Afi Escuela

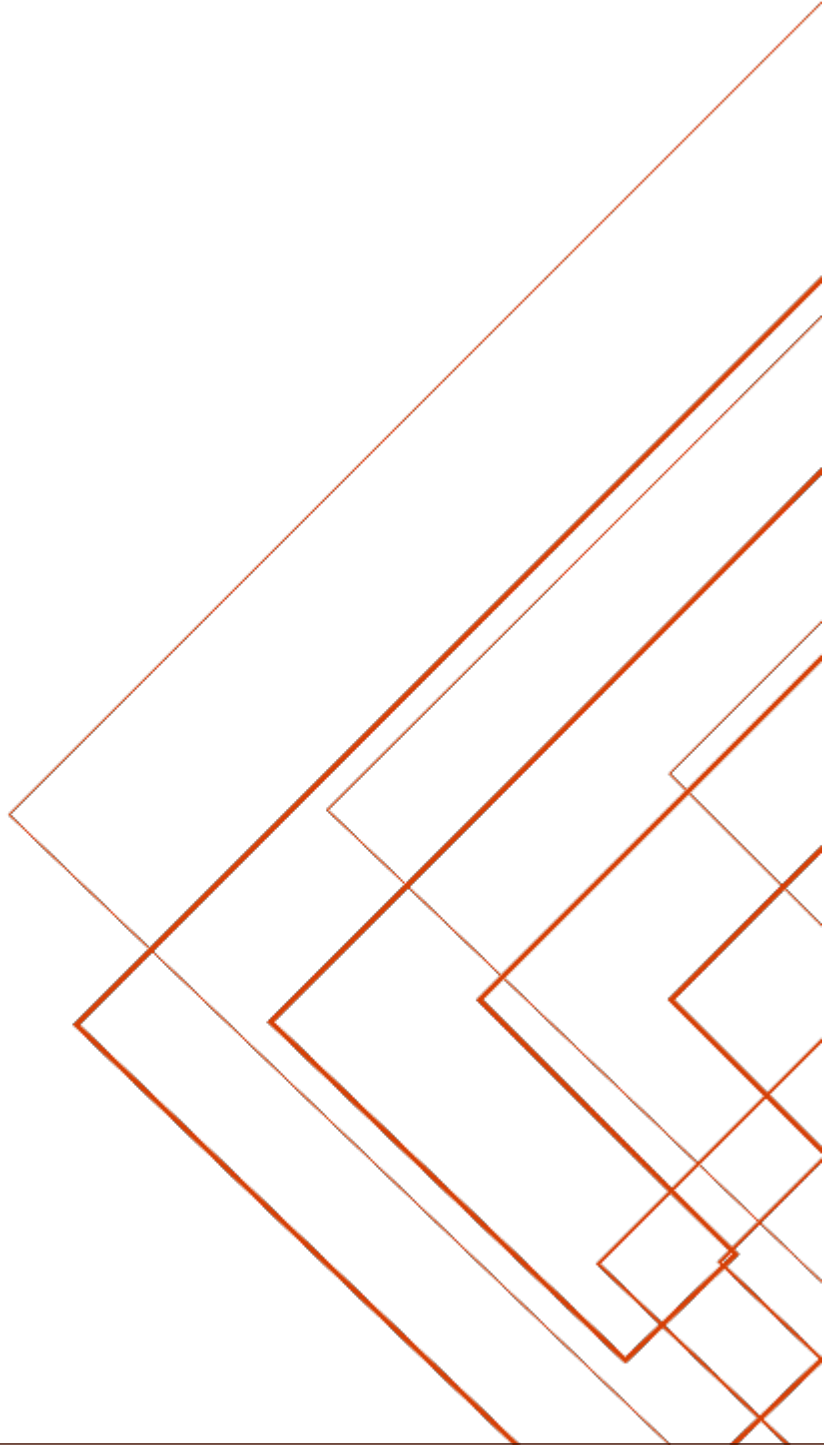
Apache Spark - Streaming

Máster en Data Sience y Big Data

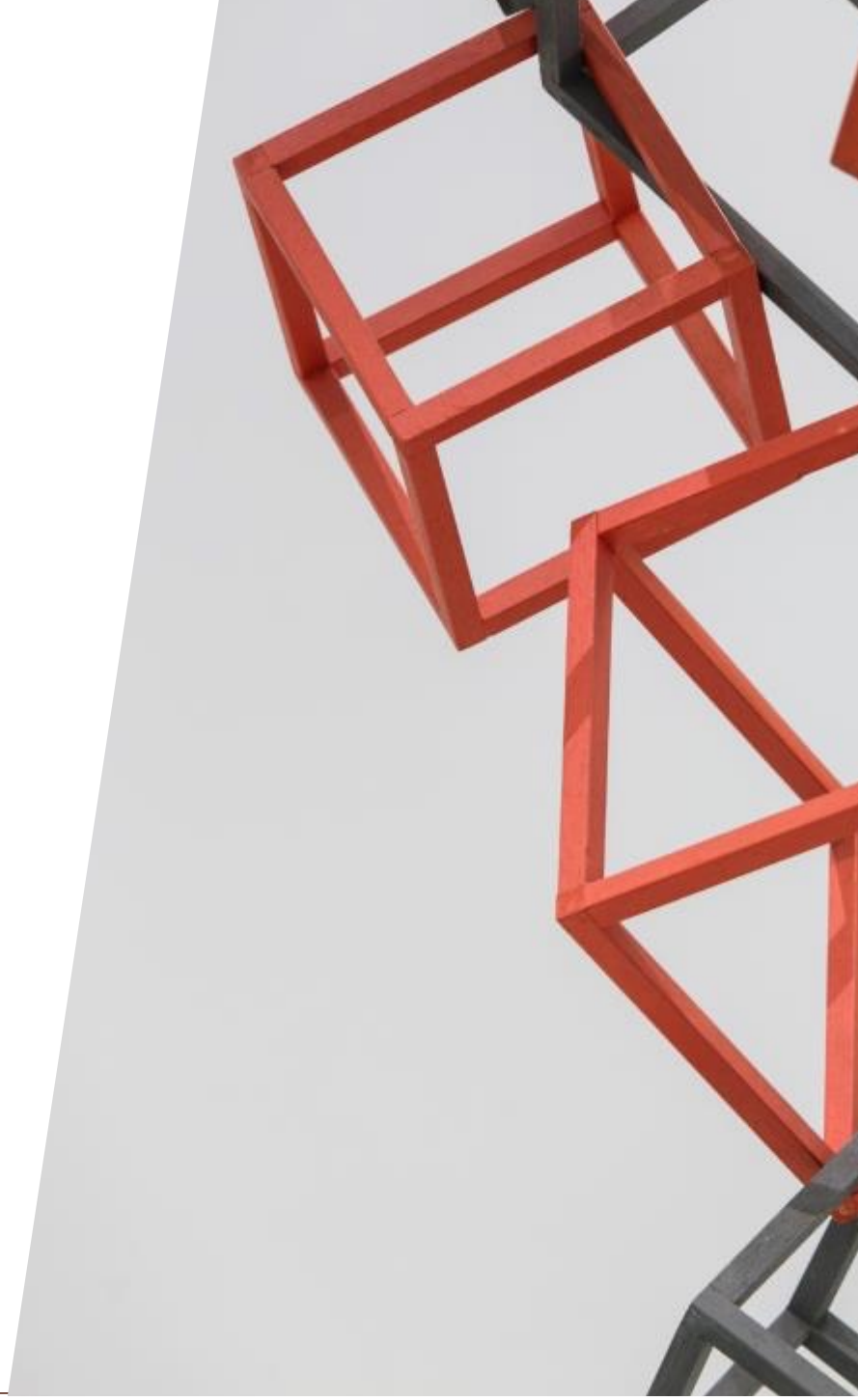
Miguel Ángel Corella
mcorella@geoblink.com

Marzo 2022

Contenido



Introducción



¿Qué es un stream de datos?

- Se trata de una **secuencia de datos**:
  - Que se recibe en **tiempo real**.
  - Que se recibe de **forma continua**.
  - Que se recibe de **forma ordenada** (bien por timestamp o por llegada).
  - Que **NO puede ser almacenada** de forma local.

- Sobre la que se reciben **consultas**:
  - Que se ejecutan en **tiempo real**.
  - Que se ejecutan de **forma continua**.
  - Que devuelven **resultados incrementales** a medida que se procesan datos.

- La **velocidad de los datos** conlleva nuevos retos para los que tecnologías como Hadoop o incluso Spark (Core y/o SQL) no están preparados.

