Procesamiento en Streaming con Spark

por Kibernum Capacitación S.A.

Procesamiento de Streaming

El procesamiento de streaming se refiere al procesamiento en tiempo real de datos que llegan de manera continua. A diferencia del procesamiento por lotes (batch processing), en el que los datos se recogen, almacenan y procesan en bloques o intervalos definidos, el procesamiento en streaming maneja flujos de datos que se generan de manera constante, permitiendo realizar análisis en tiempo real.

Ejemplo:

Imagina que tienes un sistema de monitoreo en tiempo real para detectar fraudes en transacciones bancarias. Cada vez que un cliente realiza una transacción, los datos (como el monto, la ubicación, el tipo de transacción, etc.) se generan y deben ser procesados inmediatamente para detectar cualquier patrón anómalo, como transacciones fuera de lugar o por montos inusuales.

Características de Spark Streaming

Apache Spark Streaming es una extensión de Apache Spark que permite el procesamiento en tiempo real de flujos de datos. Algunas de sus características más importantes son las siguientes:

- Escalabilidad y Tolerancia a Fallos, Spark Streaming es altamente escalable y tolerante a fallos. Puede distribuir el procesamiento de los micro-lotes a través de un clúster de nodos, lo que permite procesar grandes volúmenes de datos en paralelo.
- Interoperabilidad con otras fuentes de datos, Spark Streaming es compatible con una amplia variedad de fuentes de datos en tiempo real, como Kafka, Flume, Kinesis, HDFS, bases de datos, y más.
- Procesamiento en Tiempo Real con Tiempos de Latencia Bajos: Aunque Spark Streaming usa micro-batches, la latencia es muy baja y puede ser ajustada según las necesidades. El tamaño de cada micro-lote puede configurarse (por ejemplo, 1 segundo, 5 segundos, etc.), lo que permite ajustar el equilibrio entre latencia y rendimiento.

Principios Básicos del Procesamiento de Streaming

Micro-batching

Procesamiento de datos en pequeños lotes, no evento por evento.

Latencia Baja

Procesamiento casi en tiempo real con control de latencia.

Tolerancia a Fallos

Uso de checkpoints para garantizar la recuperación en caso de fallos.

Operaciones sobre Streams

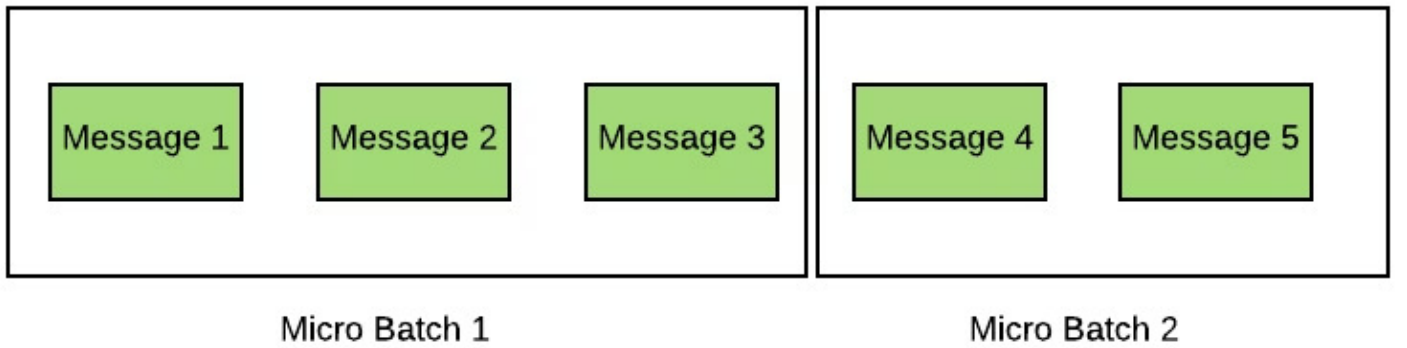
Transformaciones y acciones para manipular los datos en streaming.

Windowing

Procesamiento de datos dentro de una ventana de tiempo específica para análisis en tiempo real.

