializers/deserializers (serde) for String and Long types
final Serde<String> stringSerde = Serdes.String();
final Serde<Long> longSerde = Serdes.Long();
// Construct a `KStream` from the input topic "streams-plaintext-input", where message values
// represent lines of text (for the sake of this example, we ignore whatever may be stored
// in the message keys).
KStream<String, String> textLines = builder.stream("streams-plaintext-input",
                Consumed.with(stringSerde, stringSerde));
```



Kafka Stream API

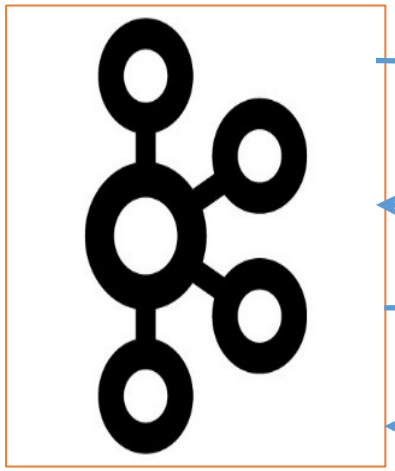



```
KStream<String, Long> dataAnalytics= textLinesStream
        .flatMapValues(textLine-> Arrays.asList(textLine.split(" ")))
        .map((k,v)->new KeyValue<>(k,v.toLowerCase()))
        .filter((k,v)->v.equals("a") || v.equals("b"))
        .groupBy((k,v)->v)
        .windowedBy(TimeWindows.of(Duration.ofSeconds(5)))
        .count(Materialized.as("count-store"))
        .toStream().map((k,v)->new KeyValue<>(k.key(),v));
dataAnalytics.to("wcData",Produced.valueSerde(Serdes.Long()));
```

P	J	Q	I	M	W	A	...	
p	j	q	i	m	w	a	...	
a	b	b	b	b	..	..	...	

a	1
b	4

Kafka Cluster



A	2	B	1	X	2	A	1
A	2	B	1	X	2	A	1
null	A B A	null	X X A	null	X B A	null	A X Y
null	A B A	null	X Y A	null	X B A	null	A X Y

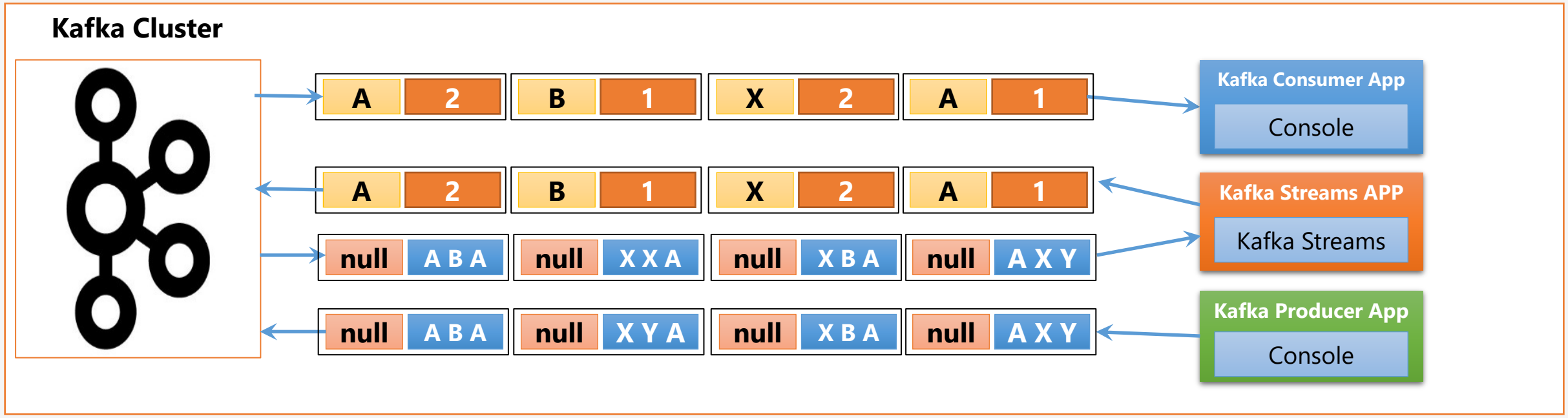
Kafka Consumer App
Console

Kafka Streams APP
Kafka Streams

Kafka Producer App
Console

Kafka Stream API

```
// Start Kafka Streams
Topology topology=streamsBuilder.build();
KafkaStreams kafkaStreams=new KafkaStreams(topology,properties);
kafkaStreams.start();
```



Kafka Stream API

```
<dependencies>
   <dependency>
        <groupId>org.apache.kafka</groupId>
        <artifactId>kafka-clients</artifactId>
        <version>2.3.0</version>
    </dependency>
    <dependency>
        <groupId>org.apache.kafka</groupId>
        <artifactId>kafka_2.12</artifactId>
        <version>2.3.0</version>
    </dependency>
    <dependency>
        <groupId>org.apache.kafka</groupId>
        <artifactId>kafka-streams</artifactId>
        <version>2.3.0</version>
    </dependency>

</dependencies>
```

Kafka Stream Producer

```
youssfi@youssfi-VirtualBox:~/kafka_2.12-2.3.0$ bin/kafka-console-consumer.sh --
bootstrap-server localhost:9092 --topic testTopic
 P J Q I M W A M C K
 J T Y Y P O G D J L
 E K P V V B F R O E
 C R U O I D W K S C
 B T B P L I X S D B
```

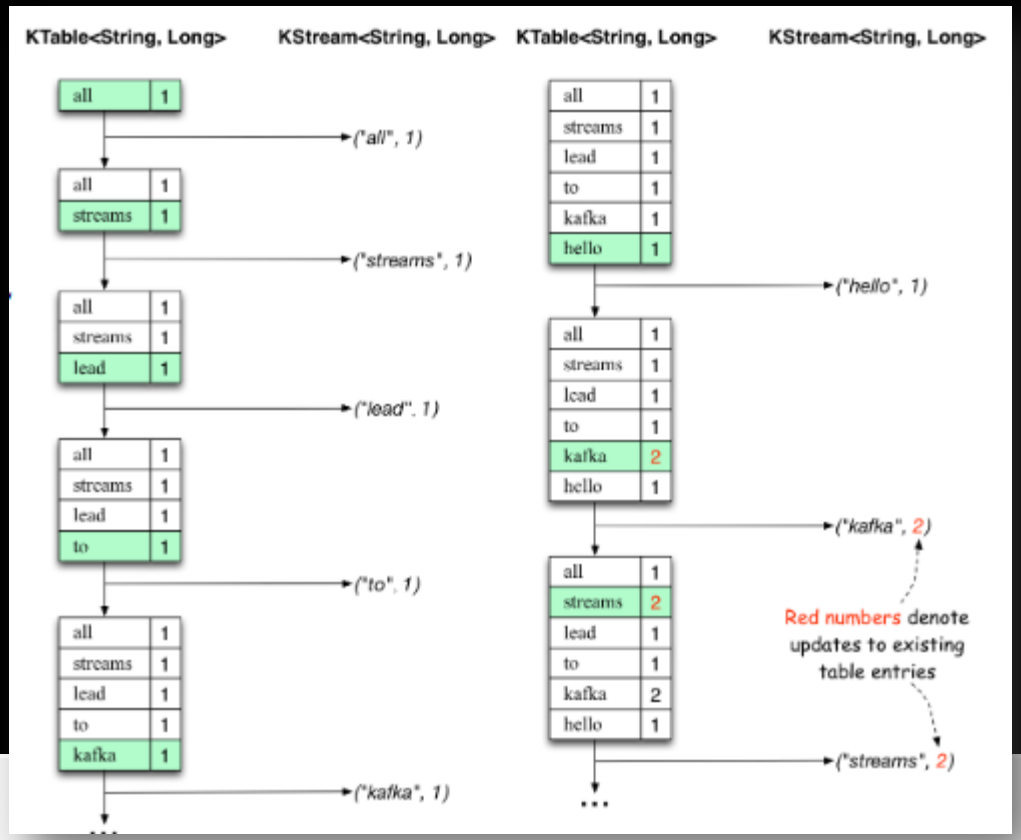
```
import org.apache.kafka.clients.producer.KafkaProducer; import org.apache.kafka.clients.producer.ProducerRecord;
import java.util.*; import java.util.concurrent.Executors; import java.util.concurrent.TimeUnit;
public class StreamProducer {
    private int counter;
    private String KAFKA_BROKER_URL="192.168.43.22:9092";    private String TOPIC_NAME="testTopic";
    private String clientID="client_prod_1";    private String message;
    public static void main(String[] args) {
        new StreamProducer();
    }
    public StreamProducer() {
        Properties properties = new Properties();
        properties.put("bootstrap.servers", KAFKA_BROKER_URL);
        properties.put("client.id", clientID);
        properties.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
        properties.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");
        KafkaProducer<String,String> producer=new KafkaProducer<String, String>(properties);
        Random random=new Random();
        List<Character> characters= new ArrayList<>();
        for (char i = 'A'; i <'Z' ; i++) {   characters.add(i);   }
        Executors.newScheduledThreadPool(1).scheduleAtFixedRate(()->{
            ++counter;  message="";  for (int i = 0; i < 10; i++) {
                message+=" "+characters.get(random.nextInt(characters.size()));
            }
            producer.send(new ProducerRecord<String, String>(TOPIC_NAME,""+(++counter),message),
                    (metadata,ex)->{
                        System.out.println("Sending Message key=>"+counter+" Value =>"+message);
                        System.out.println("Partition => "+metadata.partition()+" Offset=>"+metadata.offset());
                    });
        },1000,1000, TimeUnit.MILLISECONDS);
    }
}
```

Kafka Stream Application

```
import org.apache.kafka.common.serialization.Serdes;import org.apache.kafka.streams.KafkaStreams;
import org.apache.kafka.streams.StreamsBuilder;import org.apache.kafka.streams.StreamsConfig;
import org.apache.kafka.streams.kstream.KStream;import org.apache.kafka.streams.kstream.KTable;
import org.apache.kafka.streams.kstream.Materialized; import org.apache.kafka.streams.kstream.Produced;
import java.util.Arrays;import java.util.Properties;
public class KafkaStreamsApp {
    public static void main(String[] args) {
        Properties properties=new Properties();
        properties.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG,"192.168.43.22:9092");
        properties.put(StreamsConfig.APPLICATION_ID_CONFIG,"k-stream-app");
        properties.put(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONFIG, Serdes.String().getClass().getName());
        properties.put(StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG, Serdes.String().getClass().getName());
        properties.put(StreamsConfig.COMMIT_INTERVAL_MS_CONFIG, 1000);
        StreamsBuilder streamsBuilder=new StreamsBuilder();
        KStream<String,String> textLines=streamsBuilder.stream("testTopic");
        KTable<String,Long> wordCounts=textLines
                .flatMapValues(textLine-> Arrays.asList(textLine.split("\\W+")))
                .groupBy((k,word)->word)
                .count(Materialized.as("count-store"));
        wordCounts.toStream().to("wcTopic", Produced.with(Serdes.String(),Serdes.Long()));
        Topology topology=streamsBuilder.build();
        KafkaStreams kafkaStreams=new KafkaStreams(topology,properties);
        kafkaStreams.start();

    }
}
```

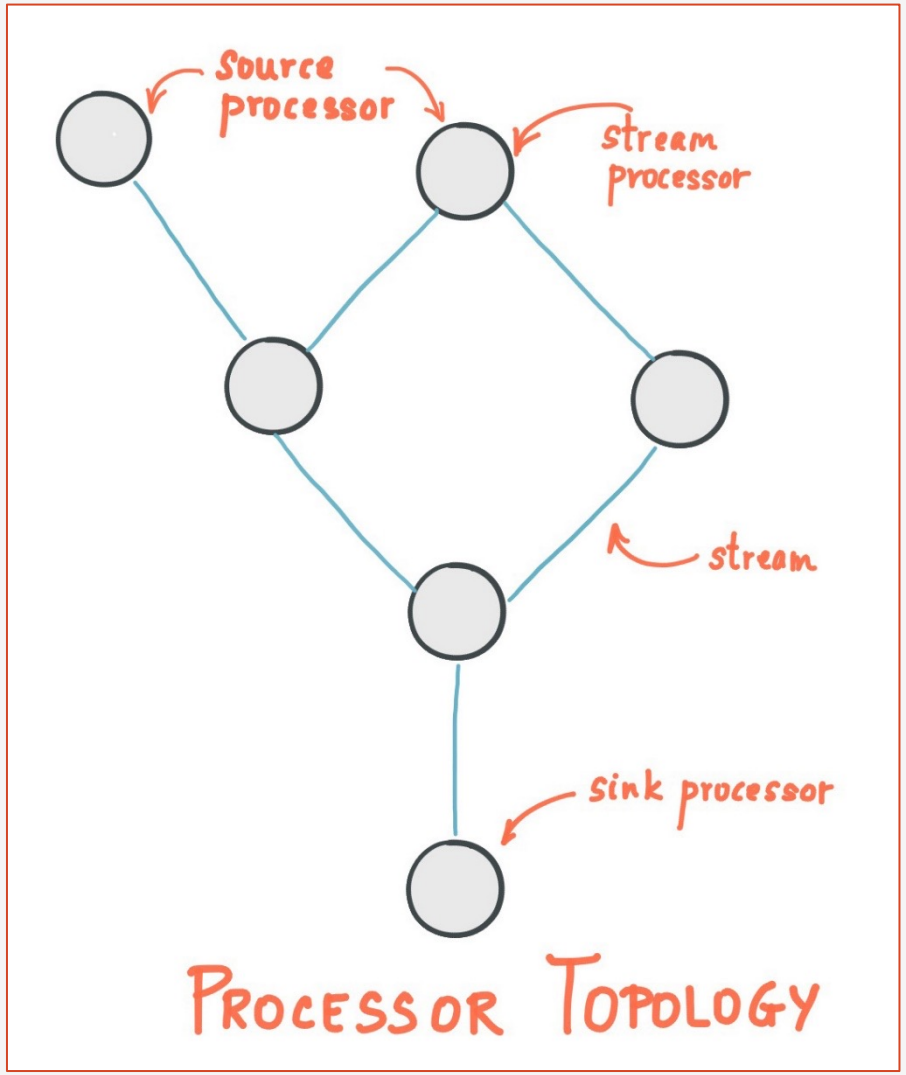
Kafka Stream API (Stream Processing Consumer)

