

Procesamiento en Streaming con Spark

El procesamiento en streaming se refiere al manejo en tiempo real de datos que llegan de manera continua, a diferencia del procesamiento por lotes que trabaja con bloques de datos en intervalos definidos. Esta tecnología permite realizar análisis instantáneos sobre flujos constantes de información.

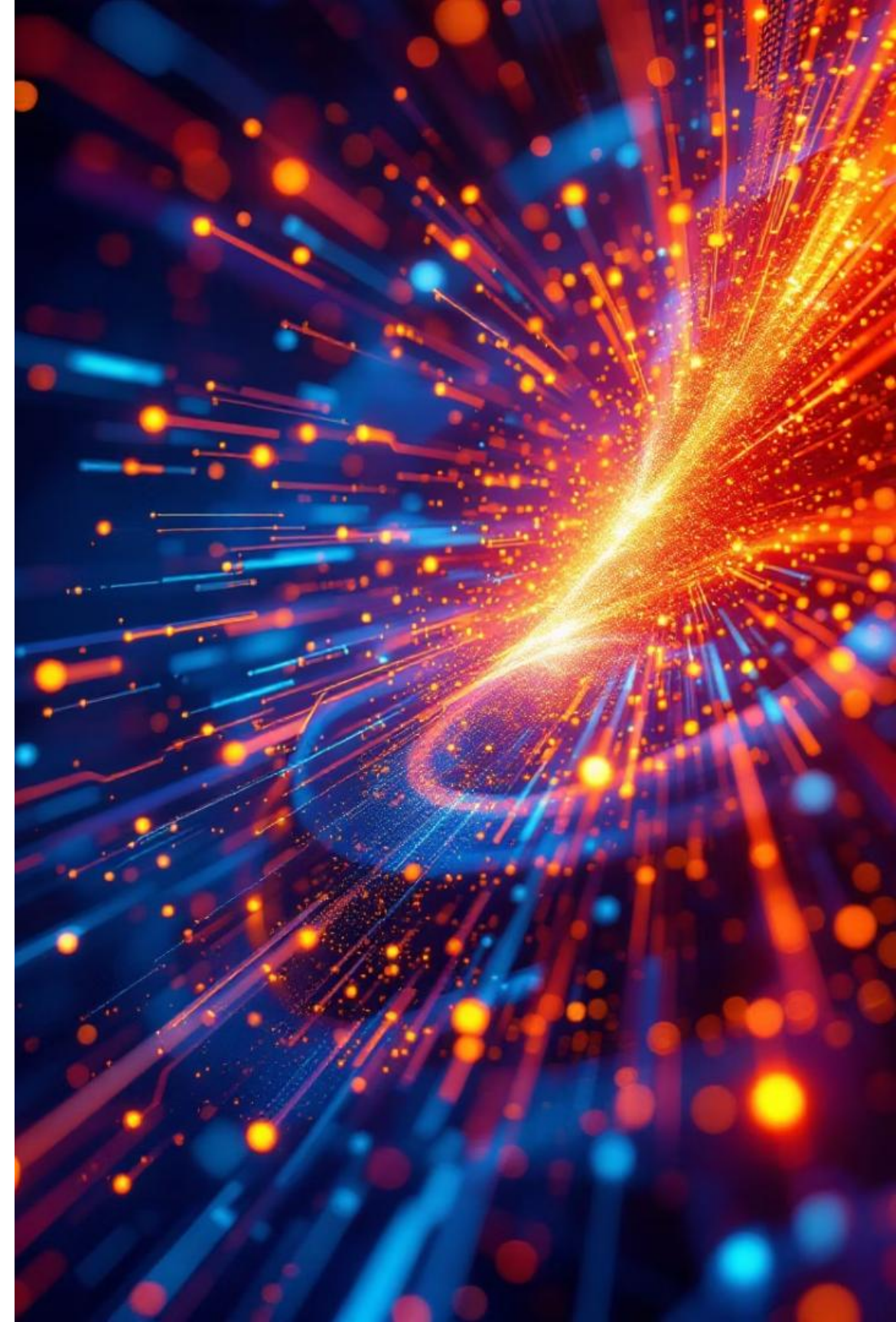
Apache Spark Streaming es una extensión de Spark diseñada específicamente para este tipo de procesamiento, ofreciendo capacidades avanzadas para manejar grandes volúmenes de datos en tiempo real con baja latencia y alta tolerancia a fallos.