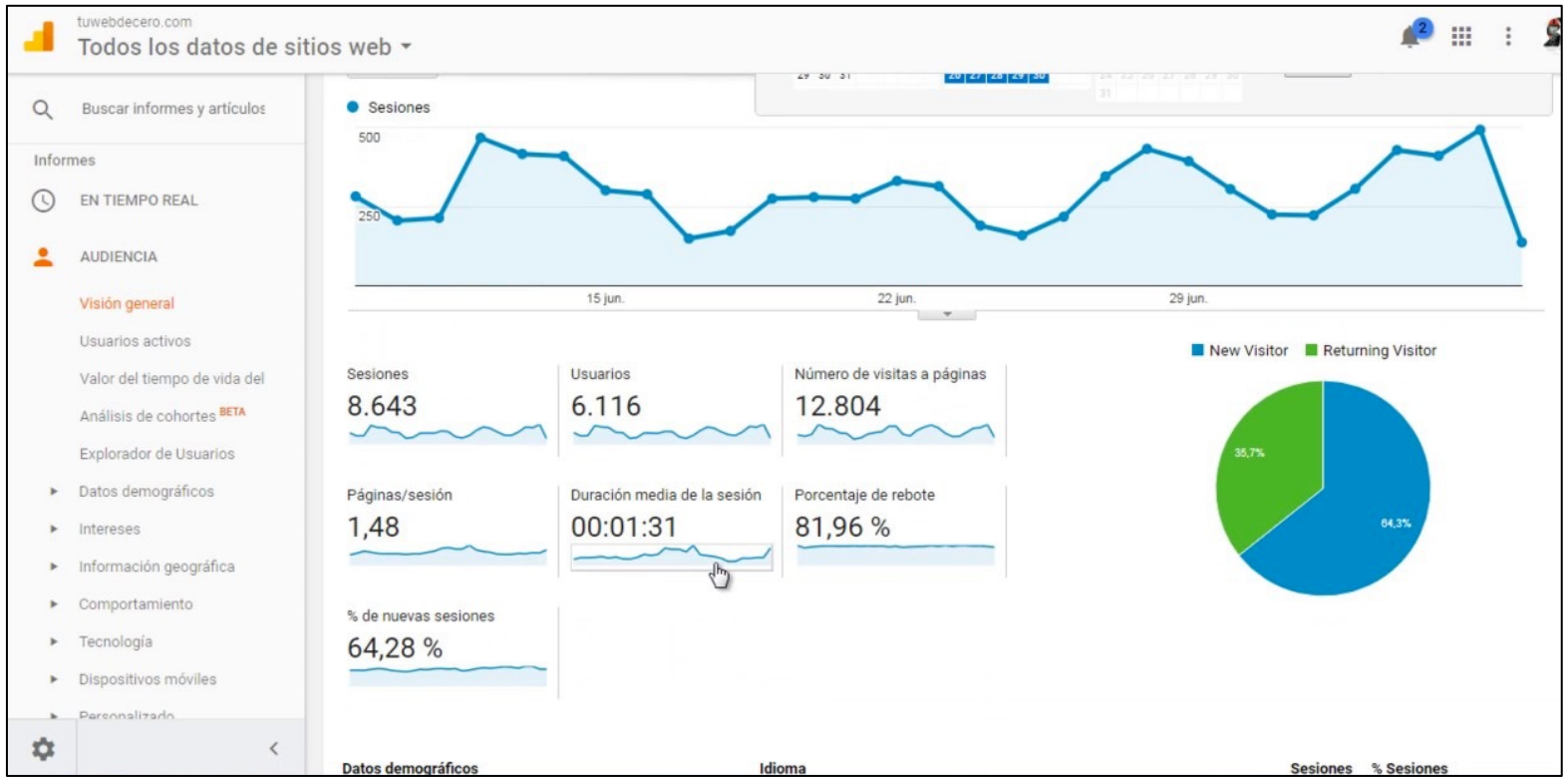
Batch vs. Streaming

- Los sistemas de **procesamiento batch** funcionan bajo la premisa de que:
  - TODOS los datos necesarios se han generado de forma previa.
  - Disponemos de TODOS los datos necesarios para la ejecución del proceso.
  - TODOS los datos están disponibles "para siempre".

- Cuando hablamos de **procesamiento en streaming** esta situación cambia y tenemos:
  - Datos generados "al vuelo" de forma paralela al procesado.
  - Los datos que no se procesen a tiempo, se pierden.
  - Necesitamos consultar los datos que se generan "al vuelo".

- En resumen, tenemos:
  - Datos generados "al vuelo" y a gran velocidad.
  - Que deben ser procesados "al vuelo" y a gran velocidad o se perderán.

