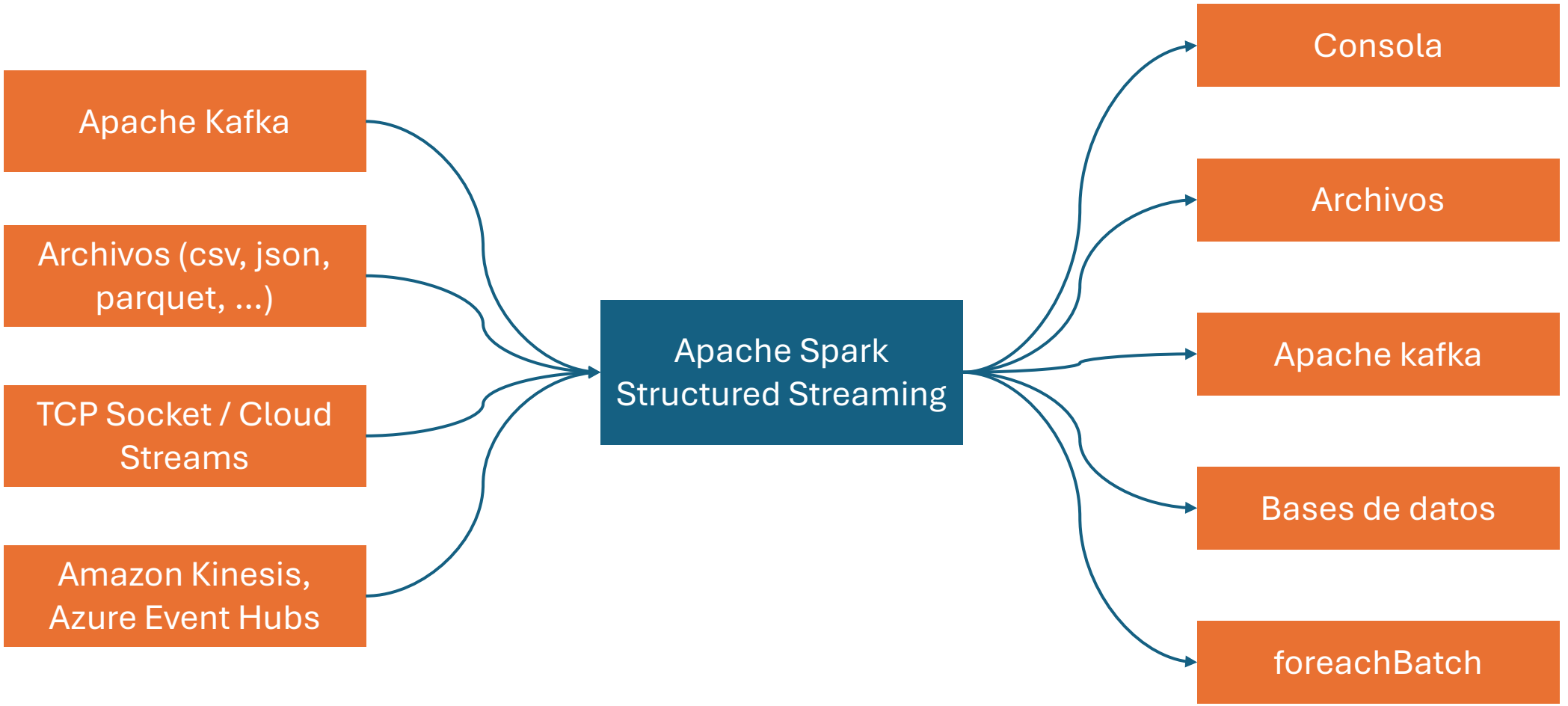
Sesión práctica de Spark para datos en Streaming

Fuentes y salidas

Spark **lee** datos de **fuentes**, los **transforma** (aplicando la lógica de la tabla infinita), y **escribe** los resultados en **salidas**.



Una Tabla Infinita (unbounded input table)

- Cada nuevo dato que llega al sistema (un clic, una venta, una lectura de sensor) se considera una nueva fila que se anexa al final de esta tabla.
- Esta "tabla" no tiene un final definido; es "no acotada" (unbounded) porque siempre pueden llegar más datos en el futuro.
- Podemos usar las mismas herramientas que ya conocemos para consultar tablas estáticas (como SQL y operaciones de DataFrame) para analizar datos en tiempo real.

Estrutura usual de un script

Etapa	Código
Leer entradas	df = spark.readStream.format(...).load()
Define la Lógica/Transformación	query = df.filter(...).join(...).groupby(...)
Define la Salida (Sink) y el Modo	query.writeStream.trigger(...).outputMode(...).format(...)
Inicia la consulta	query.start()
Espera	query.awaitTermination()

Transformaciones

Sin estado

La transformación de cada fila es independiente de las demás.

