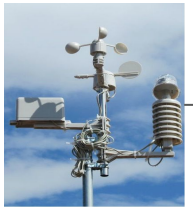
PRÁCTICA DE PROCESAMIENTO DE STREAMS

Máster IoT & Data Science
CIFF Business School

Estela Romero Muñoz

ESQUEMA DEL DISEÑO



APACHE kafka

Spark Streaming

cassandra

Dashboard

ADQUISICIÓN Y ALMACENAMIENTO INTERMEDIO

PROCESAMIENTO

ALMACENAMIENTO FINAL Y CONSULTA

CAPA DE ADQUISICIÓN Y ALMACENAMIENTO INTERMEDIO

- La fuente de datos va a ser un fichero .csv que recoge información de miles de estaciones meteorológicas cada hora.
- Este fichero va a ser ingerido por Kafka, haciendo uso de Kafka Connect, sistema para importar/exportar datos streaming a gran escala, mediante un Source Connector.
- Sólo hay un fichero, por lo que habrá una partición. Los mensajes serán cada una de las filas del fichero.

CAPA DE PROCESAMIENTO

- Los datos procedentes de Kafka serán consumidos por Spark Streaming a través del Sink Connector, usando Kafka Streams (lo veremos más adelante).
- Spark Streaming tomará el stream, en microbatches de 1 segundo, y los procesará.
- Entre otras, realizará las siguientes operaciones :
  - Copiar los datos tal cual en una tabla de una base de datos Cassandra
  - Calcular near real-time agregados de temperatura, como la temperatura máxima, la mínima y la media diarias y almacenarlos en otra tabla.
  - Calcular near real-time agregados de precipitaciones, como las precipitaciones diarias.

CAPA DE ALMACENAMIENTO FINAL Y CONSULTA (I)

- Los datos obtenidos del procesamiento de los datos en Spark son almacenados en tablas de una base de datos Cassandra.
- Sobre esas tablas se harán queries para visualizar los datos en el dashboard.
- Tablas:

```
CREATE TABLE datos_meteorologicos (
    estacion_meteorologica text,
    anyo int,
    mes int,
    dia int,
    hora int,
    temperatura double,
    precip_hora double,
    PRIMARY KEY((estacion_meteorologica), anyo, mes, dia, hora))
)CLUSTERING ORDER BY (anyo DESC, mes DESC, dia DESC, hora DESC);
```

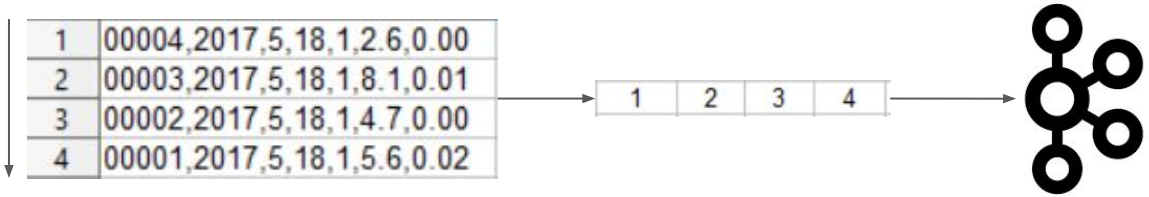
```
CREATE TABLE estacion_meteorologica (
    id text PRIMARY KEY,
    nombre text,
    pais text,
    comunidad_autonoma text,
    provincia text,
    poblacion text,
    lat double,
    long double,
    elevation double
);
```

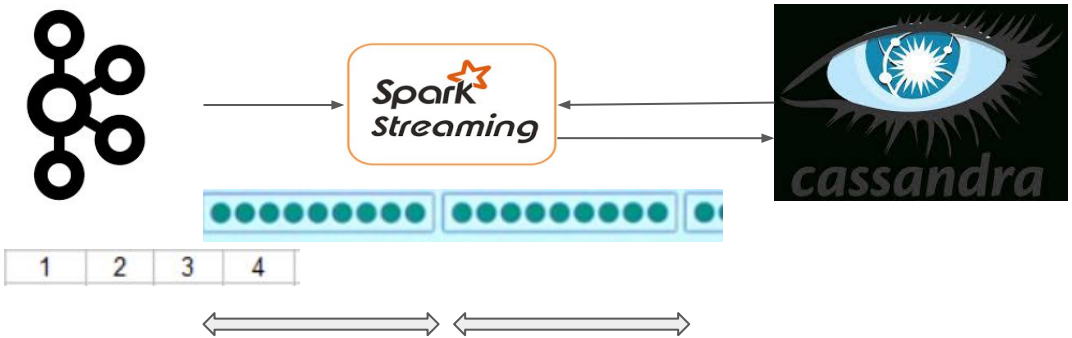
CAPA DE ALMACENAMIENTO FINAL Y CONSULTA (II)

```
CREATE TABLE datos_diaros_temperatura (
    estacion_meteorologica text,
    anyo int,
    mes int,
    dia int,
    temp_max double,
    temp_min double,
    temp_media double,
    PRIMARY KEY((estacion_meteorologica), anyo, mes, dia))
)WITH CLUSTERING ORDER BY (anyo DESC, mes DESC, dia DESC);
```