Ejemplos de streams

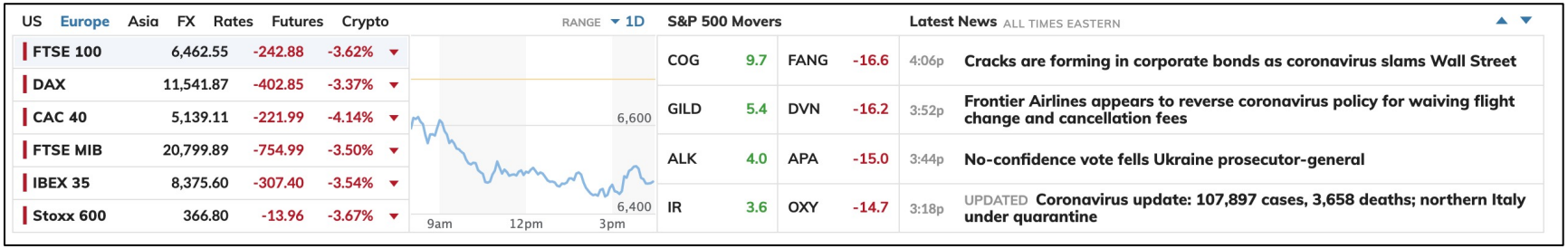


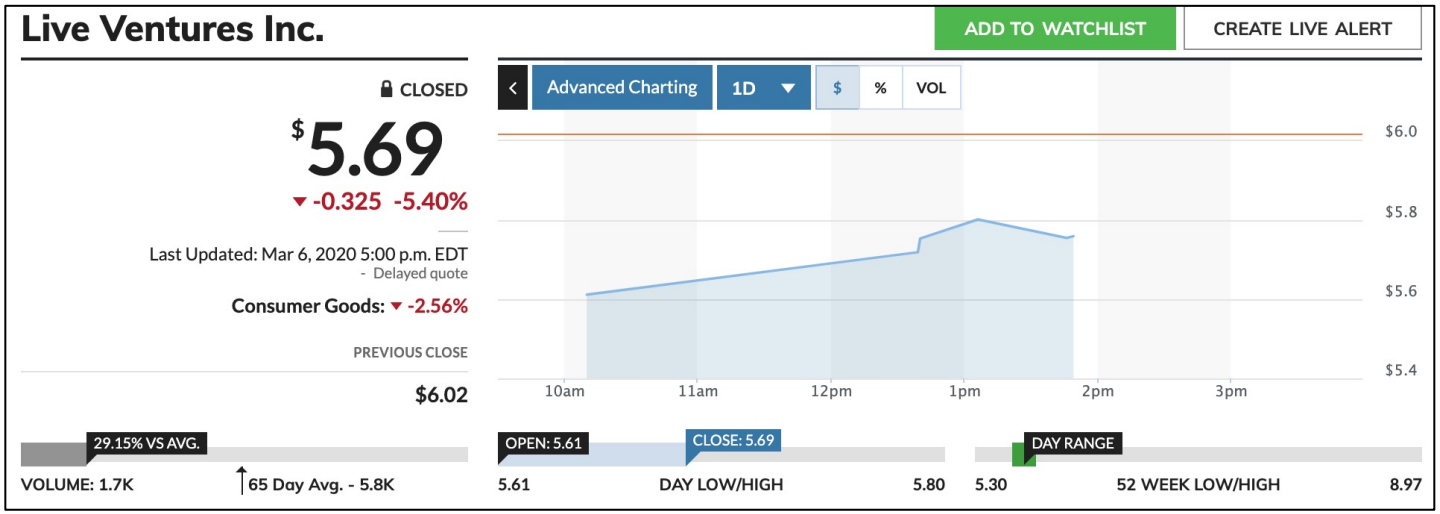
Web analytics

Ejemplos de streams

Redes sociales

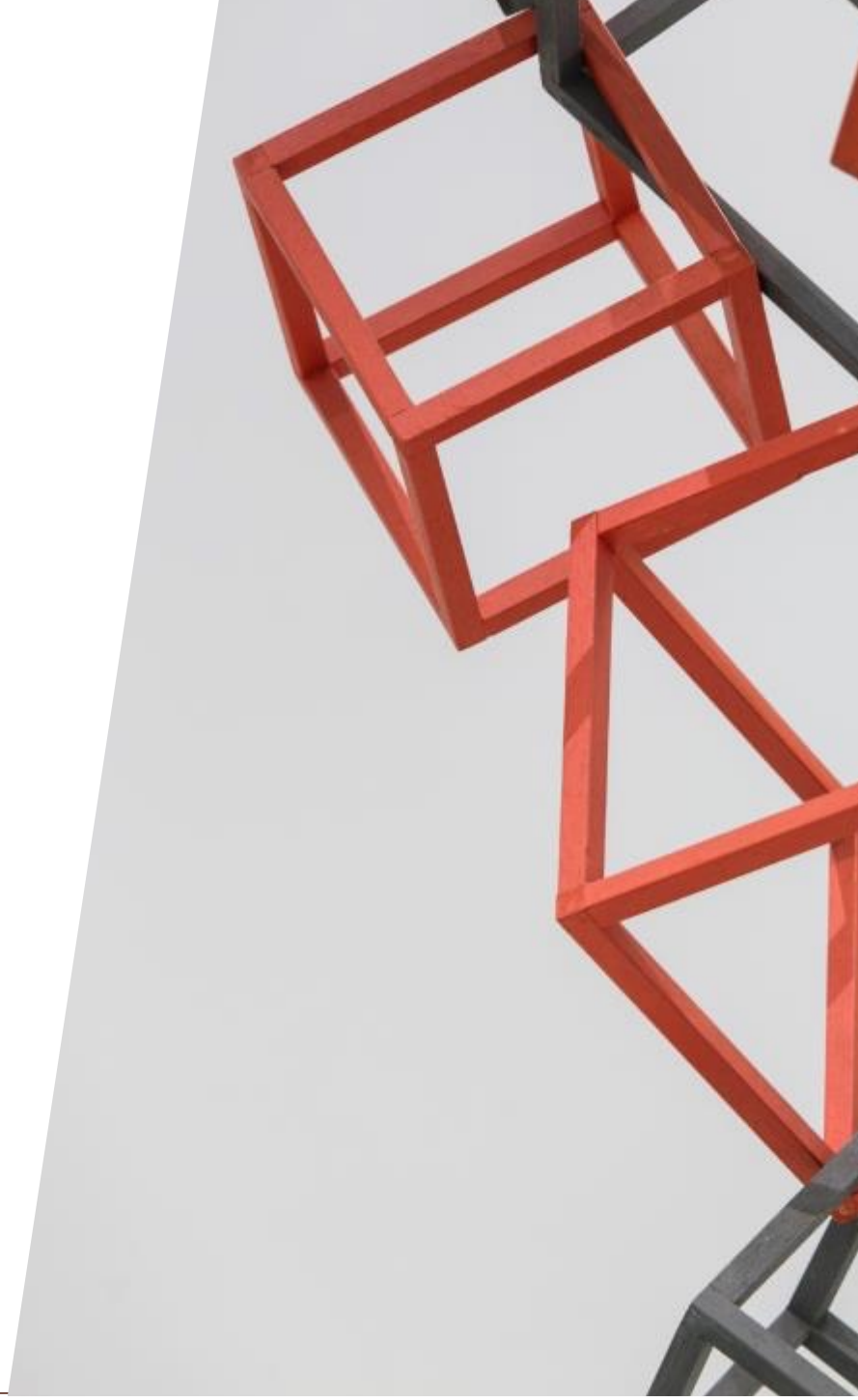
Ejemplos de streams





Bolsa

Arquitecturas de streaming



¿Qué problema intentamos resolver?

- Tenemos **flujos rápidos de datos** que tenemos que procesar a una velocidad rápida para no perder información.

- Queremos poder **consultar TODOS los datos disponibles** en el sistema en cualquier momento y en tiempo real.

- Veamos cómo lo resolveríamos con:
  - Arquitecturas tradicionales.
  - Arquitecturas distribuidas.
  - Arquitecturas de streaming.

Arquitecturas tradicionales

- En arquitecturas tradicionales, al tener un **flujo continuo de datos** tendríamos que ser capaces de ir **almacenando cada elemento recibido** mediante inserciones en, por ejemplo, una BBDD relacional.

- Al incrementar notablemente el número de inserciones individuales que se hacen contra una BBDD relacional la **probabilidad de timeout va aumentando** y podría llegar a colapsar, produciendo una pérdida de información.

- Pero podemos solucionarlo…
  - Colas de mensajes para hacer inserciones en batch.
  - Particionado "manual" de la información en múltiples servidores (sharding).

- **Problema:** estas soluciones escalan sólo hasta cierto punto.

Arquitecturas distribuidas

- Al pensar en **arquitecturas de almacenamiento distribuidas/big data** como las que nos ofrece Hadoop, el problema de inserciones continuas desaparece.

- Contamos con un sistema de almacenamiento **distribuido, con replicación y, potencialmente, escalable** hasta el infinito.

- Podremos, por tanto, **recibir y almacenar la información independientemente del flujo/velocidad** de la misma (teóricamente).

- **Problema:** si bien estas tecnologías eliminan el problema del almacenamiento, lo hacen sacrificando por completo la posibilidad de consulta (en tiempo real) ya que tienen una latencia MUY alta (incluso Spark).

Arquitecturas de streaming

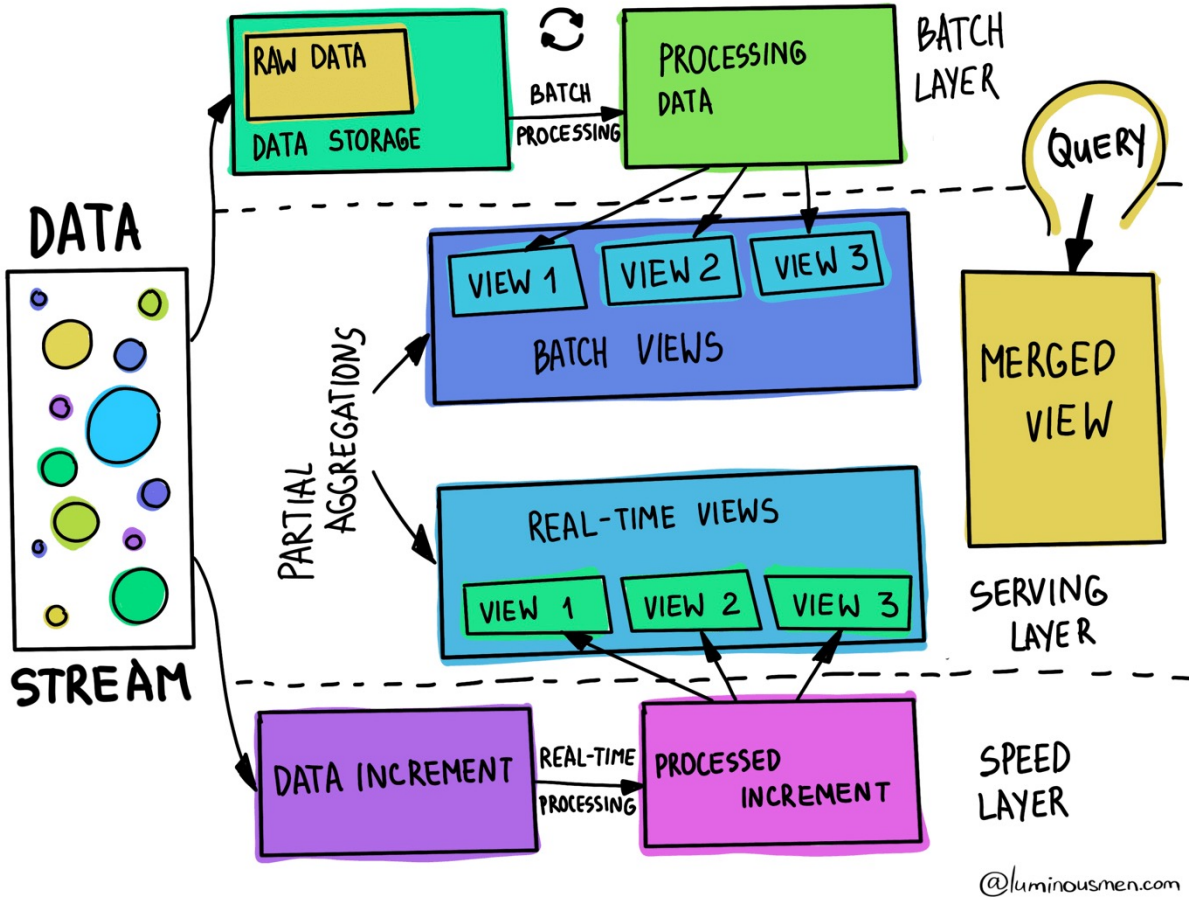
- La base de las arquitecturas de streaming para dar solución al problema planteado es el principio de: **"no existe UNA herramienta para TODO"**.