```
youssfi@youssfi-VirtualBox:~/kafka_2.12-2.3.0$ bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --
topic wcTopic --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
        827
N       347
Y       328
S       311
W       330
H       365
G       338
V       322
        828
O       309
M       325
C       331
F       274
S       312
U       322
D       333
G       339
```



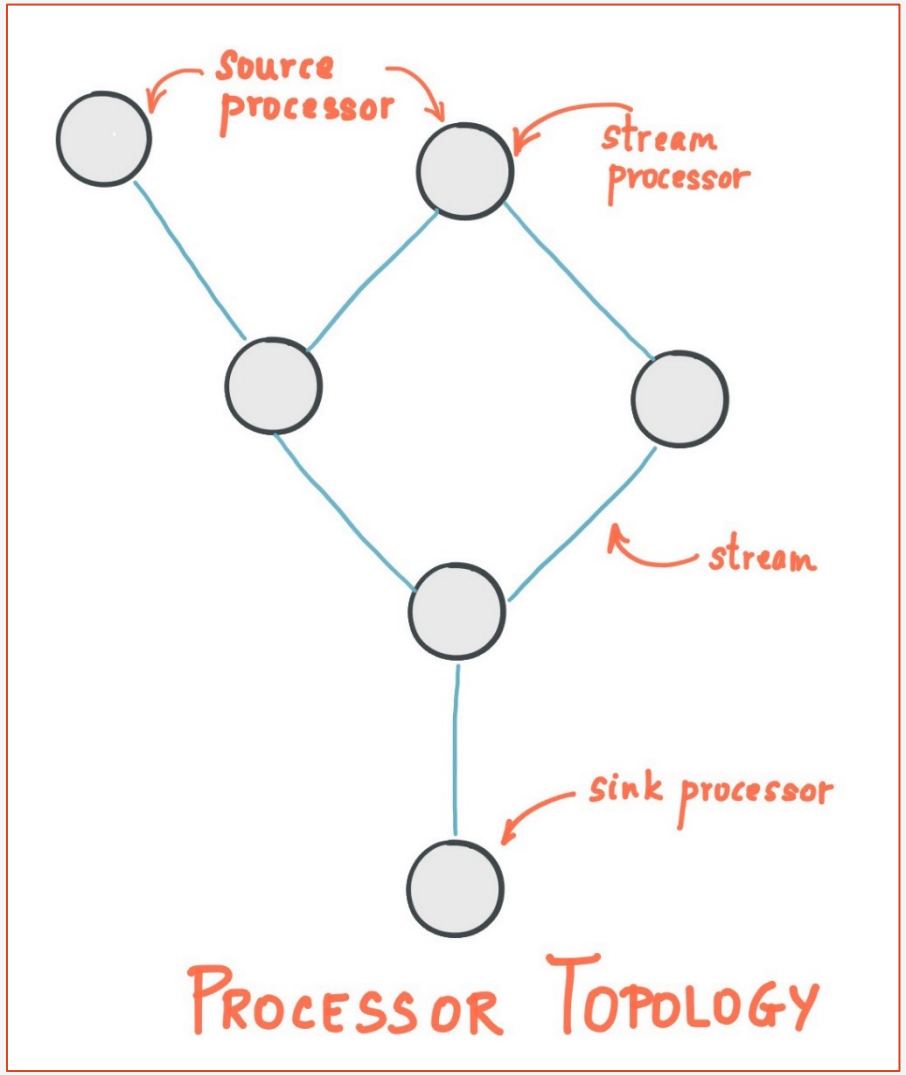
Concepts de base de Kafka : Stream Processing Topology

- Un **Stream** représente un ensemble de données illimité et constamment mis à jour. C'est une séquence d'enregistrements de données immutables, dans laquelle un enregistrement de données est défini comme une paire clé-valeur.
- **Une application de traitement de flux** est un programme qui utilise la bibliothèque Kafka Streams. Il définit sa logique de calcul par le biais d'une ou de plusieurs **topologies de processeurs (Processor Topologies)**.
- **Une topologie de processeurs** étant un graphe de processeurs de flux (**Stream Processors**) (nœuds) connectés par des Steram (arêtes).
- **Un processeur de flux (Stream Processor)** est un nœud de la topologie du processeurs; il représente une étape de traitement pour transformer les données des Streams en recevant un enregistrement en d'entrée à un instant donné à partir de ses processeurs en amont dans la topologie, en lui appliquant son **opérateur (map, ftatMap, groupBy, count, etc..)**, et peut ensuite produire un ou plusieurs enregistrements en sortie à ses processeurs en aval.



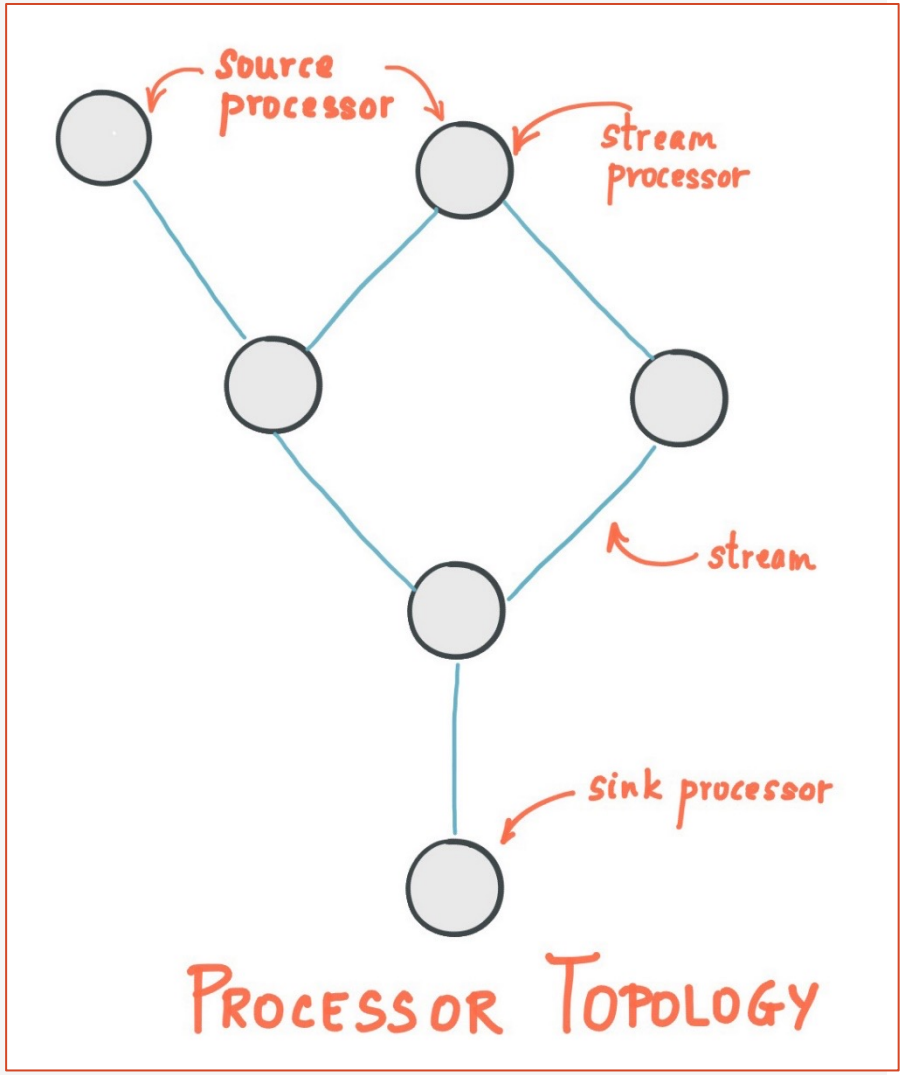
Concepts de base de Kafka : Stream Processing Topology

- Kafka Streams propose deux manières de définir la topologie de traitement de flux:
  - **Kafka Streams DSL** : fournit les opérations de transformation de données les plus courantes, telles que mappage, filtrage, jointure et agrégations prêtes à l'emploi;
  - **Lower-level Processor API** : permet aux développeurs de définir et de connecter des processeurs personnalisés, ainsi que d'interagir avec les state stores.



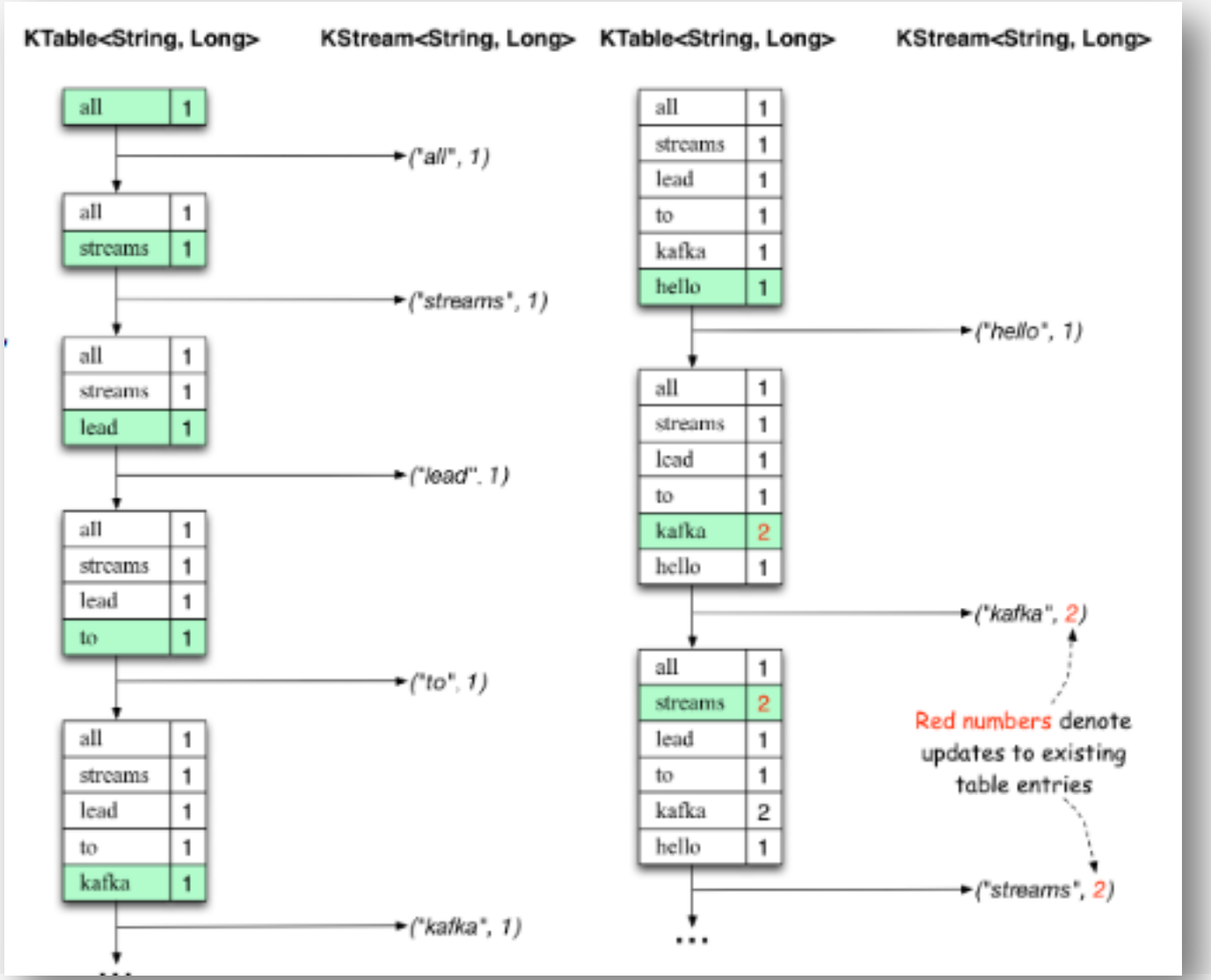
Concepts de base de Kafka : Time

- Un aspect essentiel du traitement de flux est la notion de temps, ainsi que sa modélisation et son intégration. Par exemple, certaines opérations telles que le fenêtrage (**windowing**) sont définies en fonction des intervalles d e temps.
- Les notions courantes de temps dans les flux sont:
  - **Event Time** : (Heure de l'événement) - Heure à laquelle un événement ou un enregistrement de données s'est produit, c'est-à-dire qu'il a été créé à l'origine "à la source« (Temps de capture de l'événement par un capteur GPS par exemple).
  - **Processing Time** (Durée de traitement) - Peut être de l'ordre de la milliseconde ou de la seconde (pour les pipelines Real Time basés sur Apache Kafka et Kafka Streams) ou de minutes et d'heures (Les pipelines par Batch Processing, basés sur Apache Hadoop ou Apache Spark).
  - **Ingestion Time** (Heure d'ingestion) - Heure à laquelle un événement ou un enregistrement de données est stocké dans une partition de Topic par un Broker Kafka.
- Le choix entre **Event Time** et **Ingestion Time** se fait en réalité via la configuration de Kafka (et non de Kafka Streams)
- Kafka Streams attribue un horodatage (timestamp) à chaque enregistrement de données via l'interface TimestampExtractor. Ces horodatages par enregistrement décrivent la progression d'un flux en termes de temps et sont exploités par des opérations dépendantes du temps telles que les opérations de windowing



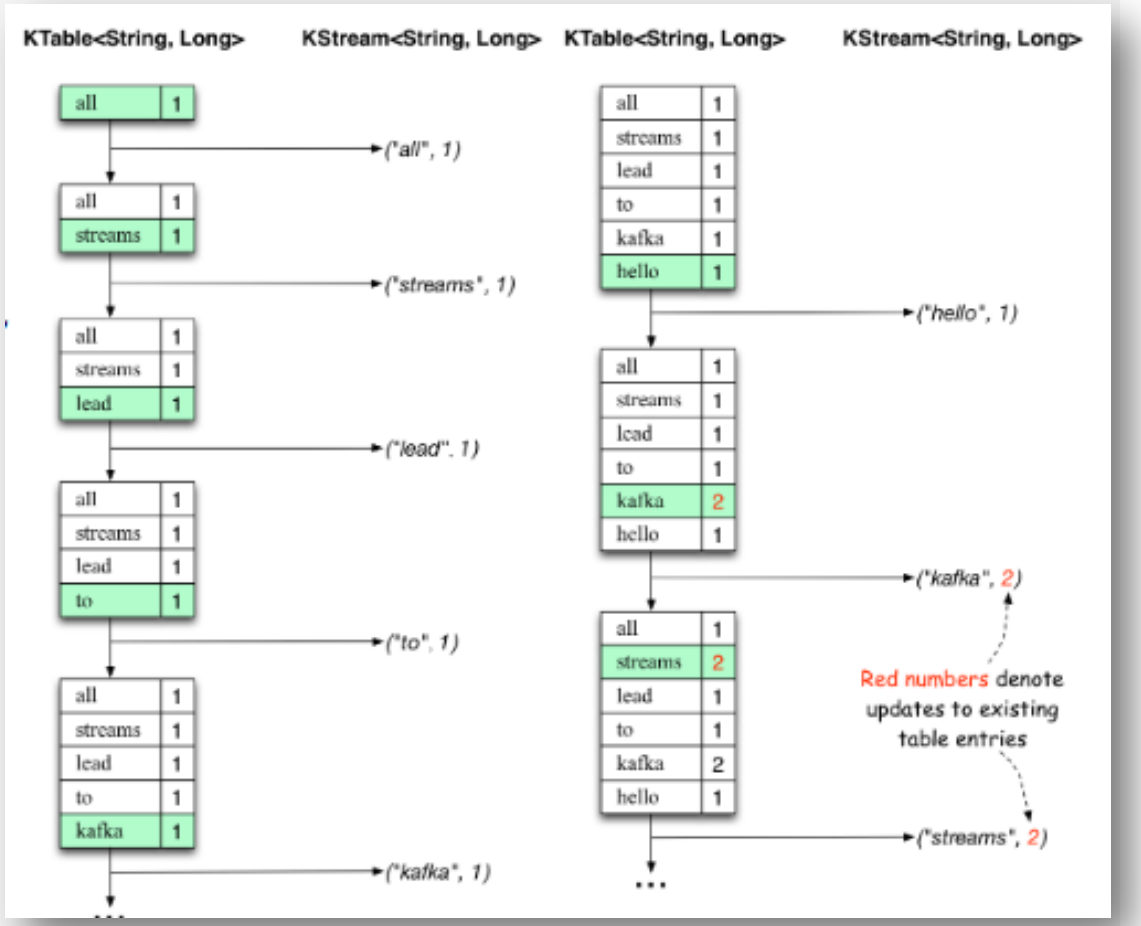
Concepts de base de Kafka : Agregations

- Une opération d'agrégation prend un flux ou une table d'entrée et génère une nouvelle table en combinant plusieurs enregistrements d'entrée en un seul enregistrement de sortie. Des exemples d'agrégations sont le calcul du nombre (**count**) ou de la somme (**sum**).
- Dans le DSL de Kafka Streams, un flux d'entrée d'une agrégation peut être un **KStream** ou un **KTable**, mais le flux de sortie sera toujours un **KTable**.
- Cela permet à Kafka Streams de mettre à jour une valeur globale lors de l'arrivée tardive d'autres enregistrements après que la valeur a été produite et émise.



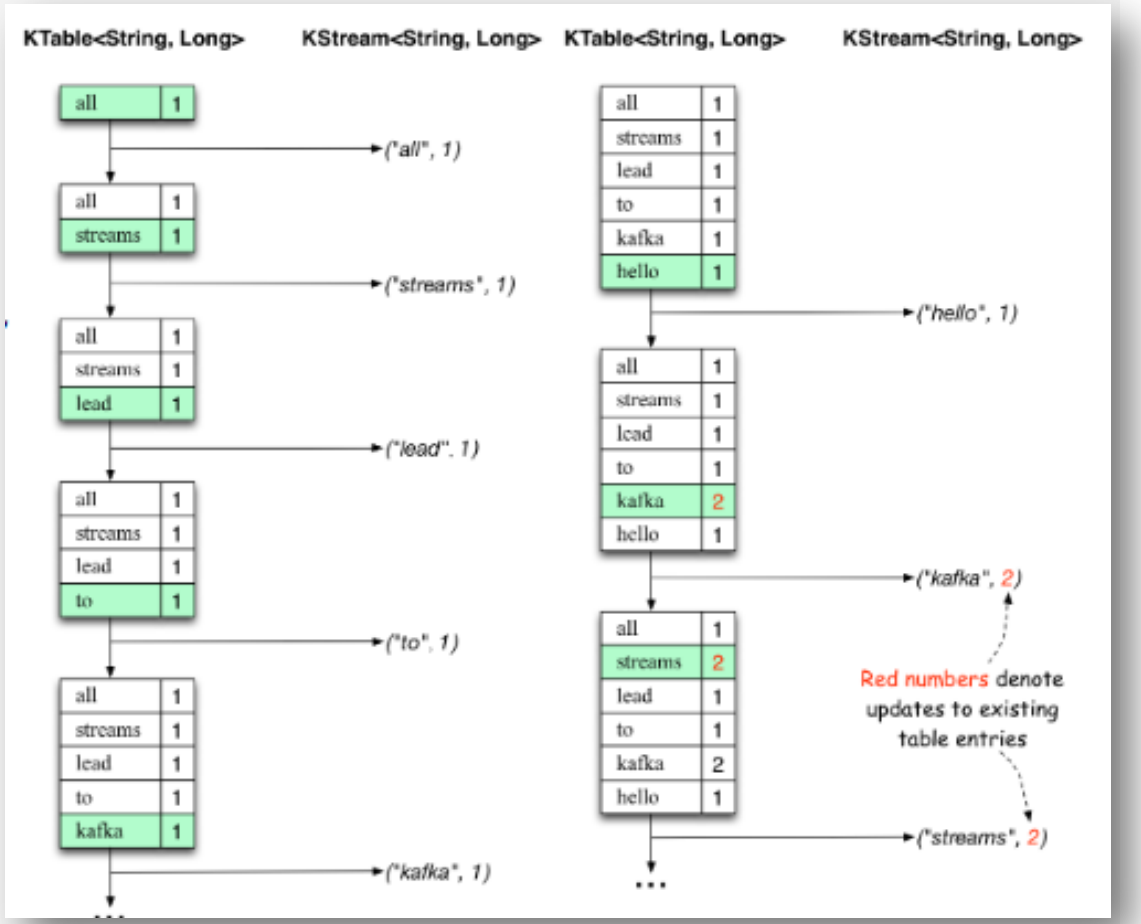
Concepts de base de Kafka : Windowing

- Le fenêtrage vous permet de contrôler comment regrouper les enregistrements ayant la même clé pour des opérations avec état telles que des agrégations ou des jointures dans des fenêtres.
- Avec les ces opérations, vous pouvez spécifier une période de rétention pour la fenêtre.
- Cette période de rétention détermine la durée pendant laquelle Kafka Stream attendra des enregistrements de données tardifs pour une fenêtre donnée.



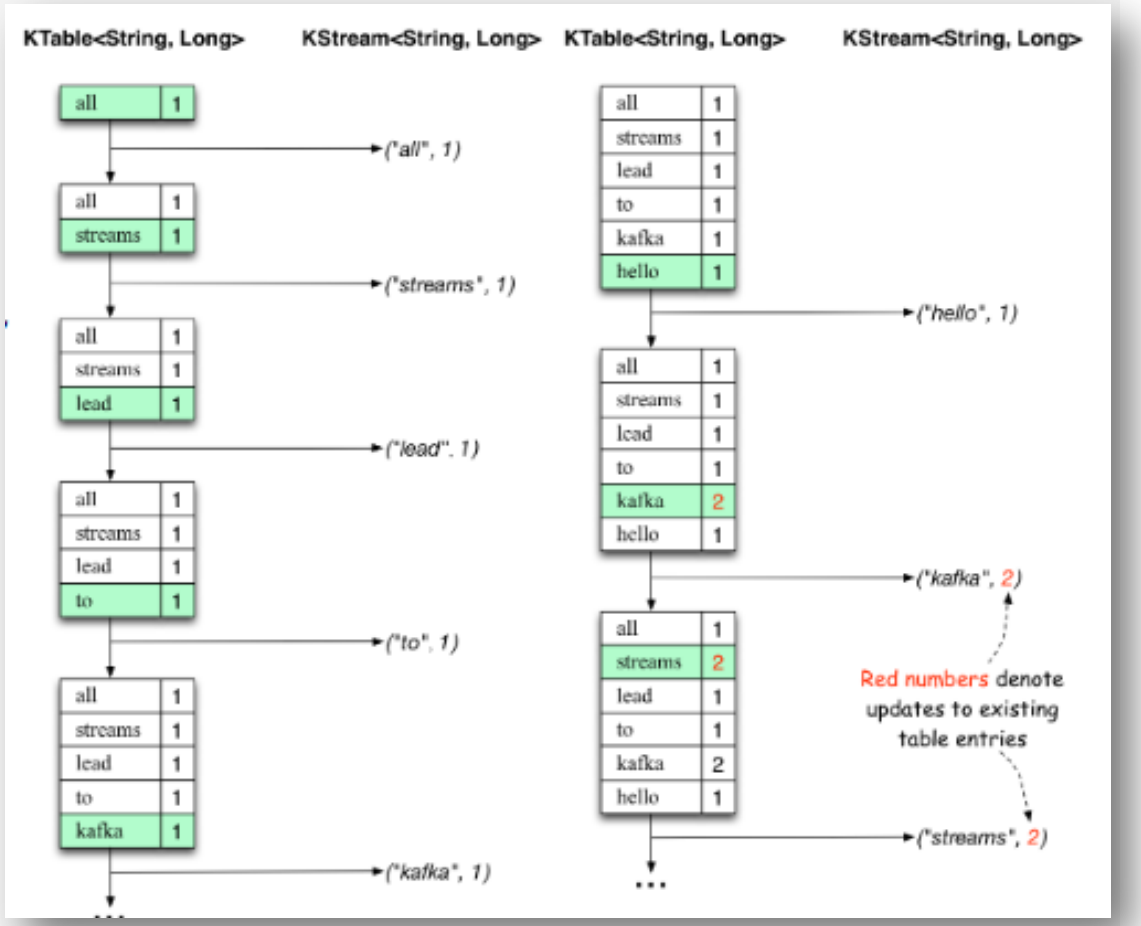
Concepts de base de Kafka : States

- Certaines applications de traitement de flux ne nécessitent pas d'état (**Statless Operations**), ce qui signifie que le traitement d'un message est indépendant du traitement de tous les autres messages.
- Cependant, le maintien de l'état (**Statful Operations**) ouvre de nombreuses possibilités pour les applications de traitement de flux sophistiquées: vous pouvez joindre des flux d'entrée (**Join**) ou regrouper (**GroupBy**) des enregistrements de données. Le DSL de Kafka Streams fournit nombre de ces opérateurs avec état (**Statful Operations**).
- Kafka Streams fournit des magasins d'état (**Data Stores**), que les applications de traitement de flux peuvent utiliser pour stocker et interroger des données.
- C'est une capacité importante lors de la mise en œuvre d'opérations avec état. Chaque tâche de Kafka Streams intègre un ou plusieurs magasins d'Etats accessibles via des API pour stocker et interroger les données nécessaires au traitement.

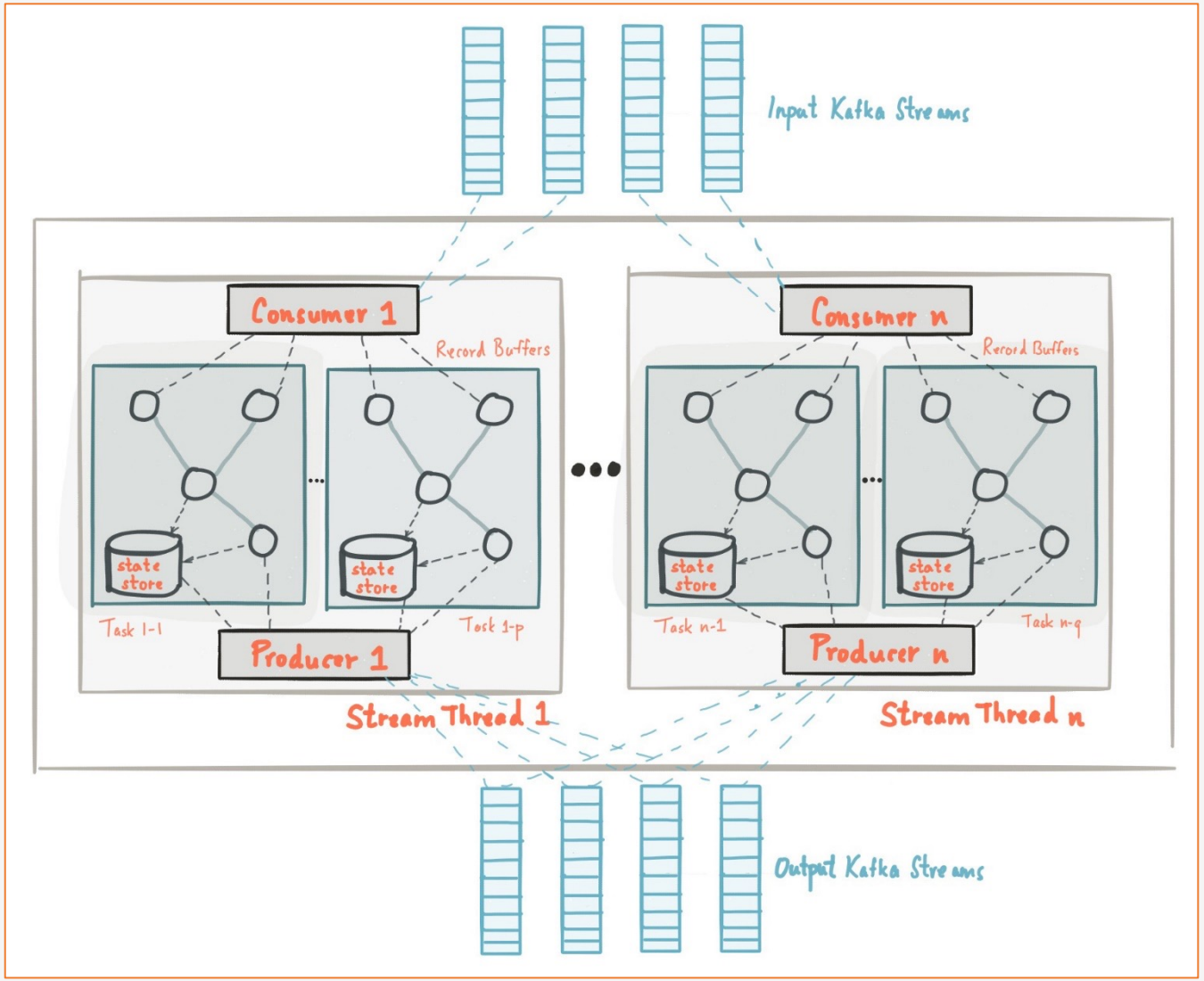


Concepts de base de Kafka : States

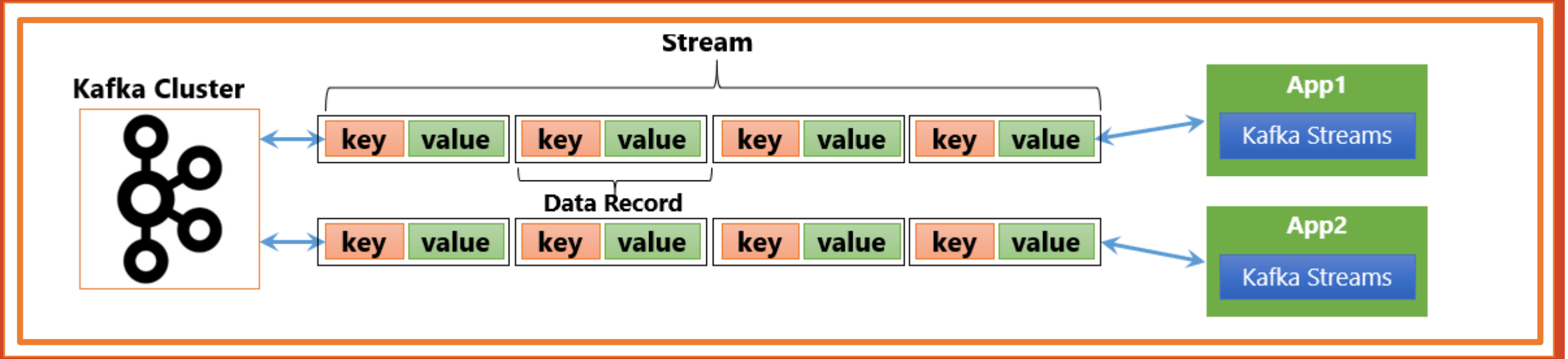
- Ces magasins d'état peuvent être **un magasin de clé-valeur persistant**, **un hashmap en mémoire** ou **une autre structure de données pratique**.
- Kafka Streams offre une tolérance aux pannes et une récupération automatique pour les magasins d'état locaux.
- Kafka Streams permet des requêtes directes en lecture seule sur les magasins d'état à l'aide de méthodes, de threads, de processus ou d'applications externes à l'application de traitement de flux ayant créé les magasins d'état.
- Ceci est fourni par une fonctionnalité appelée **Interactive Queries**.
- Tous les magasins sont nommés et Interactive Queries n'expose que les opérations de lecture de l'implémentation sous-jacente.



Concepts de base de Kafka : Architecture



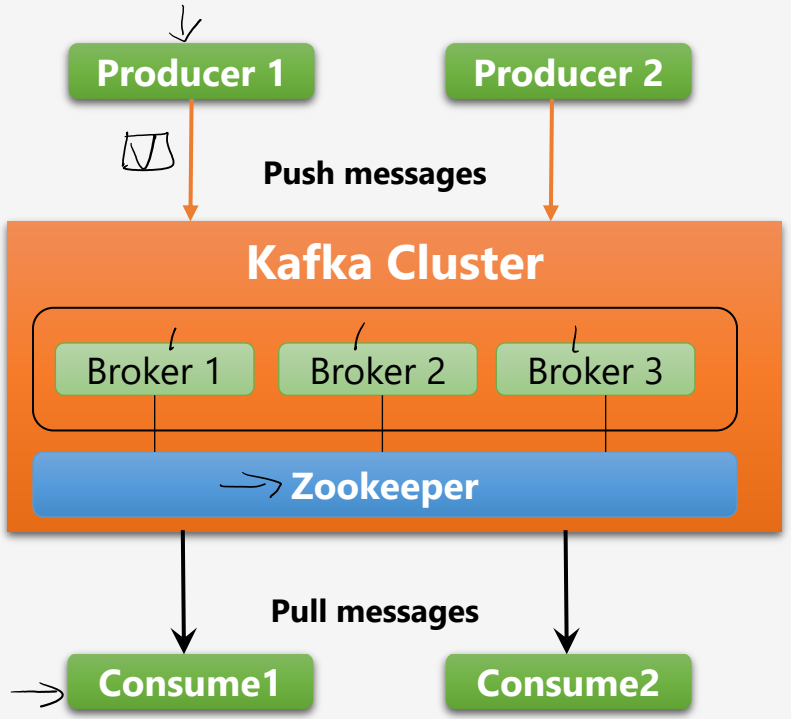
Big Data Real Time Stream Processing avec KAFKA Streams



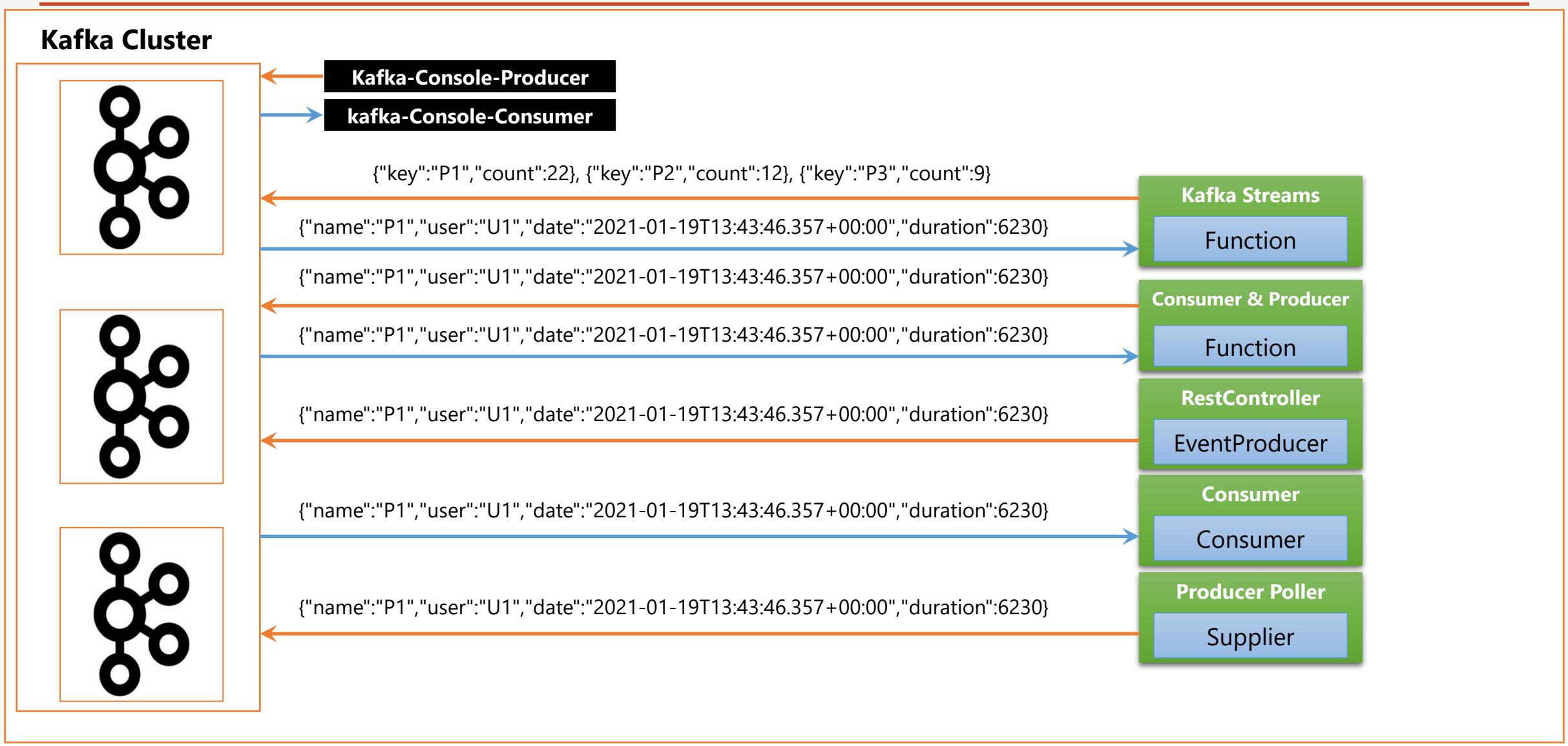
Mohamed Youssfi
Laboratoire Signaux Systèmes Distribués et Intelligence Artificielle (SSDIA)
ENSET, Université Hassan II Casablanca, Maroc
Email : med@youssfi.net
Supports de cours : http://fr.slideshare.net/mohamedyoussfi9
Chaîne vidéo : http://youtube.com/mohamedYoussfi
Recherche : http://www.researchgate.net/profile/Youssfi_Mohamed/publications

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

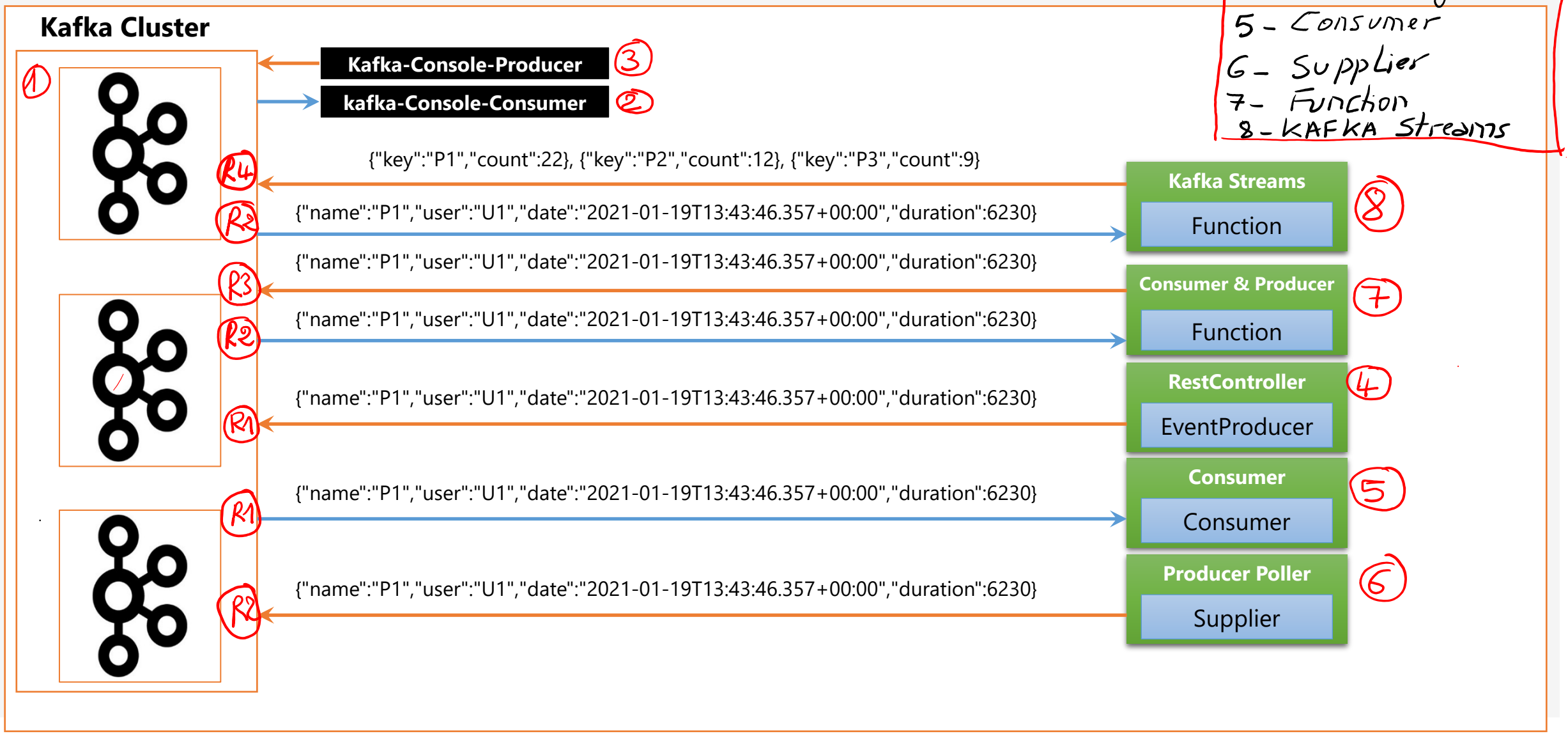


Application Spring Cloud Streams Functions



Activité pratique Spring Cloud Streams Functions et KAFKA

1- KAFKA
2- KAFKA-Console
et 3
4- RestController avec StreamBridge
5- Consumer
6- Supplier
7- Function
8- KAFKA Streams



Démarrage du Broker

1. **Lancer le serveur Zookeeper:** `> bin\windows\zookeeper-server-start.bat config/zookeeper.properties`
2. **Lancer le Boker KAFKA :** `> bin\windows\kafka-server-start.bat config/server.properties`
3. **Lancer Kafka-console-consumer :**

