



## Apache Kafka: Proceso ETL utilizando un archivo CSV como origen de datos y una base de datos MySQL como destino.

Apache Kafka es una plataforma distribuida de administración de flujo de datos diseñada para recibir datos de diversas fuentes y distribuirlas a diversos usuarios de forma rápida. Para demostrar la utilidad de Kafka y como funciona el flujo de los datos, realizaremos una aplicación Java con la cual se extraerán datos de un archivo CSV, serán convertidos en formato JSON, enviados a un servidor de Apache Kafka y agrupados en una colección de mensajes (Topic). Una base de datos diseñada para la recolección de los datos del archivo CSV será el consumidor de los mensajes almacenados en Apache Kafka.


### Requisitos Previos:

- Spring Tool Suite 4
- Java Development Kit versión 8 u 11
- Apache Maven 3.0 o posterior
- Archivos binarios de Apache Kafka (Binary downloads) “kafka\_2.12-3.1.0.tgz”
- Terminales de Símbolo del Sistema (cmd)
- MySQL Workbench 8.0 CE
- Docker (Opcional)

### Instrucciones:

Iniciaremos con la instalación de Apache Kafka en nuestro equipo Windows. Requeriremos de los archivos oficiales de Apache Kafka que pueden ser descargados en la dirección: <https://kafka.apache.org/downloads> seleccionando el archivo “kafka\_2.12-3.1.0.tgz” de la sección “Binary downloads”.

---



GET STARTEDDOCSPOWERED BYCOMMUNITY

DOWNLOAD KAFKA

## DOWNLOAD

3.1.0 is the latest release. The current stable version is 3.1.0.

You can verify your download by following these [procedures](#) and using these [KEYS](#).

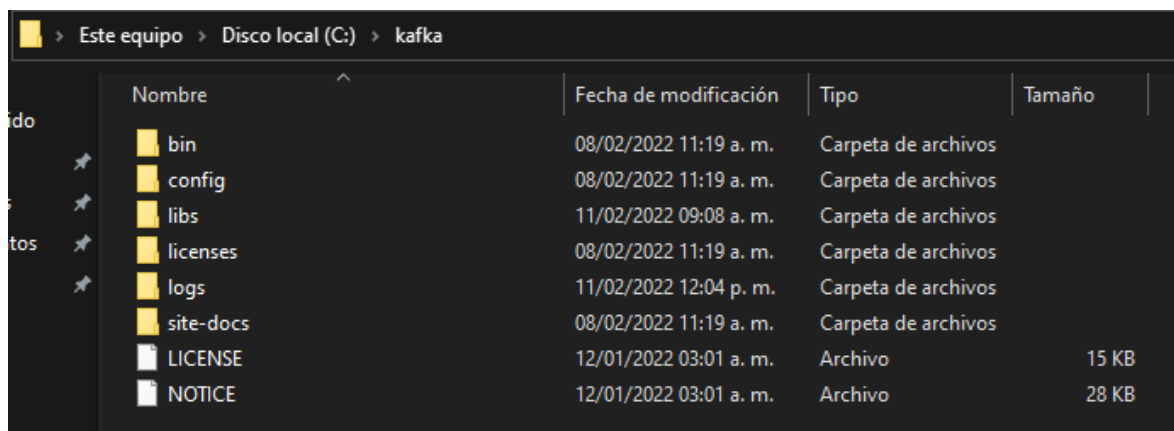
### 3.1.0

- Released January 24, 2022
- [Release Notes](#)
- Source download: [kafka-3.1.0-src.tgz](#) ([asc](#), [sha512](#))
- Binary downloads:
  - Scala 2.12 - [kafka\\_2.12-3.1.0.tgz](#) ([asc](#), [sha512](#))
  - Scala 2.13 - [kafka\\_2.13-3.1.0.tgz](#) ([asc](#), [sha512](#))

We build for multiple versions of Scala. This only matters if you are using Scala and you want a version built for the same Scala version you use. Otherwise any version should work (2.13 is recommended).



Una vez descargado el archivo, se descomprime y se almacena en una ubicación deseada, para efectos de este ejemplo, el fichero fue descomprimido y almacenado en la ruta C:\Kafka y se deberá contar con un contenido como el presentado en la imagen.



Nombre	Fecha de modificación	Tipo	Tamaño
bin	08/02/2022 11:19 a. m.	Carpeta de archivos	
config	08/02/2022 11:19 a. m.	Carpeta de archivos	
libs	11/02/2022 09:08 a. m.	Carpeta de archivos	
licenses	08/02/2022 11:19 a. m.	Carpeta de archivos	
logs	11/02/2022 12:04 p. m.	Carpeta de archivos	
site-docs	08/02/2022 11:19 a. m.	Carpeta de archivos	
LICENSE	12/01/2022 03:01 a. m.	Archivo	15 KB
NOTICE	12/01/2022 03:01 a. m.	Archivo	28 KB

Ya tenemos los archivos necesarios para correr Apache Kafka. Ahora procedemos a iniciar dos servidores que nos darán acceso a los servicios de Kafka.

El primer servidor que correremos se denomina “Zookeeper”, el cual se encarga de la gestión del clusters (uno o más servidores “Brokers”) y registro de cambios en el servidor o en los Topics.

El segundo servidor se denomina “Broker”, el cual será nuestro acceso a Apache Kafka y es donde se contienen los topics con sus respectivas particiones.

En este punto, ofreceremos dos opciones de arranque de estos servidores: por medio de terminales de comandos (cmd) o por medio de Docker. Nuestra sugerencia es realizar el proceso por medio de Docker ya que el arranque de Zookeeper y Broker será más sencillo e incluso será automático al momento de iniciar Docker.