- Por tanto, se propuso la utilización de **arquitecturas híbridas** en las que se mezclan motores de **almacenamiento y procesamiento batch** con motores de **procesamiento real time/streaming**.

- Estas arquitecturas híbridas reciben el nombre de **arquitecturas lambda**.

- En cierta forma, estas arquitecturas permiten **romper el Teorema CAP** ya que:
  - Datos distribuidos → Particionado.
  - Sistema de almacenamiento y procesamiento batch → Consistencia
  - Los datos siempre están disponibles para consulta → Disponibilidad.

Arquitecturas de streaming



Arquitectura lambda

Arquitecturas de streaming

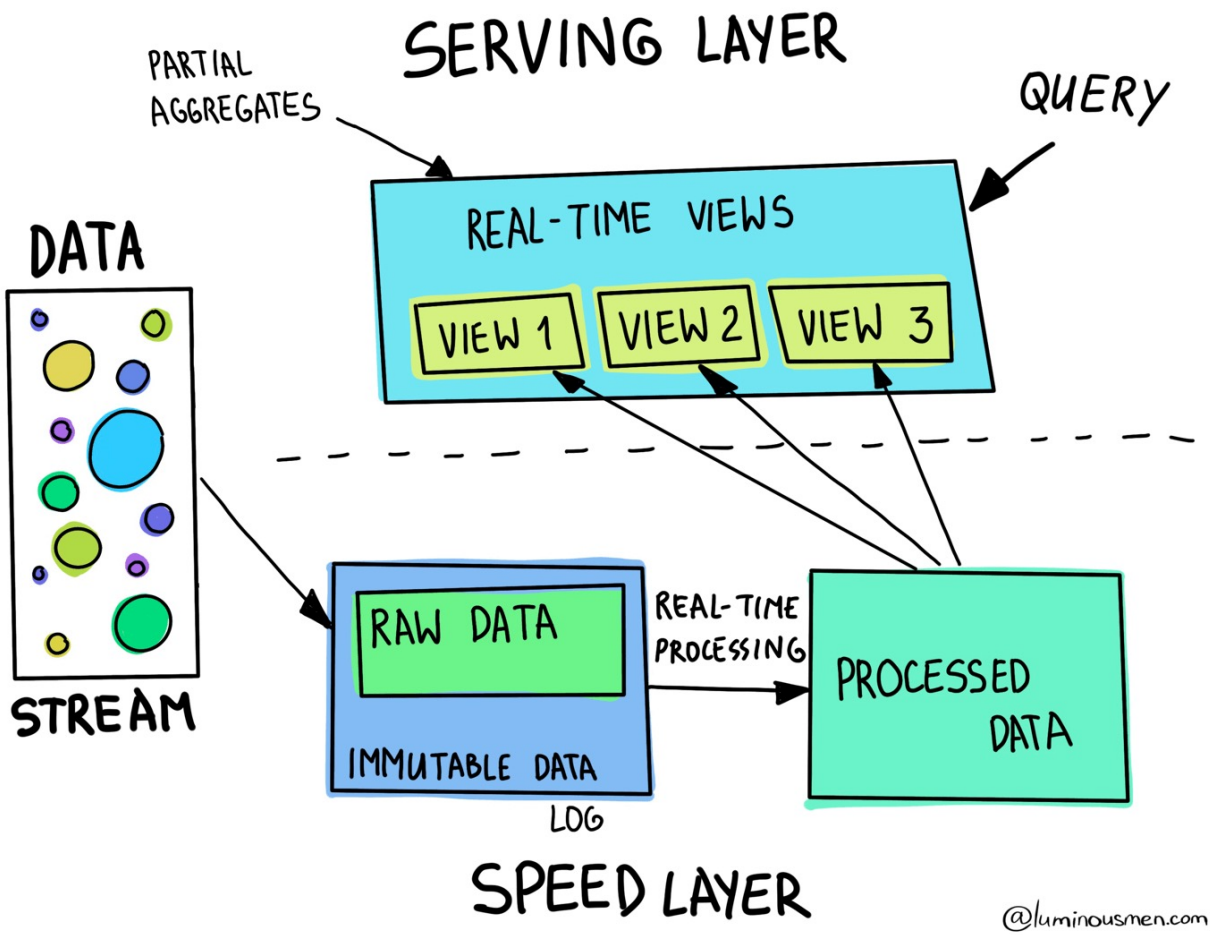
- Las **arquitecturas lambda** parecían la única solución al problema planteado, pero lógicamente, presentan **ciertas carencias**:
  - Dos sistemas complejos.
  - Dos sistemas trabajando juntos.
  - Dos procesos "idénticos" en plataformas (incluso lenguajes) distintas.

- La rápida **evolución de las tecnologías big data** unida a la, cada vez mayor, **potencia de los sistemas e infraestructuras**, han dado pie a un nuevo tipo de arquitectura.

- Esta última versión apuesta por la **eliminación de la capa batch** y el tratamiento de TODO el **flujo de datos** como **un único stream** continuo.

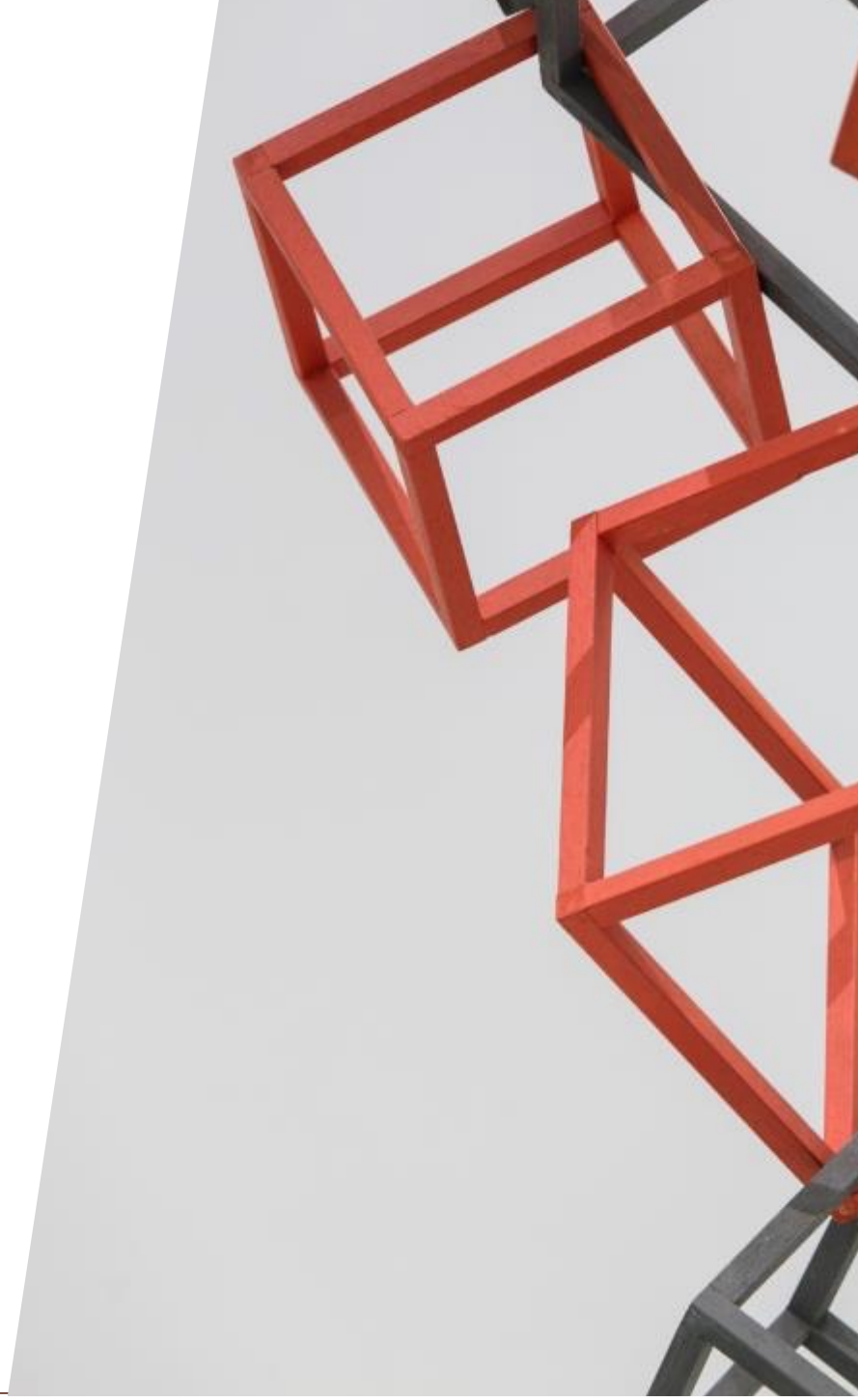
- Estas arquitecturas reciben el nombre de **arquitecturas kappa**.