Microbatching

Micro-batching es un concepto clave en Apache Spark Streaming. Se refiere al proceso de dividir un flujo de datos en intervalos de tiempo pequeños, llamados micro-lotes, que luego se procesan en su totalidad antes de continuar con el siguiente micro-lote. Es una técnica que permite simular el procesamiento en tiempo real mientras se aprovecha la eficiencia del procesamiento por lotes.



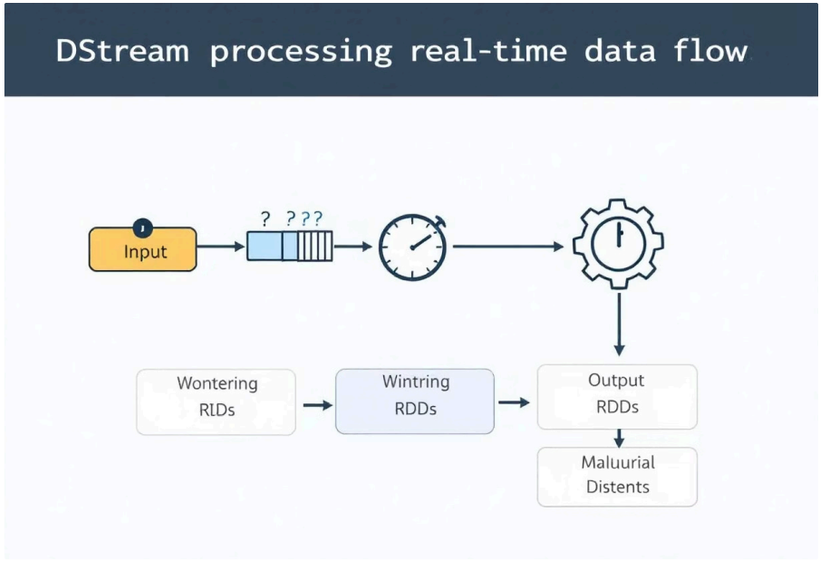
DStreams en Spark Streaming

Qué es un DStream

DStream (short for Discretized Stream) es una de las abstracciones fundamentales en Apache Spark Streaming. Un DStream representa un flujo de datos en tiempo real, y es una secuencia de RDDs (Resilient Distributed Datasets) que se procesan a medida que llegan.

Ejemplo:

Imagina que tienes un sistema de monitoreo de logs de servidores y quieres contar cuántos accesos recibes por segundo. Los logs se transmiten continuamente a través de un flujo de datos y Spark Streaming los procesa en tiempo real usando un DStream.



Casos de Uso Comunes

DStream (Discretized Stream) es una de las abstracciones clave en Spark Streaming, y se utiliza para procesar flujos de datos en tiempo real. A continuación, te menciono algunos de los casos de uso comunes de DStream en aplicaciones de procesamiento en tiempo real:

Monitoreo de Redes Sociales

Tweets, Posts, etc.

Detección de Fraudes

En Transacciones Financieras

Monitoreo de Logs

Y Eventos del Sistema

Análisis de Datos IoT

En Tiempo Real de Datos de Sensores

Recomendaciones

En Tiempo Real

Análisis de Tráfico Web

En Tiempo Real

Procesamiento de Eventos

En Juegos en Línea

Procesamiento con Spark Streaming

El procesamiento con Spark Streaming permite procesar grandes flujos de datos en tiempo real utilizando las capacidades de Spark para manejar big data. En lugar de procesar los datos por lotes (batch processing), Spark Streaming permite procesar datos continuamente, lo que es esencial para aplicaciones que requieren análisis en tiempo real, como detección de fraudes, monitoreo de eventos en vivo, análisis de redes sociales, etc.