### Arranque de servidores Kafka por medio de CMD

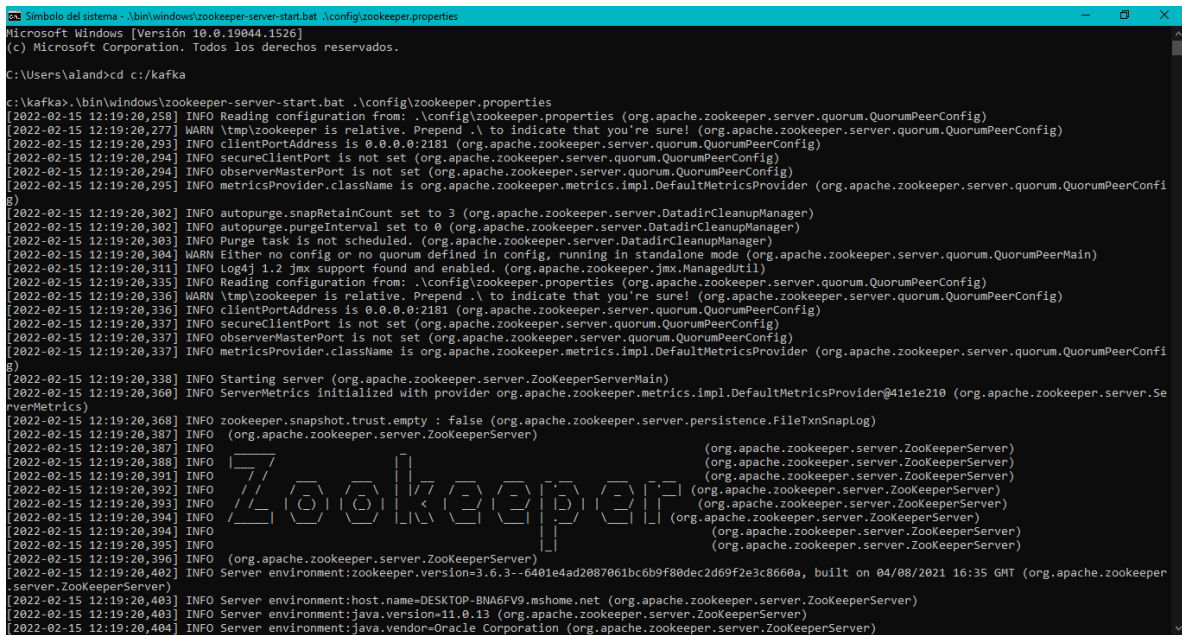
Abriremos dos terminales de comandos (cmd), en una estará corriendo Zookeeper y en otra Broker.

En ambos casos, nos tendremos que colocar en la carpeta donde fueron alojados los archivos de Apache Kafka. En este ejemplo hemos almacenado los archivos en una carpeta alojada en nuestro disco local, por lo que nos ubicaremos en ella con el siguiente comando

```
cd c:/kafka
```

Ahora debemos arrancar primero el servidor Zookeeper con el siguiente comando:

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



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

[illegible]

Es importante que las ventanas estén abiertas todo el tiempo que se requiera el uso de Apache Kafka, así como respetar la sintaxis de los comandos como añadir el punto inicial de cada comando.



## Arranque de servidores Kafka por medio de Docker

Para realizar este procedimiento, necesitaremos del programa Docker Desktop. Su descarga, así como los requisitos necesarios para su funcionamiento pueden ser consultados en la siguiente página: <https://docs.docker.com/desktop/windows/install/>

Una vez descargado e instalado el programa, procedemos a realizar un archivo de configuración llamado **docker-compose.yml**

El archivo puede ser realizado en block de notas o en Notepad++. Copiamos y pegamos el siguiente código y guardamos el archivo con el nombre y extensión ya mencionados.

```
version: "3.7"

networks:
  kafka-net:
    name: kafka-net
    driver: bridge

services:
  zookeeper:
    image: zookeeper:3.7.0
    container_name: zookeeper
    restart: always
    networks:
      - kafka-net
    ports:
      - "2181:2181"

  kafka:
    image: wurstmeister/kafka:2.13-2.7.0
    container_name: kafka
    restart: always
    networks:
      - kafka-net
    ports:
      - "9092:9092"

  environment:
    KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: DOCKER_INTERNAL:PLAINTEXT,DOCKER_EXTERNAL:PLAINTEXT
    KAFKA_LISTENERS: DOCKER_INTERNAL://:29092,DOCKER_EXTERNAL://:9092
    KAFKA_ADVERTISED_LISTENERS: DOCKER_INTERNAL://kafka:29092,DOCKER_EXTERNAL://{DOCKER_HOST_IP:-
127.0.0.1}:9092
    KAFKA_INTER_BROKER_LISTENER_NAME: DOCKER_INTERNAL
    KAFKA_ZOOKEEPER_CONNECT: "zookeeper:2181"
    KAFKA_BROKER_ID: 1
    KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
    depends_on:
      - zookeeper
```



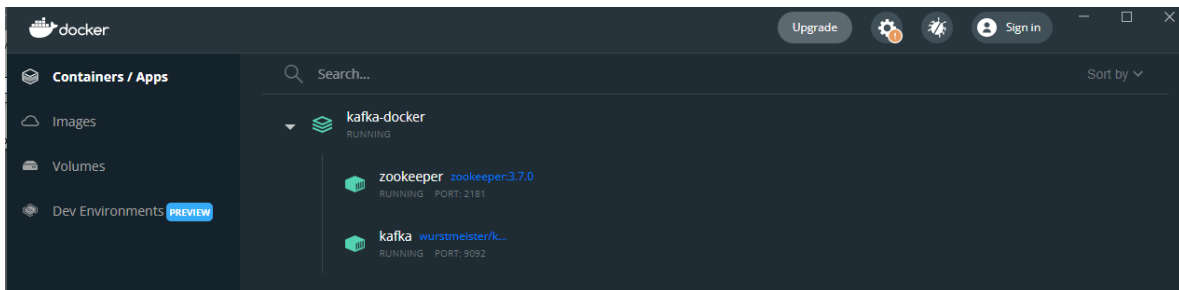
Debemos guardar el archivo en un folder deseado, en este tutorial elegimos crear un folder nuevo en el disco local llamado “Kafka-docker”.