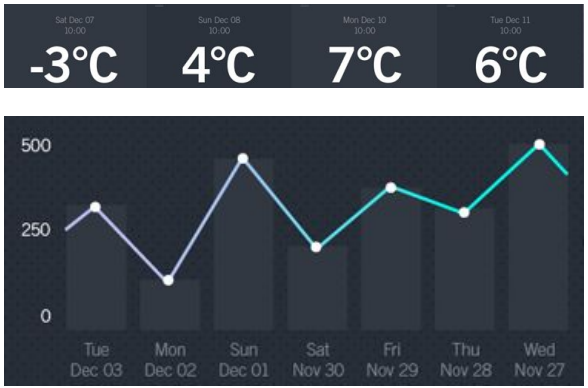
```
CREATE TABLE datos_diaros_precipitaciones (
    estacion_meteorologica text,
    anyo int,
    mes int,
    dia int,
    precipitacion counter,
    PRIMARY KEY((estacion_meteorologica), anyo, mes, dia))
)WITH CLUSTERING ORDER BY (anyo DESC, mes DESC, dia DESC);
```

UN POCO MÁS DETALLADO (I)...





Operaciones sobre microbatches de 1 seg



UN POCO MÁS DETALLADO (II)...

- Miles de estaciones meteorológicas nos van a enviar sus datos de temperatura y precipitación cada hora, de manera secuencial.
- Vamos a simular esos datos con un fichero .csv donde cada fila contiene la información que una estación en concreto nos envía a una hora en concreto.
- Los datos han sido insertados de tal forma que la primera fila contiene la información más reciente.
- Este fichero .csv es ingerido por Kafka mediante Kafka Connect y las filas son consumidas por Spark Streaming a través de Kafka Streams.
- El motor de computación va a ser Scala.
- Se crea el contexto en Spark Streaming

```
val sc = ...
val ssc = new StreamingContext(sc, Seconds(1))
```

- Después creamos los streams transformando las filas en formato String separadas por comas que nos llegan de Kafka en elementos de la clase DatosMeteorologicos que cuadran con los elementos de la tabla de almacenamiento de Cassandra

```
val kafkaStream = KafkaUtils.createStream[String, String, StringDecoder, StringDecoder](ssc,
kafkaParams.Map(KafkaTopicRaw -> 1), StorageLevel.DISK_ONLY_2).map(_._2.split(","))
.map(DatosMeteorologicos(_))
```

UN POCO MÁS DETALLADO (III)...

- Una de las operaciones a realizar es almacenar directamente los datos tal cual llegan a la tabla datos_meteorologicos.

```
kafkaStream.saveToCassandra(CassandraKeyspace, CassandraTableDatos)
```

- Otra operación sería , por ejemplo, calcular el agregado precipitaciones diarias.

```
kafkaStream.map{tiempo => tiempo.estacionid, mes.anyo, tiempo.mes, tiempo.dia, tiempo.precipHora)
}.saveToCassandra(CassandraKeySpace, CassandraTablePrecipDiaria)
```

ARGUMENTACIÓN

¿POR QUÉ KAFKA?

- Kafka Connect nos permite importar y exportar datos streaming a gran escala y nos permite definir los offset.
- Teniendo Kafka Connect en el extremo de entrada a la capa de procesamiento garantizamos escalabilidad y tolerancia a fallos, y crear un conector es relativamente simple .

¿POR QUÉ SPARK STREAMING?

- La llegada de los datos es secuencial, una vez cada hora por estación. No está basada en eventos asíncronos. La gran cantidad de número de estaciones meteorológicas y el hecho de estar calculando agregados cada hora hace que surja la necesidad de procesar en microbatches.
- Al trabajar con Resilient Distributed Datasets (RDDs), podemos reusar RDDs ya existentes y realizar una serie de operaciones con agregados que Cassandra no permite.

¿POR QUÉ CASSANDRA?

- Cassandra es una gran base de datos para almacenar datos relacionados con series temporales. Se puede elegir cómo se almacenan estos datos en disco y tiene escalabilidad lineal. Además, es datacenter y rack aware por lo que se puede usar para hacer analítica directamente sobre ella.

FIN