Arquitecturas de streaming



Arquitectura kappa

Streaming con Spark

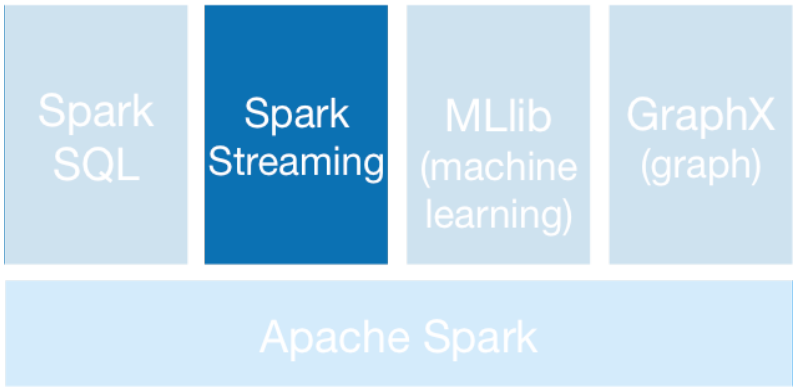


Streaming con Spark

- El objetivo principal de Spark ha sido **unificar en un único framework** de trabajo de las **funcionalidades más demandadas** dentro de las tecnologías **big data**.

- Hasta el momento, hemos visto que cubre:
  - Procesamiento básico de información → Spark Core.
  - Procesamiento y consulta de información estructurada → Spark SQL.
  - Machine learning sobre información distribuida → Spark ML.

- Como es de esperar, Spark también incluye una solución **procesamiento en streaming**.



Streaming con Spark

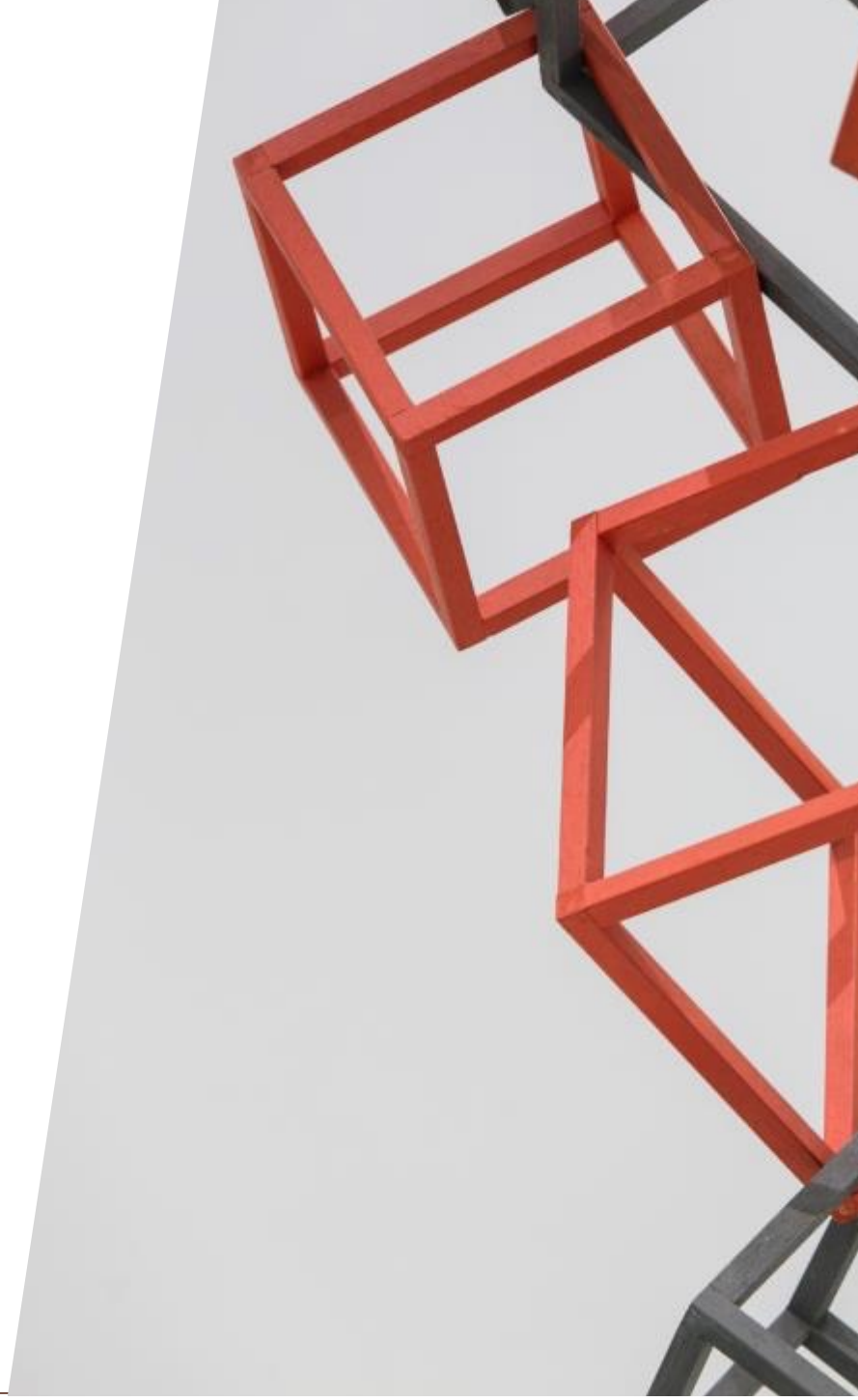
- Spark ofrece **dos paradigmas** bien diferenciados para la **representación de datos**:
  - RDDs: datos no estructurados.
  - DataFrames: datos estructurados.

- Esta duplicidad hace que dispongamos de **dos formas de trabajo** que, además, han trascendido a los otros módulos del framework.
  - Spark Core / Spark MLLib → RDDs.
  - Spark SQL / Spark ML → DataFrames.

- En el caso de **streaming**, tenemos la **misma situación**:
  - Spark Streaming → RDDs (+ Spark Core).
  - Structured Streaming → DataFrames (+ Spark SQL).

- Si bien en el caso de **MLlib vs. Spark ML**, los propios **desarrolladores** se han decantado por **una opción**, en el caso de streaming la situación no es tan clara.

Spark Streaming



¿Qué es Spark Streaming?

- Es una **extensión de Spark** que permite **procesamiento (no almacenamiento) de streams** de datos de un modo escalable, rápido y tolerante a fallos.

- Al estar implementada **sobre Spark Core**, trata los **flujos de datos como RDDs** y permite realizar "casi" todas las **transformaciones y acciones** que permitía Spark Core.

¿Cómo funciona Spark Streaming?

- **REALMENTE**, **no hace procesamiento en streaming** como tal ya que no lanza un proceso por cada elemento recibido.

- Spark Streaming lanza un **proceso periódico cada X segundos** sobre un RDD construido con la información recibida desde el proceso anterior.
  - **Facilita el desarrollo** de procesos, ya que es idéntico a un proceso batch.
  - **Dificulta la configuración** ya que hay que "ajustar" el período de proceso.
  - **Dificulta el mantenimiento/escalabilidad** ya que el período puede variar con el tiempo.

SparkContext vs. StreamingContext

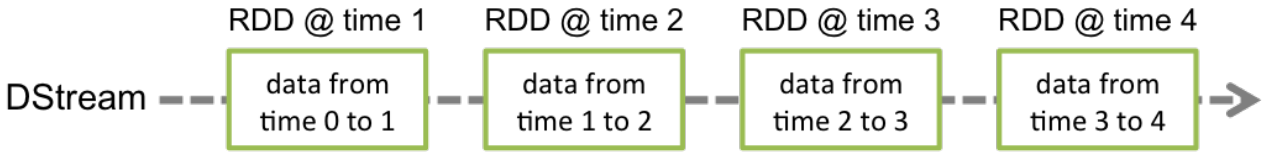
- Para poder hacer uso de **Spark Streaming**, lo primero que hay que hacer es crear un objeto de tipo **StreamingContext**.