Abrimos una ventana de comandos (cmd) y nos ubicamos en la carpeta donde está alojado el archivo creado, posteriormente colocamos el siguiente comando:

**docker-compose up -d**

```
c:\kafka-docker>docker-compose up -d
[+] Running 3/3
 - Network kafka-net      Created           0.8s
 - Container zookeeper    Started           2.5s
 - Container kafka        Started           4.3s
c:\kafka-docker>
```

Ahora comprobaremos en nuestra aplicación Docker Desktop la creación del contenedor “kafka-docker” con los servidores “kafka” y “zookeeper”, así como visualizar que están corriendo en este momento.



Ahora, cada vez que la aplicación Docker Desktop se ejecute (por defecto se ejecuta automáticamente al iniciar la computadora), también esta corriendo los servidores necesarios para utilizar Apache Kafka.

## Archivo CSV

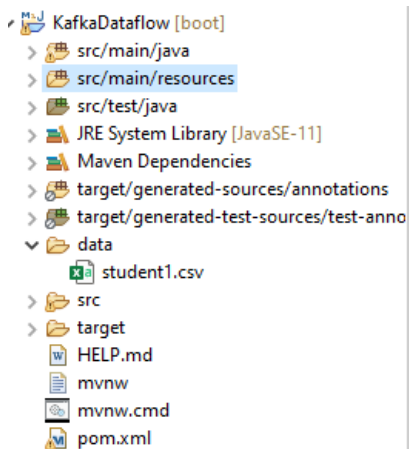
Prepararemos el archivo CSV con el que trabajaremos en este ejemplo. Abrimos el block de notas o Notepad++ y escribimos lo siguiente:

```
studentid,name,department,subject,marks
100,"Amir","CS","DBMS",98
200,"Isbu Noor","IT","Networking",87
300,"Moshin Khan","CE","DBMS",74
400,"Amjath","IT","Networking",99
500,"Muntazir","IT","Security",81
600,"Nasim","IT","Networking",76
700,"Sultan","CS","DBMS",65
800,"Alotaibi","CS","Networking",70
900,"Maria","IT","Cloud",80
1000,"Pedro","CS","DBMS",93
```



Es importante respetar el formato presentado y asegurarnos que no haya un espacio en el último dato ya que pueden presentarse errores. Guardamos en una ubicación accesible el archivo con el nombre y extensión **student1.csv**.

En nuestro proyecto, creamos una nueva carpeta llamada **data** y colocamos dentro de ella el archivo student1.csv recién creado. El esquema se verá así:



### Base de Datos destino.

Realizaremos la base de datos, así como las tablas y columnas necesarias para este ejemplo. Utilizaremos la herramienta MySQL Workbench 8.0 CE, la cual puede ser descargada en la siguiente liga: <https://dev.mysql.com/downloads/workbench/>

Para este ejemplo, realizaremos el proceso desde la instancia local de MySQL (servidor localhost:3306). Iniciamos sesión en la instancia y procedemos a la creación de la base de datos destino, junto con las tablas y columnas requeridas. El siguiente código es el script completo con los requisitos del proyecto:

```
create database kafka;
use kafka;
```

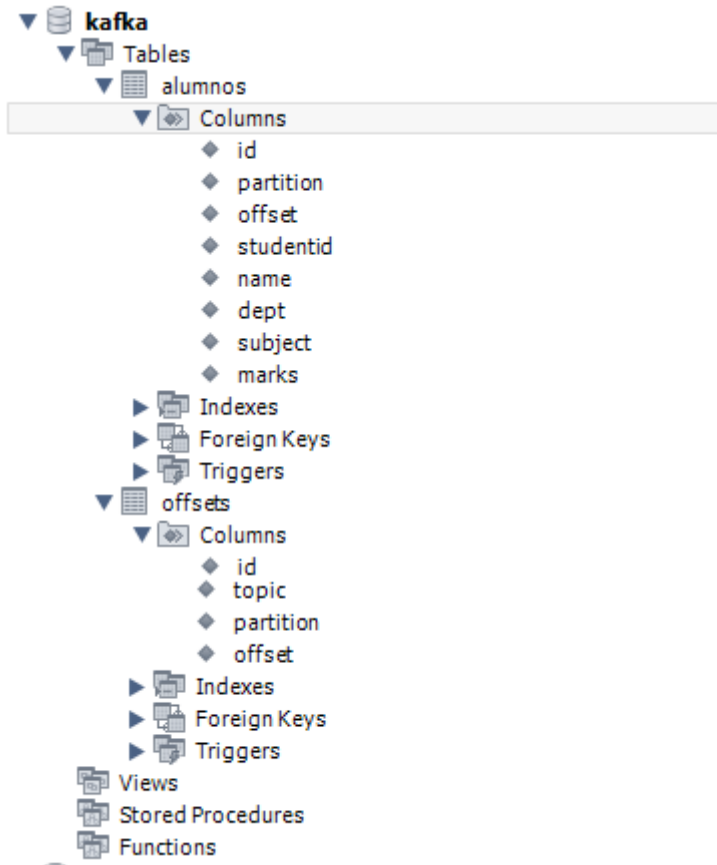
```
CREATE TABLE `kafka`.`offsets` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `topic` VARCHAR(100) NULL,
  `partition` VARCHAR(100) NULL,
  `offset` VARCHAR(100) NULL,
  PRIMARY KEY (`id`));
```

```
CREATE TABLE `kafka`.`alumnos` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `partition` VARCHAR(100) NULL,
  `offset` VARCHAR(100) NULL,
  `studentid` INT NULL,
  `name` VARCHAR(100) NULL,
```



```
`dept` VARCHAR(100) NULL,  
`subject` VARCHAR(100) NULL,  
`marks` DOUBLE NULL,  
PRIMARY KEY (`id`));
```

Hemos creado con éxito nuestra base de datos destino, el esquema se verá así:



### Aplicación Java.

Con los servidores corriendo correctamente, utilizaremos Spring Tools Suite 4 para la creación de un proyecto Java que nos permita leer un archivo CSV, pasar los datos a un Topic de Apache Kafka y contar con un consumidor que estará escuchando los cambios en el Topic, recibirá los datos y los enviará a una base de datos de MySQL.