```
> bin\windows\kafka-console-consumer.bat --bootstrap-server localhost:9092 --topic R4 --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.StringDeserializer
```

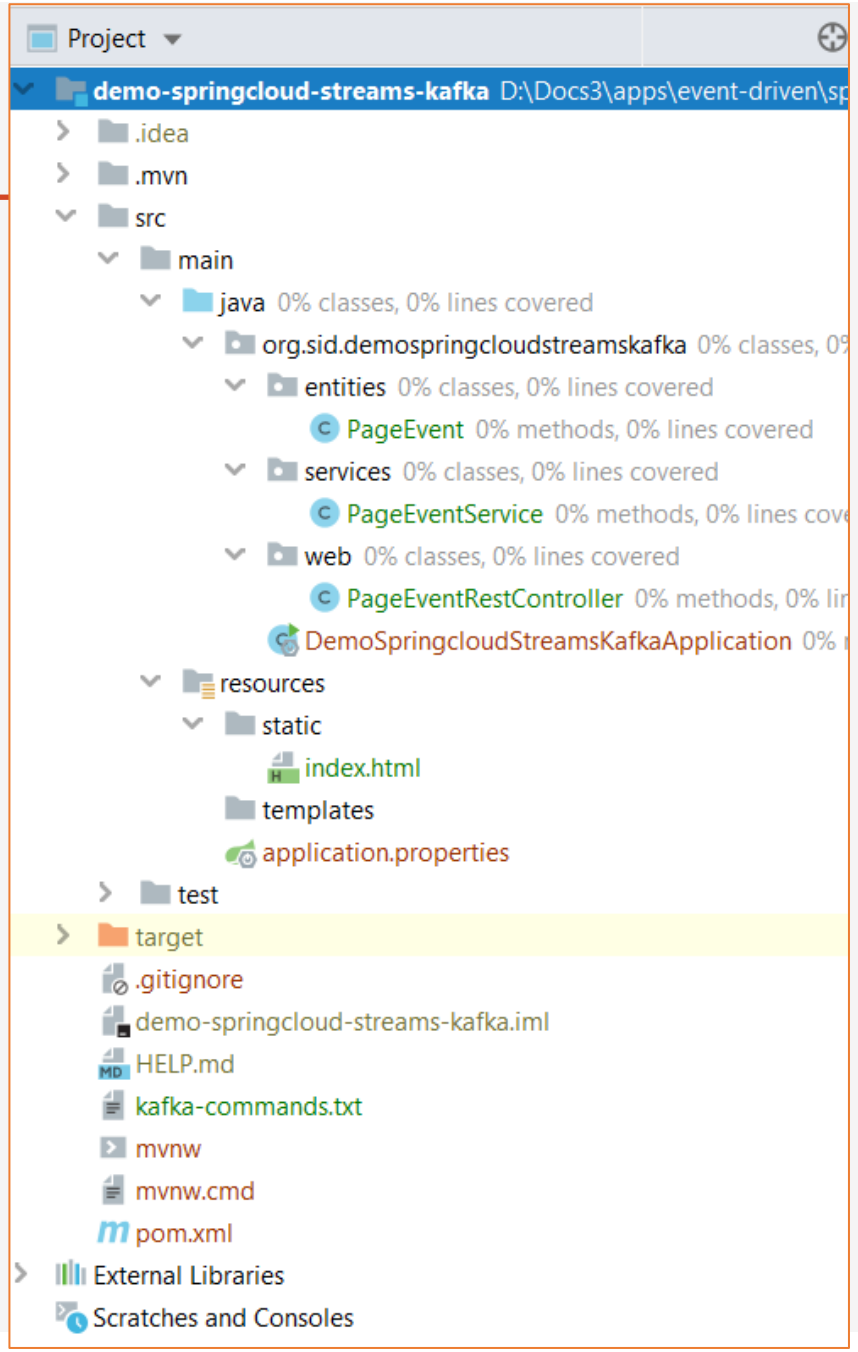
4. **Lancer kafka-console-producer :** `> bin\windows\kafka-console-producer.bat --broker-list localhost:9092 --topic R4`

```
D:\Tools>cd kafka_2.13-2.7.0

D:\Tools\kafka_2.13-2.7.0>bin\windows\kafka-console-consumer.bat --bootstrap-server localhost:9092 --topic R4
--property print.key=true --property print.value=true --property key.deserializer=org.apache.kafka.common.seri
alization.StringDeserializer --property value.deserializer=org.apache.kafka.common.serialization.StringDeseria
lizer
null    Hello
null    Test
null    Me
```

```
D:\Tools>cd kafka_2.13-2.7.0

D:\Tools\kafka_2.13-2.7.0>bin\windows\kafka-console-producer.bat --broker-list localhost:9092 --topic R4
>Hello
>Test
>Me
>
```

Structure du Projet

```xml
<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>
```



Spring Cloud Streams Consumer

```
@RestController
public class PageEventRestController {
    @Autowired
    private StreamBridge streamBridge;

    @GetMapping("/publish/{topic}/{name}")
    public PageEvent publish(@PathVariable String topic,
@PathVariable String name){
        PageEvent pageEvent=new
PageEvent(name,Math.random()>0.5?"U1":"U2",new Date(),new
Random().nextInt(9000));
        streamBridge.send(topic,pageEvent);
        return pageEvent;
    }

}
```

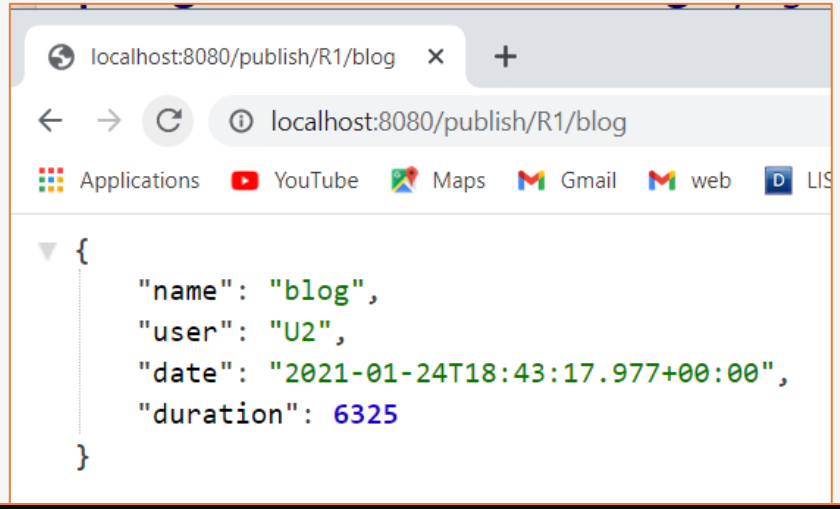
```
@Data @NoArgsConstructor
@AllArgsConstructor @ToString
public class PageEvent {
    private String name;
    private String user;
    private Date date;
    private long duration;
}
```

Spring Cloud Streams Consumer

application.properties

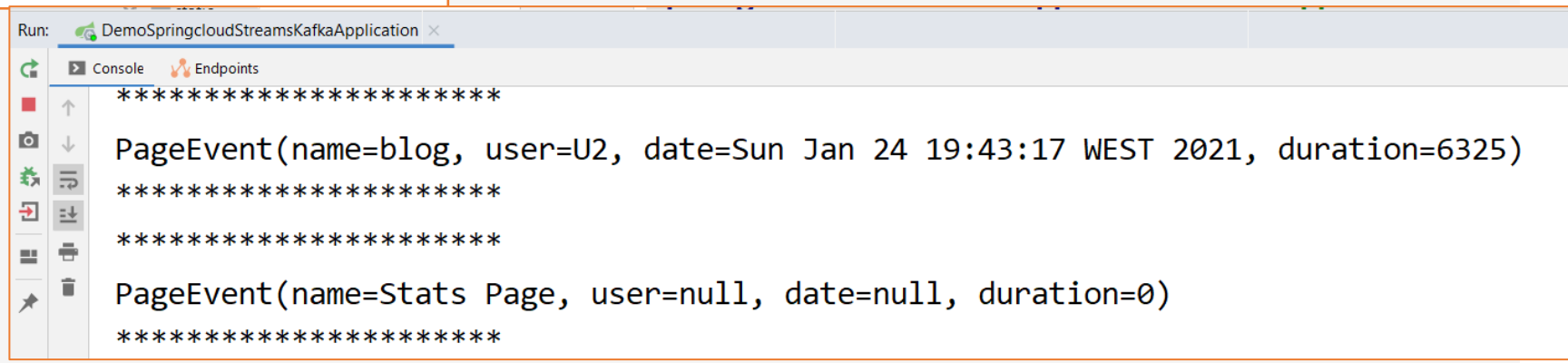
```
spring.application.name=app1
spring.cloud.stream.bindings.pageEventConsumer-in-0.destination=R1
```

```java
@Service
public class PageEventService {

    @Bean
    public Consumer<PageEvent> pageEventConsumer(){
        return (input)->{
            System.out.println("**********************");
            System.out.println(input.toString());
            System.out.println("**********************");
        };
    }
}
```




```
D:\Tools\kafka_2.13-2.7.0>bin\windows\kafka-console-producer.bat --broker-list localhost:9092 --topic R1
>{"name":"Stats Page"}
>
```

```java
@Data @NoArgsConstructor
@AllArgsConstructor @ToString
public class PageEvent {
    private String name;
    private String user;
    private Date date;
    private long duration;
}
```



Spring Cloud Streams Supplier

application.properties

```
spring.cloud.stream.bindings.pageEventSupplier-out-0.destination=R2
spring.cloud.stream.poller.fixed-delay=100
```

```
@Service
public class PageEventService {

    @Bean
public Supplier<PageEvent> pageEventSupplier(){
    return ()-> new PageEvent(
            Math.random()>0.5?"P1":"P2",
            Math.random()>0.5?"U1":"U2",
            new Date(),
            new Random().nextInt(9000));
}
}
```

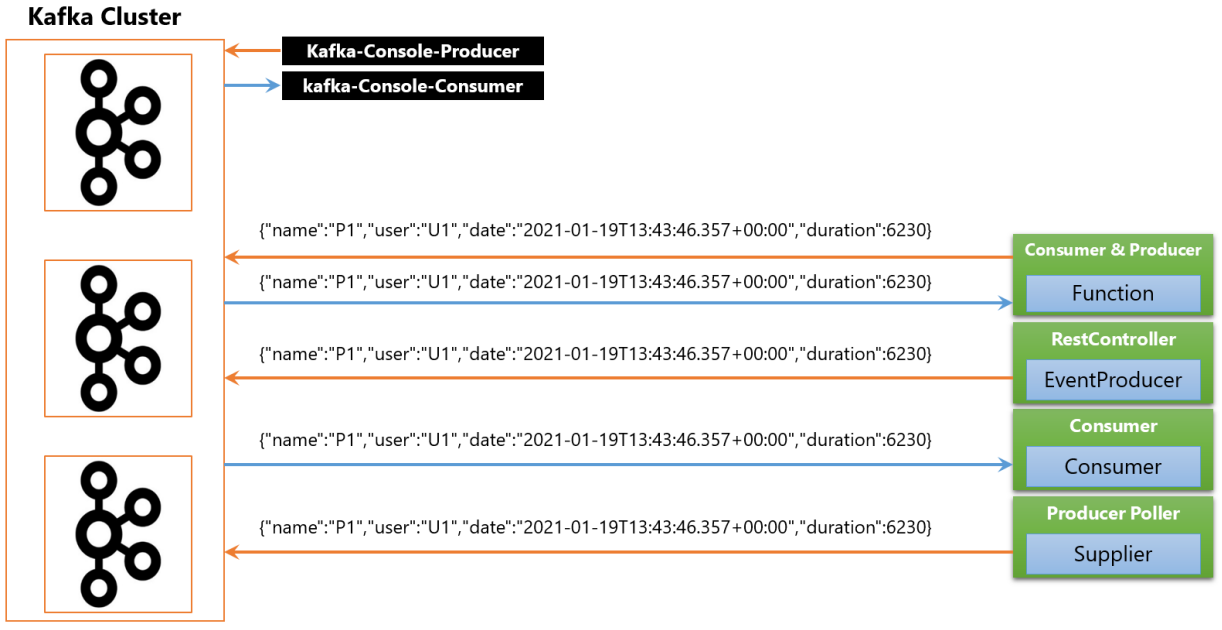
```
D:\Tools\kafka_2.13-2.7.0>bin\windows\kafka-console-consumer.bat --bootstrap-server localhost:9092 --topic R2
{"name":"P1","user":"U1","date":"2021-01-24T18:53:10.780+00:00","duration":5004}
{"name":"P1","user":"U2","date":"2021-01-24T18:53:11.003+00:00","duration":8236}
{"name":"P2","user":"U1","date":"2021-01-24T18:53:11.106+00:00","duration":8152}
{"name":"P2","user":"U2","date":"2021-01-24T18:53:11.207+00:00","duration":3991}
{"name":"P2","user":"U1","date":"2021-01-24T18:53:11.311+00:00","duration":3917}
{"name":"P2","user":"U1","date":"2021-01-24T18:53:11.415+00:00","duration":5603}
{"name":"P2","user":"U1","date":"2021-01-24T18:53:11.518+00:00","duration":5323}
```

Consumer & Producer : Function

application.properties

```
spring.cloud.stream.bindings.pageEventFunction-in-0.destination=R2
spring.cloud.stream.bindings.pageEventFunction-out-0.destination=R3
```

```
@Service
public class PageEventService {
 @Bean
 public Function<PageEvent,PageEvent> pageEventFunction(){
    return (input)->{
      input.setName("L:"+input.getName().length()); input.setUser("User => 1"); return input;
    };
}
}
```



C:\Windows\system32\cmd.exe - bin\windows\kafka-console-consumer.bat --bootstrap-server localhost:9092 --topic R2

```
{"name":"P2","user":"U2","date":"2021-01-24T19:47:23.188+00:00","duration":3764}
{"name":"P1","user":"U1","date":"2021-01-24T19:47:23.290+00:00","duration":2699}
{"name":"P2","user":"U1","date":"2021-01-24T19:47:23.393+00:00","duration":6024}
{"name":"P1","user":"U1","date":"2021-01-24T19:47:23.496+00:00","duration":8039}
{"name":"P2","user":"U1","date":"2021-01-24T19:47:23.598+00:00","duration":3470}
{"name":"P1","user":"U2","date":"2021-01-24T19:47:23.700+00:00","duration":7947}
{"name":"P1","user":"U2","date":"2021-01-24T19:47:23.802+00:00","duration":1840}
```

C:\Windows\system32\cmd.exe - bin\windows\kafka-console-consumer.bat --bootstrap-server localhost:9092 --topic R3

```
{"name":"Page:2","user":"User=>1","date":"2021-01-24T19:47:22.879+00:00","duration":6953}
{"name":"Page:2","user":"User=>1","date":"2021-01-24T19:47:22.982+00:00","duration":6786}
{"name":"Page:2","user":"User=>1","date":"2021-01-24T19:47:23.085+00:00","duration":654}
{"name":"Page:2","user":"User=>1","date":"2021-01-24T19:47:23.188+00:00","duration":3764}
{"name":"Page:2","user":"User=>1","date":"2021-01-24T19:47:23.290+00:00","duration":2699}
{"name":"Page:2","user":"User=>1","date":"2021-01-24T19:47:23.393+00:00","duration":6024}
{"name":"Page:2","user":"User=>1","date":"2021-01-24T19:47:23.496+00:00","duration":8039}
{"name":"Page:2","user":"User=>1","date":"2021-01-24T19:47:23.598+00:00","duration":3470}
{"name":"Page:2","user":"User=>1","date":"2021-01-24T19:47:23.700+00:00","duration":7947}
{"name":"Page:2","user":"User=>1","date":"2021-01-24T19:47:23.802+00:00","duration":1840}
```

Kafka Streams

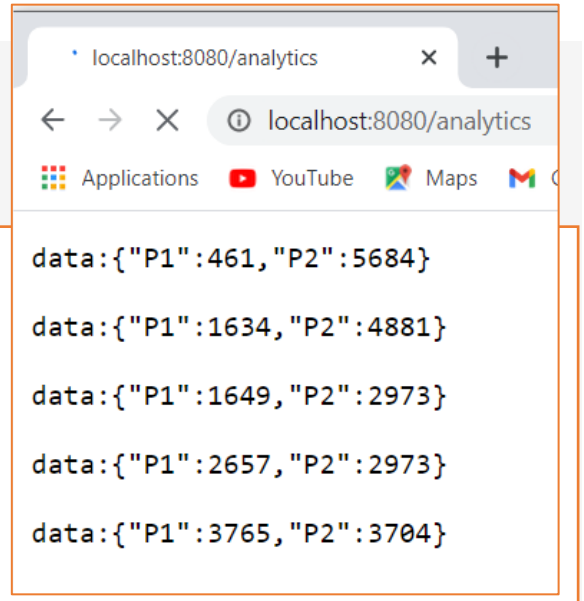
application.properties

```
spring.kafka.streams.application-id=app2
spring.cloud.stream.bindings.kStreamFunction-in-0.destination=R2
spring.cloud.stream.bindings.kStreamFunction-out-0.destination=R4
spring.cloud.stream.kafka.streams.binder.configuration.commit.interval.ms=1000
```

```
@Bean
public Function<KStream<String,PageEvent>, KStream<String,Long>> kStreamFunction(){
    return (input)->{
      return input
              .filter((k,v)->v.getDuration()>100)
              .map((k,v)->new KeyValue<>(v.getName(),v.getDuration()))
              .groupByKey(Grouped.with(Serdes.String(),Serdes.Long()))
              .windowedBy(TimeWindows.of(Duration.ofSeconds(1)))
              .reduce((acc, v) -> {long sum=(acc+v)/2;return sum; },Materialized.as("stats-store"))
              .toStream()
              .map((k,v)->new KeyValue<>("=>"+k.window().startTime()+k.window().endTime()+":"+k.key(),v));
    };
}
```

```
D:\Tools\kafka_2.13-2.7.0>bin\windows\kafka-console-consumer.bat --bootstrap-server localhost:9092 --topic R4 --property p
rint.key=true --property print.value=true --property key.deserializer=org.apache.kafka.common.serialization.StringDeserial
izer --property value.deserializer=org.apache.kafka.common.serialization.LongDeserializer
=>2021-01-24T19:47:30Z2021-01-24T19:47:31Z:P1   3339
=>2021-01-24T19:47:31Z2021-01-24T19:47:32Z:P2   4860
=>2021-01-24T19:47:31Z2021-01-24T19:47:32Z:P1   2670
=>2021-01-24T20:00:09Z2021-01-24T20:00:10Z:P1   1143
=>2021-01-24T20:00:09Z2021-01-24T20:00:10Z:P2   4336
=>2021-01-24T20:00:10Z2021-01-24T20:00:11Z:P1   4770
=>2021-01-24T20:00:10Z2021-01-24T20:00:11Z:P2   5803
=>2021-01-24T20:00:11Z2021-01-24T20:00:12Z:P1   1961
=>2021-01-24T20:00:11Z2021-01-24T20:00:12Z:P2   5052
```

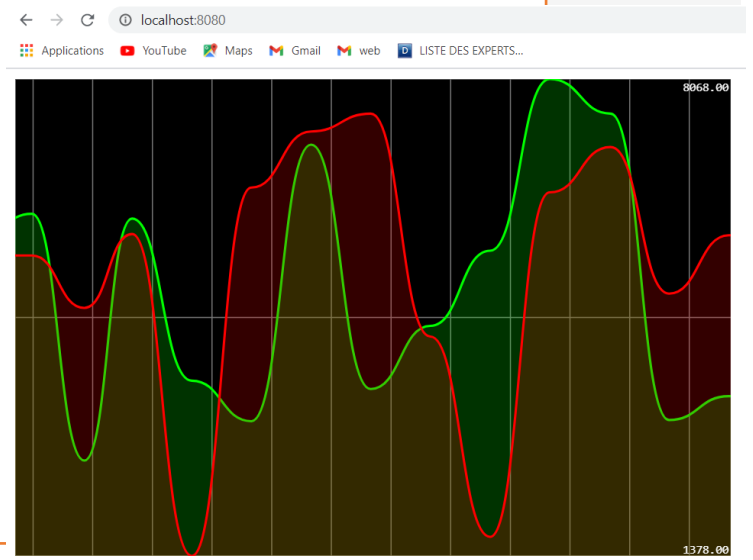
Interroger Kafka-streams Store Client Server Sent Event




```java
@RestController
public class PageEventRestController {
    @Autowired  private InteractiveQueryService interactiveQueryService;
    @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> stats =
                   interactiveQueryService.getQueryableStore("stats-store", QueryableStoreTypes.windowStore());
                    Instant now=Instant.now();
                    Instant from=now.minusMillis(5000);
                    KeyValueIterator<Windowed<String>, Long> fetchAll = stats.fetchAll(from, now);
                    //WindowStoreIterator<Long> fetch = stats.fetch(name, from, now);
                    while (fetchAll.hasNext()){
                        KeyValue<Windowed<String>, Long> next = fetchAll.next();
                        stringLongMap.put(next.key.key(),next.value);
                    }
                    return stringLongMap;
                }).share();
    }
}
```

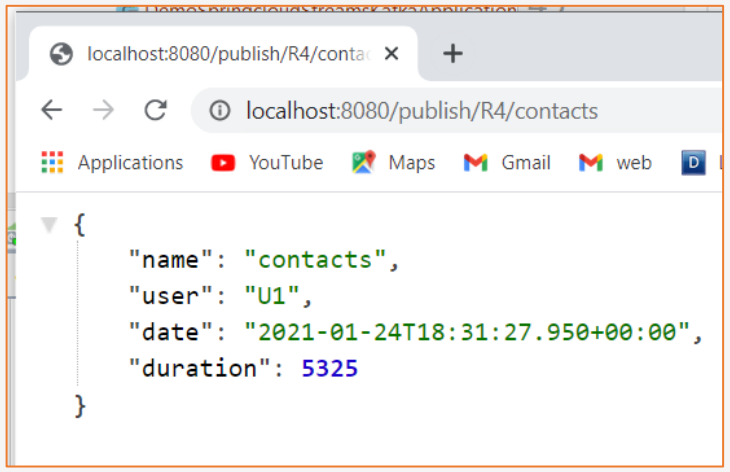
Client HTML Server Sent Event

```
<!DOCTYPE html><html lang="en">
<head>
    <meta charset="UTF-8"> <title>Analytics</title>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/smoothie/1.34.0/smoothie.min.js"></script>
</head>
<body>
<canvas id="chart2" width="600" height="400"></canvas>
<script>
    var index=-1;
    randomColor = function() { ++index;  if (index >= colors.length) index = 0; return colors[index];}
    var pages=["P1","P2"];
    var colors=[
        { sroke : 'rgba(0, 255, 0, 1)', fill : 'rgba(0, 255, 0, 0.2)' },{ sroke : 'rgba(255, 0, 0, 1)', fill : 'rgba(255, 0, 0, 0.2)'}
    ];
    var courbe = []; var smoothieChart = new SmoothieChart({tooltip: true});
    smoothieChart.streamTo(document.getElementById("chart2"), 500);
    pages.forEach(function(v){
        courbe[v]=new TimeSeries(); col = randomColor();
        smoothieChart.addTimeSeries(courbe[v], {strokeStyle : col.sroke, fillStyle : col.fill, lineWidth : 2
        });
    });
    var stockEventSource= new EventSource("/analytics");
    stockEventSource.addEventListener("message", function (event) {
        pages.forEach(function(v){
            val=JSON.parse(event.data)[v];
            courbe[v].append(new Date().getTime(),val);
        });
    });
</script>
</body>
</html>
</body>
</html>
```



Kafka Consumer

```java
@Data @NoArgsConstructor
@AllArgsConstructor @ToString
public class PageEvent {
    private String name;
    private String user;
    private Date date;
    private long duration;
}
```

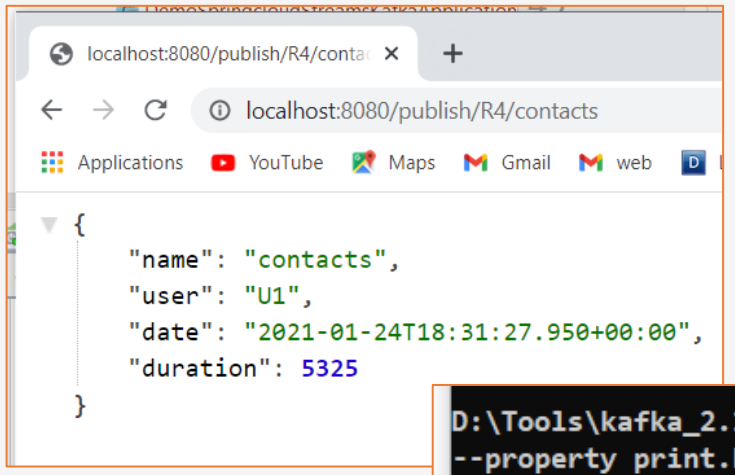