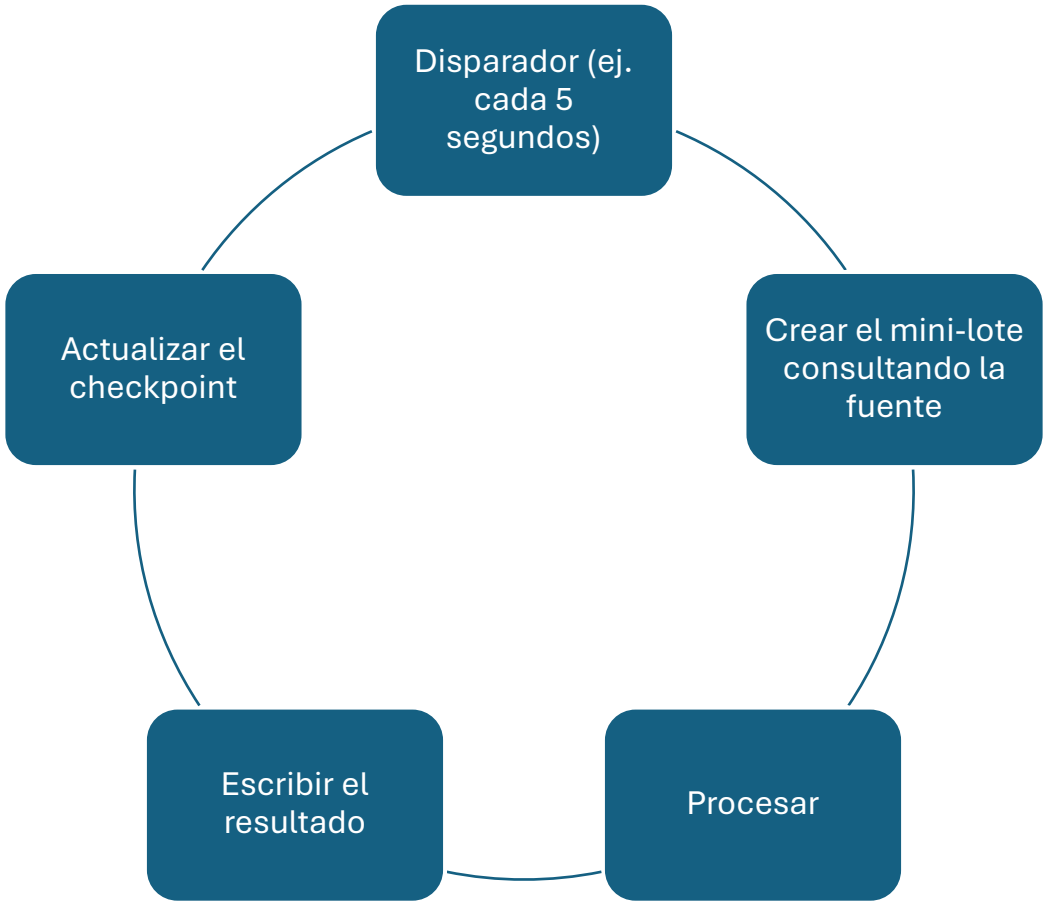
select, filter, withColumn, explode, drop, join, ...

Con estado

Se requiere mantener un estado a lo largo del tiempo para poderlas llevar a cabo.

groupBy, join, dropDuplicates

Micro-Lotes (Micro-Batch Processing)