- Spark Streaming está construido "sobre" Spark Core, por lo que un objeto de tipo **StreamingContext SIEMPRE contiene** un objeto de tipo **SparkContext.**

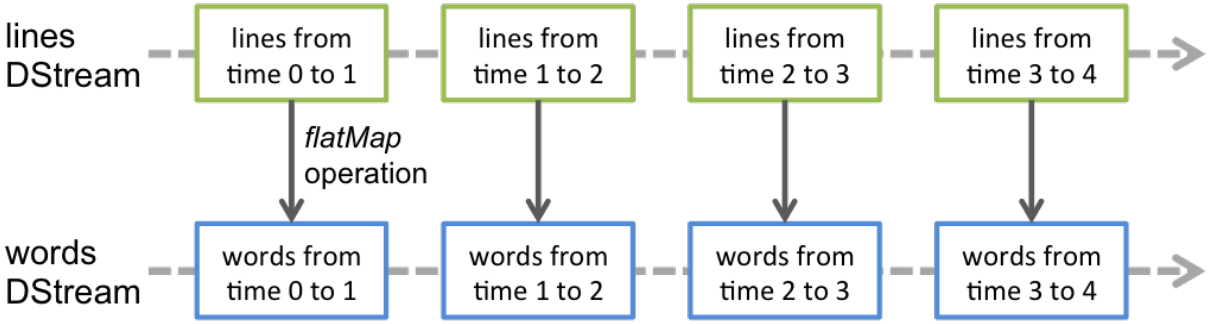
- De hecho, para llevar a cabo la **creación de un objeto StreamingContext** deberemos pasar como parámetro:
  - Un **objeto SparkContext** previamente creado: en el que se especificará la configuración de la aplicación de Spark (IP del nodo maestro, nombre de la aplicación, etc.).
  - Un **tiempo en segundos** de duración para cada batch de procesamiento.

DStreams

- Un **DStream o Discretized Stream** es una abstracción para representar un **flujo continuo de datos** que, internamente, se traduce a una **serie ordenada de RDDs**.



- **Cualquier operación** (transformación o acción) aplicada **sobre un DStream**, se traducirá a **operaciones equivalentes** aplicadas a los **RDDs subyacentes**.



SparkContext vs. StreamingContext

1. Abrir un **contexto de Spark** y un **contexto de Streaming**.
2. **Definir los inputs** del proceso mediante la creación de DStreams.
3. **Definir el conjunto de transformaciones y acciones** sobre cada DStream.
4. **Marcar el inicio** de la recepción de datos.
5. **Esperar** hasta que haya un error o se finalice manualmente.
6. **Cerrar el contexto** de Streaming (que por defecto, cierra el de Spark).

Inputs de Spark Streaming

- Spark Streaming pone a nuestra disposición un **conjunto amplio de posibles fuentes de datos** para los DStreams.
  - Directorios compatibles con HDFS API (HDFS, AWS S3, NFS…).
  - Sockets TCP.
  - Apache Flume.
  - Apache Kafka.
  - …
- Además, nos ofrece **utilidades para escribir receptores propios** (de una forma bastante sencilla).
- Incluso nos permite (principalmente, con fines de testing) la **definición de inputs** a partir de **series de RDDs** previamente creadas.

Inputs de Spark Streaming

- Spark Streaming no tiene un objetivo de consulta directa de resultados (ya que no se sabe, realmente, cuándo llegarán los datos).

- Es por ello, que la salida del proceso siempre será (salvo en tiempo de desarrollo/test) el almacenamiento de datos en un sistema externo.

- Spark Streaming pone a nuestra disposición un **conjunto amplio de posibles fuentes de datos** para los DStreams.
  - Directorios compatibles con HDFS API (HDFS, AWS S3, NFS…)
  - Bases de datos NoSQL (Cassandra, HBase…)
  - Consola (sólo para debug/test).
  - …

Transformaciones

- Un DStream funciona de forma idéntica a un RDD, por lo que las transformaciones son prácticamente equivalentes.

Transformaciones mediante aplicación de una función de usuario

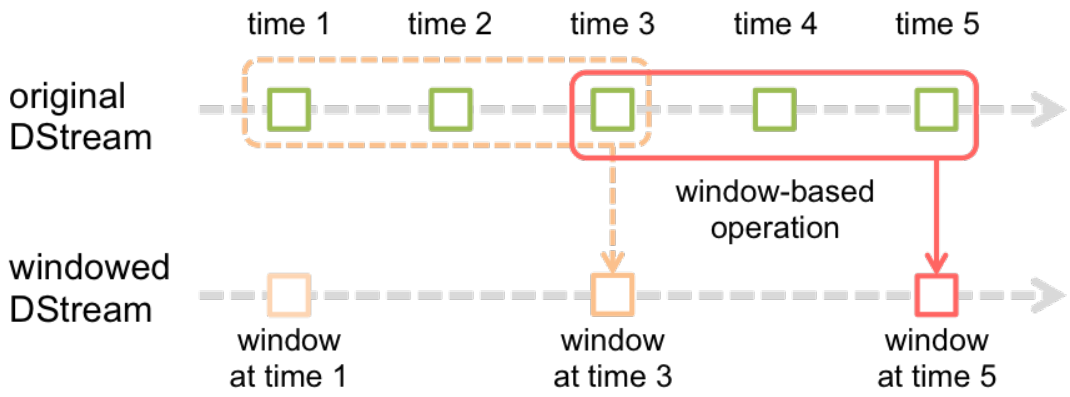
Instrucción	Descripción
ds.map(func)	Aplica "func" a cada elemento del DStream (un resultado por elemento)
ds.flatMap(func)	Aplica "func" a cada elemento del DStream y "aplana" el resultado
ds.filter(func)	Filtra aquellos elementos del DStream para los que "func" devuelve False
ds.reduce(func)	Agrega hasta que sólo queda un elemento por cada RDD del DStream
ds.reduceByKey(func)	Agrega hasta que sólo queda un elemento por cada PairRDD del DStream

Transformaciones aplicadas sobre parejas de DStreams

Instrucción	Descripción
ds.union(otherDs)	Devuelve la unión de dos DStreams
ds.join(otherDs)	Devuelve la unión de los PairRDD de cada uno de los DStreams

Transformaciones

- Dado el **carácter periódico y ordenado de los DStreams**, aparecen en este caso un nuevo tipo de funciones: **funciones de ventana móvil**.



- Estas funciones necesitarán especificar:
  - **Tamaño de ventana**: número de batchs (RDDs) a usar en la agregación.
  - **Intervalo de ventana**: cada cuantos batchs (RDDs) lanzar la agregación.

Transformaciones

Transformaciones aplicadas sobre ventanas móviles

Instrucción	Descripción
ds.window(size, interval)	Devuelve un Dstream creados a partir de las ventanas
ds.reduceByWindow(func, size, interval)	Agrega los RDDs creados a partir de las ventanas
ds.reduceByKeyAndWindow(func, size, interval)	Agrega los PairRDDs creados a partir de las ventanas

Acciones

- Dado que el resultado de un proceso de Streaming no está pensado para su consumo "al vuelo", el conjunto de acciones disponibles es más reducido que en Spark Core.