Crearemos un proyecto nuevo (Spring Starter Project) llamado “KafkaDataflow”, por el momento no requeriremos agregar dependencias por medio del menú de Spring Tools ya que colocaremos el código del archivo pom.xml más adelante. Para este tutorial, seguiremos la siguiente configuración del proyecto.



#### New Spring Starter Project

⚠ A project with name 'KafkaDataflow' already exists in the workspace.



Service URL:

Name:

☒ Use default location

Location:

Type:  Packaging:

Java Version:  Language:

Group:

Artifact:

Version:

Description:

Package:

Working sets

☐ Add project to working sets

Working sets:

Una vez creado el proyecto, nos dirigiremos al archivo **pom.xml** para agregar las dependencias requeridas para este ejemplo.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.6.3</version>
    <relativePath /> <!-- lookup parent from repository -->
  </parent>
  <groupId>com.adl</groupId>
  <artifactId>KafkaDataflow</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>KafkaDataflow</name>
  <description>Demo project for Spring Boot</description>
  <properties>
    <java.version>11</java.version>
  </properties>
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter</artifactId>
    </dependency>

    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
    </dependency>

    <dependency>
      <groupId>org.slf4j</groupId>
      <artifactId>slf4j-simple</artifactId>
      <version>1.7.25</version>
    </dependency>
  </dependencies>
</project>
```





```
<!-- https://mvnrepository.com/artifact/org.springframework.kafka/spring-kafka -->
<dependency>
  <groupId>org.springframework.kafka</groupId>
  <artifactId>spring-kafka</artifactId>
  <version>2.8.2</version>
</dependency>

<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>8.0.15</version>
</dependency>

<!-- https://mvnrepository.com/artifact/com.zaxxer/HikariCP -->
<dependency>
  <groupId>com.zaxxer</groupId>
  <artifactId>HikariCP</artifactId>
  <version>3.2.0</version>
</dependency>

<dependency>
  <groupId>org.apache.kafka</groupId>
  <artifactId>kafka-clients</artifactId>
  <version>3.1.0</version>
</dependency>
<dependency>
  <groupId>org.apache.logging.log4j</groupId>
  <artifactId>log4j-slf4j-impl</artifactId>
  <version>2.11.0</version>
</dependency>

<dependency>
  <groupId>commons-lang</groupId>
  <artifactId>commons-lang</artifactId>
  <version>2.5</version>
</dependency>

<dependency>
  <groupId>com.fasterxml.jackson.core</groupId>
  <artifactId>jackson-databind</artifactId>
  <version>2.11.0</version>
</dependency>

<dependency>
  <groupId>com.opencsv</groupId>
  <artifactId>opencsv</artifactId>
  <version>5.1</version>
</dependency>
</dependencies>

<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>

</project>
```



Continuando con la creación de archivos de configuración, se deben realizar tres archivos que nos ayudarán a conectar nuestro proyecto con el servidor de Apache Kafka y con la base de datos destino de MySQL, así como un archivo de configuración para utilizar log4j2.

Empezaremos con el archivo de configuración para conectar los datos del archivo CSV con un Topic de Apache Kafka. En un block de notas o un nuevo archivo de Notepad++ colocamos el siguiente código:

```
bootstrap.servers=localhost:9092
group.id=group1
topic=nuevosEstudiantes
enable.auto.commit=false
min.batch.size=1
number.of.consumers=3
```

Para este tutorial, utilizaremos un Topic llamado “nuevosEstudiantes”, el cual no está creado aun en el servidor Kafka, pero que será creado y utilizado al momento de correr el proyecto completado.

Guardamos el archivo con el nombre **Consumer.properties** y lo colocamos en una ubicación accesible ya que debemos traspasar el archivo a la carpeta **src/main/resources** dentro del proyecto.

Ahora, realizaremos el archivo de configuración de la base de datos destino. Para ello volveremos a seguir los pasos para la creación del archivo anterior y colocaremos el siguiente código:

```
jdbcUrl=jdbc:mysql://localhost:3306/kafka?useUnicode=true&characterEncoding=UTF-8&serverTimezone=UTC&useSSL=false
dataSource.user=root
dataSource.password=password
dataSource.cachePrepStmts=true
dataSource.prepStmtCacheSize=250
dataSource.prepStmtCacheSqlLimit=2048
maximumPoolSize=10
autoCommit=false
connectionTimeout=2000
```

Se deberá de cambiar los parámetros de data.Source.user y data.Source.password por las credenciales utilizadas para acceder al servidor de MySQL donde está alojada la base de datos creada para este proyecto.

Guardamos el archivo con el nombre **DB.properties** y lo colocamos en una ubicación accesible ya que debemos traspasar el archivo a la carpeta **src/main/resources** dentro del proyecto.

Por último, realizaremos el archivo de configuración de log4j2 utilizando un nuevo archivo del block de notas o Notepad++ y colocaremos el siguiente código:

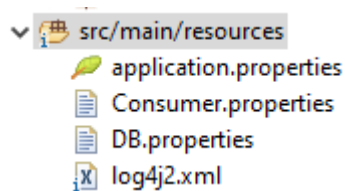
```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="ERROR">
  <Appenders>
```



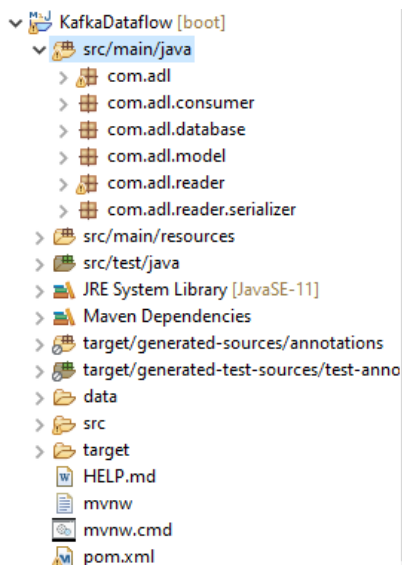
```
<Console name="stdout" target="SYSTEM_OUT">
  <PatternLayout pattern="%d] (%c) - %p %m %n" />
</Console>
</Appenders>
<Loggers>
  <Root level="error">
    <AppenderRef ref="stdout"></AppenderRef>
  </Root>
  <Logger name="org.apache.kafka.clients" level="warn" additivity="false">
    <AppenderRef ref="stdout" />
  </Logger>
  <Logger name="tu.cit.examples.kafkaapis" level="trace" additivity="false">
    <AppenderRef ref="stdout"></AppenderRef>
  </Logger>
</Loggers>
</Configuration>
```

Guardamos el archivo con el nombre **log4j2.xml** y lo colocamos en una ubicación accesible ya que debemos traspasar el archivo a la carpeta **src/main/resources** dentro del proyecto.

El esquema de la carpeta **src/main/resources** deberá verse de la siguiente manera:



Continuamos con la creación de nuestros paquetes de clases (package) dentro de la carpeta **src/main/java** obteniendo un esquema como el presentado en la siguiente imagen:





Empezaremos configurando la clase principal del proyecto llamada **KafkaDataflowApplication** ubicada en el paquete **com.adl**

Colocaremos el siguiente código:

```
package com.adl;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.Properties;

import org.apache.kafka.clients.admin.NewTopic;
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.ProducerConfig;
import org.apache.kafka.clients.producer.ProducerRecord;
import org.apache.kafka.common.serialization.StringSerializer;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
import org.springframework.kafka.config.TopicBuilder;

import com.adl.consumer.Consumer;
import com.adl.consumer.ConsumerGroup;
import com.adl.consumer.ConsumerThread;
import com.adl.database.MySQLDatabase;
import com.adl.model.student;
import com.adl.reader.ReadCSV;
import com.adl.reader.serializer.JsonSerializer;

@SpringBootApplication
public class KafkaDataflowApplication {

    static String nombreTopic= "nuevosEstudiantes";

    @Bean
    public NewTopic topic() {
        return TopicBuilder.name(nombreTopic)
            .partitions(5)
            .replicas(1)
            .build();
    }

    public static void main(String[] args) throws FileNotFoundException, IOException {
        final Logger logger = LogManager.getLogger();
        SpringApplication.run(KafkaDataflowApplication.class, args);

        Properties props = new Properties();
        props.put(ProducerConfig.CLIENT_ID_CONFIG, "my-app-readcsv");
        props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");
```



```
        props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
StringSerializer.class.getName());
        props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
JsonSerializer.class);

        logger.info("Producer has been created...Start sending Student Record ");

        KafkaProducer<String,student> producer = new
KafkaProducer<String,student>(props);

        ReadCSV readCSV = new ReadCSV();
        List studentList = readCSV.ReadCSVFile();
        for (Object studentObject : studentList) {
            student stdobject = (student) studentObject;
            producer.send(new ProducerRecord<String,
student>(nombreTopic,stdobject.getDept(),stdobject));
        }
        logger.info("Producer has sent all employee records successfully...");
        producer.close();

        String databaseConfig = "src\\main\\resources\\DB.properties";
        MySQLDatabase database = new MySQLDatabase(databaseConfig);

        String consumerConfig = "src\\main\\resources\\Consumer.properties";
        Properties consumerProperties = new Properties();
        consumerProperties.load(new FileInputStream(consumerConfig));

        ConsumerGroup consumerGroup = new ConsumerGroup(consumerProperties,
database);

        List<Consumer> consumers = new ArrayList<>();
        for (int i = 0; i <
Integer.parseInt(consumerProperties.getProperty("number.of.consumers")); i++) {
            consumers.add(new ConsumerThread(consumerProperties, database));
        }

        consumerGroup.assignListConsumers(consumers);

        consumerGroup.run();

    }

}
```

En esta clase se creará el Topic de Apache Kafka con el que trabajaremos, así como hará uso de los archivos de configuración de Kafka y la base de datos a su vez que invocará a la clase que leerá los datos del CSV y los colocará en una lista.

Continuamos con el paquete **com.adl.consumer** en donde estaremos realizando tres clases para el manejo de nuestro consumidor así como una interfaz para la base de datos.



Dentro del paquete, creamos la interfaz **Consumer.java** y le colocaremos el siguiente código:

```
package com.adl.consumer;

import org.apache.kafka.common.TopicPartition;

import java.util.concurrent.CountDownLatch;

public interface Consumer extends Runnable {
    void shutdown();
    String topic();
    String groupID();
    String bootstrapServer();
    void setLatch(CountDownLatch latch);
    long position(TopicPartition partition);
    void seek(TopicPartition partition, long offset);
}
```

Continuaremos con la creación de la clase **ConsumerGroup.java** y le colocaremos el siguiente código:

```
package com.adl.consumer;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import com.adl.database.Database;
import java.util.ArrayList;
import java.util.List;
import java.util.Properties;
import java.util.concurrent.CountDownLatch;

public class ConsumerGroup {

    private final Logger logger = LoggerFactory.getLogger(ConsumerGroup.class.getName());
    private CountDownLatch latch;
    private List<Consumer> consumers = new ArrayList<>();
    private Database database;

    private final String bootstrapServer;
    private final String groupID;
    private final String topic;

    public ConsumerGroup(Properties properties, Database database) {
        this.bootstrapServer = properties.getProperty("bootstrap.servers");
        this.groupID = properties.getProperty("group.id");
        this.topic = properties.getProperty("topic");
        this.database = database;
    }

    public void assignListConsumers(List<Consumer> consumers) {
        this.consumers = consumers;

        latch = new CountDownLatch(this.consumers.size());
    }
}
```



```
        for (Consumer consumer : this.consumers) {
            if (!consumer.topic().equals(this.topic) ||
!consumer.bootstrapServer().equals(this.bootstrapServer) ||
!consumer.groupID().equals(this.groupID)) {
                throw new IllegalArgumentException("Parameter of Consumer Group and
Consumer is different");
            }

            consumer.setLatch(latch);
        }
    }

    public void run() {
        for (Consumer consumer: consumers) {
            Thread thread = new Thread(consumer);
            thread.start();
        }

        Runtime.getRuntime().addShutdownHook(new Thread(() -> {
            logger.info("Caught shutdown hook");

            for (Consumer consumer : consumers) {
                consumer.shutdown();
            }

            await(this.latch);
            database.shutdown();

            logger.info("Consumer Group has exited");
        }));

        await(latch);
        database.shutdown();
    }

    private void await(CountDownLatch latch) {
        try {
            latch.await();
        } catch (InterruptedException e) {
            logger.error("Application got interrupted", e);
        } finally {
            logger.info("Application is closing");
        }
    }
}
```

Creamos una nueva clase llamada **ConsumerThread.java** y colocamos el siguiente código:

```
package com.adl.consumer;
```

```
import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.KafkaConsumer;
import org.apache.kafka.common.TopicPartition;
import org.apache.kafka.common.errors.WakeupException;
import org.apache.kafka.common.serialization.StringDeserializer;
import org.slf4j.Logger;
```



```
import org.slf4j.LoggerFactory;
import org.springframework.kafka.support.serializer.JsonDeserializer;
import com.adl.database.Database;
import com.adl.model.student;
import java.sql.*;
import java.time.Duration;
import java.util.*;
import java.util.concurrent.CountDownLatch;

public class ConsumerThread implements Consumer {

    private final Logger logger =
        LoggerFactory.getLogger(ConsumerThread.class.getName());

    private String bootstrapServer;
    private String topic;
    private String groupId;
    private KafkaConsumer<String, student> consumer;
    private List<ConsumerRecord<String, student>> buffer = new ArrayList<>();

    private long minBatchSize;
    private CountDownLatch latch;
    private Database database;

    public ConsumerThread(Properties properties, Database database) {
        this.latch = new CountDownLatch(0); // default is 0 so this consumerThread can
run independent
        this.database = database;

        this.bootstrapServer = properties.getProperty("bootstrap.servers");
        this.groupID = properties.getProperty("group.id");
        this.topic = properties.getProperty("topic");
        this.minBatchSize = Long.parseLong(properties.getProperty("min.batch.size"));

        this.consumer = new KafkaConsumer<>(properties, new StringDeserializer(), new
JsonDeserializer<>(student.class));
        this.consumer.subscribe(Collections.singletonList(this.topic), new
MyConsumerRebalanceListener(this, this.database));
    }

    @Override
    public void run() {
        try {
            while (true) {
                ConsumerRecords<String, student> records =
this.consumer.poll(Duration.ofMillis(1000));

                for (ConsumerRecord<String, student> record : records) {
                    buffer.add(record);
                }

                if (buffer.size() >= this.minBatchSize) {
                    database.insertMessageToDB(buffer);
                    buffer.clear();
                }
            }
        }
    }
}
```





```
    }

    } catch (WakeupException e) {
        logger.info("Received shutdown signal");
    } catch (BatchUpdateException batchUpdateException) {
        logger.error("Batch Update exception", batchUpdateException);
    } catch (Exception e) {
        logger.error("Database Exception", e);
        e.printStackTrace();
    } finally {
        this.consumer.close();
        this.latch.countDown();
    }
}

@Override
public String topic() {
    return this.topic;
}

@Override
public String groupID() {
    return this.groupID;
}

@Override
public String bootstrapServer() {
    return this.bootstrapServer;
}

// set latch to join a consumer group and run together
@Override
public void setLatch(CountDownLatch latch) {
    this.latch = latch;
}

@Override
public long position(TopicPartition partition) {
    return this.consumer.position(partition);
}

@Override
public void seek(TopicPartition partition, long offset) {
    this.consumer.seek(partition, offset);
}

public void shutdown() {
    this.consumer.wakeup();
}
}
```



La ultima clase a crear en este paquete será la llamada **MyConsumerRebalanceListener** y contendrá el siguiente código:

```
package com.adl.consumer;

import org.apache.kafka.clients.consumer.ConsumerRebalanceListener;
import org.apache.kafka.common.TopicPartition;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import com.adl.database.Database;
import java.util.Collection;

public class MyConsumerRebalanceListener implements ConsumerRebalanceListener {

    private Logger logger = LoggerFactory.getLogger(MyConsumerRebalanceListener.class);
    private Database database;
    private Consumer consumer;

    public MyConsumerRebalanceListener(Consumer consumer, Database database) {
        this.consumer = consumer;
        this.database = database;
    }

    @Override
    public void onPartitionsRevoked(Collection<TopicPartition> partitions) {
        try {
            for (TopicPartition partition : partitions) {
                this.database.saveOffset(partition.topic(), partition.partition(),
consumer.position(partition));
            }
        } catch (Exception e) {
            logger.error("Database Exception", e);
            this.consumer.shutdown();
        }
    }

