Dependencias necesarias

name := "TestScalaNuevo"

version := "0.1"

scalaVersion := "2.11.12"

libraryDependencies += "org.apache.spark" %% "spark-sql" % "2.4.0"% "Provided"

libraryDependencies += "org.apache.spark" %% "spark-streaming-kafka-0-10" % "2.4.0"

libraryDependencies += "org.apache.spark" %% "spark-sql-kafka-0-10" % "2.4.0"

EJERCICIO1. PROBAR COMUNICACIÓN KAFKA

1. Primero Abrir un terminal de Linux y lanzamos el servicio zookeeper

bin/zookeeper-server-start.sh config/zookeeper.properties

2. En un segundo terminal de Linux lanzamos el servicio de KAFKA

bin/kafka-server-start.sh config/server.properties

los dos quedarán bloqueados para podamos trabajar.

3. Vamos a testear:
Vamos a publicar y consumir un mensaje "Hello World" para asegurarnos de que el servidor de Kafka se está comportando correctamente. La publicación de mensajes en Kafka requiere:

• Un productor, que permite la publicación de registros y datos a temas.
• Un consumidor, que lee mensajes y datos de temas.

NOSOTROS NO LO NECESITAMOS POR QUE YA LO HICE YO

CD /home/kafka/ kafka_2.12-2.4.0/
Primero creamos el topic TutorialTopic :
bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic TutorialTopic

- Publicamos el mensaje (PRODUCTOR) "Hello, World" to the TutorialTopic :

echo "Hello, World" | bin/kafka-console-producer.sh --broker-list localhost:9092 --topic TutorialTopic > /dev/null

- Creamos el CONSUMIDOR que lee los mensajes que el PRODUCTOR a publicado: bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic TutorialTopic --from-beginning
- Si todo ha ido bien debemos ver en el terminal:

Output

Hello, World

<u>Si tenemos dos terminales en uno voy lanzando el productor tengo que ver que en el otro terminal (consumidor recibe los mensajes que yo mando)</u>

libraryDependencies += "org.apache.spark" %% "spark-streaming-kafka-0-10" % "2.1.0"

Ejercicio 2

CREAR UN PRODUCTOR EN SCALA y recibir el mensaje desde el consumidor de KAFKA

Antes tenemos que crear el topic sino no tirará

```
import java.util.Properties
import org.apache.kafka.clients.producer.{KafkaProducer, ProducerRecord}
object Ejemplo {
 def main(args: Array[String]): Unit = {
   val props: Properties = new Properties()
   props.put("bootstrap.servers", "localhost:9092")
   props.put("key.serializer",
     "org.apache.kafka.common.serialization.StringSerializer")
   props.put("value.serializer",
     "org.apache.kafka.common.serialization.StringSerializer")
   props.put("acks", "all")
```

```
  val producer = new KafkaProducer[String, String](props)
  val topic = "topic1"

  try {
   for (i <- 0 to 15) {
    val record = new ProducerRecord[String, String](topic, i.toString, "Mi mensajenuevo " + i)
    val metadata = producer.send(record)
    printf(s"sent record(key=%s value=%s) " +
     "meta(partition=%d, offset=%d)\n",
     record.key(), record.value(),
     metadata.get().partition(),
     metadata.get().offset())
   }
  } catch {
   case e: Exception => e.printStackTrace()
  } finally {
   producer.close()
  }
 }
}
```

EL CONSUMIDOR DESDE EL TERMINAL DE LINUX´

bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic topic_text1--from-beginning

EJERCICIO CONSUMIDOR SCALA Y PRODUCTOR DESDE KAFKA TERMINAL LINUX

PRODUCTOR

Echo "Hello, World" | bin/kafka-console-producer.sh --broker-list localhost:9092 --topic topic_text

CONSUMIDOR

```
import java.util.{Collections, Properties}
import java.util.regex.Pattern
import org.apache.kafka.clients.consumer.KafkaConsumer
import scala.collection.JavaConverters._
object ejemplo  extends App {

 val props:Properties = new Properties()
 props.put("group.id", "test")
 props.put("bootstrap.servers","localhost:9092")
 props.put("key.deserializer",
  "org.apache.kafka.common.serialization.StringDeserializer")
 props.put("value.deserializer",
  "org.apache.kafka.common.serialization.StringDeserializer")
 props.put("enable.auto.commit", "true")
 props.put("auto.commit.interval.ms", "1000")
 val consumer = new KafkaConsumer(props)
 val topics = List("topic_text")
 try {
  consumer.subscribe(topics.asJava)
  while (true) {
   val records = consumer.poll(10)
   for (record <- records.asScala) {
    println("Topic: " + record.topic() +",Value: " + record.value())}
  }
 }catch{
  case e:Exception => e.printStackTrace()
 }finally {
  consumer.close()
 }
```

```
}
```