```java
@Service
public class PageEventService {

    @Bean
    public Consumer<PageEvent> pageEventConsumer(){
        return (input)->{
            System.out.println("**********************");
            System.out.println(input.toString());
            System.out.println("**********************");
        };
    }
    @Bean
    public Supplier<PageEvent> pageEventSupplier(){
        return ()-> new PageEvent(
                Math.random()>0.5?"P1":"P2",
                Math.random()>0.5?"U1":"U2",
                new Date(),
                new Random().nextInt(9000));
    }
    @Bean
    public Function<PageEvent,PageEvent> pageEventFunction(){
        return (input)->{
          input.setName("L:"+input.getName().length());
          input.setUser("UUUUUU");
          return input;
        };
    }
    @Bean
    public Function<KStream<String,PageEvent>, KStream<String,Long>>
```

Kafka-Stream et Spring Boot : Kafka Producer App

```java
@Data @NoArgsConstructor
@AllArgsConstructor @ToString
public class PageEvent {
    private String name;
    private String user;
    private Date date;
    private long duration;
}
```

```java
@RestController
public class PageEventRestController {
    @Autowired
    private StreamBridge streamBridge;
    @Autowired
    private InteractiveQueryService interactiveQueryService;
    @GetMapping("/publish/{topic}/{name}")
    public PageEvent publish(@PathVariable String topic, @PathVariable String name){
        PageEvent pageEvent=new PageEvent(name,Math.random()>0.5?"U1":"U2",new
Date(),new Random().nextInt(9000));
        streamBridge.send(topic,pageEvent);
        return pageEvent;
    }
}
```



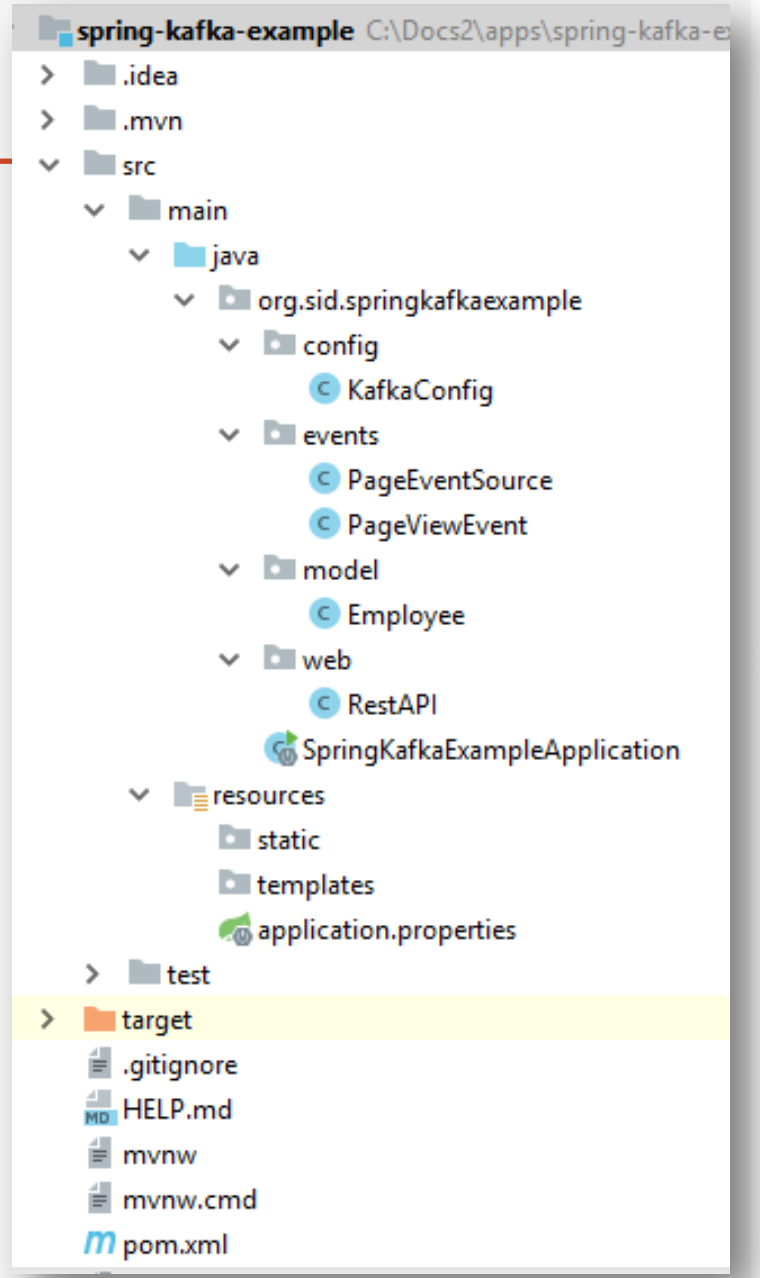

```
D:\Tools\kafka_2.13-2.7.0>bin\windows\kafka-console-consumer.bat --bootstrap-server localhost:9092 --topic R4
--property print.key=true --property print.value=true --property key.deserializer=org.apache.kafka.common.seri
alization.StringDeserializer --property value.deserializer=org.apache.kafka.common.serialization.StringDeseria
lizer
null    Hello
null    Test
null    Me
null    {"name":"contacts","user":"U1","date":"2021-01-24T18:31:27.950+00:00","duration":5325}
```

Kafka-Stream et Spring Boot : Kafka Producer App

```xml
<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>
```

application.properties

```
server.port=8083
```



Kafka-Stream et Spring Boot : Kafka Producer App

```java
package org.sid.springkafkaexample.events;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.ToString;
@Data
@AllArgsConstructor
@NoArgsConstructor
@ToString
public class PageViewEvent {
    private String userId;
    private String page;
    private int duration;
}
```

```java
package org.sid.springkafkaexample.web;
import org.sid.springkafkaexample.events.PageViewEvent;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.kafka.core.KafkaTemplate;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("kafka")
public class RestAPI {
    @Autowired
    private KafkaTemplate<String, PageViewEvent> kafkaTemplate;
    private String topic="pvts";
    @GetMapping("/publish/{user}-{page}-{duration}")
    public PageViewEvent publishMessage(
            @PathVariable String user,
            @PathVariable String page,
            @PathVariable int duration){
        PageViewEvent pageViewEvent=new PageViewEvent(user,page,duration);
        kafkaTemplate.send(topic,pageViewEvent.getUserId(),pageViewEvent);
        return pageViewEvent;
    }
}
```

Kafka-Stream et Spring Boot : Kafka Producer App

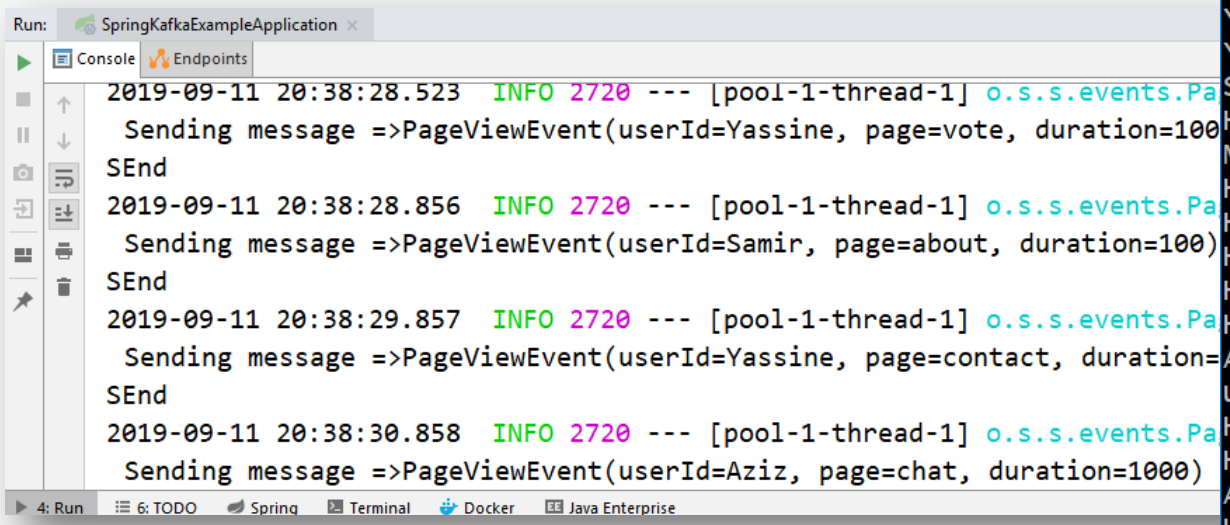
```
package org.sid.springkafkaexample.events;

import lombok.extern.slf4j.Slf4j; import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.ApplicationArguments; import org.springframework.boot.ApplicationRunner;
import org.springframework.kafka.core.KafkaTemplate; import org.springframework.stereotype.Component;
import java.util.Arrays; import java.util.List; import java.util.Random;
import java.util.concurrent.Executors; import java.util.concurrent.TimeUnit;
@Component
@Slf4j
public  class PageEventSource implements ApplicationRunner {
    @Autowired
    private KafkaTemplate<String,PageViewEvent>  kafkaTemplate;
    @Override
    public void run(ApplicationArguments applicationArguments)  {
        System.out.println("..................................................;");
        List<String> names= Arrays.asList("Hassan","Mohamed","Hanane","Yassine","Samir","Aziz");
        List<String> pages= Arrays.asList("blog","chat","profile","about","contact","vote","search");
        Runnable runnable=()->{
            String rPage=pages.get(new Random().nextInt(pages.size()));
            String rName=names.get(new Random().nextInt(names.size()));
            PageViewEvent pageViewEvent=new PageViewEvent(rName,rPage,Math.random()>0.5?100:1000);
// PageViewEvent pageViewEvent=new PageViewEvent(rName,rPage,100+(int)(Math.random()*1000));
            System.out.println("SEnd");
            kafkaTemplate.send("pvts",pageViewEvent.getUserId(),pageViewEvent);
            log.info("Sending message =>"+pageViewEvent.toString());
        };
        System.out.println("-----------------------------------------");
        Executors.newScheduledThreadPool(1).scheduleAtFixedRate(runnable,1,1, TimeUnit.SECONDS);
//Executors.newScheduledThreadPool(1).scheduleAtFixedRate(runnable,100,100, TimeUnit.MILLISECONDS);
    }
}
```

Kafka-Stream et Spring Boot : Kafka Producer App

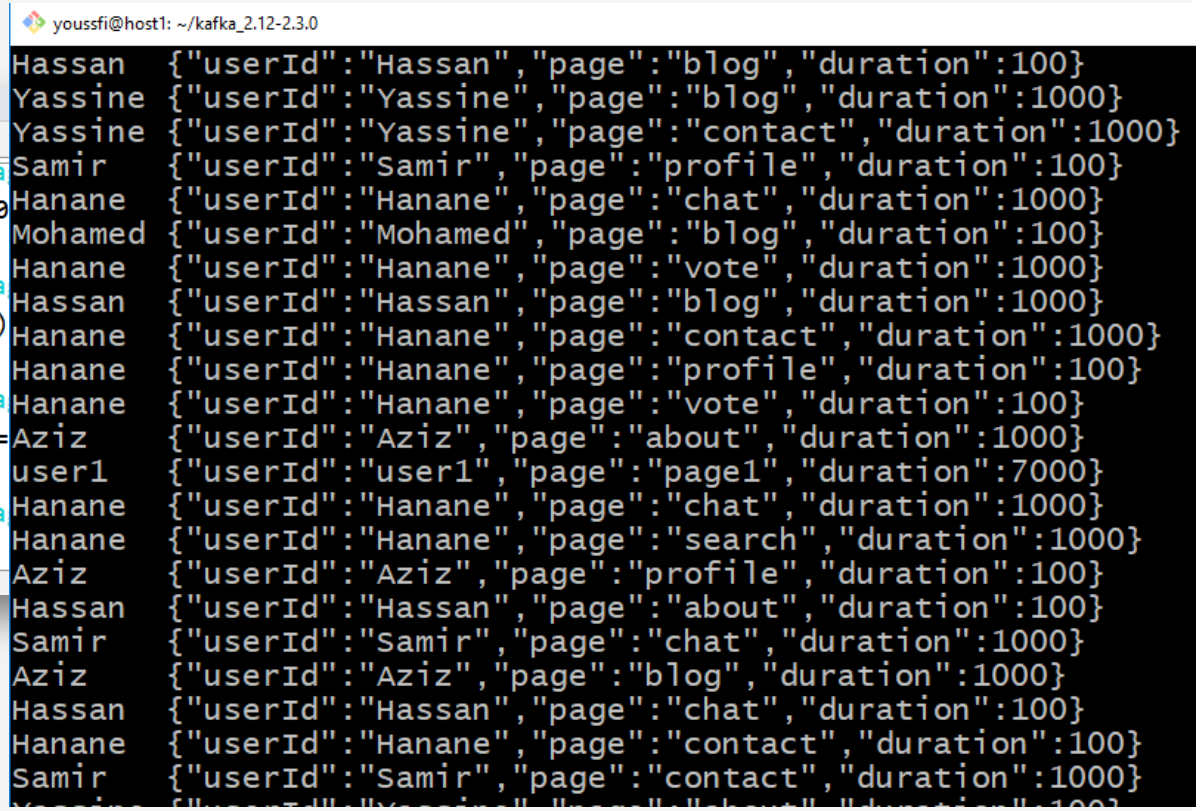
```
package org.sid.springkafkaexample.config;

import org.apache.kafka.clients.producer.ProducerConfig; import org.apache.kafka.common.serialization.StringSerializer;
import org.sid.springkafkaexample.events.PageViewEvent; import org.sid.springkafkaexample.model.Employee;
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, PageViewEvent> producerFactory(){
        Map<String,Object> config=new HashMap<>();
        config.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,"192.168.57.3:9092");
        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, PageViewEvent> kafkaTemplate(){
        return new KafkaTemplate<>(producerFactory());
    }
}
```

Kafka-Stream et Spring Boot : Kafka Producer App

Producer App

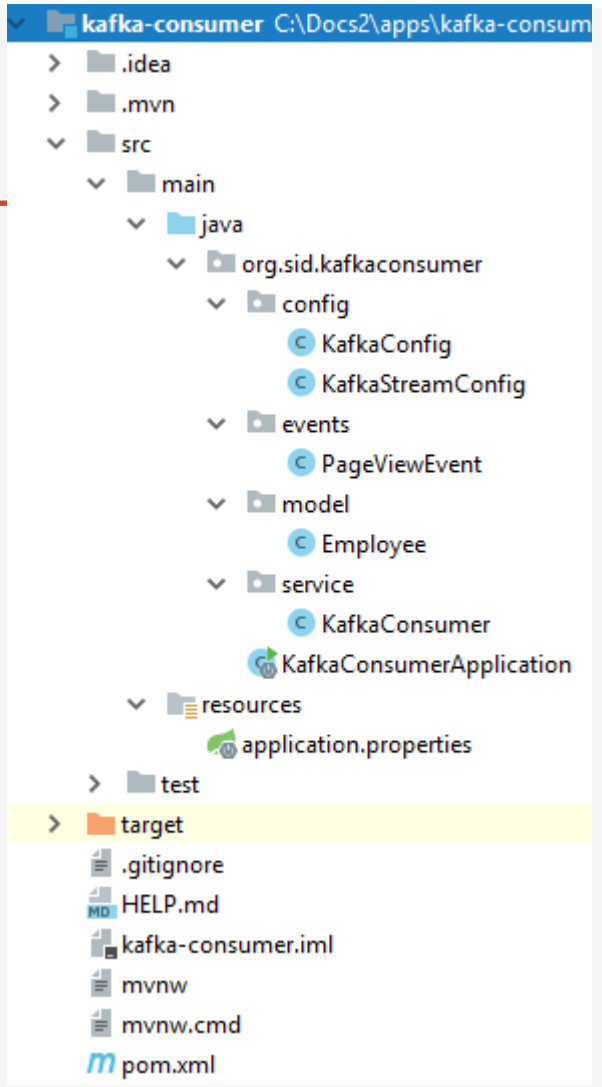


Kafka Consumer Console



Kafka-Stream et Spring Boot : Kafka Stream Appli

```xml
<dependency>
    <groupId>org.springframework.kafka</groupId>
    <artifactId>spring-kafka</artifactId>
</dependency>
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.9.6</version>
</dependency>
<dependency>
    <groupId>org.apache.kafka</groupId>
    <artifactId>kafka-streams</artifactId>
</dependency>

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



application.properties

```
#spring.kafka.consumer.bootstrap-servers=192.168.57.3:9092
#spring.kafka.consumer.group-id=enset_uh2c
```

Kafka-Stream et Spring Boot : Kafka Stream Appli

```
package org.sid.kafkaconsumer.config;
import com.fasterxml.jackson.databind.ObjectMapper; import org.apache.kafka.common.serialization.Serdes;
import org.apache.kafka.streams.KeyValue; import org.apache.kafka.streams.StreamsConfig;
import org.apache.kafka.streams.kstream.*; import org.sid.kafkaconsumer.events.PageViewEvent;
import org.springframework.context.annotation.Bean; import org.springframework.context.annotation.Configuration;
import org.springframework.kafka.annotation.EnableKafka; import org.springframework.kafka.annotation.EnableKafkaStreams;
import org.springframework.kafka.annotation.KafkaStreamsDefaultConfiguration;import org.springframework.kafka.config.KafkaStreamsConfiguration;
import org.springframework.kafka.config.StreamsBuilderFactoryBean; import java.io.IOException; import java.util.HashMap; import java.util.Map;
@Configuration
@EnableKafka
@EnableKafkaStreams
public class KafkaStreamConfig {
    ObjectMapper objectMapper=new ObjectMapper();
    @Bean(name = KafkaStreamsDefaultConfiguration.DEFAULT_STREAMS_CONFIG_BEAN_NAME)
    public KafkaStreamsConfiguration kafkaStreamsConfiguration(){
        Map<String,Object> conf=new HashMap<>();
       conf.put(StreamsConfig.APPLICATION_ID_CONFIG, "Page-Event-Stream");
       conf.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "192.168.57.3:9092");
       conf.put(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONFIG, Serdes.String().getClass());
       conf.put(StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG, Serdes.String().getClass());
       conf.put(StreamsConfig.COMMIT_INTERVAL_MS_CONFIG, 1000);
        return new KafkaStreamsConfiguration(conf);
    }
 @Bean
    public StreamsBuilderFactoryBean factoryBean(){
       return new StreamsBuilderFactoryBean(kafkaStreamsConfiguration());
    }
```

Kafka-Stream et Spring Boot : Kafka Stream Appli

```
    @Bean
    public KStream<String, Integer> process() throws Exception{
        KStream<String, String> stream = factoryBean().getObject().stream("pvts");
        KStream<String, Integer> counts = stream
                .map((k, v) -> new KeyValue<>(k, pageViewEvent(v)))
                .filter((k, v) -> v.getDuration() > 200)
                .map((k,v)->new KeyValue<>(k,v.getDuration()));
                //.groupByKey()
                //.windowedBy(TimeWindows.of(4000))
                //.count(Materialized.as("Page_Count")).toStream();
        counts.to("pvs-out-5",Produced.valueSerde(Serdes.Integer()));
        //stream.map((k,v)->new KeyValue<>(v,k)).to("xxx");
        return counts;
    }
    public PageViewEvent pageViewEvent(String strObject){
        PageViewEvent pageViewEvent = new PageViewEvent();
        try {
            pageViewEvent = objectMapper.readValue(strObject, PageViewEvent.class);
        } catch (IOException e) {
            e.printStackTrace();
        }
        return pageViewEvent;
    }

}
```

Kafka-Stream et Spring Boot : Kafka Stream Appli

```
$ bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic pvs-out-5 --
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.IntegerDeserializer
Mohamed 1000
Hassan  1000
Yassine 1000
Mohamed 1000
Mohamed 1000
Yassine 1000
Aziz    1000
```

Kafka-Stream et Spring Boot : Kafka Stream Appli

```
@Configuration
@EnableKafka
@EnableKafkaStreams
public class KafkaStreamConfig {
    ObjectMapper objectMapper=new ObjectMapper();
    @Bean(name = KafkaStreamsDefaultConfiguration.DEFAULT_STREAMS_CONFIG_BEAN_NAME)
    public KafkaStreamsConfiguration kafkaStreamsConfiguration(){
        Map<String,Object> conf=new HashMap<>();
       conf.put(StreamsConfig.APPLICATION_ID_CONFIG, "Page-Event-Stream");
       conf.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "192.168.57.3:9092");
       conf.put(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONFIG, Serdes.String().getClass());
       conf.put(StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG,
Serdes.String().getClass());
       conf.put(StreamsConfig.COMMIT_INTERVAL_MS_CONFIG, 1000);
        conf.put(StreamsConfig.POLL_MS_CONFIG, 1000);
        return new KafkaStreamsConfiguration(conf);
    }
   @Bean
    public StreamsBuilderFactoryBean factoryBean(){
       return new StreamsBuilderFactoryBean(kafkaStreamsConfiguration());
    }
```

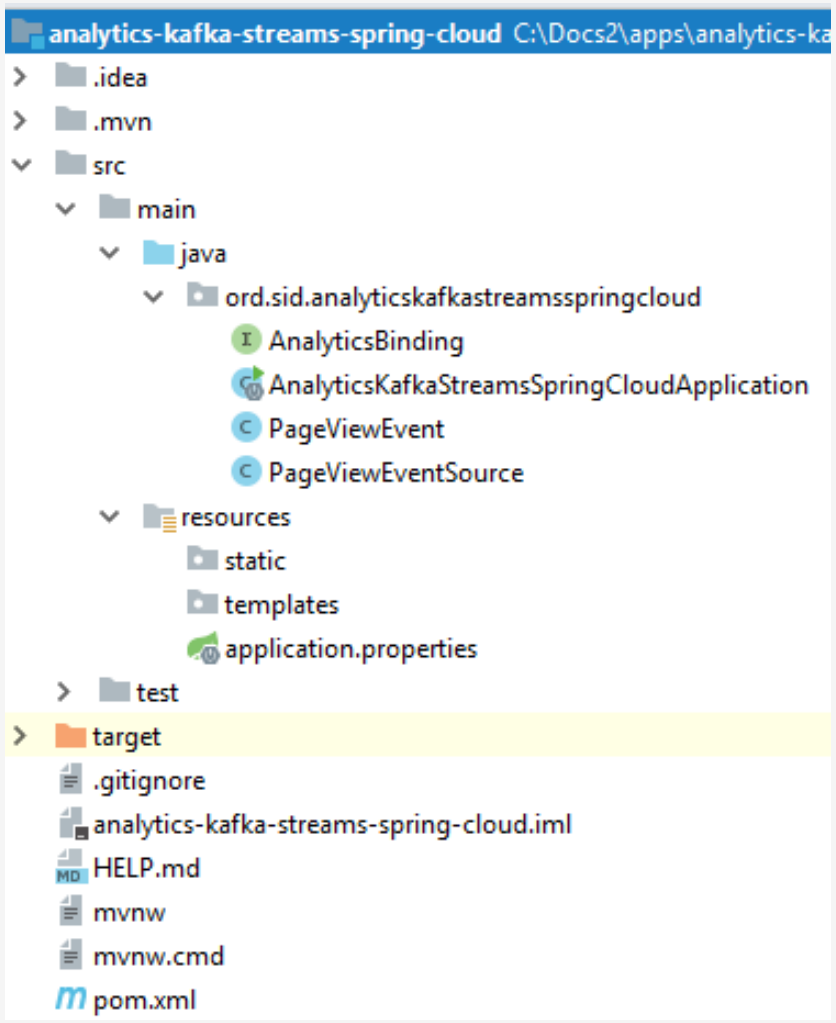
Kafka-Stream et Spring Boot : Kafka Stream Appli

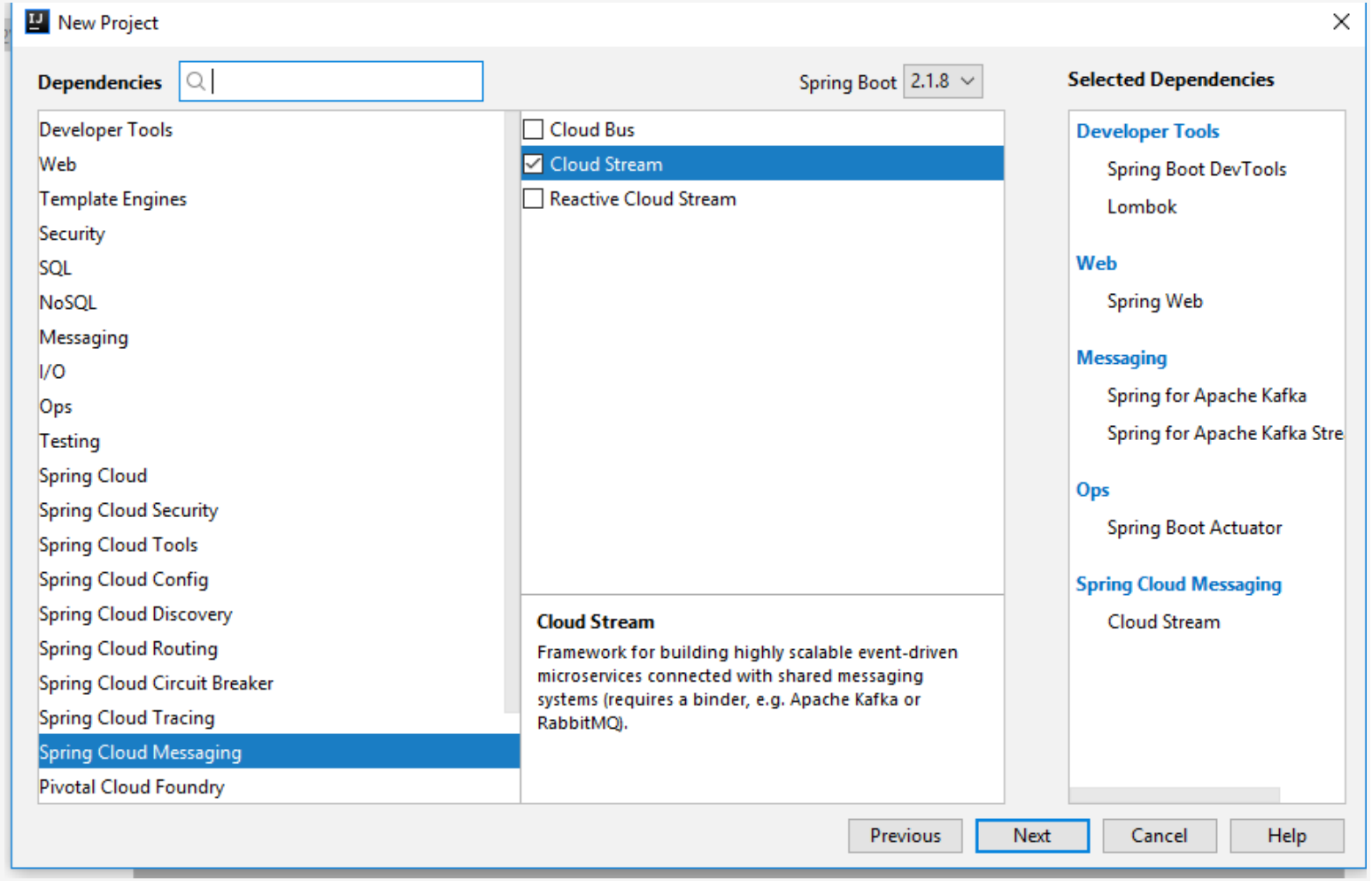
```
@Bean
    public KStream<String, Long> process() throws Exception{
        KStream<String, String> stream = factoryBean().getObject().stream("pvts");
        KStream<String, Long> counts = stream
                .map((k, v) -> new KeyValue<>(k, pageViewEvent(v)))
                .filter((k, v) -> v.getDuration() > 80)
                .map((k,v)->new KeyValue<>(v.getPage(),"0"))
                .groupByKey()
                .windowedBy(TimeWindows.of(5000))
                .count(Materialized.as("Page_Count")).toStream()
                .map((k,v)->new KeyValue<>(k.key(),v));
        counts.to("pvs-out-6",Produced.valueSerde(Serdes.Long()));
        return counts;
    }
    public PageViewEvent pageViewEvent(String strObject){
        PageViewEvent pageViewEvent = new PageViewEvent();
        try {
             pageViewEvent = objectMapper.readValue(strObject, PageViewEvent.class);
             System.out.println(pageViewEvent.getPage());
           } catch (IOException e) {
            e.printStackTrace();
        }
        return pageViewEvent;
    }
}
```