 **por Kibernetum Capacitación S.A.**



Preguntas de Activación de Contenidos

- 1) ¿Qué es Spark SQL y cómo se integra con Apache Spark?
- 2) ¿Cuál es la diferencia entre un **DataFrame** y un **Dataset** en Spark?
- 3) ¿Qué es el Catalyst Optimizer y cómo contribuye al rendimiento de las consultas en Spark SQL?





Características del Procesamiento en Streaming



Escalabilidad y Tolerancia a Fallos

Spark Streaming distribuye el procesamiento de micro-lotes a través de un clúster de nodos, permitiendo procesar grandes volúmenes de datos en paralelo con alta resistencia a fallos.



Interoperabilidad

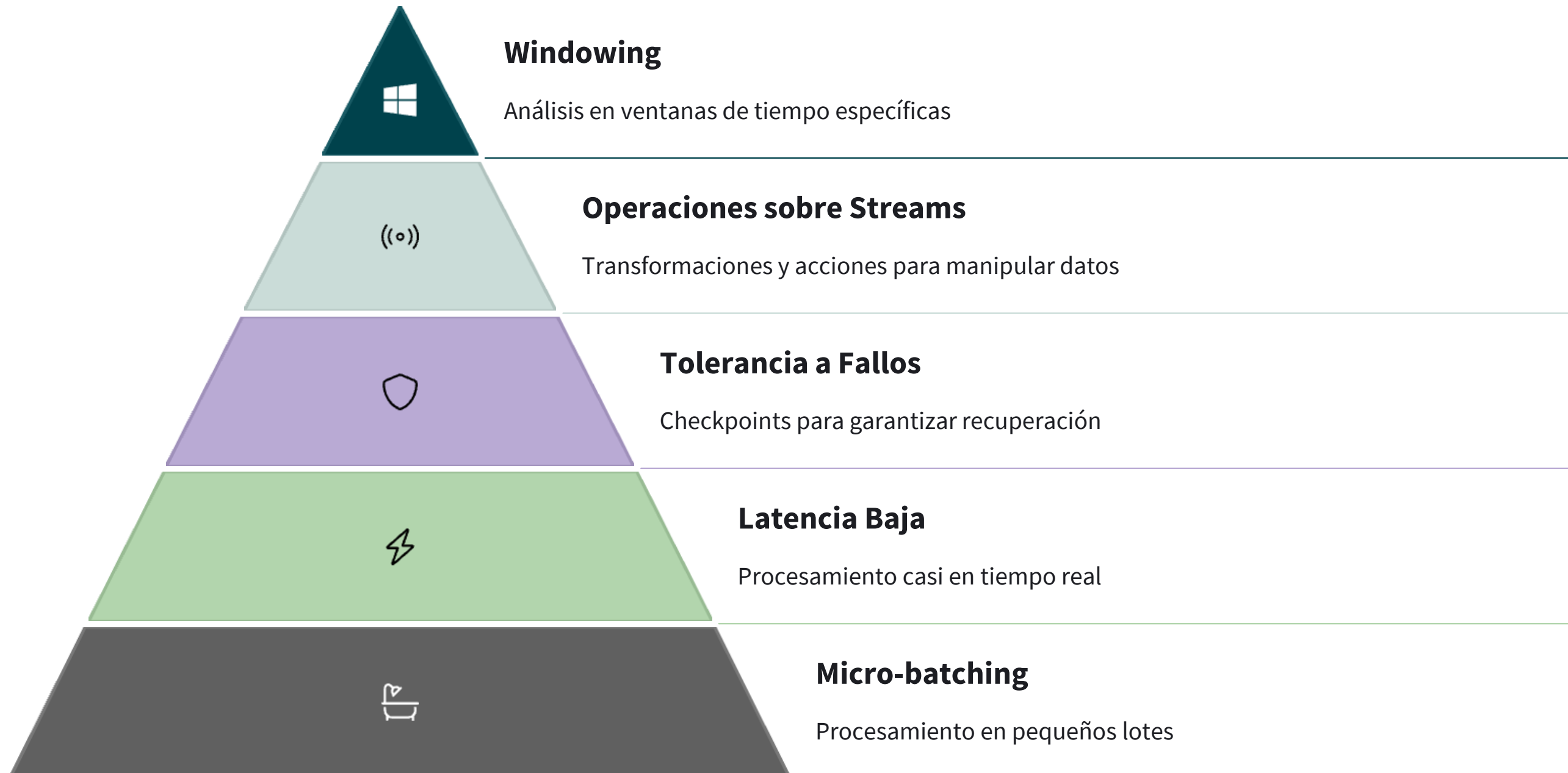
Compatible con diversas fuentes de datos en tiempo real como Kafka, Flume, Kinesis, HDFS y múltiples bases de datos.