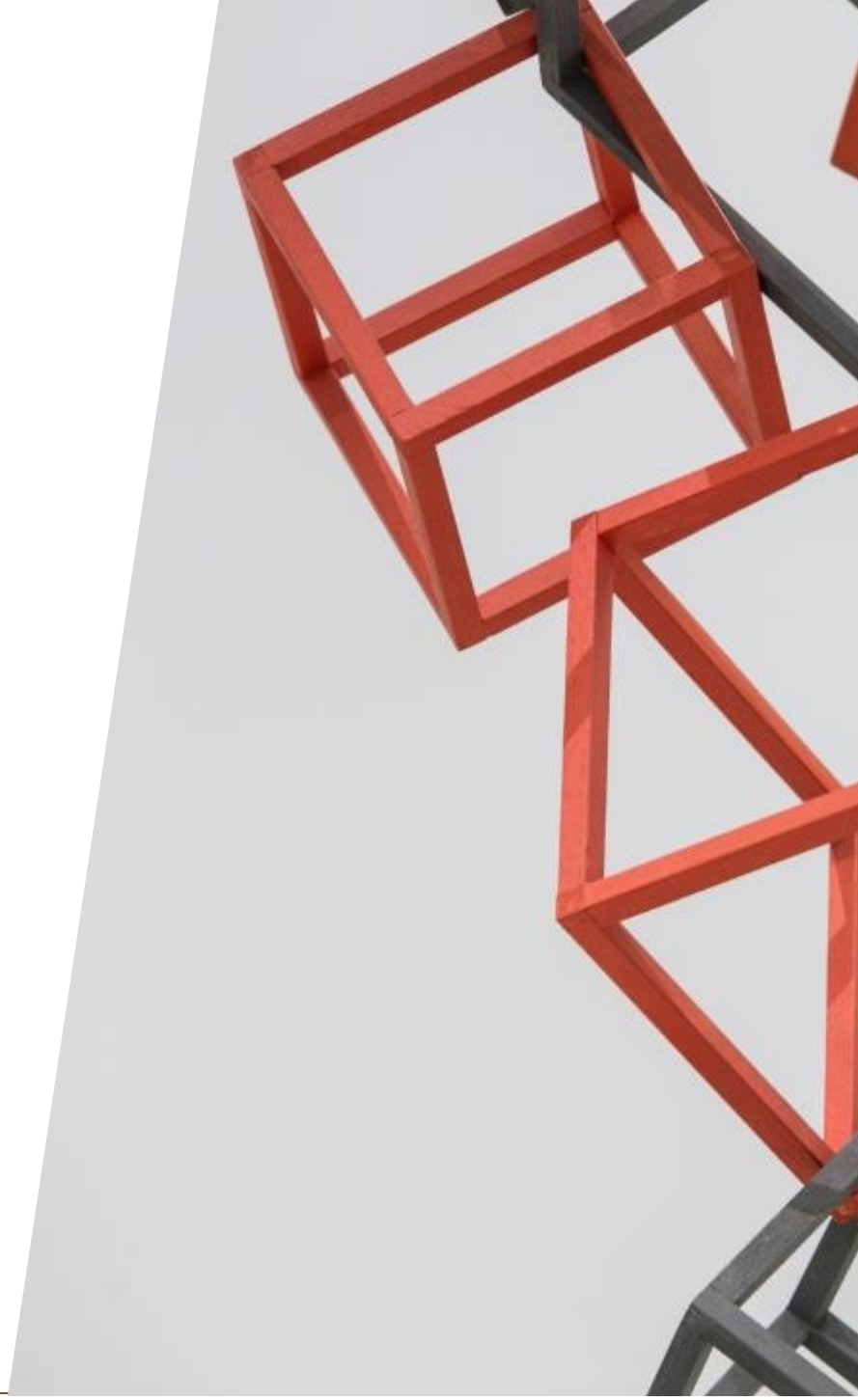
Acciones de almacenamiento/consulta de salida

Instrucción	Descripción
ds.pprint()	Escribe el resultado de cada batch al terminal del nodo maestro (debug/test)
ds.saveAsTextFiles(preffix)	Guarda el resultado en ficheros de nombre preffix-<time_in_ms>.

Juguemos un poco...

spark_streaming.py

Structured Streaming



¿Qué es Structured Streaming?

- Es una **extensión de Spark** que permite **procesamiento (no almacenamiento) de streams** de datos de un modo escalable, rápido y tolerante a fallos.

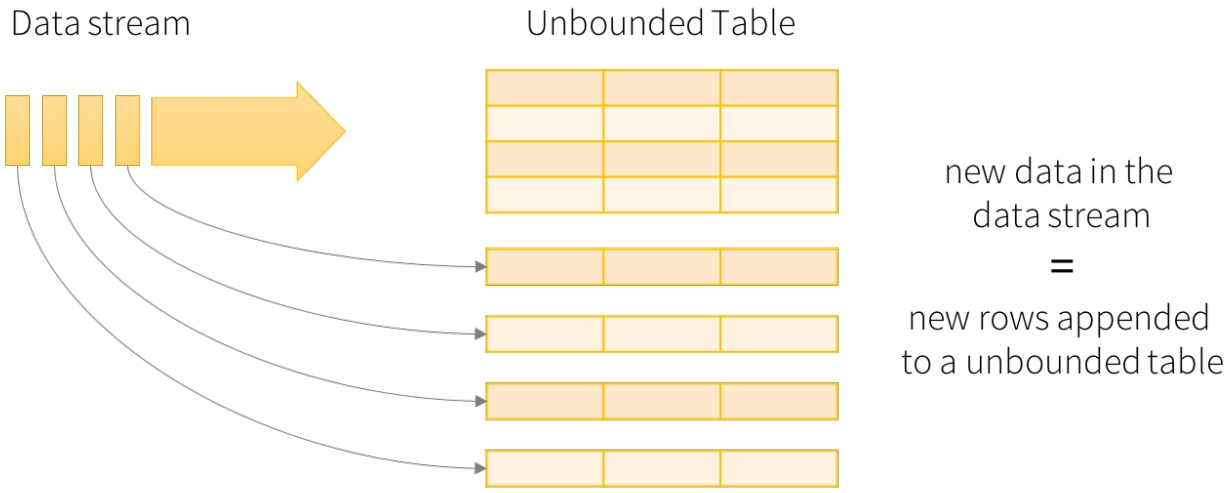
- La principal diferencia con Spark Streaming es que en lugar de estar implementada sobre Spark Core, lo está sobre Spark SQL.

- Esto nos ofrecerá algunas diferencias importantes sobre Spark Streaming:
  - La estructura de datos interna con la que trabajaremos no será RDD sino DataFrame.
  - Definiremos el proceso EXACTAMENTE igual que lo haríamos en Spark SQL.
  - El motor de Spark SQL se encargará de procesar los bloques de información y añadirlos al DataFrame de forma incremental y continua (es decir, la misma query dará resultados distintos cada vez).

- En definitiva, trabajar con **Structured Streaming** es como trabajar con **Spark SQL** sobre una **tabla que crece continuamente**.

¿Cómo funciona Structured Streaming?

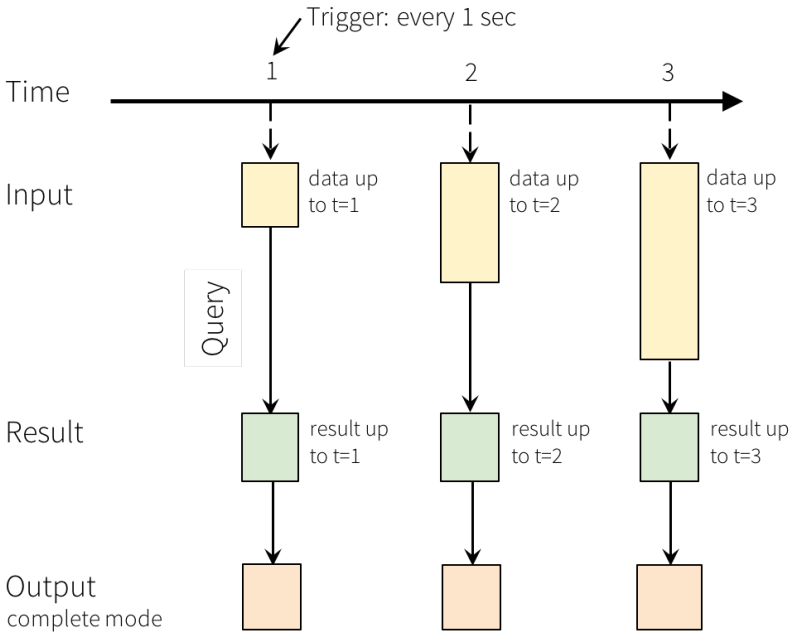
- Structured Streaming trata **cada elemento de nuestro stream** de datos como un **nuevo registro que añadir a una tabla**.



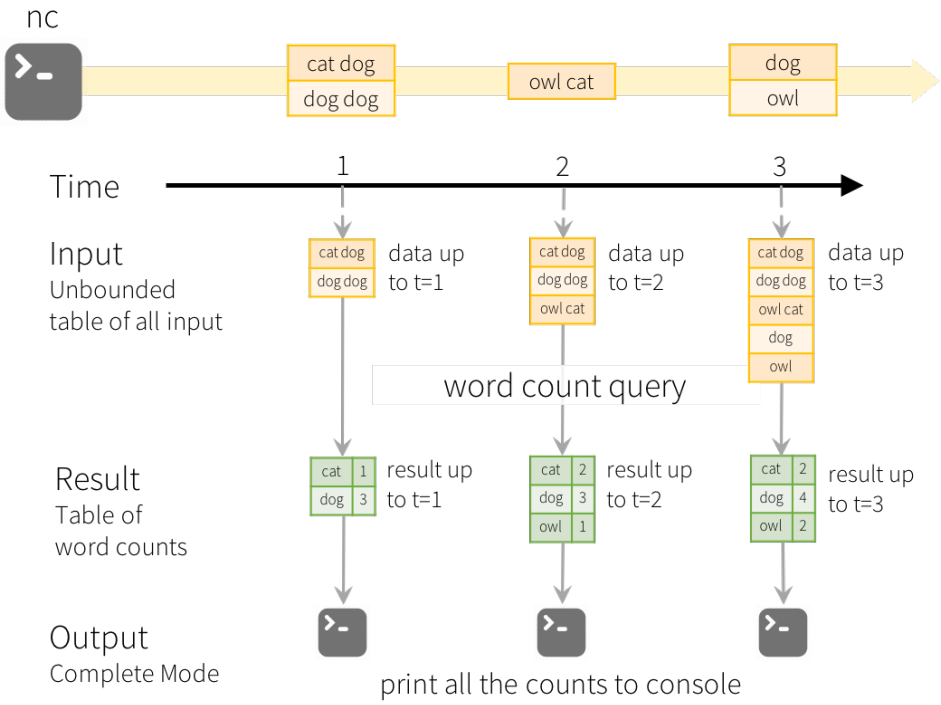
Data stream as an unbounded table

¿Cómo funciona Structured Streaming?