Kafka-Stream et Spring Boot : Kafka Stream Appli

```
$ bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic pvs-out-6 --
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
search  8
blog    5
chat    4
vote    5
contact 7
profile 6

contact 9
vote    6
blog    6
search  10
chat    7
about   1

blog    1
chat    1
...
```

Spring Cloud avec Kafka





Spring Cloud avec Kafka

```xml
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-actuator</artifactId>
    </dependency>
    <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.springframework.boot</groupId>
        <artifactId>spring-boot-devtools</artifactId>
        <scope>runtime</scope>
        <optional>true</optional>
    </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.cloud</groupId>
        <artifactId>spring-cloud-stream-test-support</artifactId>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.springframework.kafka</groupId>
        <artifactId>spring-kafka-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>
```

Spring Cloud avec Kafka

PageViewEvent.java

```
package ord.sid.analyticskafkastreamsspringcloud;

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

@Data
@NoArgsConstructor
@AllArgsConstructor
@ToString
public class PageViewEvent{
    private String userId,page;
    private long duration;
}
```

Spring Cloud avec Kafka

AnalyticsBindings.java

```
package ord.sid.analyticskafkastreamsspringcloud;

import org.springframework.cloud.stream.annotation.Output;
import org.springframework.messaging.MessageChannel;

public interface AnalyticsBinding{
    String PAGE_VIEWS_OUT="pvout";
    @Output(PAGE_VIEWS_OUT)
    MessageChannel pageViewsOut();
}
```

Spring Cloud avec Kafka

PageViewEventSource.java

```
package ord.sid.analyticskafkastreamsspringcloud;
import lombok.extern.slf4j.Slf4j; import org.springframework.boot.ApplicationArguments; import org.springframework.boot.ApplicationRunner;
import org.springframework.kafka.support.KafkaHeaders; import org.springframework.messaging.Message;
import org.springframework.messaging.MessageChannel; import org.springframework.messaging.support.MessageBuilder;
import org.springframework.stereotype.Component; import java.util.Arrays; import java.util.List; import java.util.Random;
import java.util.concurrent.Executors; import java.util.concurrent.TimeUnit;
@Component
@Slf4j
public class PageViewEventSource implements ApplicationRunner {
    private MessageChannel pageViewsOutChannel;
    @SuppressWarnings("SpringJavaInjectionPointsAutowiringInspection")
    public PageViewEventSource(AnalyticsBinding analyticsBinding) {
        this.pageViewsOutChannel = analyticsBinding.pageViewsOut();
    }
```

Spring Cloud avec Kafka

PageViewEventSource.java

```
@Override
    public void run(ApplicationArguments args) throws Exception {
        System.out.println("..................................................;");
        List<String> names= Arrays.asList("Hassan","Mohamed","Hanane","Yassine","Samir","Aziz");
        List<String> pages= Arrays.asList("blog","chat","profile","about","contact","vote","search");
        Runnable runnable=()->{
            String rPage=pages.get(new Random().nextInt(pages.size()));
            String rName=names.get(new Random().nextInt(names.size()));
            PageViewEvent pageViewEvent=new PageViewEvent(rName,rPage,100+(int)(Math.random()*1000));
            Message<PageViewEvent> eventMessage = MessageBuilder
                    .withPayload(pageViewEvent)
                    .setHeader(KafkaHeaders.MESSAGE_KEY, pageViewEvent.getUserId().getBytes())
                    .build();
            try {
                this.pageViewsOutChannel.send(eventMessage); log.info("Sending"+eventMessage.toString());
            } catch (Exception e) {              e.printStackTrace();
            }
            log.info("Sending message =>"+pageViewEvent.toString());
        };
        System.out.println("-------------------------------------------");
        Executors.newScheduledThreadPool(1).scheduleAtFixedRate(runnable,1000,1000, TimeUnit.MILLISECONDS);
    }
}
```

Spring Cloud avec Kafka

application.properties

```
#defaults
spring.cloud.stream.kafka.streams.binder.configuration.commit.interval.mms=1000
spring.cloud.stream.kafka.streams.binder.configuration.default.key.serde=org.apache.kafka.common.serialization.Serdes$StringSerde
spring.cloud.stream.kafka.streams.binder.configuration.default.value.serde=org.apache.kafka.common.serialization.Serdes$StringSerde

# Page view out Bindings

spring.cloud.stream.bindings.pvout.destination=pvs
spring.cloud.stream.bindings.pvout.producer.header-mode=raw
# Kafka Brokers hosts
spring.kafka.bootstrap-servers=192.168.57.3:9092
```

Spring Cloud avec Kafka

```
package ord.sid.analyticskafkastreamsspringcloud;
import lombok.extern.slf4j.Slf4j;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.stream.annotation.EnableBinding;
@SpringBootApplication
@EnableBinding(AnalyticsBinding.class)
@Slf4j
public class AnalyticsKafkaStreamsSpringCloudApplication {
    public static void main(String[] args) {
        SpringApplication.run(AnalyticsKafkaStreamsSpringCloudApplication.class, args);
    }
}
```

Spring Cloud avec Kafka

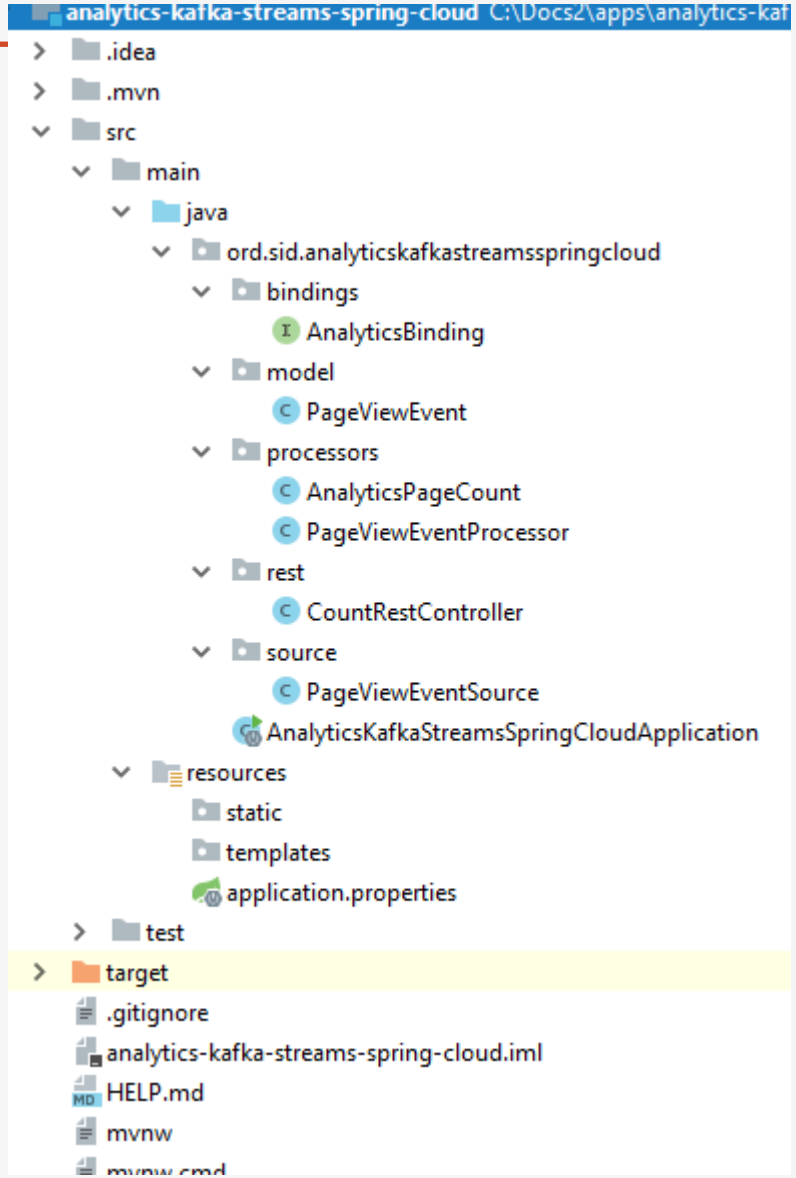
Application Console

```
2019-09-12 17:28:28.624  INFO 14516 --- [pool-1-thread-1] lyticsKafkaStreamsSpringCloudApplication : Sending message
GenericMessage [payload=PageViewEvent(userId=Hanane, page=about, duration=593), headers={id=e129e113-f7ee-a068-a603-
797eb1f9a1f3, kafka_messageKey=[B@7c2c7662, contentType=application/json, timestamp=1568305708500}]
2019-09-12 17:28:28.624  INFO 14516 --- [pool-1-thread-1] lyticsKafkaStreamsSpringCloudApplication : Sending message
=>PageViewEvent(userId=Hanane, page=about, duration=593)
2019-09-12 17:28:29.492  INFO 14516 --- [pool-1-thread-1] lyticsKafkaStreamsSpringCloudApplication : Sending message
GenericMessage [payload=PageViewEvent(userId=Hanane, page=chat, duration=1046), headers={id=6a6b0911-efd9-52e6-ab2d-
caa38db80e97, kafka_messageKey=[B@321d4e3, contentType=application/json, timestamp=1568305709492}]
```

Kafka Consumer Console

```
$ bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic pvs --
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.StringDeserializer
Hanane  {"userId":"Hanane","page":"search","duration":926}
Hanane  {"userId":"Hanane","page":"chat","duration":955}
Hassan  {"userId":"Hassan","page":"search","duration":403}
Hanane  {"userId":"Hanane","page":"contact","duration":140}
Mohamed {"userId":"Mohamed","page":"contact","duration":126}
Aziz    {"userId":"Aziz","page":"profile","duration":1069}
Hassan  {"userId":"Hassan","page":"search","duration":879}
Yassine {"userId":"Yassine","page":"about","duration":999}
Samir   {"userId":"Samir","page":"search","duration":195}
```

Spring Cloud avec Kafka : Partie Spring Cloud Kafka Streams



Spring Cloud avec Kafka : Partie Spring Cloud Kafka Streams

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-actuator</artifactId>
    </dependency>
    <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.springframework.boot</groupId>
        <artifactId>spring-boot-devtools</artifactId>
        <scope>runtime</scope>
        <optional>true</optional>
    </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.cloud</groupId>
        <artifactId>spring-cloud-stream-test-support</artifactId>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.springframework.kafka</groupId>
        <artifactId>spring-kafka-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>
```

Spring Cloud avec Kafka : Partie Spring Cloud Kafka Streams

```java
package ord.sid.analyticskafkastreamsspringcloud.model;

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

@Data
@NoArgsConstructor
@AllArgsConstructor
@ToString
public class PageViewEvent{
    private String userId,page;
    private long duration;
}
```

```java
package ord.sid.analyticskafkastreamsspringcloud.bindings;

import
ord.sid.analyticskafkastreamsspringcloud.model.PageViewEvent;
import org.apache.kafka.streams.kstream.KStream;
import org.apache.kafka.streams.kstream.KTable;
import org.springframework.cloud.stream.annotation.Input;
import org.springframework.cloud.stream.annotation.Output;
import org.springframework.messaging.MessageChannel;

public interface AnalyticsBinding{
    String PAGE_VIEWS_OUT="pvout";
    String PAGE_VIEWS_IN="pvin";
    String PAGE_VIEW_COUNT_OUT = "pcout";
    String PAGE_VIEW_COUNT_IN = "pcin";
    String PAGE_COUNT_MV = "PAGE_COUNT";

    @Output(PAGE_VIEWS_OUT)
    MessageChannel pageViewsOut();
    @Input(PAGE_VIEWS_IN)
    KStream<String, PageViewEvent> pageViewsIn();

    @Output(PAGE_VIEW_COUNT_OUT)
    KStream<String,Long> pageCountOutput();
    @Input(PAGE_VIEW_COUNT_IN)
    KTable<String,Long> pageCountInput();
}
```

Spring Cloud avec Kafka : Partie Spring Cloud Kafka Streams

```
package
ord.sid.analyticskafkastreamsspringcloud.source;

import lombok.extern.slf4j.Slf4j;
import
ord.sid.analyticskafkastreamsspringcloud.bindings.Analy
ticsBinding;
import
ord.sid.analyticskafkastreamsspringcloud.model.PageView
Event;
import org.springframework.boot.ApplicationArguments;
import org.springframework.boot.ApplicationRunner;
import org.springframework.kafka.support.KafkaHeaders;
import org.springframework.messaging.Message;
import org.springframework.messaging.MessageChannel;
import
org.springframework.messaging.support.MessageBuilder;
import org.springframework.stereotype.Component;

import java.util.Arrays;
import java.util.List;
import java.util.Random;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;
```

```
@Component
@Slf4j
public class PageViewEventSource implements ApplicationRunner {
    private MessageChannel pageViewsOutChannel;
    @SuppressWarnings("SpringJavaInjectionPointsAutowiringInspection")
    public PageViewEventSource(AnalyticsBinding analyticsBinding) {
        this.pageViewsOutChannel = analyticsBinding.pageViewsOut();
    }
    @Override
    public void run(ApplicationArguments args) throws Exception {
        System.out.println("..................................................;");
        List<String> names= Arrays.asList("Hassan","Mohamed","Hanane","Yassine","Samir","Aziz");
        List<String> pages= Arrays.asList("blog","chat","profile","about","contact","vote","search");
        Runnable runnable=()->{
            String rPage=pages.get(new Random().nextInt(pages.size()));
            String rName=names.get(new Random().nextInt(names.size()));
            PageViewEvent pageViewEvent=new PageViewEvent(rName,rPage,100+(int)(Math.random()*1000));
            //kafkaTemplate.send("pvts",pageViewEvent.getUserId(),pageViewEvent);
            Message<PageViewEvent> eventMessage = MessageBuilder
                    .withPayload(pageViewEvent)
                    .setHeader(KafkaHeaders.MESSAGE_KEY, pageViewEvent.getUserId())
                    .build();
            try {
                this.pageViewsOutChannel.send(eventMessage);
                log.info("Sending message "+eventMessage.toString());
            } catch (Exception e) {
                e.printStackTrace();
            }
        };
        System.out.println("-----------------------------------------");
        Executors.newScheduledThreadPool(1).scheduleAtFixedRate(runnable,1000,1000, TimeUnit.MILLISECONDS);
    }
}
```

Spring Cloud avec Kafka : Partie Spring Cloud Kafka Streams

```
package ord.sid.analyticskafkastreamsspringcloud.processors;

import lombok.extern.slf4j.Slf4j; import ord.sid.analyticskafkastreamsspringcloud.bindings.AnalyticsBinding;
import ord.sid.analyticskafkastreamsspringcloud.model.PageViewEvent; import org.apache.kafka.streams.KeyValue;
import org.apache.kafka.streams.kstream.KStream; import org.apache.kafka.streams.kstream.KTable;
import org.apache.kafka.streams.kstream.Materialized; import org.apache.kafka.streams.kstream.TimeWindows;
import org.springframework.cloud.stream.annotation.Input; import org.springframework.cloud.stream.annotation.StreamListener;
import org.springframework.messaging.handler.annotation.SendTo; import org.springframework.stereotype.Component; import java.util.Date;
@Component
@Slf4j
public class PageViewEventProcessor {
    @StreamListener
    @SendTo({AnalyticsBinding.PAGE_VIEW_COUNT_OUT})
    public KStream<String, Long> process(@Input(AnalyticsBinding.PAGE_VIEWS_IN) KStream<String, PageViewEvent>
events){
        return events
                .filter((k, v) -> v.getDuration() > 5)
                //.map((k,v)->new KeyValue<>(v.getPage(),"0"));
                .map((k, v) -> new KeyValue<>(v.getPage(), "0"))
                .groupByKey()
                //.windowedBy(TimeWindows.of(10000))
                .count(Materialized.as(AnalyticsBinding.PAGE_COUNT_MV)).toStream();
//.map((k,v)->new KeyValue<>("["+new Date(k.window().start())+"=>"+new Date(k.window().end())+"]"+":=>"+k.key(),v));

    }
}
```

Spring Cloud avec Kafka : Partie Spring Cloud Kafka Streams

```
#spring.cloud.stream.kafka.streams.binder.application-id=page-views-count-app-2
spring.kafka.producer.key-serializer=org.apache.kafka.common.serialization.StringSerializer
spring.kafka.consumer.key-deserializer=org.apache.kafka.common.serialization.StringSerializer
#defaults
spring.cloud.stream.kafka.streams.binder.configuration.commit.interval.mms=1000
spring.cloud.stream.kafka.streams.binder.configuration.default.key.serde=org.apache.kafka.common.serialization.Serdes$StringSerde
spring.cloud.stream.kafka.streams.binder.configuration.default.value.serde=org.apache.kafka.common.serialization.Serdes$StringSerde
# Page view out Bindings
spring.cloud.stream.bindings.pvout.destination=pvs
spring.cloud.stream.bindings.pvout.producer.header-mode=raw
#Page view in
spring.cloud.stream.bindings.pvin.destination=pvs
spring.cloud.stream.bindings.pvin.consumer.header-mode=raw
spring.cloud.stream.kafka.streams.bindings.pvin.consumer.application-id=page-views-app-id
# Kafka Brokers hosts
spring.kafka.bootstrap-servers=192.168.57.3:9092
#Page Count out
spring.cloud.stream.bindings.pcout.destination=pcs2
spring.cloud.stream.bindings.pcout.producer.use-native-encoding=true
spring.cloud.stream.kafka.streams.bindings.pcout.producer.key-serde=org.apache.kafka.common.serialization.Serdes$StringSerde
spring.cloud.stream.kafka.streams.bindings.pcout.producer.value-serde=org.apache.kafka.common.serialization.Serdes$LongSerde
spring.cloud.stream.bindings.pcout.producer.header-mode=raw
#Page Count In
spring.cloud.stream.bindings.pcin.destination=pcs2
spring.cloud.stream.bindings.pcin.consumer.use-native-decoding=true
spring.cloud.stream.bindings.pcin.group=pcs-gr
#spring.cloud.stream.bindings.pcin.content-type=application/json
spring.cloud.stream.kafka.streams.bindings.pcin.consumer.key-serde=org.apache.kafka.common.serialization.Serdes$StringSerde
spring.cloud.stream.kafka.streams.bindings.pcin.consumer.value-serde=org.apache.kafka.common.serialization.Serdes$LongSerde
spring.cloud.stream.bindings.pcin.consumer.header-mode=raw
spring.cloud.stream.kafka.streams.bindings.pcin.consumer.application-id=analytics-app-id
```

Spring Cloud avec Kafka : Partie Spring Cloud Kafka Streams

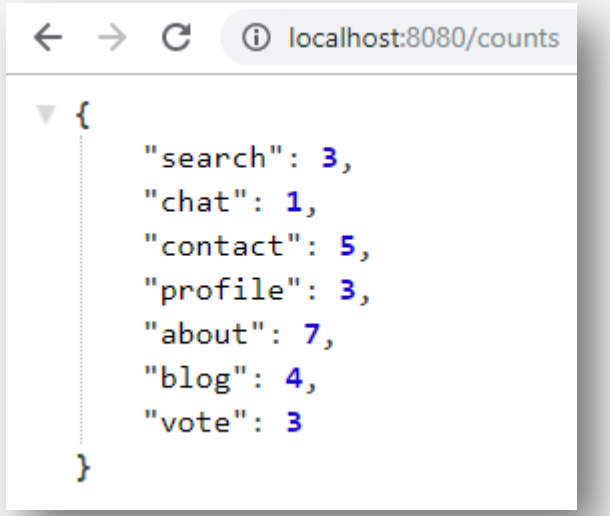
```
package ord.sid.analyticskafkastreamsspringcloud.processors;
import lombok.extern.slf4j.Slf4j;
import ord.sid.analyticskafkastreamsspringcloud.bindings.AnalyticsBinding;
import org.apache.kafka.streams.kstream.KTable;
import org.springframework.cloud.stream.annotation.Input;
import org.springframework.cloud.stream.annotation.StreamListener;
import org.springframework.stereotype.Component;
@Component
@Slf4j
public class AnalyticsPageCount {
  @StreamListener
  public void processData(@Input(AnalyticsBinding.PAGE_VIEW_COUNT_IN) KTable<String,Long> counts){
      counts.toStream().foreach((k,v)->{
          log.info(k+"=>"+v);
      });
  }
}
```

Spring Cloud avec Kafka : Partie Spring Cloud Kafka Streams

```
package ord.sid.analyticskafkastreamsspringcloud.rest;
import ord.sid.analyticskafkastreamsspringcloud.bindings.AnalyticsBinding;import org.apache.kafka.streams.KeyValue;
import org.apache.kafka.streams.state.KeyValueIterator;import org.apache.kafka.streams.state.QueryableStoreTypes;
import org.apache.kafka.streams.state.ReadOnlyKeyValueStore;import org.springframework.cloud.stream.binder.kafka.streams.InteractiveQueryService;
import org.springframework.web.bind.annotation.GetMapping;import org.springframework.web.bind.annotation.RestController;
import java.util.HashMap;import java.util.Map;
@RestController
public class CountRestController {
    //@Depricated private QueryableStoreRegistry queryableStoreRegistry;
    private InteractiveQueryService queryableStoreRegistry;
    public CountRestController(InteractiveQueryService queryableStoreRegistry) {
        this.queryableStoreRegistry = queryableStoreRegistry;
    }
    @GetMapping("/counts")
    public Map<String,Long> counts(){
        Map<String,Long> results=new HashMap<>();
        ReadOnlyKeyValueStore<String, Long> queryableStoreType =
this.queryableStoreRegistry.getQueryableStore(AnalyticsBinding.PAGE_COUNT_MV, QueryableStoreTypes.keyValueStore());
        KeyValueIterator<String,Long> all=queryableStoreType.all();
        while(all.hasNext()){
            KeyValue<String,Long> item=all.next();
            results.put(item.key,item.value);
        }
        return results;
    }
}
```

Spring Cloud avec Kafka : WindowStore

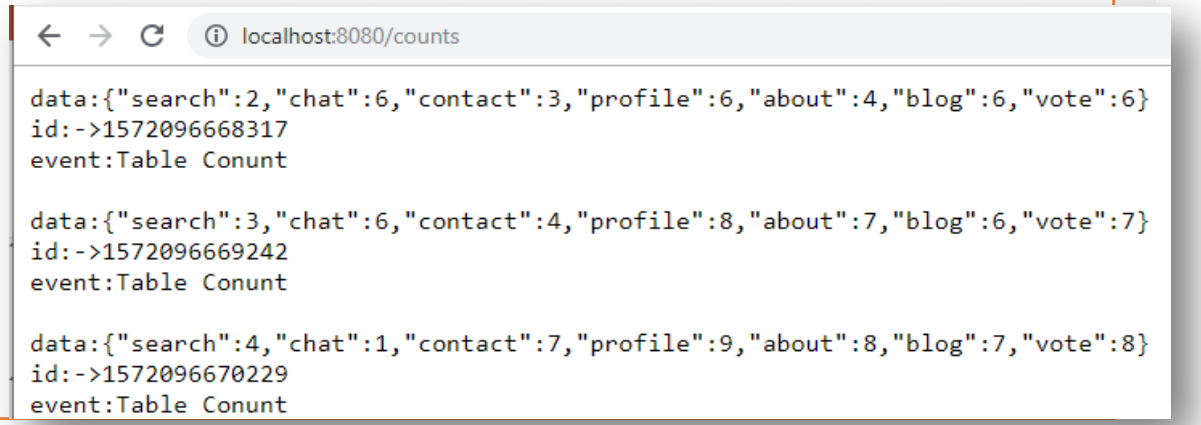
```
public Map<String,Long> counts(){
    Map<String,Long> results=new HashMap<>();
    ReadOnlyWindowStore<String, Long> queryableStoreType =
this.queryableStoreRegistry.getQueryableStore(AnalyticsBinding.PAGE_COUNT_MV,
QueryableStoreTypes.windowStore());
    KeyValueIterator<Windowed<String>,Long> all=queryableStoreType.all();

    while(all.hasNext()){
        KeyValue<Windowed<String>,Long> item=all.next();
        results.put(item.key.key(),item.value);
    }
    return results;
}
```



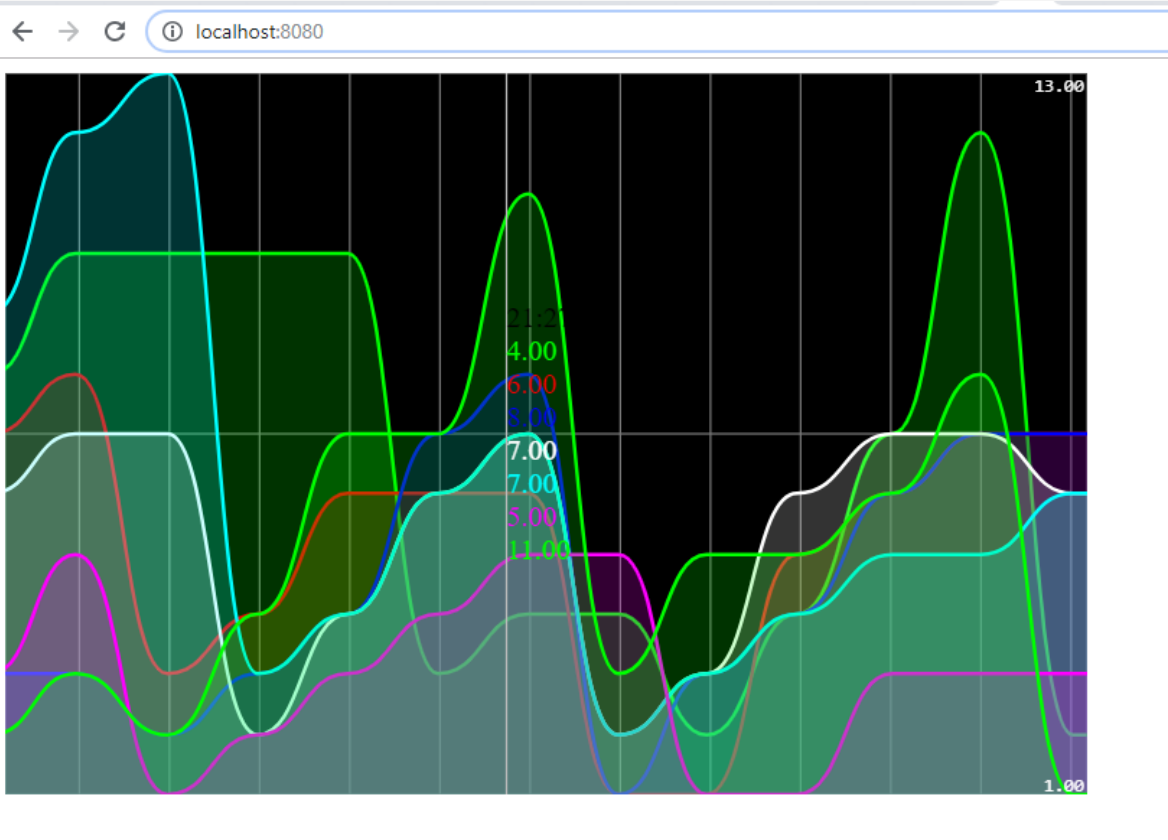