Baja Latencia

Aunque utiliza micro-batches, la latencia es muy baja y ajustable según necesidades, permitiendo configurar el tamaño de cada micro-lote para equilibrar latencia y rendimiento.

Principios Básicos del Procesamiento de Streaming



Microbatching: Clave en Spark Streaming

División del Flujo

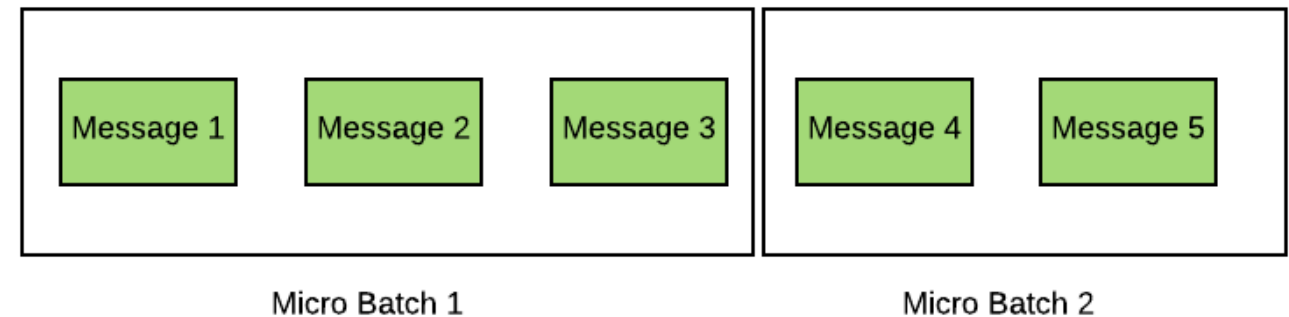
El flujo continuo de datos se divide en intervalos de tiempo pequeños llamados micro-lotes.

Procesamiento Completo

Cada micro-lote se procesa en su totalidad antes de continuar con el siguiente, simulando procesamiento en tiempo real.

Eficiencia Combinada

Esta técnica aprovecha la eficiencia del procesamiento por lotes mientras mantiene la apariencia de procesamiento en tiempo real.



DStreams: Fundamento de Spark Streaming

Definición

DStream (Discretized Stream) es una abstracción fundamental que representa un flujo de datos en tiempo real, funcionando como una secuencia de RDDs (Resilient Distributed Datasets) que se procesan a medida que llegan.

Casos de Uso

- Monitoreo de redes sociales
- Detección de fraudes financieros
- Análisis de logs y eventos
- Procesamiento de datos IoT
- Recomendaciones en tiempo real