    @Override
    public void onPartitionsAssigned(Collection<TopicPartition> partitions) {
        try {
            for (TopicPartition partition : partitions) {
                this.consumer.seek(partition, this.database.getOffset(partition.topic(),
partition.partition()));
            }
        } catch (Exception e) {
            logger.error("Database Exception", e);
            this.consumer.shutdown();
        }
    }
}
```

Ahora, colocaremos dentro del paquete **com.adl.database** una interfaz y una clase.

La interfaz tendrá de nombre **Database.java** y tendrá el siguiente código:



```
package com.adl.database;

import org.apache.kafka.clients.consumer.ConsumerRecord;
import com.adl.model.student;
import java.sql.SQLException;
import java.util.List;

public interface Database {

    void shutdown();
    void insertMessageToDB(List<ConsumerRecord<String, student>> records) throws
Exception;
    void saveOffset(String topic, int partition, long offset) throws Exception;
    long getOffset(String topic, int partition) throws SQLException;
}
```

La clase se llamará **MySQLDatabase** y contendrá el siguiente código:

```
package com.adl.database;

import com.adl.model.student;
import com.zaxxer.hikari.HikariConfig;
import com.zaxxer.hikari.HikariDataSource;
import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.HashMap;
import java.util.List;

public class MySQLDatabase implements Database {

    private HikariDataSource db;
    private final Logger logger = LoggerFactory.getLogger(Database.class.getName());

    public MySQLDatabase(String configFile) {
        HikariConfig config = new HikariConfig(configFile);
        db = new HikariDataSource(config);
    }

    @Override
    public void shutdown() {
        db.close();
    }

    @Override
    public void insertMessageToDB(List<ConsumerRecord<String, student>> records) throws
SQLException {
        Connection connection = db.getConnection();
```



```
String query = "INSERT IGNORE INTO `alumnos`(`partition`, `offset`, `studentid`,  
`name`, `dept`, `subject`, `marks`) VALUE (?, ?, ?, ?, ?, ?, ?)";  
PreparedStatement preparedStatementInsert = connection.prepareStatement(query);  
  
ConsumerRecord<String, student> lastRecord = null;  
HashMap<Integer, Long> offsetMap = new HashMap<>();  
  
for (ConsumerRecord<String, student> record : records) {  
    logger.info("Key: " + record.key() + ", Value: " + record.value() + " -  
Partition: " + record.partition() + ", Offset: " + record.offset());  
  
    preparedStatementInsert.setInt(1, record.partition());  
    preparedStatementInsert.setLong(2, record.offset());  
    preparedStatementInsert.setInt(3, record.value().getStudentid());  
    preparedStatementInsert.setString(4, record.value().getName());  
    preparedStatementInsert.setString(5, record.value().getDept());  
    preparedStatementInsert.setString(6, record.value().getSubject());  
    preparedStatementInsert.setDouble(7, record.value().getMarks());  
  
    preparedStatementInsert.addBatch();  
  
    if (offsetMap.get(record.partition()) == null ||  
offsetMap.get(record.partition()) < record.offset()) {  
        offsetMap.put(record.partition(), record.offset());  
    }  
  
    lastRecord = record;  
}  
  
preparedStatementInsert.executeBatch();  
  
if (lastRecord != null) {  
    PreparedStatement preparedStatementSaveOffset =  
getSaveOffsetStatement(connection, lastRecord.topic(), offsetMap);  
    preparedStatementSaveOffset.executeBatch();  
}  
  
connection.commit();  
connection.close();  
}  
  
@Override  
public void saveOffset(String topic, int partition, long offset) throws SQLException  
{  
    Connection connection = db.getConnection();  
  
    HashMap<Integer, Long> offsetMap = new HashMap<>(partition, offset);  
    PreparedStatement preparedStatement = getSaveOffsetStatement(connection, topic,  
offsetMap);  
    preparedStatement.executeBatch();  
  
    connection.commit();  
    connection.close();  
}
```



```
@Override
public long getOffset(String topic, int partition) throws SQLException {
    Connection connection = db.getConnection();

    String query = "SELECT `offset` FROM `offsets` WHERE `topic` = ? AND `partition`
= ? LIMIT 1";
    PreparedStatement preparedStatement = connection.prepareStatement(query);
    preparedStatement.setString(1, topic);
    preparedStatement.setInt(2, partition);
    ResultSet resultSet = preparedStatement.executeQuery();

    if (resultSet.next()) {
        return resultSet.getLong("offset") + 1L;
    }

    resultSet.close();
    connection.close();

    return 0;
}

private PreparedStatement getSaveOffsetStatement(Connection connection, String topic,
HashMap<Integer, Long> offsetMap) throws SQLException {
    String query = "INSERT INTO `offsets`(`topic`, `partition`, `offset`) value (?,
?, ?) ON DUPLICATE KEY UPDATE `offset` = ?";

    PreparedStatement preparedStatement = connection.prepareStatement(query);

    Long offset;
    for (Integer partition : offsetMap.keySet()) {
        offset = offsetMap.get(partition);
        preparedStatement.setString(1, topic);
        preparedStatement.setInt(2, partition);
        preparedStatement.setLong(3, offset);
        preparedStatement.setLong(4, offset);
        preparedStatement.addBatch();
    }

    return preparedStatement;
}
}
```

Continuaremos con el paquete **com.adl.model** el cual contendrá una sola clase llamada **student.java** con el siguiente código:

```
package com.adl.model;

import com.opencsv.bean.CsvBindByName;

public class student {
    @CsvBindByName
    public int studentid;

    @CsvBindByName
    public String name;
}
```



```
@CsvBindByName(column="department")
public String dept;

@CsvBindByName
public String subject;