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.

```python
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.

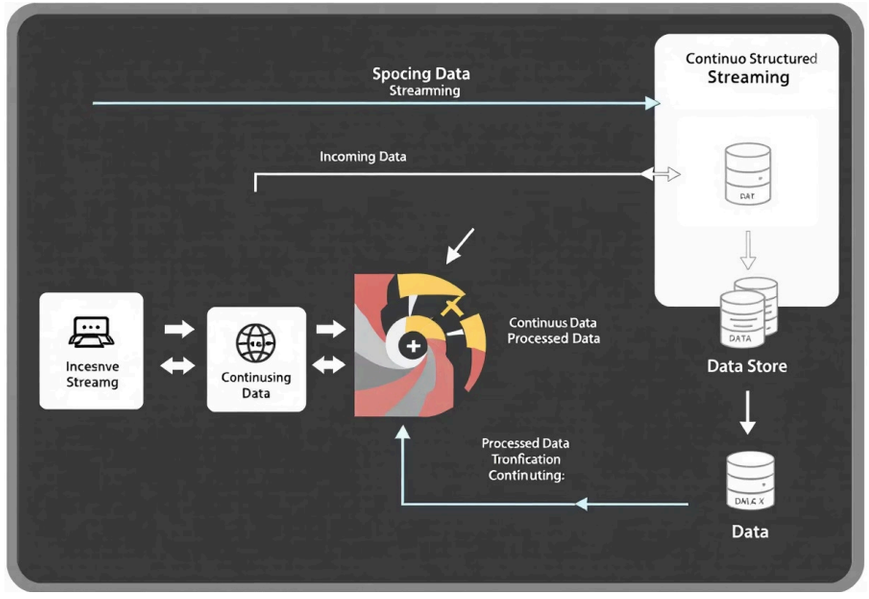
Integraciones disponibles:

- Apache Kafka
- Apache Flume
- Amazon Kinesis
- Sockets
- HDFS
- Cassandra
- RabbitMQ

Spark Structured Streaming

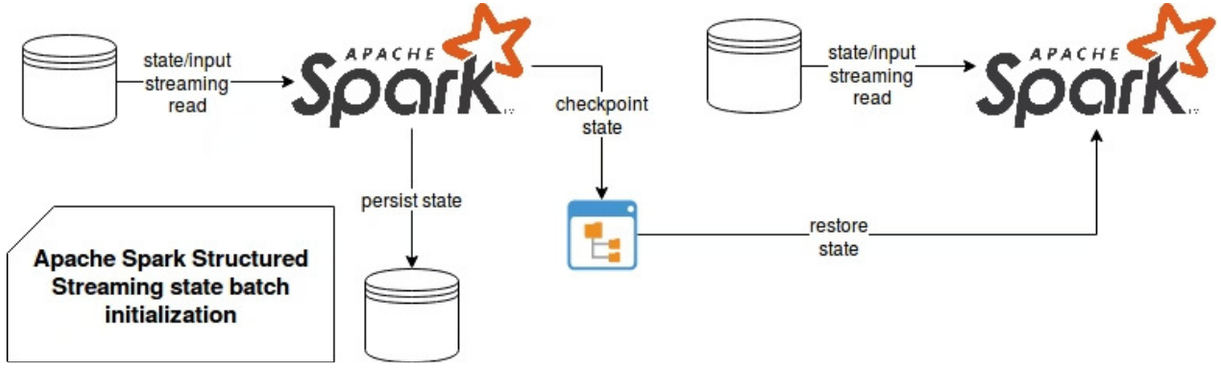
Qué es Structured Streaming

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 con Spark Structured Streaming

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.

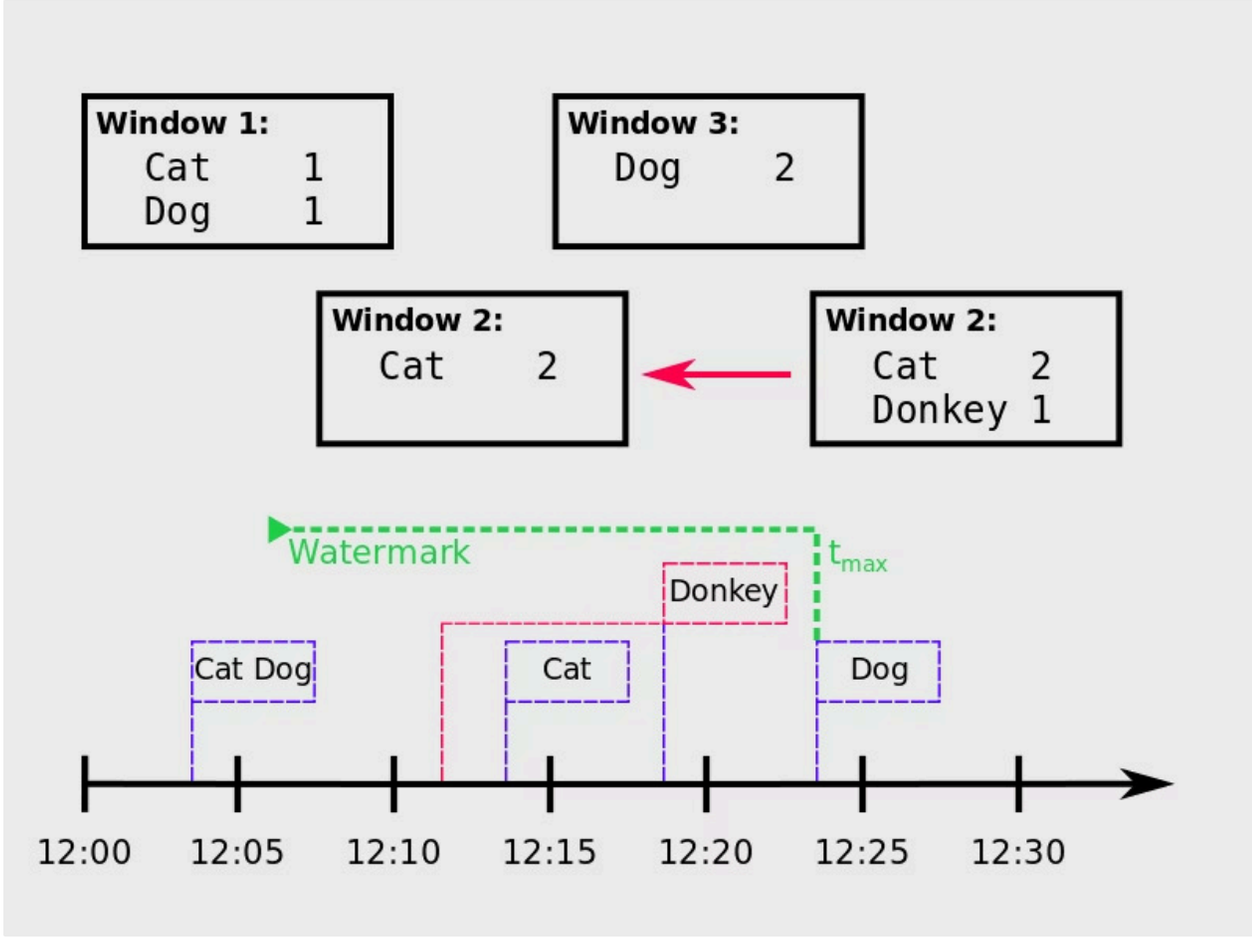


Watermark y Processing Time

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.

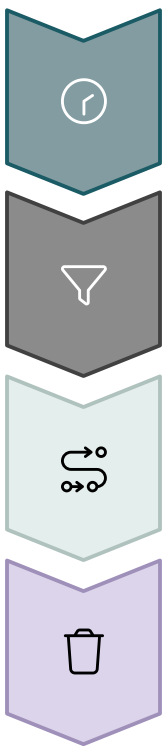
Para abordar este problema, Structured Streaming introduce el concepto de watermarking, que le indica a Spark que permita cierta tolerancia en los datos desordenados. Esto se hace permitiendo que los eventos lleguen con un retraso aceptable y aun así sean procesados correctamente.



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.

- ***Ejemplo:*** Si un flujo de datos tiene una frecuencia de llegada de eventos de 1 segundo y un intervalo de procesamiento de 5 segundos, entonces el processing time indica que los datos serán procesados cada 5 segundos, y el sistema esperará hasta que el intervalo esté completo antes de ejecutar las transformaciones.



Llegada de Eventos

Los eventos llegan continuamente con timestamps

Aplicación de Watermark

Se establece un límite de tolerancia para eventos tardíos

Procesamiento

Los eventos dentro del límite se procesan normalmente

Descarte

Los eventos demasiado tardíos se descartan

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, [])
}
```

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)
```

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 palabras

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)
```

Paso 5: Mostrar resultados

Mostrar los resultados en la consola:

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