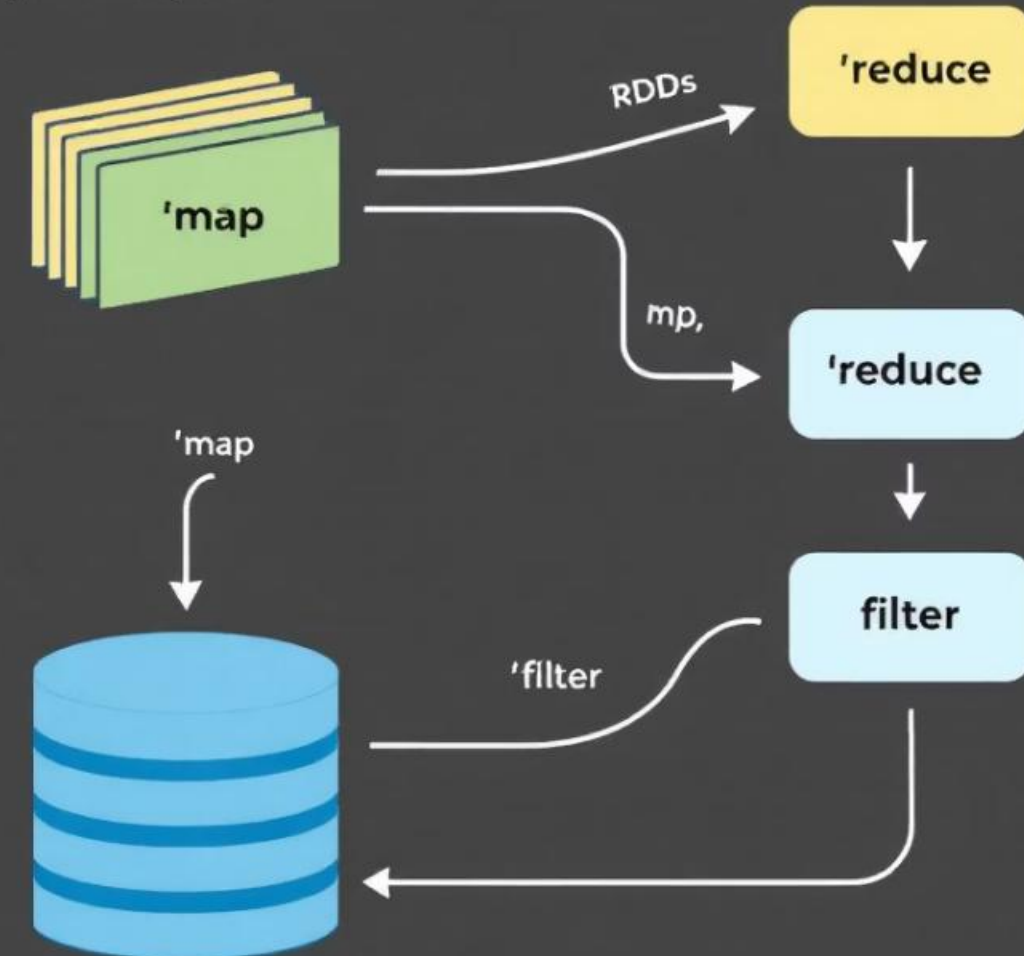
Ejemplo Práctico

En un sistema de monitoreo de logs de servidores, los datos se transmiten continuamente y Spark Streaming los procesa en tiempo real usando un DStream para contar accesos por segundo.

DStream

Is continuous stream of data into, divided it into discrete Resilient Distributed Datasets (RDDs)