```
@StreamListener
@SendTo(AnalyticsBinding.PAGE_VIEW_COUNT)
public KStream<String,Long> process(@Input(AnalyticsBinding.PAGE_VIEW_IN)
KStream<String,PageViewEvent> pageViewEventKStream){

    return
            pageViewEventKStream.filter((k,v)->v.getDuration()>5)
            .map((k,v)->new KeyValue<>(v.getPage(),"0"))
                    .groupByKey()
            .windowedBy(TimeWindows.of(5000))
                    .count(Materialized.as(AnalyticsBinding.PAGE_COUNT_MV))
                    .toStream()
            .map((k,v)->new KeyValue<>(k.key(),v));
}
```

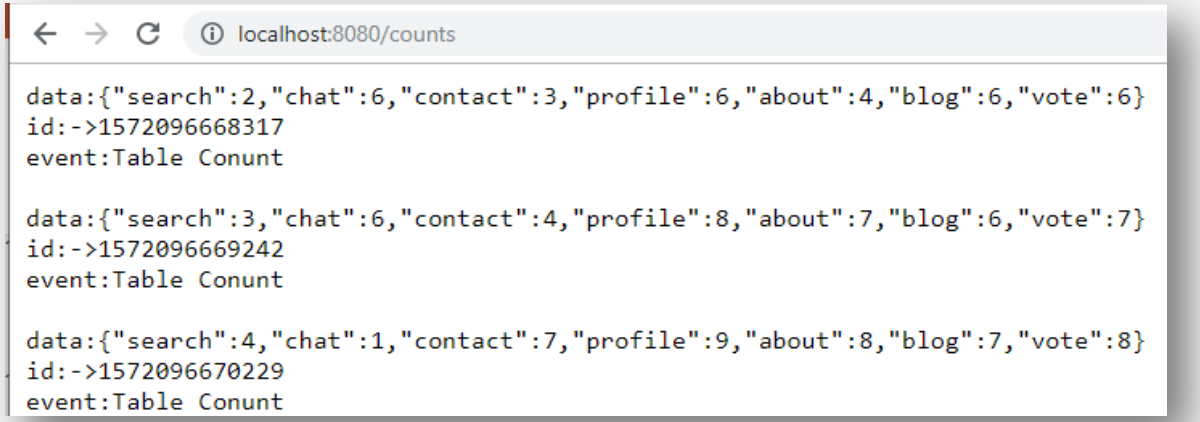
Spring Cloud avec Kafka : WindowStore with Server Sent Event

```
@GetMapping(value = "/counts",produces = MediaType.APPLICATION_STREAM_JSON_VALUE)
public SseEmitter counts() throws Exception{

    SseEmitter emitter = new SseEmitter();
   Executors.newScheduledThreadPool(1).scheduleAtFixedRate(()-> {
       try {
           Map<String, Long> results = new HashMap<>();
           ReadOnlyWindowStore<String, Long> queryableStoreType =
this.queryableStoreRegistry.getQueryableStore(AnalyticsBinding.PAGE_COUNT_MV,
QueryableStoreTypes.windowStore());
           KeyValueIterator<Windowed<String>, Long> all = queryableStoreType.all();
           while (all.hasNext()) {
               KeyValue<Windowed<String>, Long> item = all.next();
               results.put(item.key.key(), item.value);
           }
           SseEmitter.SseEventBuilder event = SseEmitter.event()
                   .data(new ObjectMapper().writeValueAsString(results))
                   .id("->" + System.currentTimeMillis())
                   .name("Table Conunt");
           emitter.send(event);
       } catch (Exception ex) {
           emitter.completeWithError(ex);
       }
   },1000,1000,TimeUnit.MILLISECONDS);
       return emitter;
}
```



Spring Cloud avec Kafka : WindowStore with Server Sent Event





Spring Cloud avec Kafka : WindowStore with Server Sent Event

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Analytics</title>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/smoothie/1.34.0/smoothie.min.js"></script>
</head>
<body>

<canvas id="chart2" width="600" height="400"></canvas>
<script>
    var index=-1;
    randomColor = function() {
        ++index;
        if (index >= colors.length) index = 0; return colors[index];
    }
    var pages=["search","chat","contact","profile","about","blog","vote"];
    var colors=[
        { sroke : 'rgba(0, 255, 0, 1)', fill : 'rgba(0, 255, 0, 0.2)' },
        { sroke : 'rgba(255, 0, 0, 1)', fill : 'rgba(255, 0, 0, 0.2)'},
        { sroke : 'rgba(0, 0, 255, 1)', fill : 'rgba(0, 0, 255, 0.2)'},
        { sroke : 'rgba(255, 255, 255, 1)', fill : 'rgba(255, 255, 255, 0.2)'},
        { sroke : 'rgba(0, 255, 255, 1)', fill : 'rgba(0, 255, 255, 0.2)'},
        { sroke : 'rgba(255, 0, 255, 1)', fill : 'rgba(255, 0, 255, 0.2)'}
        ];
```

Spring Cloud avec Kafka : WindowStore with Server Sent Event

```
var courbe = [];
    var smoothieChart = new SmoothieChart({tooltip: true});
    smoothieChart.streamTo(document.getElementById("chart2"), 500);
    pages.forEach(function(v){
       courbe[v]=new TimeSeries();
        col = randomColor();
        smoothieChart.addTimeSeries(courbe[v], {strokeStyle : col.sroke, fillStyle : col.fill, lineWidth : 2
        });
    });
    var stockEventSource= new EventSource("counts");
    stockEventSource.addEventListener("message", function (event) {
        pages.forEach(function(v){
            val=JSON.parse(event.data)[v];
            courbe[v].append(new Date().getTime(),val);
        });
    }, false);

</script>
</body>
</html>
</body>
</html>
```

Spring Cloud avec Kafka : Partie Spring Cloud Kafka Streams

```java
package ord.sid.analyticskafkastreamsspringcloud;
import ord.sid.analyticskafkastreamsspringcloud.bindings.AnalyticsBinding;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.stream.annotation.EnableBinding;

@SpringBootApplication
@EnableBinding(AnalyticsBinding.class)
public class AnalyticsKafkaStreamsSpringCloudApplication {
    public static void main(String[] args) {
        SpringApplication.run(AnalyticsKafkaStreamsSpringCloudApplication.class, args);
    }
}
```

Spring Cloud avec Kafka : Partie Spring Cloud Kafka Streams

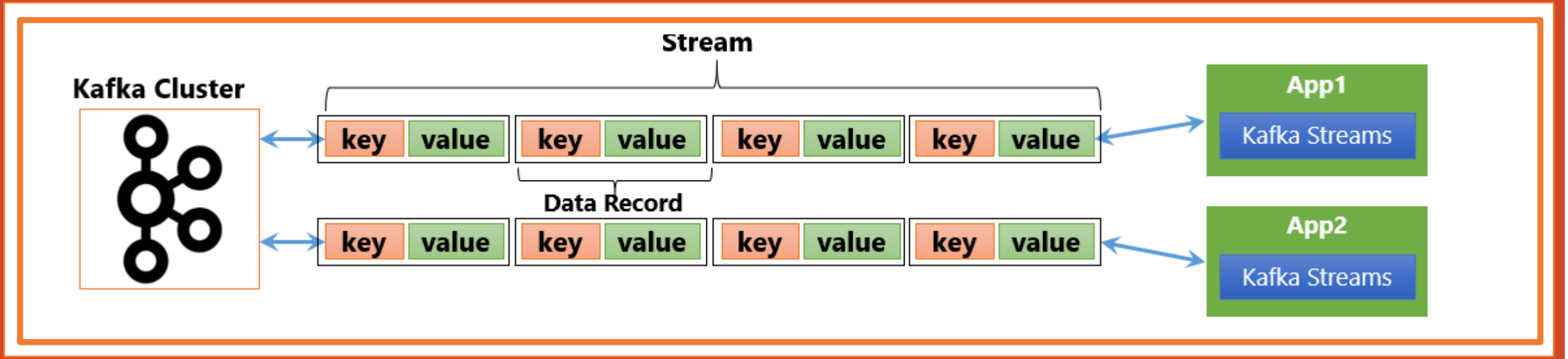
```
2019-09-13 17:32:01.572  INFO 18160 --- [pool-1-thread-1] o.s.a.source.PageViewEventSource         : Sending message GenericMessage
[payload=PageViewEvent(userId=Hassan, page=about, duration=1092), headers={id=af90bc14-23cb-d912-5b39-776945977044,
kafka_messageKey=Hassan, contentType=application/json, timestamp=1568392321572}]
2019-09-13 17:32:02.573  INFO 18160 --- [pool-1-thread-1] o.s.a.source.PageViewEventSource         : Sending message GenericMessage
[payload=PageViewEvent(userId=Hanane, page=chat, duration=583), headers={id=24409bf8-4c8d-e110-1d05-300f4238eb58,
kafka_messageKey=Hanane, contentType=application/json, timestamp=1568392322572}]
2019-09-13 17:32:03.573  INFO 18160 --- [pool-1-thread-1] o.s.a.source.PageViewEventSource         : Sending message GenericMessage
[payload=PageViewEvent(userId=Samir, page=contact, duration=1066), headers={id=22e05b15-90ce-23ac-2a48-85272f88c045,
kafka_messageKey=Samir, contentType=application/json, timestamp=1568392323573}]
2019-09-13 17:32:04.572  INFO 18160 --- [pool-1-thread-1] o.s.a.source.PageViewEventSource         : Sending message GenericMessage
[payload=PageViewEvent(userId=Samir, page=search, duration=795), headers={id=8c327648-6fdb-ed1d-707b-08f0f4f101f5,
kafka_messageKey=Samir, contentType=application/json, timestamp=1568392324572}]
2019-09-13 17:32:04.623  INFO 18160 --- [read-2-producer] org.apache.kafka.clients.Metadata        : Cluster ID: OTtUmHuBS1SmXvIxNVzYtA
2019-09-13 17:32:04.624  INFO 18160 --- [-StreamThread-2] o.s.a.processors.AnalyticsPageCount      : profile=>265
2019-09-13 17:32:04.624  INFO 18160 --- [-StreamThread-2] o.s.a.processors.AnalyticsPageCount      : blog=>283
2019-09-13 17:32:04.624  INFO 18160 --- [-StreamThread-2] o.s.a.processors.AnalyticsPageCount      : search=>272
2019-09-13 17:32:04.624  INFO 18160 --- [-StreamThread-2] o.s.a.processors.AnalyticsPageCount      : vote=>308
2019-09-13 17:32:04.624  INFO 18160 --- [-StreamThread-2] o.s.a.processors.AnalyticsPageCount      : about=>248
2019-09-13 17:32:04.624  INFO 18160 --- [-StreamThread-2] o.s.a.processors.AnalyticsPageCount      : chat=>260
2019-09-13 17:32:04.625  INFO 18160 --- [-StreamThread-2] o.s.a.processors.AnalyticsPageCount      : contact=>281
```

Spring Cloud avec Kafka : Partie Spring Cloud Kafka Streams

```
0$ bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic pvs --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.StringDeserializer
Hassan  {"userId":"Hassan","page":"vote","duration":844}
Samir   {"userId":"Samir","page":"search","duration":551}
Mohamed {"userId":"Mohamed","page":"about","duration":704}
Samir   {"userId":"Samir","page":"contact","duration":972}
Samir   {"userId":"Samir","page":"chat","duration":883}
Hassan  {"userId":"Hassan","page":"vote","duration":135}
Hanane  {"userId":"Hanane","page":"blog","duration":236}
Samir   {"userId":"Samir","page":"vote","duration":731}
Hassan  {"userId":"Hassan","page":"search","duration":503}
Aziz    {"userId":"Aziz","page":"contact","duration":484}
Samir   {"userId":"Samir","page":"about","duration":810}
Samir   {"userId":"Samir","page":"chat","duration":141}
Aziz    {"userId":"Aziz","page":"search","duration":1011}
Yassine {"userId":"Yassine","page":"contact","duration":178}
Hanane  {"userId":"Hanane","page":"profile","duration":424}
Samir   {"userId":"Samir","page":"about","duration":1042}
Mohamed {"userId":"Mohamed","page":"vote","duration":923}
Hassan  {"userId":"Hassan","page":"blog","duration":494}
Hassan  {"userId":"Hassan","page":"search","duration":225}
Mohamed {"userId":"Mohamed","page":"about","duration":985}
Yassine {"userId":"Yassine","page":"contact","duration":730}
Hanane  {"userId":"Hanane","page":"contact","duration":947}
```

```
$ bin/kafka-console-consumer.sh --bootstrap-server
localhost:9092 --topic pcs2 --property print.key=true --property
print.value=true --property
key.deserializer=org.apache.kafka.common.serialization.StringDes
erializer --property
value.deserializer=org.apache.kafka.common.serialization.LongDes
erializer
profile 265
blog    283
search  272
vote    308
about   248
chat    260
contact 281
```

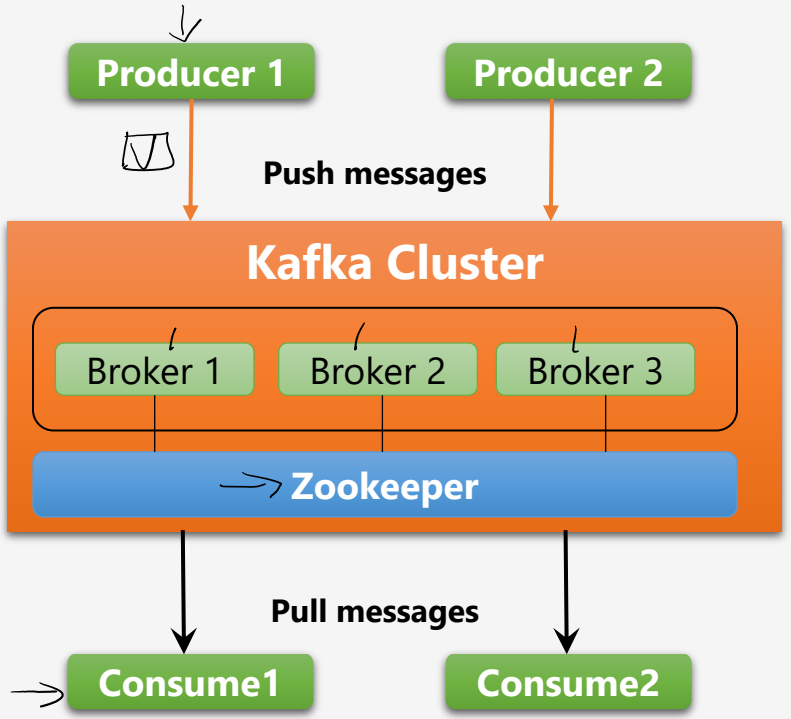
Big Data Real Time Stream Processing avec KAFKA Streams



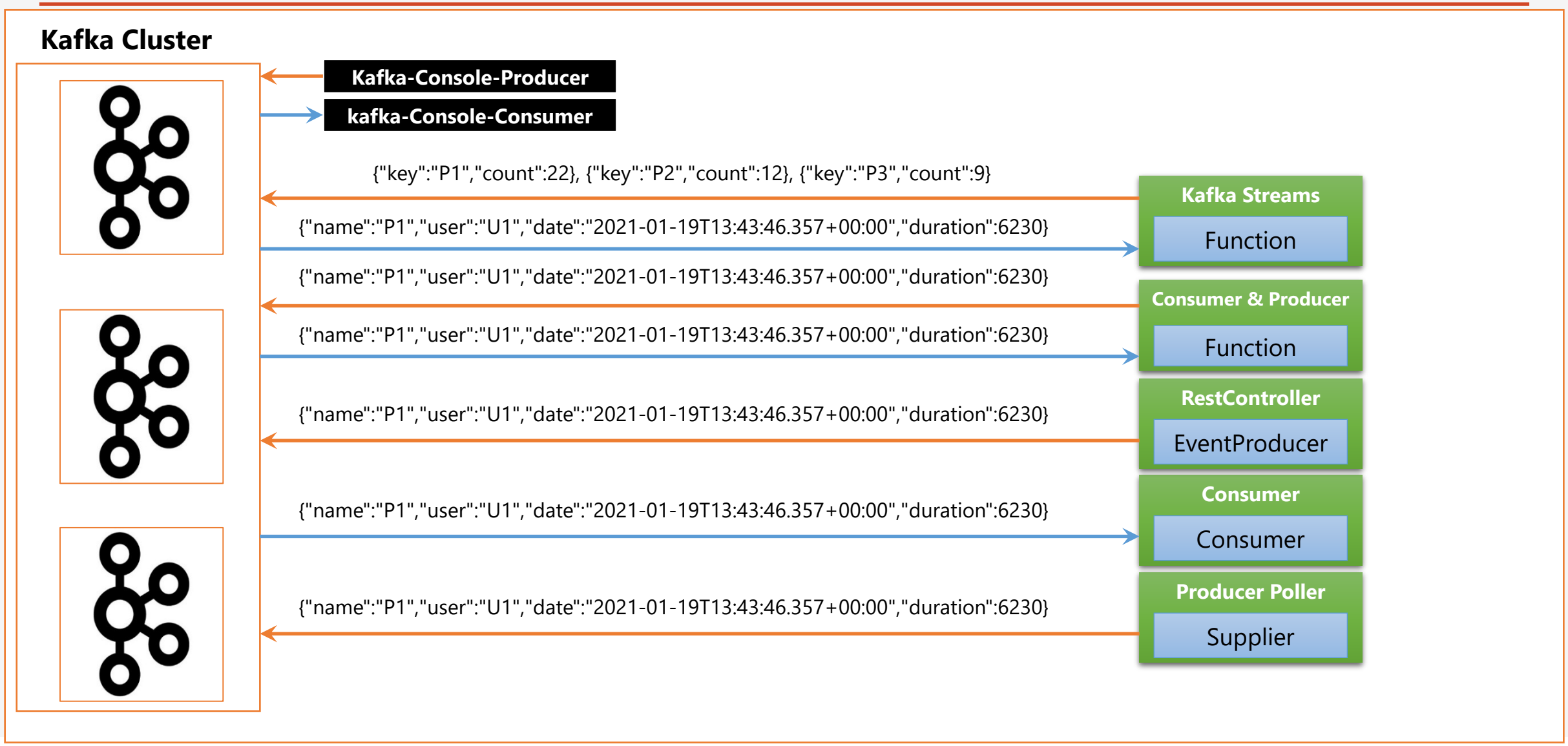
Mohamed Youssfi
Laboratoire Signaux Systèmes Distribués et Intelligence Artificielle (SSDIA)
ENSET, Université Hassan II Casablanca, Maroc
Email : med@youssfi.net
Supports de cours : http://fr.slideshare.net/mohamedyoussfi9
Chaîne vidéo : http://youtube.com/mohamedYoussfi
Recherche : http://www.researchgate.net/profile/Youssfi_Mohamed/publications

Les API de Kafka

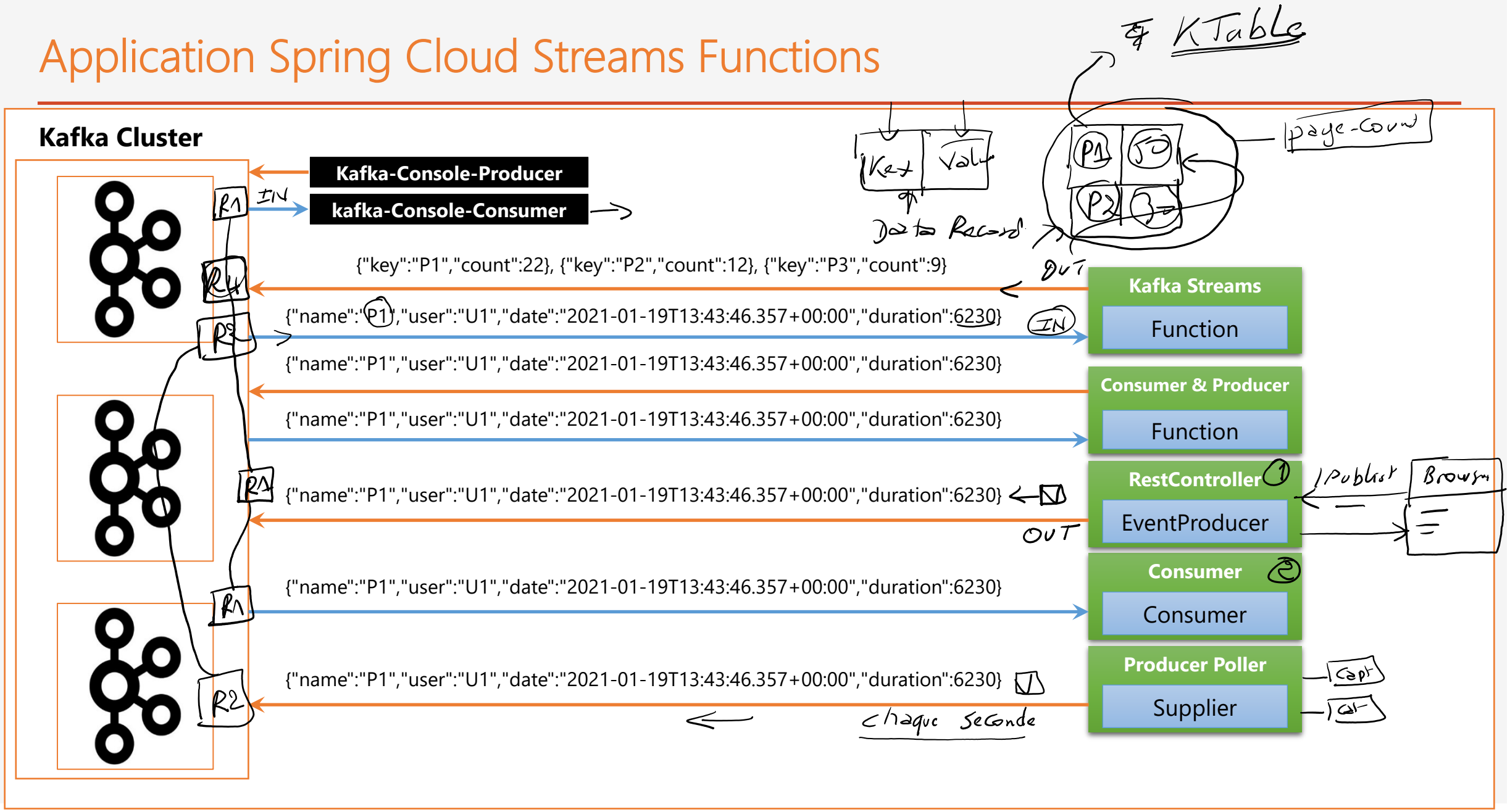
- Kafka a quatre API principales:
  - **Producer API**: Permet à une application de publier un flux d'enregistrements vers un ou plusieurs Topics (Sujets) Kafka.
  - **Consumer API**: Permet à une application de s'abonner à un ou plusieurs Topics et de traiter le flux d'enregistrements qui lui sont transmis.
  - **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 fourni des API clientes pour différents langages : Java, C++, Node JS, .Net, PHP, Python, etc…



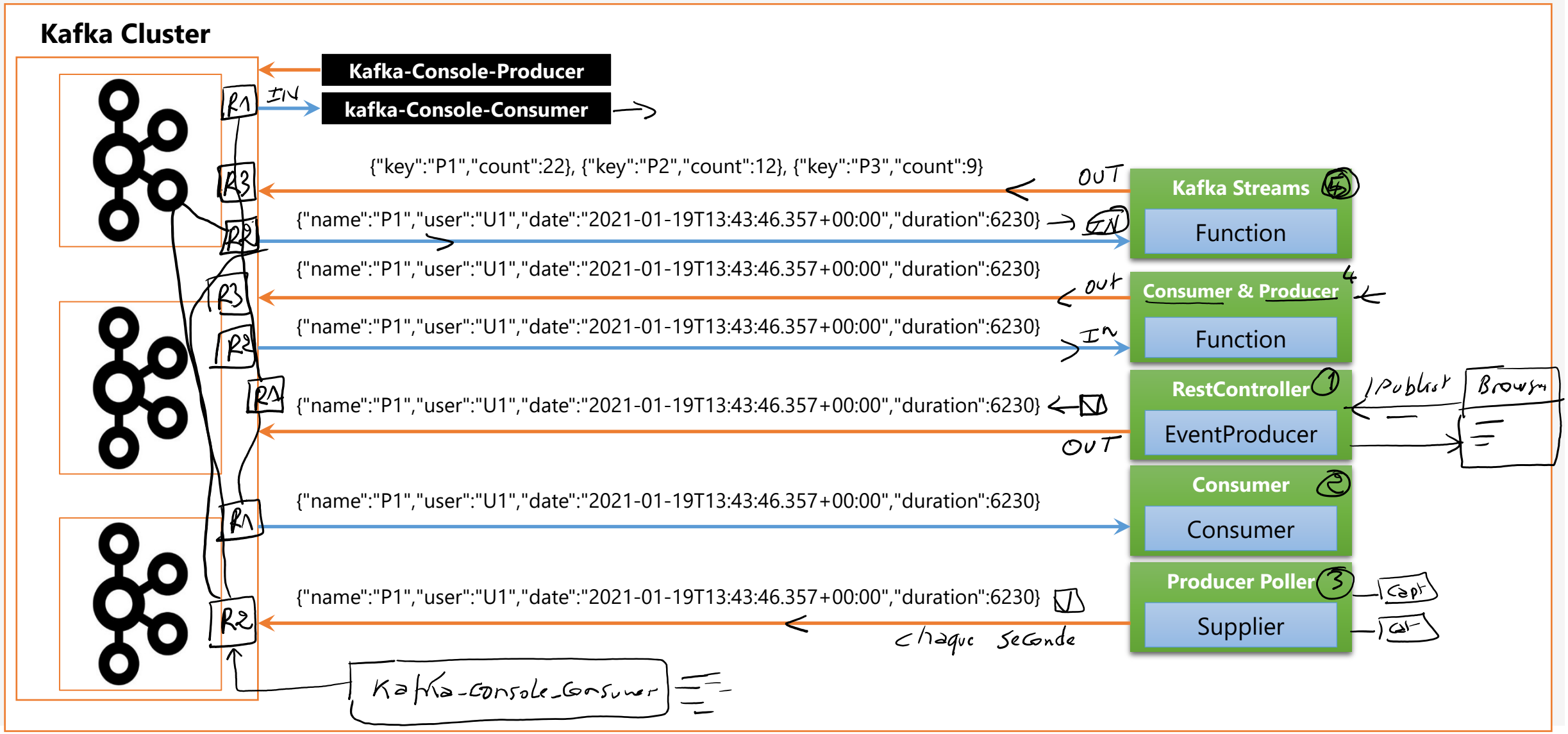
Application Spring Cloud Streams Functions