EJERCICIO CONSUMIDOR SCALA Y PRODUCTOR SCALA

PRODUCTOR

```
import java.util.Properties
import org.apache.kafka.clients.producer.{KafkaProducer, ProducerRecord}
object Ejemplo {
  def main(args: Array[String]): Unit = {
    val props: Properties = new Properties()
    props.put("bootstrap.servers", "localhost:9092")
    props.put("key.serializer",
      "org.apache.kafka.common.serialization.StringSerializer")
    props.put("value.serializer",
      "org.apache.kafka.common.serialization.StringSerializer")
    props.put("acks", "all")
    val producer = new KafkaProducer[String, String](props)
    val topic = "topic1"

    try {
      for (i <- 0 to 15) {
        val record = new ProducerRecord[String, String](topic, i.toString, "Mi mensajenuevo " + i)
        val metadata = producer.send(record)
        printf(s"sent record(key=%s value=%s) " +
          "meta(partition=%d, offset=%d)\n",
          record.key(), record.value(),
          metadata.get().partition(),
          metadata.get().offset())
      }
    } catch {
```

```
    case e: Exception => e.printStackTrace()
   } finally {
    producer.close()
   }
  }
}
```

CONSUMIDOR

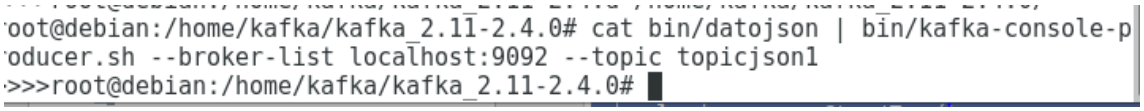
```
import org.apache.kafka.clients.consumer.KafkaConsumer
import scala.collection.JavaConverters._
object casefichero {
 def main(args: Array[String]): Unit = {

   val props: Properties = new Properties()
   props.put("group.id", "test")
   props.put("bootstrap.servers", "localhost:9092")
   props.put("key.deserializer",
    "org.apache.kafka.common.serialization.StringDeserializer")
   props.put("value.deserializer",
    "org.apache.kafka.common.serialization.StringDeserializer")
   props.put("enable.auto.commit", "true")
   props.put("auto.commit.interval.ms", "1000")
   val consumer = new KafkaConsumer(props)
   val topics = List("topic1")
   try {
    consumer.subscribe(topics.asJava)
    while (true) {
     val records = consumer.poll(10)
     for (record <- records.asScala) {
      println("Topic: " + record.topic() +
```

```
          ",Key: " + record.key() +
          ",Value: " + record.value() +
          ", Offset: " + record.offset() +
          ", Partition: " + record.partition())
       }
     }
   } catch {
     case e: Exception => e.printStackTrace()
   } finally {
     consumer.close()
   }
 }
}
```

LECTURA DE UN JSON EN KAFKA –SPARK

En el terminal




```
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.types.{IntegerType, StringType, StructType}
import org.apache.spark.sql.functions.{col, from_json}

import org.apache.spark.sql.sources.v2.reader._


object casefichero {
  def main(args: Array[String]): Unit = {
    val spark = SparkSession.builder().appName("MiApp").master("local[*]").getOrCreate()
    import spark.implicits._
```

```
    val df = spark.readStream
      .format("kafka")
      .option("kafka.bootstrap.servers", "localhost:9092")
      .option("subscribe", "topicjson1")
      .option("startingOffsets", "earliest") // From starting
      .load()

   df.printSchema()
    val personStringDF = df.selectExpr("CAST(value AS STRING)")
    val schema = new StructType()
      .add("nombre",StringType)
      .add("edad",IntegerType)



print("consulta")
    val personDF = personStringDF.select(from_json(col("value"), schema).as("data"))
      .select("data.*")

    print( "abriendo la escritura")
    personDF.writeStream
      .format("console")
      .outputMode("append")
      .start()
      .awaitTermination()

  }
}
```

LECTURA DE JSON CON KAFKA Y FILTRADO DE LOS DATOS A ELIMINAR

```
import org.apache.spark.sql.SparkSession
```

```
import org.apache.spark.sql.types.{IntegerType, StringType, StructType}
import org.apache.spark.sql.functions.{col, from_json}

import org.apache.spark.sql.sources.v2.reader._


object casefichero {
  def main(args: Array[String]): Unit = {
    val spark = SparkSession.builder().appName("MiApp").master("local[*]").getOrCreate()
    import spark.implicits._

    val df = spark.readStream
      .format("kafka")
      .option("kafka.bootstrap.servers", "localhost:9092")
      .option("subscribe", "topic5")
      .option("startingOffsets", "earliest") // From starting
      .load()

    df.printSchema()
    val personStringDF = df.selectExpr("CAST(value AS STRING)")
    val schema = new StructType()
      .add("nombre",StringType)
      .add("edad",IntegerType)


    print("consulta")
    val personDF = personStringDF.select(from_json(col("value"), schema).as("data"))
      .select("data.*").filter("data.nombre != 'pablo'")
```

```
    print( "abriendo la escritura")
    personDF.writeStream
      .format("console")
      .outputMode("append")
      .start()
      .awaitTermination()

  }
}
```