DStreams processed by functions

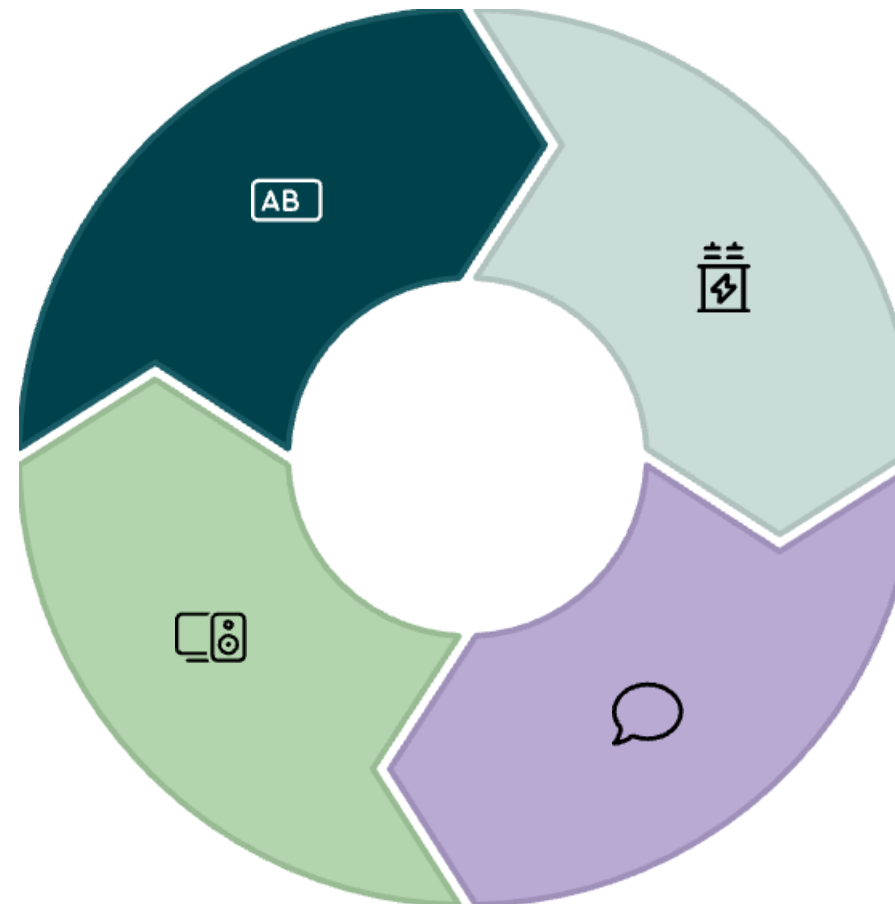


Procesamiento con Spark Streaming



Recepción de Datos

Ingesta continua desde diversas fuentes



Transformación

Aplicación de operaciones sobre DStreams

Análisis

Procesamiento en tiempo real de los datos

Salida
Almacenamiento o visualización de resultados

Uso de DStreams

En **Spark Streaming**, los **DStreams** (Discretized Streams) son la abstracción principal para representar flujos de datos en tiempo real. Un **DStream** es una secuencia de RDDs (Resilient Distributed Datasets) que se generan en intervalos de tiempo específicos (micro-batches). Los DStreams permiten realizar transformaciones y acciones sobre datos en tiempo real de una manera similar a cómo se hace con RDDs en el procesamiento por lotes.



```
from pyspark import SparkContext
from pyspark.streaming import StreamingContext

# Crear un contexto de Spark y un StreamingContext con 5 segundos de intervalo de micro-lote
sc = SparkContext(appName="DStreamExample")
ssc = StreamingContext(sc, 5)

# Crear un DStream a partir de un flujo de datos de un socket
stream_data = ssc.socketTextStream("localhost", 9999)

# Aquí el DStream 'stream_data' representará un flujo de datos recibido de un puerto 9999
|
```


Integración con Fuentes de Datos de Streaming

Una de las características más poderosas de Spark Streaming es su capacidad para integrarse con diversas **fuentes de datos de streaming**. Esto permite a las aplicaciones procesar flujos de datos en tiempo real desde una variedad de orígenes, como **Kafka**, **Flume**, **Sockets**, **HDFS**, y **Kinesis**. A través de estas integraciones, Spark Streaming puede recibir datos de manera continua, procesarlos y generar resultados en tiempo real.

Integraciones disponibles:

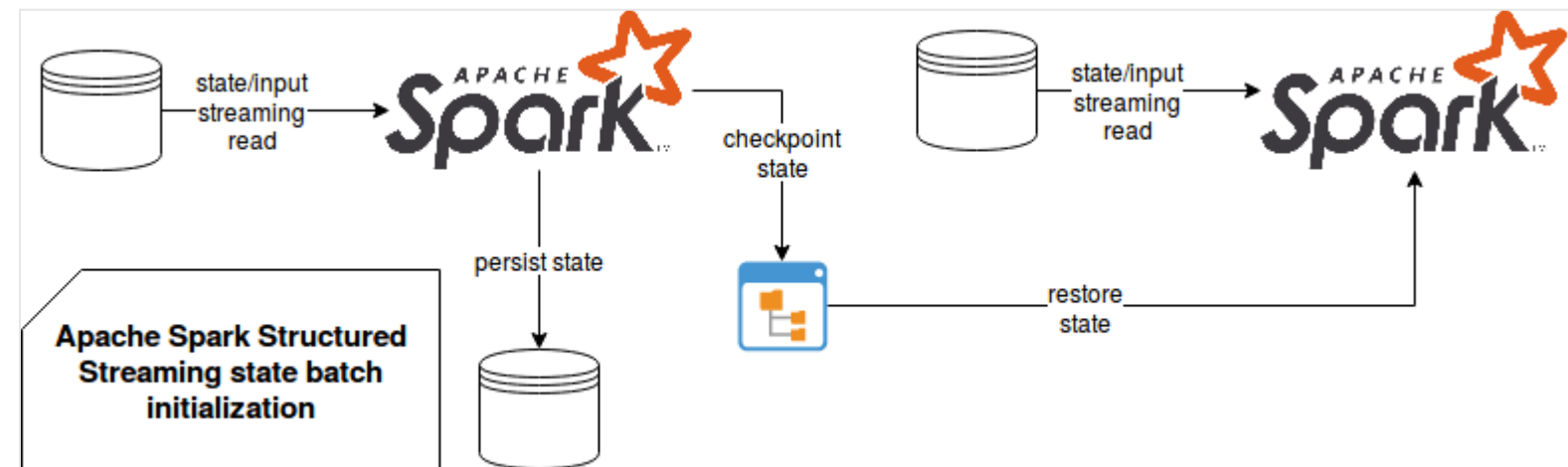
- Apache Kafka
- Apache Flume
- Amazon Kinesis
- Sockets
- HDFS (Hadoop Distributed File System)
- Cassandra
- RabbitMQ