- La **query** que se configure **sobre el stream de datos**, irá recuperando registros de una **tabla cada vez más grande**.



Programming Model for Structured Streaming



Model of the Quick Example

SparkSession

- Para poder hacer uso de Structured Streaming, **no será necesaria la creación de un nuevo objeto de contexto** como ocurría con Spark Streaming..
- Structured Streaming está implementado sobre Spark SQL y persigue **abstraer al desarrollador de las diferencias** existentes entre trabajar en **modo batch y en modo streaming**.
- Por ello, **sólo será necesaria** la creación de una **sesión de Spark (SparkSession)** igual que ocurría en Spark SQL.
- **Lo único que variará** a la hora de trabajar en streaming o en batch, será el modo en el que **definimos los inputs y los outputs** de nuestro proceso.

SparkContext vs. StreamingContext

1. Abrir una **sesión de Spark**.
2. **Definir los inputs** del proceso mediante la lectura de streams.
3. **Definir el conjunto de consultas** y salida de las mismas.
4. **Marcar el inicio** de la recepción de datos de cada consulta.
5. **Esperar** hasta que haya un error o se finalice manualmente.
6. **Cerrar la sesión** de Spark.

Inputs de Structured Streaming

- Structured Streaming pone a nuestra disposición un **conjunto amplio de posibles fuentes de datos** para los DStreams.
  - Directorios compatibles con HDFS API (HDFS, AWS S3, NFS…).
  - Sockets TCP.
  - Apache Kafka.
  - …

Outputs de Structured Streaming

- Structured Streaming no tiene un objetivo de consulta directa de resultados (ya que no se sabe, realmente, cuándo llegarán los datos).

- Es por ello, que la salida del proceso siempre será (salvo en tiempo de desarrollo/test) el almacenamiento de datos en un sistema externo.

- Structured Streaming pone a nuestra disposición un **conjunto amplio de posibles fuentes de datos** para los DStreams.
  - Directorios compatibles con HDFS API (HDFS, AWS S3, NFS…)
  - Kafka
  - Consola/Memoria (sólo para debug/test)
  - …

Outputs de Structured Streaming

- Si bien hemos comentado que la salida de las consultas realizadas sobre streams siempre serán resultados sobre tablas incrementales, es importante saber que este comportamiento es configurable.

- Structured Streaming nos permite "controlar" hasta cierto punto qué resultados se envían a la salida de un proceso mediante tres modos de salida distintos:
  - **Complete mode:** la tabla generada a la salida de cada consulta es siempre completa (incremental).
  - **Append mode:** sólo los nuevos registros son enviados a la salida (esto sólo funciona en queries que no puedan modificar los resultados de filas existentes).
  - **Update mode:** sólo los registros nuevos y/o que hayan cambiado se envían a la salida.

- Aunque generalmente, trabajaremos con el "complete mode" por defecto.

Operaciones sobre streams

- Dado que en Structured Streaming **no existe una diferencia** clara entre **streams y DataFrames**, las **operaciones** que se podrán realizar son (prácticamente) las mismas que las permitidas en **Spark SQL**.

- Lo único que habrá que **tener en cuenta** es que las operaciones se aplicarán sobre una **tabla incremental**.

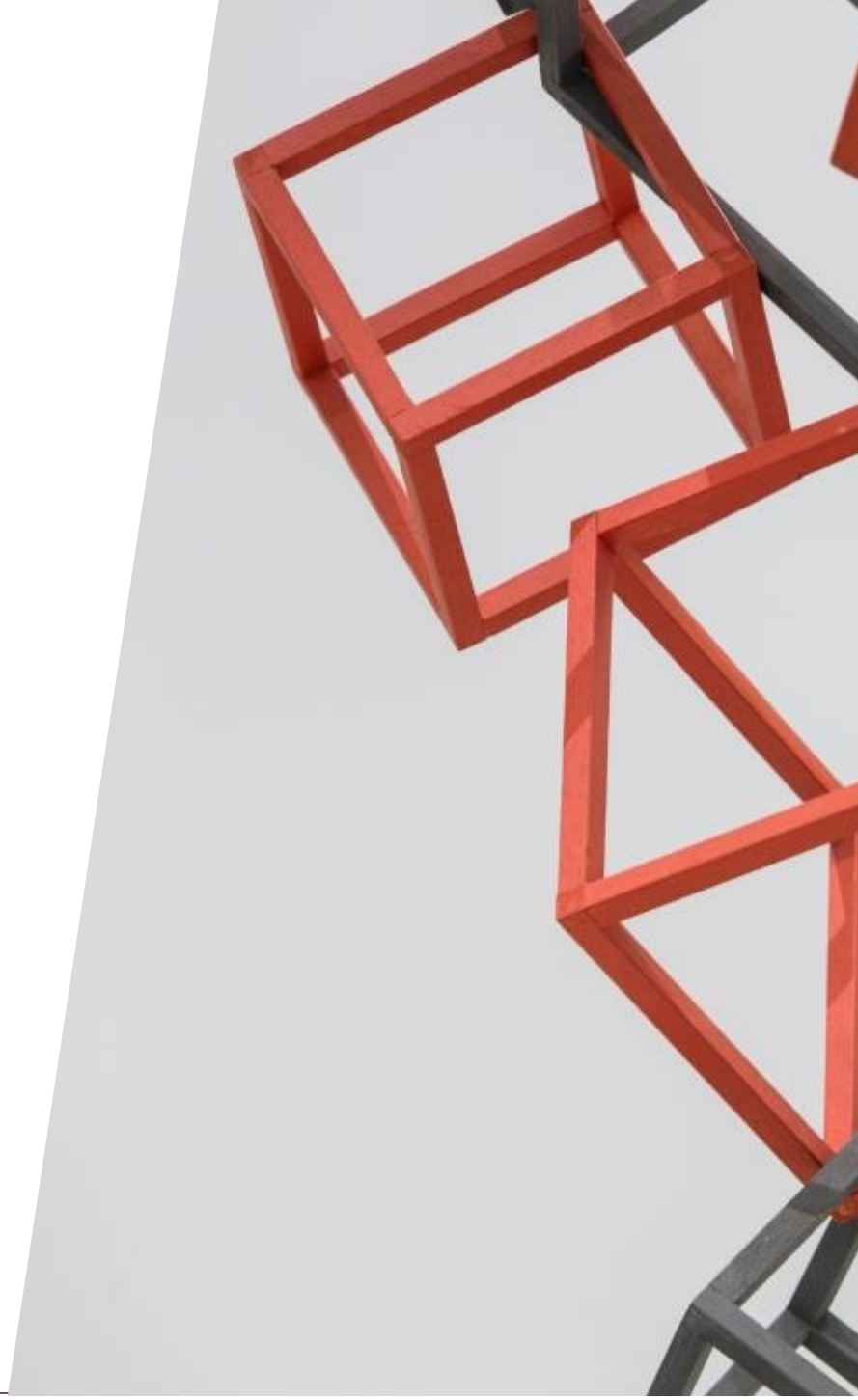
- Las únicas **operaciones que no se podrán realizar** son:
  - Límite sobre número de resultados.
  - Eliminación de duplicados.
  - Ordenaciones.
  - …

- En definitiva, todas aquellas **operaciones cuyos resultados dependan** de tener el **set de datos completo** disponible.

Juguemos un poco...

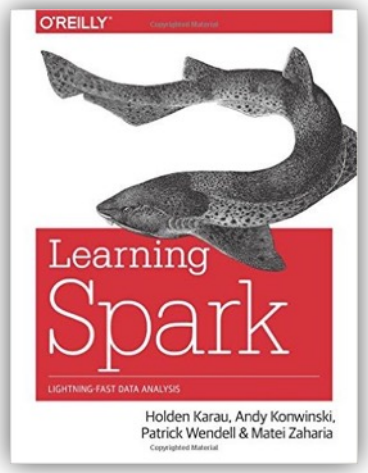
structured_streaming.py

Referencias



Referencias

- Documentación oficial:
  - https://spark.apache.org
- Tutoriales online:
  - https://www.tutorialspoint.com/apache_spark/
- Libros:



Afi Escuela