Application Spring Cloud Streams Functions

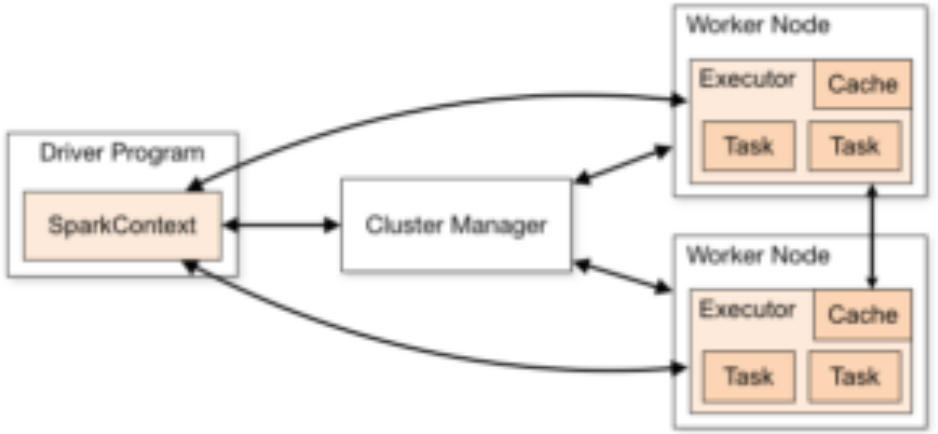


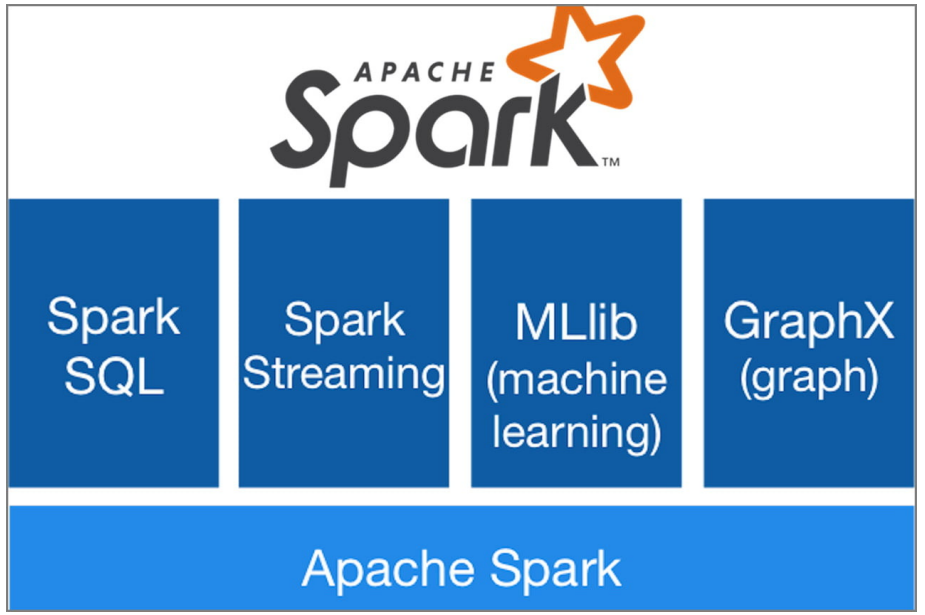
Application Spring Cloud Streams Functions



Apache Spark

- Apache Spark est un framework de traitements distribué en Big Data
- Celui-ci a originellement été développé par AMPLab, de l'Université UC Berkeley, en 2009 et passé open source sous forme de projet Apache en 2010.
- C'est un moteur de calcul distribué sur des données massives à partir des sources de données distribuées provenant de tout système de partitionnement de données sur lesquels on peut appliquer des traitements parallèles comme HDFS, Mongo Db, Cassandra, ElasticSearch etc.
- Spark est un moteur d'exécution avec des opérateurs de haut niveau (Map, Reduce, Group, Join)
- Des opérateurs qui prennent des fonctions en paramètres et appliquent ces fonctions à des données dans un environnement distribuées.





Apache Spark

- La principale innovation de Spark par rapport aux autres technologies c'est l'introduction du concept de Collections Résidentes en Mémoire (RDD)
- Les RDD est un moyen qui permet d'améliorer considérablement certains traitements y compris les traitement itératifs par rapport à un système comme Hadoop Map Reduce.
- Car Hadoop s'appuie beaucoup sur les écritures sur le disque, ce qui introduit une latence importante pendant les traitement.
- Spark, au contraire, va essayer de faire le maximum en mémoire. Ce qui pourrait poser le problème de pertes de données en cas de panne.
- Spark dispose d'un mécanisme de reprise sur panne permettant de faire des calculs en mémoire en préservant la capacité à être resistants aux panes.

Apache Spark

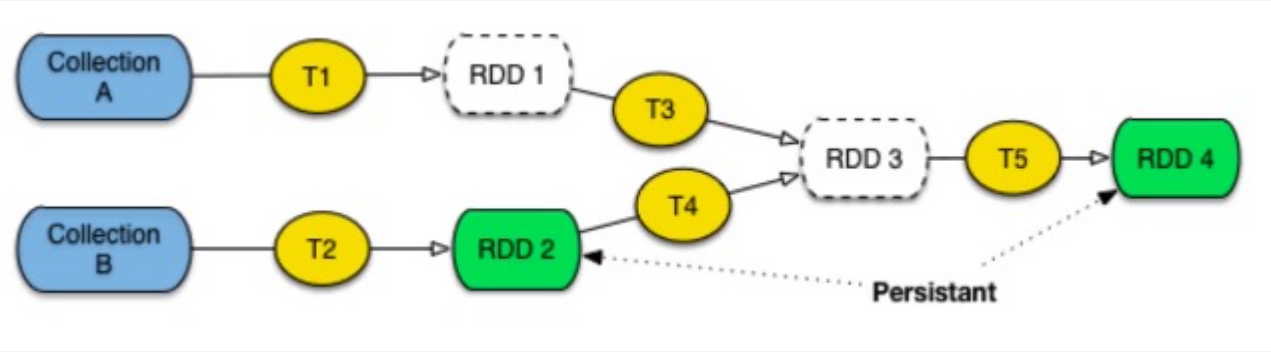
- Autres avantages Spark
  - Spark fournit plus d'opérateurs que le cas de Hadoop.
  - Syntaxe simplifiée (Java, Scala, Python)
  - Des structures de données plus agréable à manipuler (Data Frames et DataSet)
  - Large communauté : Disponibilité de beaucoup d'algorithmes de fouilles de données
  - De nombreuses librairies pour la fouille de donnés MLIB, le traitement des graphes, le traitements des flux (Streaming), etc.
  - Le fait que l'on puisse stocker des collections résilientes en mémoire et de les réutiliser dans le cadre de plusieurs workflow est un avantage important pour des algorithmes de fouilles de données itératifs comme K-means.

Resilient Distributed DataSet (RDD)

- C'est une collection d'objets java calculé à partir d'une source de données (MongoDB, un flux, un autre RDD, etc..)
- Un RDD peut être marqué persistant : Il est donc placé en mémoire RAM et conservé par Spark.
- Dans le cas d'une reprise sur panne, Spark ne fait pas une reprise sur panne à partir d'une sérialisation des données sur disque, mais par une reconstitution par recalcul des RDD à partir des opérations qui les ont créées.
- Dans un workflow, le fait de marquer un RDD persistant évitera justement de refaire le calcul après une reprise.
- Un RDD est un bloc non modifiable (immutable). Si nécessaire, il est entièrement recalculé.

Workflow avec RDD dans Spark

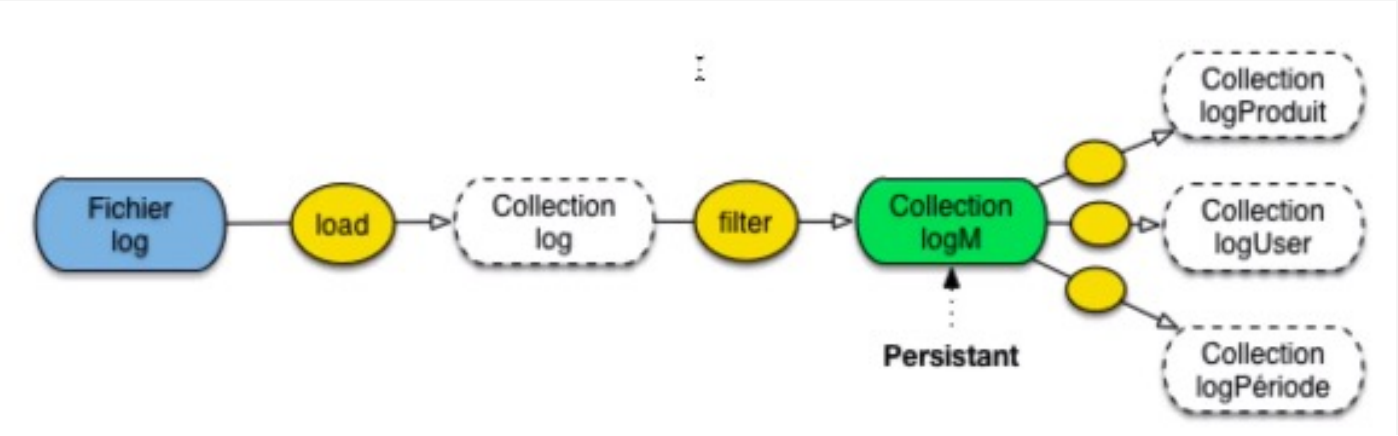
- Des transformations créent des RDD à partir d'une ou deux sources de données en utilisant des opérateurs unaire comme Map et Reduce ou des opérateurs binaires comme le Group et Le Join



- Des transformations créent des RDD à partir d'une ou deux sources de données.

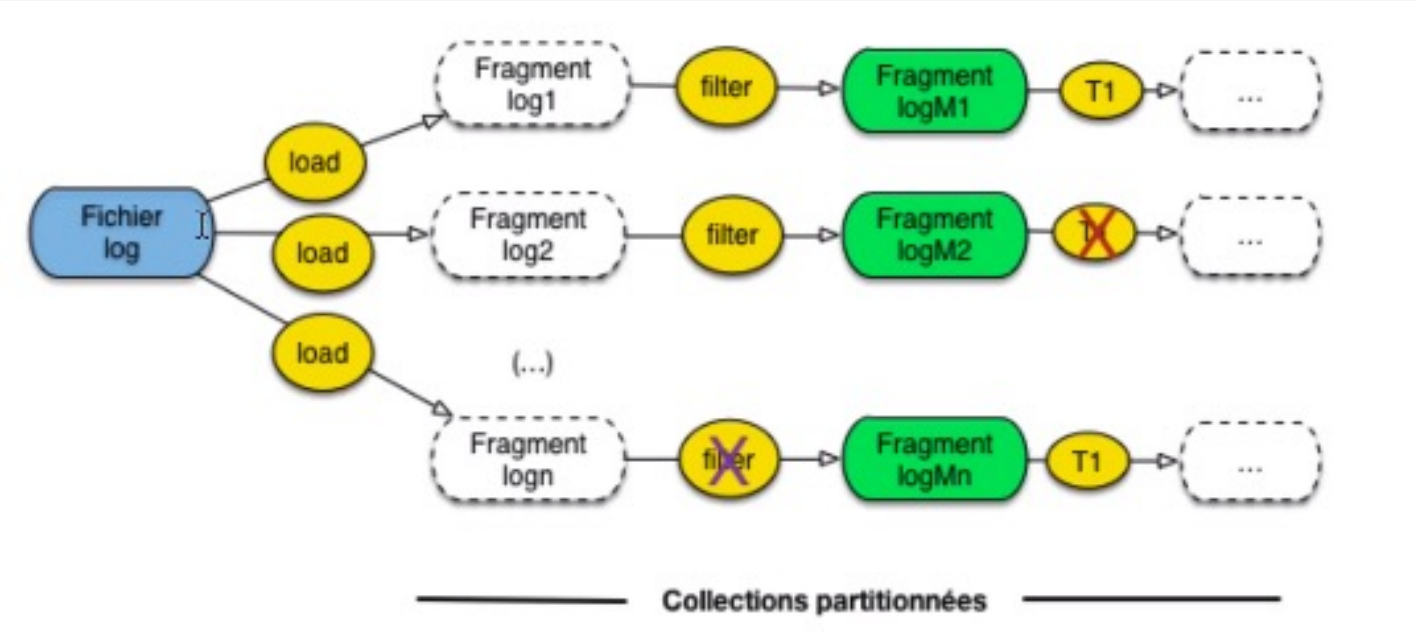
Exemple

- On veut analyser le fichier journal (log) d'une application dont un des modules (M) est suspect.
- On construit un programme qui charge le log, ne conserve que les messages produits par le module M et les analyse.
- On peut analyser par produit, par utilisateur, par période, etc
- Avec Haddop, pour chaque analyse, on reprend le traitement depuis la première étape
- Avec Spark, dans notre workflow, après filtrage des messages M, on peut marquer le résultat RDD persistant, ce qui recalculer cette collection pour les autres opérations d'analyse.



Reprise sur Pane

- Un RDD est une collection partitionnée
- Quand une panne survient dans un nœud du workflow, le recalcul est effectué à partir d'un fragment F persistant qui précède le nœud.



DataFarmes et DataSet

- Un RDD, du point de vue du programmeur, c'est un conteneur d'objets java.
- Le type précis de ces objets n'est pas connu par Spark. Par conséquent :
  - Tout ce que Spark peut faire, c'est appliquer la sérialisation/désérialisation java
  - Aucun accès aux objets grâce à un langage déclaratif n'est possible.
  - Et donc pas d'optimisation, et la nécessité de tout écrire sous forme de fonctions java.
- Les Dataset sont des RDD dont on connait le schéma de la structure des données.
- Depuis la version 1.6 : on dispose de RDD améliorés : les Datasets. On peut les traiter comme des tables relationnelles distribuées

Installation de Spark sous Linux

- Installation de Java :
  - $ sudo apt-add-repository ppa:webupd8team/java
  - $ sudo apt-get update
  - $ sudo apt-get install oracle-java8-installer
- Installation de Scala :
  - $ wget http://www.scala-lang.org/files/archive/scala-2.11.8.tgz
  - $ sudo mkdir /usr/local/src/scala
  - $ sudo tar xvf scala-2.11.8.tgz -C /usr/local/src/scala/
- Installation de Spark
  - $ wget https://www.apache.org/dyn/closer.lua/spark/spark-2.4.4/spark-2.4.4-bin-hadoop2.7.tgz
  - sudo mkdir /usr/lib/spark
  - $ sudo tar xvf spark-2.2.0-bin-hadoop2.7.tgz -C /usr/lib/spark
- Configurer les variables d'environnement :
  - export JAVA_HOME=/usr/lib/jvm/java-8-oracle
  - export SCALA_HOME=/usr/local/src/scala/scala-2.11.8
  - export SPARK_HOME=/usr/lib/spark/spark-2.2.0-bin-hadoop2.7
  - export PATH=$SCALA_HOME/bin:$JAVA_HOME/bin:$SPARK_HOME/bin:$PATH

Installation de Spark sous Linux

```
$ java -version
openjdk version "1.8.0_222"
OpenJDK Runtime Environment (build 1.8.0_222-8u222-b10-1ubuntu1~18.04.1-b10)
OpenJDK 64-Bit Server VM (build 25.222-b10, mixed mode)

$ scala -version
Scala code runner version 2.11.8 -- Copyright 2002-2016, LAMP/EPFL
youssfi@host1:~$
```

Lancement de spark-shell

```
$ spark-shell
19/09/04 12:34:06 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Spark context Web UI available at http://host1:4040
Spark context available as 'sc' (master = local[*], app id = local-1567596859950).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.4.4
      /_/

Using Scala version 2.11.12 (OpenJDK 64-Bit Server VM, Java 1.8.0_222)
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

Lancement de spark-shell

```
$ cat input-data.txt
Mr. Yassine et Mme Amal sont deux professeurs
Mr. Yassine assure le cours de génie logiciel
Mme. Amal assure le cours de communication
les étudiants se préparent à passer l'examen de génie logiciel
hassan a eu comme note 15 en GL
hanane a eu comme note 12 en GL
Malak a eu comme note 13 en GL
```

Lancement de spark-shell

```
scala> val data=sc.textFile("input-data.txt")
data: org.apache.spark.rdd.RDD[String] = input-data.txt MapPartitionsRDD[1] at
textFile at <console>:24
```

- Nous avons créé un premier RDD à partie de la source de données input-data.txt
- Spark propose des *actions* directement applicable à un RDD et produisant des résultats scalaires. (Un RDD est interfacé comme un objet auquel nous pouvons appliquer des méthodes.)

Lancement de spark-shell

```
scala> data.count()
res0: Long = 7
```

- Nombre de documents dans le RDD

Lancement de spark-shell

```
scala> data.first()
res1: String = Mr. Yassine et Mme Amal sont deux professeurs
```

- Premier documents du RDD

Lancement de spark-shell

```
scala> data.collect()
res2: Array[String] = Array(Mr. Yassine et Mme Amal sont deux professeurs, Mr. Yassine assure le cours de génie logiciel, Mme. Amal assure le cours de communication, les étudiants se préparent à passer l'examen de génie logiciel, hassan a eu comme note 15 en GL, hanane a eu comme note 12 en GL, Malak a eu comme note 13 en GL)
```

- Récupération du RDD complet

Lancement de spark-shell

```
scala> val notes=data.filter({line=>line.contains("GL")})
notes: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[2] at filter at
<console>:25
```

- Appliquer l'opérateur filter pour ne retenir que les documents contenant le mot GL

Lancement de spark-shell

```
scala> val words=notes.flatMap({line=>line.split(" ")})
words: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[3] at flatMap at <console>:25


scala> notes.collect()
res3: Array[String] = Array(hassan a eu comme note 15 en GL, hanane a eu comme note 12 en GL, Malak a eu comme note 13 en GL)
```

- La méthode split décompose une chaîne de caractères (ici, en prenant comme séparateur un espace).
- Notez l'opérateur flatMap qui produit plusieurs documents (ici un terme) pour un document en entrée (ici une ligne).

Lancement de spark-shell

```
scala> words.count()
res4: Long = 24

scala> words.collect()
res5: Array[String] = Array(hassan, a, eu, comme, note, 15, en, GL, hanane, a, eu, comme, note, 12, en, GL, Malak, a, eu, comme, note, 13, en, GL)
```

Lancement de spark-shell

```
scala> val termUnit=words.map({word=>(word,1)})
termUnit: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[4] at map
at <console>:25
```

- On introduit la notion de comptage: chaque terme vaut 1. L'opérateur map produit un document en sortie pour chaque document en entrée.
- On peut s'en servir ici pour enrichir chaque terme avec son compteur initial.

Lancement de spark-shell

```
scala> val compteursTermes=termUnit.reduceByKey({(a,b)=>a+b})
compteursTermes: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[5] at
reduceByKey at <console>:25


scala> compteursTermes.collect()
res6: Array[(String, Int)] = Array((Malak,1), (note,3), (15,1), (eu,3),
(hassan,1), (13,1), (en,3), (a,3), (12,1), (comme,3), (GL,3), (hanane,1))
```

- Cette étape regroupe les termes et effectue la somme de leurs compteurs: c'est un opérateur reduceByKey.
- On passe à l'opérateur une fonction de réduction, ici notée littéralement dans la syntaxe Scala. Une telle fonction prend en entrée deux paramètres: un accumulateur (ici a) et la nouvelle valeur à agréger à l'accumulateur (ici b).
- L'agrégation est ici simplement la somme.

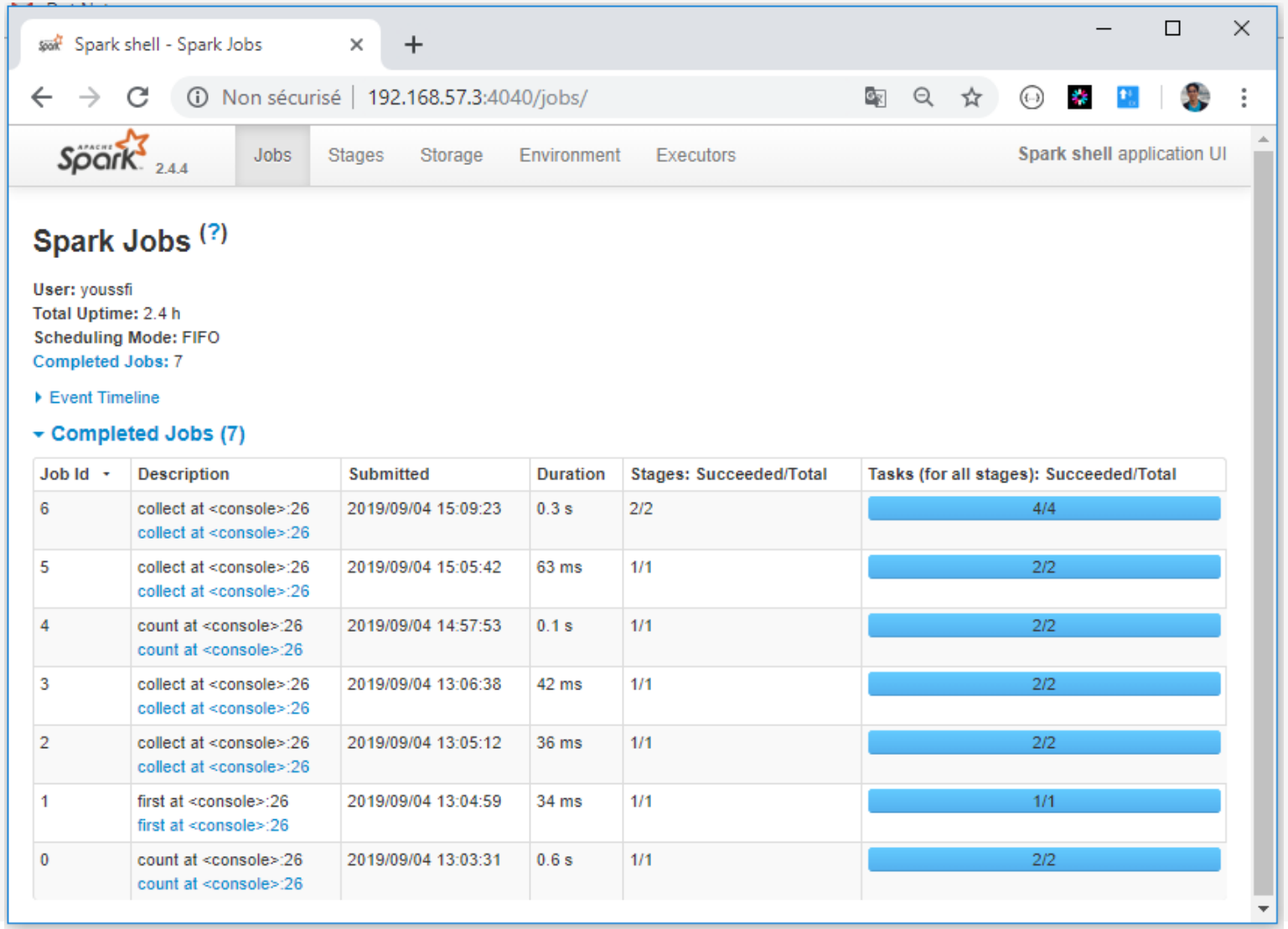
Lancement de spark-shell