Spark Structured Streaming

Spark Structured Streaming es una extensión de Apache Spark que facilita el procesamiento de datos en tiempo real utilizando un modelo de programación basado en **Datasets** y **DataFrames**, que es más eficiente y fácil de usar que el enfoque tradicional basado en DStreams. A diferencia de Spark Streaming, que utiliza micro-lotes para procesar los datos, Structured Streaming permite un modelo de programación más declarativo, similar al procesamiento de datos en batch, pero sin sacrificar la capacidad de trabajar con datos en tiempo real.

PROCESAMIENTO EN TIEMPO REAL

El procesamiento en tiempo real con Spark Structured Streaming permite procesar flujos de datos de manera continua y casi instantánea. A diferencia de los enfoques tradicionales de procesamiento por lotes (batch processing), donde los datos se procesan después de acumularse, Structured Streaming permite realizar análisis sobre los datos mientras van llegando, con un enfoque basado en DataFrames y Datasets.

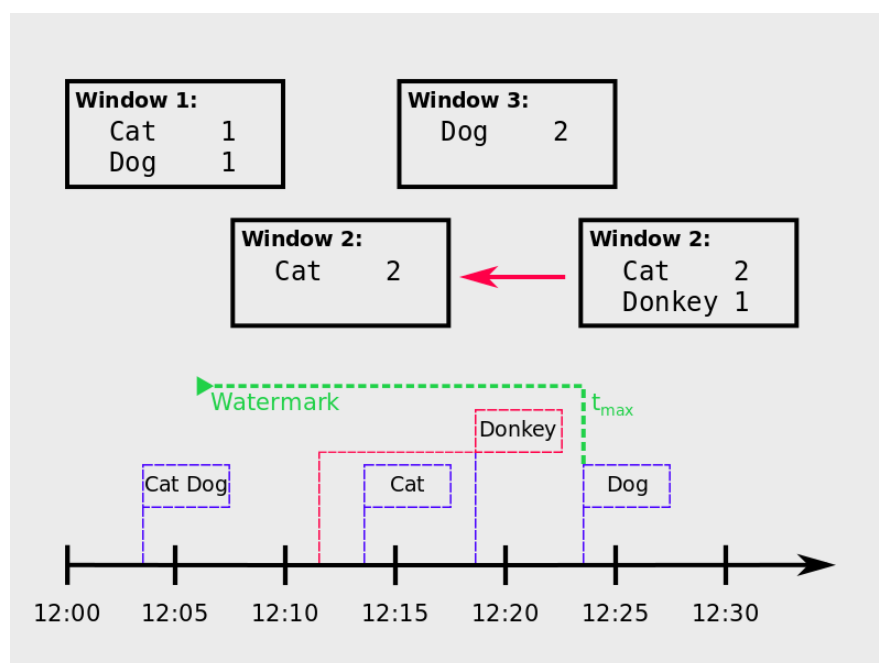


Watermark y Processing Time

Watermarking se refiere a la forma en que Spark maneja los eventos que llegan con retraso o fuera de orden, en comparación con los tiempos de evento (event time).

En flujos de datos en tiempo real, a menudo los eventos llegan fuera de orden. Esto puede suceder debido a diversos factores como latencia en la red o almacenamiento de los eventos. Si no gestionamos los eventos que llegan tarde, las agregaciones y cálculos sobre estos eventos pueden ser incorrectos.

Para abordar este problema, Structured Streaming introduce el concepto de watermarking, que le indica a Spark que permita cierta tolerancia en los datos desordenados. Esto se hace permitiendo que los eventos lleguen con un **retraso aceptable** y aun así sean procesados correctamente.



Processing Time se refiere al tiempo real en el que los datos se procesan. Es decir, el momento en que los micro-lotes se ejecutan en Spark Streaming, generalmente asociado con la ejecución de las transformaciones y acciones sobre los flujos de datos.