@CsvBindByName
public double marks;

public int getStudentid() {
    return studentid;
}

public void setStudentid(int studentid) {
    this.studentid = studentid;
}

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public String getDept() {
    return dept;
}

public void setDept(String dept) {
    this.dept = dept;
}

public String getSubject() {
    return subject;
}

public void setSubject(String subject) {
    this.subject = subject;
}

public double getMarks() {
    return marks;
}

public void setMarks(double marks) {
    this.marks = marks;
}
}
```

El paquete **com.adl.reader** contendrá la clase que leerá los datos desde el archivo CSV. Tendrá de nombre **ReadCSV.java** y tendrá el siguiente código:

```
package com.adl.reader;
```



```
import com.adl.model.student;
import com.opencsv.CSVReader;
import com.opencsv.bean.CsvToBean;
import com.opencsv.bean.CsvToBeanBuilder;

import java.io.FileReader;
import java.util.List;

public class ReadCSV {
    public String csvFileName = "data/student1.csv";
    public List stdlist;
    public List ReadCSVFile() {

        try {
            CSVReader csvReader = new CSVReader(new FileReader(csvFileName));

            CsvToBean csvToBean = new CsvToBeanBuilder(csvReader)
                .withType(student.class)
                .withIgnoreLeadingWhiteSpace(true).build();

            stdlist = csvToBean.parse();

            csvReader.close();
        } catch (Exception FileNotFoundException){
            System.out.println("File is not available...");
        }

        return stdlist;
    }
}
```

El último paquete será el **com.adl.reader.serializer** y contendrá la clase **JsonSerializer.java** con el siguiente código:

```
package com.adl.reader.serializer;

import org.apache.kafka.common.errors.SerializationException;
import org.apache.kafka.common.serialization.Serializer;
import com.fasterxml.jackson.databind.ObjectMapper;
import java.util.Map;

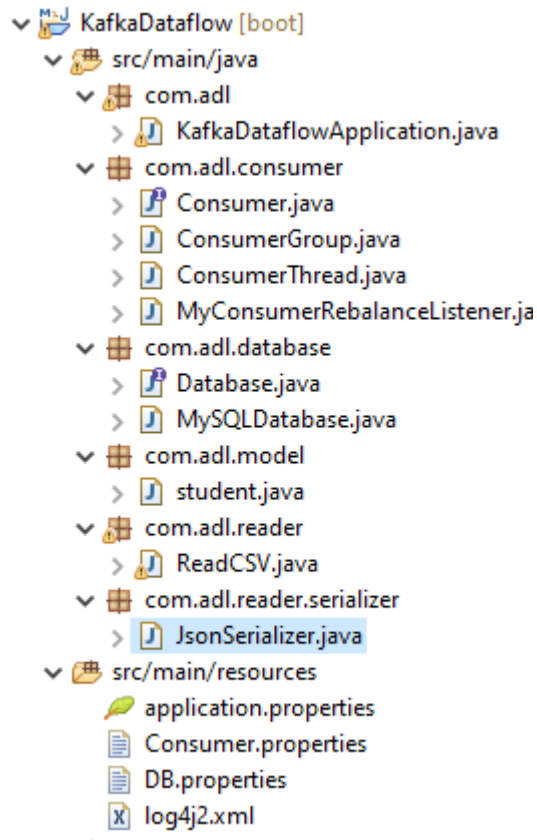
public class JsonSerializer<T> implements Serializer<T> {
    private final ObjectMapper objectMapper = new ObjectMapper();
    public JsonSerializer(){}

    // @Override
    public void configure(Map<String,?> config, boolean isKey){}
    public byte[] serialize(String topic, T data) {
        if (data == null){
            return null;
        }

        try{
            return objectMapper.writeValueAsBytes(data);
        } catch (Exception e){
            throw new SerializationException("Error serializing JSON message",e);
        }
    }
}
```



```
}  
}  
  
public void close(){}  
  
}
```



### Probar la aplicación

Ya tenemos todo listo para probar la aplicación. Nos aseguraremos de que se encuentren corriendo correctamente los servidores Kafka y Zookeeper, así como tener acceso al contenido de la base de datos con el query:

```
select * from `kafka`.`alumnos`;
```

Adicionalmente, podemos crear un consumidor extra en una ventana de comandos que nos permitirá ver en tiempo real los datos que recibe el topic. Su creación se hace mediante la ventana de comandos (cmd) posicionándonos en la carpeta donde está alojado Kafka **cd C:\kafka** y colocando el siguiente comando:

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



Por último, verificaremos nuestra base de datos y comprobaremos si fueron enviados correctamente los datos:



	id	partition	offset	studentid	name	dept	subject	marks
▶	1	3	0	100	Amir	CS	DBMS	98
	2	3	1	700	Sultan	CS	DBMS	65
	3	3	2	800	Alotaibi	CS	Networking	70
	4	3	3	1000	Pedro	CS	DBMS	93
	5	3	4	100	Amir	CS	DBMS	98
	6	3	5	700	Sultan	CS	DBMS	65
	7	3	6	800	Alotaibi	CS	Networking	70
	8	3	7	1000	Pedro	CS	DBMS	93
	9	4	0	300	Moshin Khan	CE	DBMS	74
	10	0	0	200	Isbu Noor	IT	Networking	87
	11	4	1	300	Moshin Khan	CE	DBMS	74
	12	0	1	400	Amjath	IT	Networking	99
	13	0	2	500	Muntazir	IT	Security	81
	14	0	3	600	Nasim	IT	Networking	76
	15	0	4	900	Maria	IT	Cloud	80
	16	0	5	200	Isbu Noor	IT	Networking	87
	17	0	6	400	Amjath	IT	Networking	99
	18	0	7	500	Muntazir	IT	Security	81
	19	0	8	600	Nasim	IT	Networking	76

Repositorio GitHub:

El proyecto generado en este ejemplo puede ser descargado en el siguiente repositorio de GitHub:

**<https://github.com/PADSA-github/Java-Avanzado/tree/main/Apache-Kafka>**