```
scala> compteursTermes.persist()
res7: compteursTermes.type = ShuffledRDD[5] at reduceByKey at <console>:25

scala>
```

- Finalement, si on souhaite conserver en mémoire le RDD final pour le soumettre à divers traitements, il suffit d'appeler l'opérateur persist()

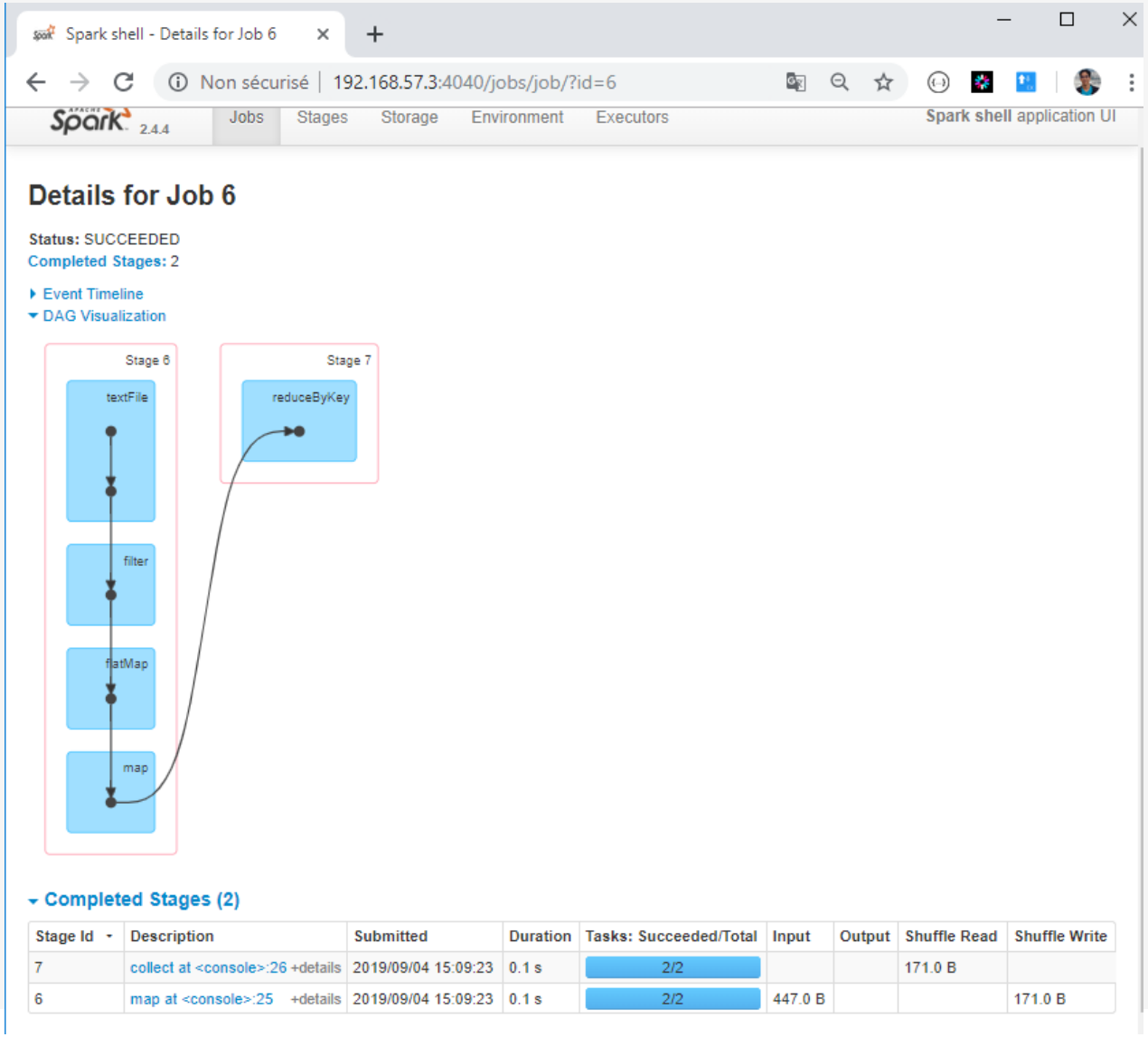
Lancement de spark-shell

- Spark dispose d'une interface Web qui permet de consulter les entrailles du système et de mieux comprendre ce qui est fait.
- Elle est accessible sur le port 4040, donc à l'URL http://localhost:4040 pour une exécution du *shell*.
- Pour explorer les informations fournies par cette interface, nous allons exécuter notre *workflow*, assemblé en une seule chaîne d'instructions Scala.



Lancement de spark-shell

- Cliquez sur le nom du job pour obtenir des détails sur les étapes du calcul
- Spark nous dit que l'exécution s'est faite en deux étapes.
  - La première comprend les transformations textFile, flatMap et `` map``,
  - la seconde la transformation `` reduceByKey``.
- les deux étapes sont séparées par une phase de shuffle.



Etape Spark (Stage)

- Une étape (Stage) dans Spark regroupe un ensemble d'opérations possible à 'exécuter localement, sur une seule machine, sans avoir à effectuer des échanges réseau.
- C'est une généralisation de la phase de Map dans un environnement MapReduce.
- Les étapes sont logiquement séparées par des phases de shuffle qui consistent à redistribuer les données afin de les regrouper selon certains critères
- Quand le traitement s'effectue sur des données partitionnées, une étape est effectuée en parallèle sur les fragments, et Spark appelle tâche l'exécution de l'étape sur un fragment particulier, pour une machine particulière.
- En résumé :
    - Un job est l'exécution d'une chaîne de traitements (workflow) dans une environnement distribué.
    - Un job est découpé en étapes, chaque étape étant un segment du workflow qui peut s'exécuter localement.
    - L'exécution d'une étape se fait par un ensemble de tâches, une par machine hébergeant un fragment du RDD servant de point d'entrée à l'étape.

Spark avec Java

```
import org.apache.spark.SparkConf;
import org.apache.spark.SparkContext;
import org.apache.spark.api.java.JavaPairRDD;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.JavaSparkContext;
import org.apache.spark.storage.StorageLevel;
import scala.Tuple2; import java.util.Arrays; import java.util.Scanner;

public class FirstSparkApp {
    public static void main(String[] args) {
        // Spark Configuration
        SparkConf sparkConf=new SparkConf()
                .setAppName("Spak Test App")
                .setMaster("local[2]");
        // Spark Context

        JavaSparkContext sc = new JavaSparkContext(sparkConf);
```

Spark avec Java

```
// Créer un premier RDD à partir d'une source de donnée : Fichier Texte
        JavaRDD<String> lines=sc.textFile("input-data.txt");
        System.out.println("----------------------");
        System.out.println(lines.count());
        System.out.println("----------------------");
        System.out.println("----------------------");
        // Première ligne du RDD
        System.out.println(lines.first());
        System.out.println("----------------------");
        lines.collect().forEach(line->{
            System.out.println(line);
        });
        System.out.println("----------------------");
        // Filtrer le RDD pour ne retenir que les lignes contenant GL
        JavaRDD<String> notes=lines.filter(line->line.contains("GL"));
        System.out.println("----------------------");
        notes.collect().forEach(line->{
            System.out.println(line);
        });
        System.out.println("----------------------");
```

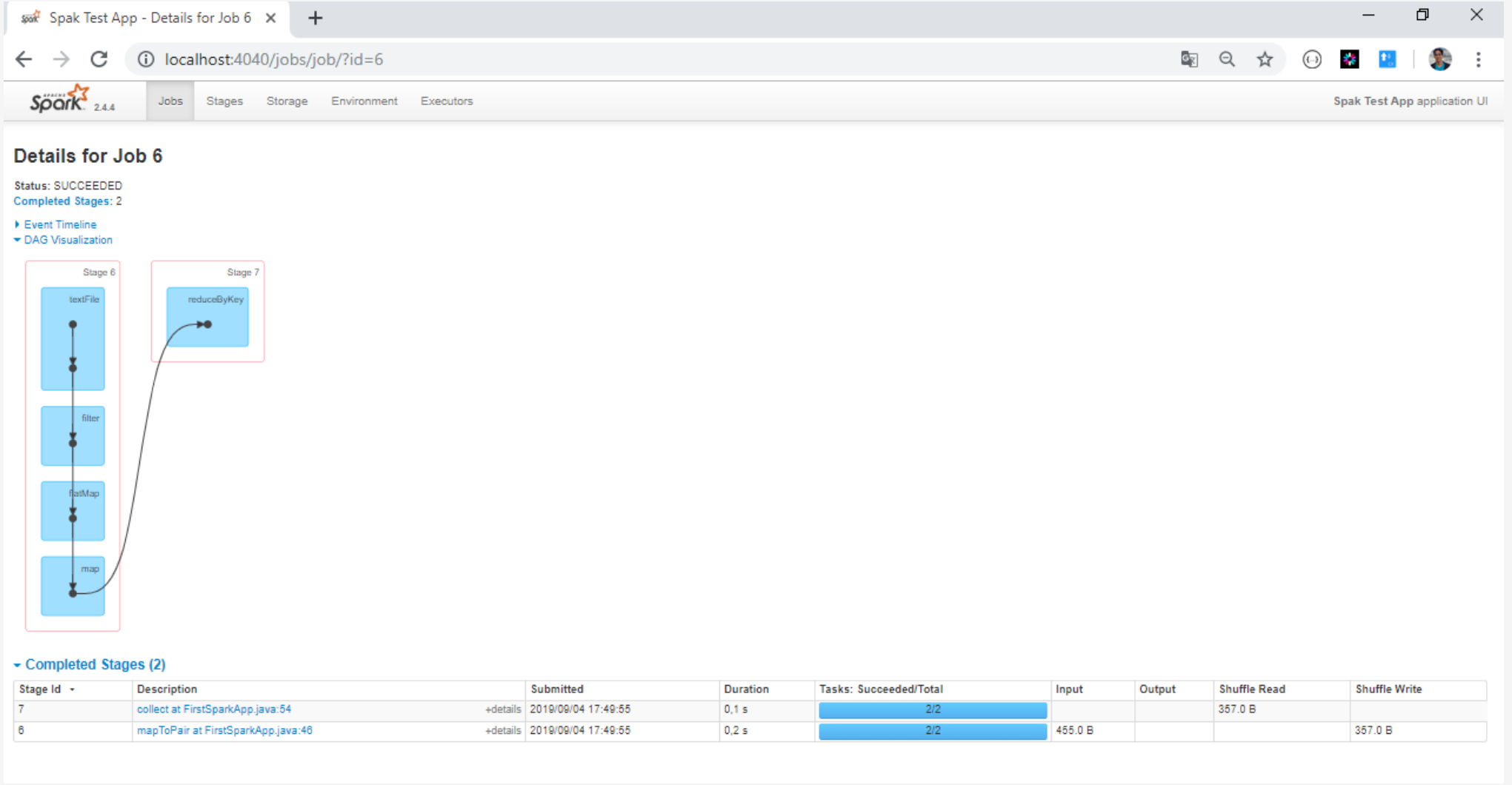
Spark avec Java

```
// Faire un split sur toutes les lignes pour récupérer une liste de termes
        JavaRDD<String> termesGL=notes.flatMap(line-> Arrays.asList(line.split(" ")).iterator());
         System.out.println("----------------------");
         termesGL.collect().forEach(word->{
             System.out.println(word);
         });
         System.out.println("----------------------");
// Faire un Map pour avoir un RDD sous forme d'une collection de paires clé terme et valeur 1
         JavaPairRDD<String,Integer> termesGLMap=termesGL.mapToPair(term->new Tuple2(term,1));
         System.out.println("----------------------");
         termesGLMap.collect().forEach(word->{
             System.out.println(word);
         });
         System.out.println("----------------------");
// Faire un reduce par clé du RDD pour compter le nombred'occurences de chaque terme
         JavaPairRDD<String,Integer> compteurTermes=termesGLMap.reduceByKey((a,b)->a+b);
         System.out.println("----------------------");
         compteurTermes.collect().forEach(word->{
             System.out.println(word);
         });
```

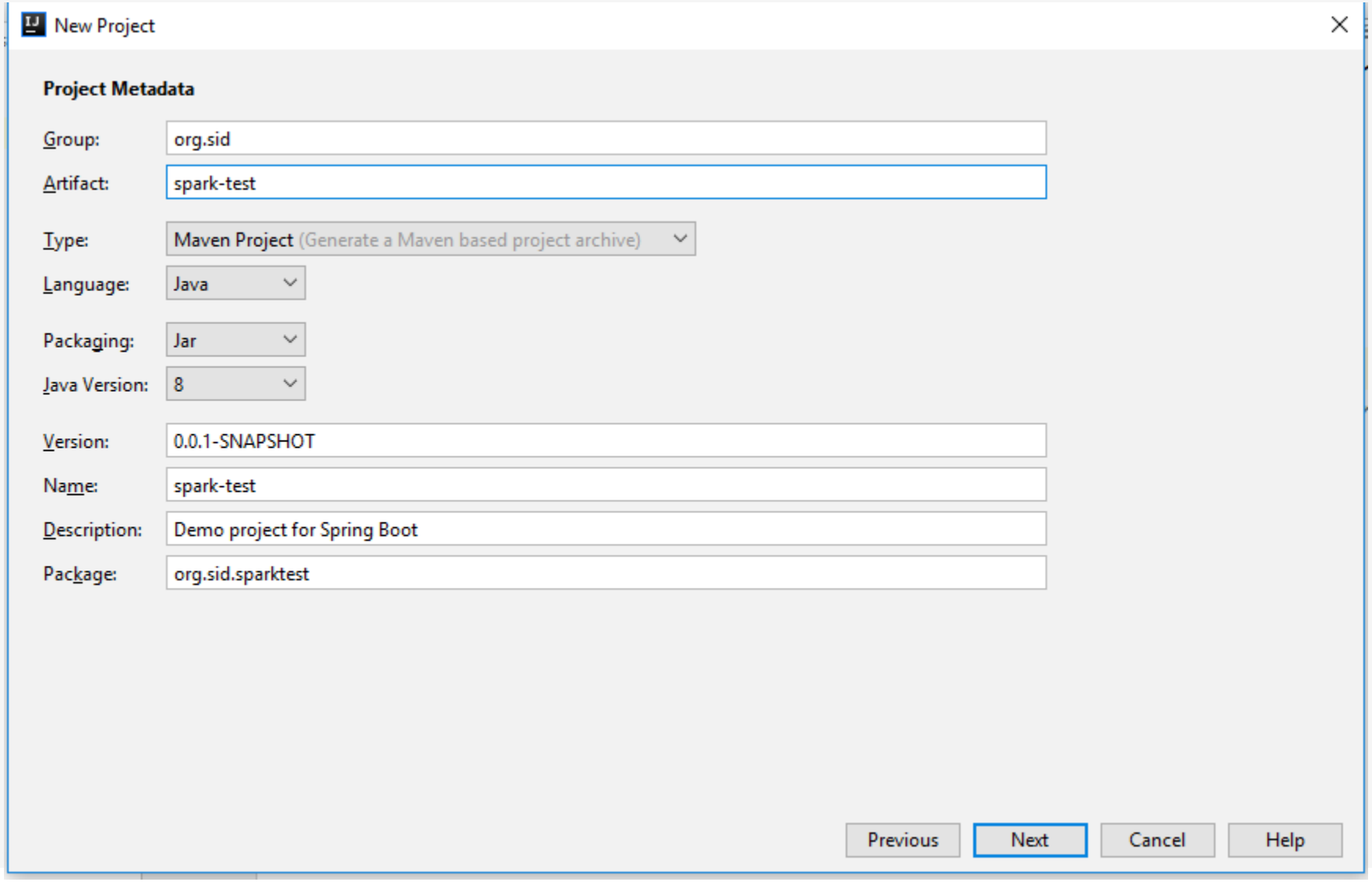
Spark avec Java

```
System.out.println("----------------------");
        // Persister le RDD en mémoire et en disk
        compteurTermes.persist(StorageLevel.MEMORY_AND_DISK());
        // Eviter de quitter l'application pour pemettre de voir l'interface web de spark
        System.out.println("Taper une touche pour terminer");

        new Scanner(System.in).next();

    }
}
```

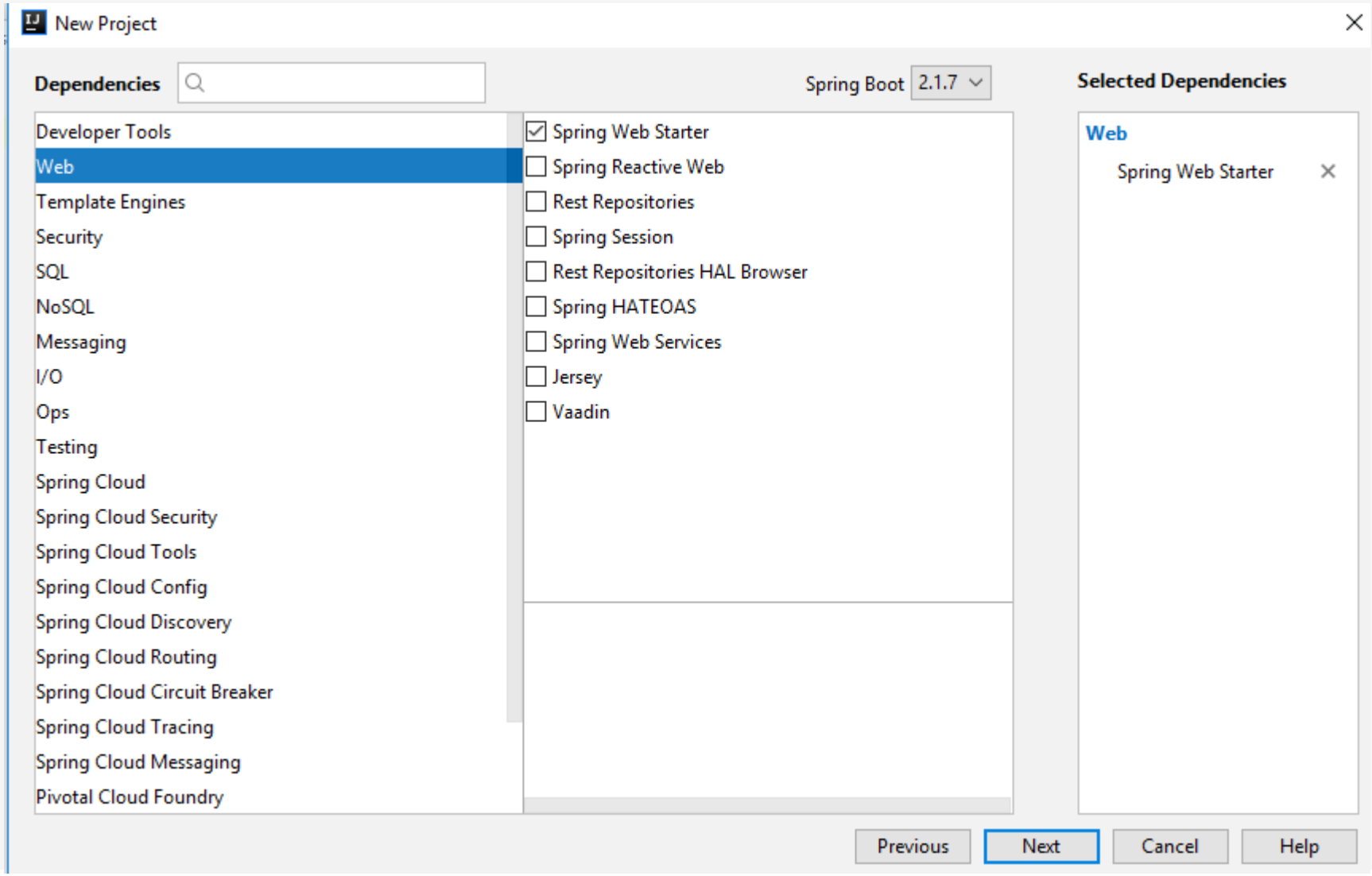
Application Java Spark



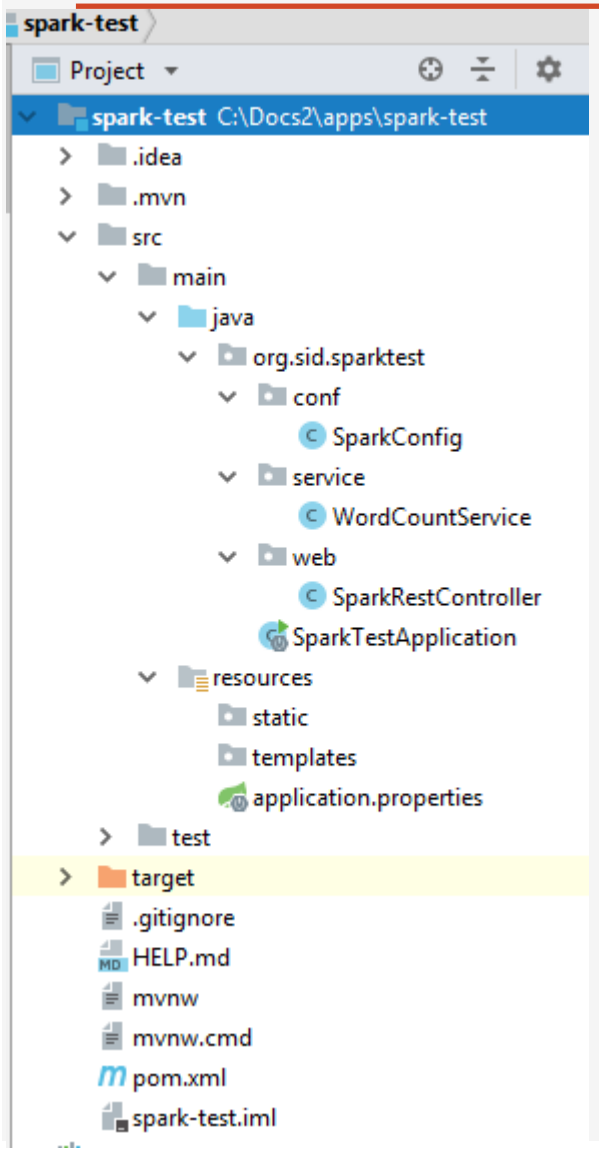
Application Spring Boot avec Spark



Application Spring Boot avec Spark



Application Spring Boot avec Spark




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

    <dependency>
        <groupId>org.apache.spark</groupId>
        <artifactId>spark-core_2.11</artifactId>
        <version>2.4.4</version>
    </dependency>

</dependencies>
```

applications.properties

```
spark.app.name=Spring Spark Word Count Application
spark.master=local[2]
```

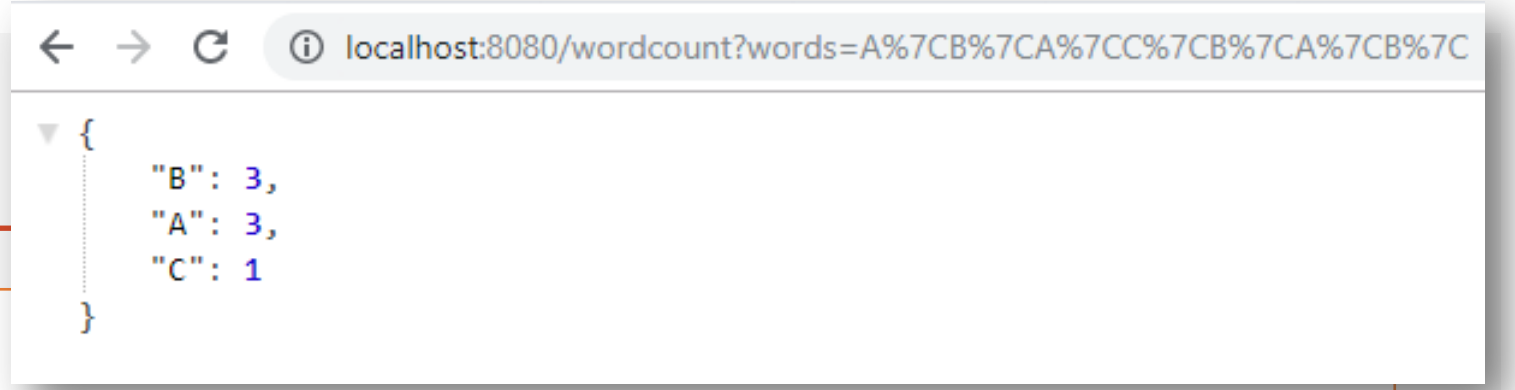
SparkConfig

```
package org.sid.sparktest.conf;

import org.apache.spark.SparkConf;
import org.apache.spark.api.java.JavaSparkContext;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
@Configuration
public class SparkConfig {
    @Value("${spark.app.name}")
    private String appName;
    @Value("${spark.master}")
    private String masterURI;

    @Bean
    public JavaSparkContext sc() {
        SparkConf sparkConf = new SparkConf().setAppName(appName).setMaster(masterURI);
        return new JavaSparkContext(sparkConf);
    }
}
```

WordCountService

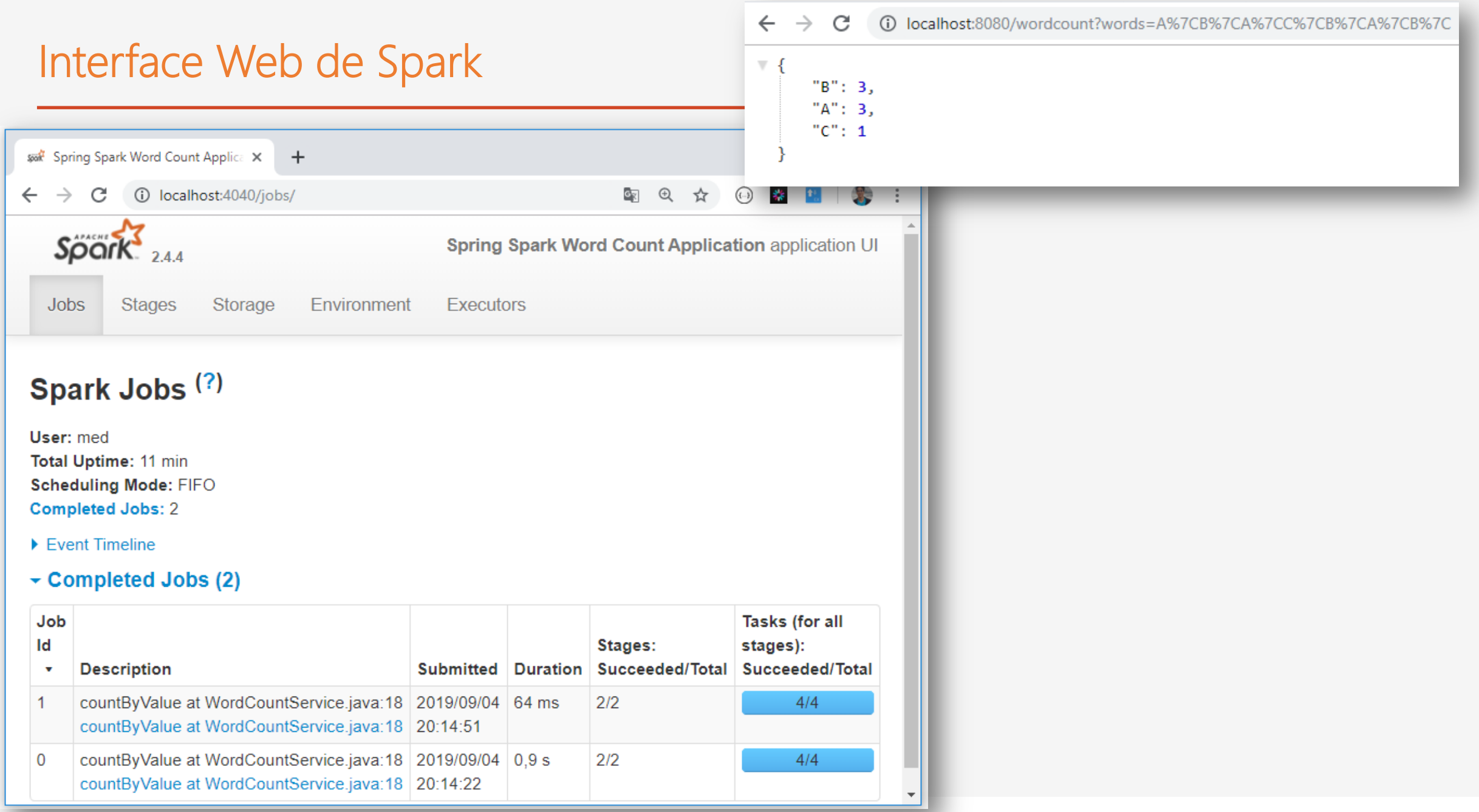
```
package org.sid.sparktest.service;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.JavaSparkContext;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import java.util.List; import java.util.Map;

@Service
public class WordCountService {
    @Autowired
    private JavaSparkContext sparkContext;

    public Map<String,Long>  getCount(List<String> listData){
        JavaRDD<String> wordsRDD=sparkContext.parallelize(listData);
        Map<String,Long> wordsCount=wordsRDD.countByValue();
        return wordsCount;
    }
}
```

SparkRestController




```
package org.sid.sparktest.web;

import org.sid.sparktest.service.WordCountService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;
import java.util.Arrays; import java.util.List; import java.util.Map;

@RestController
public class SparkRestController {
    @Autowired
    private WordCountService wordCountService;
    @GetMapping("/wordcount")
    public Map<String,Long> worksCount(String words){
        List<String> list= Arrays.asList(words.split("\\|"));
        return wordCountService.getCount(list);
    }
}
```

Interface Web de Spark



Interface Web de Spark