El Processing Time es importante porque es el marco temporal que Spark usa para organizar y ejecutar operaciones de transformación sobre los datos.

Actividad Práctica Guiada: Procesamiento de Datos en Tiempo Real con Spark Structured Streaming

En esta actividad, vamos a crear una aplicación simple de **Spark Structured Streaming** que lee datos en tiempo real desde un socket, procesa los datos y muestra los resultados en consola.

Objetivo de la actividad:

- Leer un flujo de datos desde un **socket**.
- Realizar una transformación simple para dividir los mensajes en palabras.
- Contar las palabras que aparecen en el flujo de datos.
- Mostrar los resultados en la consola.

Desarrollo:

Paso 1: Configurar el ambiente de Spark

Primero, crea un archivo de Python (por ejemplo, `structured_streaming_example.py`). En este archivo, vamos a configurar el entorno de **Spark**

```
from pyspark.sql import SparkSession


# Crear una sesión de Spark
spark = SparkSession.builder \
    .appName("StructuredStreamingExample") \
    .getOrCreate()

# Mostrar información de la sesión
spark.version
| go(f, seed, [])
}
```


Actividad Práctica Guiada: Procesamiento de Datos en Tiempo Real con Spark Structured Streaming

Paso 2: Crear un flujo de datos desde un socket

Vamos a leer datos en tiempo real desde un socket. Para eso, utilizaremos el método `socketTextStream` de **StreamingContext**.



```
from pyspark.streaming import StreamingContext

# Crear un contexto de streaming con intervalos de micro-lote de 1 segundo
ssc = StreamingContext(spark.sparkContext, 1)

# Crear un DStream para recibir datos desde un socket en localhost, puerto 9999
stream_data = ssc.socketTextStream("localhost", 9999)
```

Actividad Práctica Guiada: Procesamiento de Datos en Tiempo Real con Spark Structured Streaming

Paso 3: Transformación simple: dividir cada línea de texto en palabras:

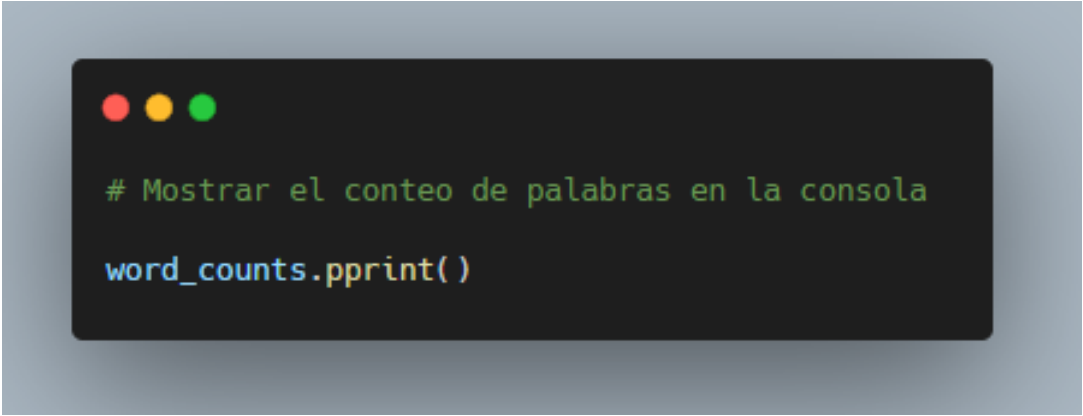
```
# Transformar el flujo de texto en palabras  
words = stream_data.flatMap(lambda line: line.split(" "))
```

Paso 4: Contar la cantidad de veces que aparece cada palabra en el flujo de datos

```
# Contar cuántas veces aparece cada palabra  
word_counts = words.map(lambda word: (word, 1)).reduceByKey(lambda x, y: x + y)
```

Actividad Práctica Guiada: Procesamiento de Datos en Tiempo Real con Spark Structured Streaming

Paso 5: Mostrar los resultados en la consola:

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. It contains two lines of code: a comment in green and a function call in blue and yellow.

```
# Mostrar el conteo de palabras en la consola  
word_counts.pprint()
```



Preguntas de Reflexión Final

- 1) ¿Qué diferencia existe entre el procesamiento por lotes y el procesamiento de streaming?
- 2) ¿Qué es un DStream en Spark Streaming y cómo se utiliza?
- 3) ¿Qué es el watermarking en Spark Structured Streaming y para qué se utiliza?