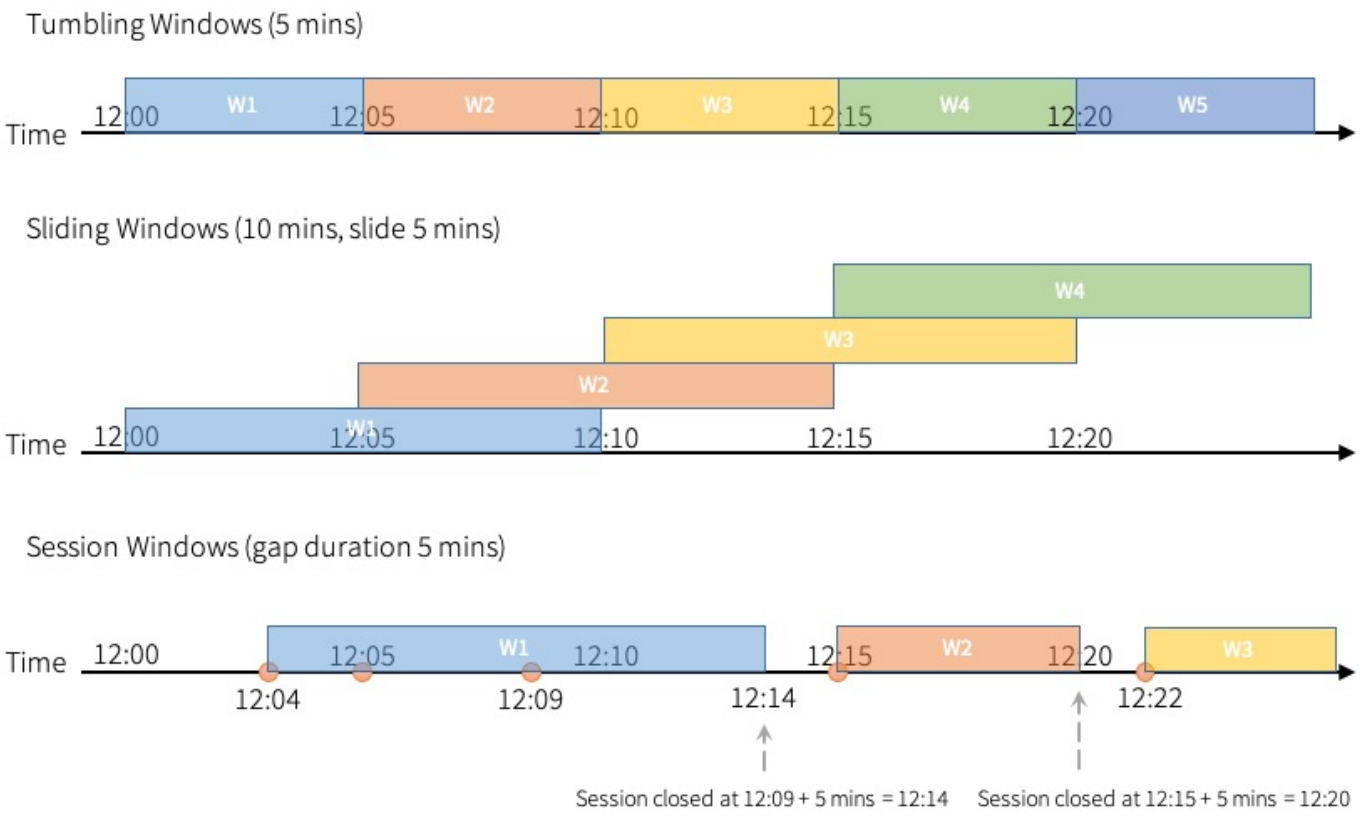


- **Rendimiento**: Es más eficiente que procesar cada evento individualmente, ya que aprovecha al máximo el motor de optimización de Spark.
- **Simplicidad**: Permite usar la misma API de DataFrames para el procesamiento en batch y en streaming. El código es prácticamente idéntico.
- **Tolerancia a Fallos**: Si la aplicación falla, al reiniciarse lee el último checkpoint y sabe exactamente desde dónde continuar, garantizando que los datos no se pierdan ni se dupliquen.

Disparadores (triggers)

Disparador		Código
processingTime	*Intenta* ejecutar el proceso creando un lote cada cierto tiempo.	df = spark.readStream . trigger(processingTime=“5 seconds”) …
once	Ejecuta un solo lote y termina	df = spark.readStream . trigger(once=True) …
availableNow	Ejecuta todos los lotes y termina	df = spark.readStream . trigger(availableNow=True) …
continuous	Procesa cada dato tan pronto llega	df = spark.readStream . trigger(continuous=“5 seconds”) …

Ventanas



Tumbling

```
df.groupBy(f.window("timestamp", "10 minutes")).count()
```

Sliding

```
df.groupBy(f.window("timestamp", "10 minutes", "5 minutes")).count()
```

Session

```
df.groupBy(session_window("timestamp", "5 minutes")).count()
```

Manejo de datos atrasados con watermarks

- Por latencia de red o fallos, los datos pueden llegar **tarde y desordenados**. Spark no puede esperar indefinidamente o la memoria se llenará.

- La marca de agua le dice a Spark cuánto tiempo debe esperar para cerrar la ventana

```
# Se define el watermark sobre la columa
df_wm = df.withWatermark("timestamp", "10 minutes")

# Se utiliza la columna en una ventana
df_ wm.groupBy(
        window("timestamp", "5 minutes")
).count()
```

Joins entre streams

Stream	Static	Inner	✅, sin estado
		Left	✅, sin estado
		Full	❌
		Semi	✅, sin estado
Stream	Stream	Inner	✅, watermark a ambos lados para limpiar el estado
		Left	⚠️, debe tener watermark a la izquierda
		Full	⚠️, debe tener watermark en un lado
		Semi	⚠️, debe tener watermark a la derecha

```
df1 = df.withWatermark(“time1", "2 hours")
df2 = df.withWatermark(“time2", "3 hours")

df1.join(
    df2,
    f.expr("""
        id1 = id2 AND
        time1 >= time2 AND
        time1 <= time2 + interval 1 hour
    """)
)
```

Modos de salida (Output Modes)

Disparador		Código
append	Escribe nuevas filas a la salida sin modificar las demás. Por lo tanto debe esperar a que ce cierren las ventanas para poder escribir.	df.writeStream.outputMode("append") ...
update	Escribe nuevas filas y actualiza sólo aquellas que han cambiado.	df.writeStream.outputMode("update") ...
complete	Emite toda la tabla de resultado en cada lote.	df.writeStream.outputMode("complete") ...

Groupby, watermarks y output mode

```
(
    df
    .groupby("llave").count()
    .writeStream
    .outputMode("append")
    .option("checkpointLocation", "checkpoint/")
    .format("parquet")
    .start("file.parquet")
)
```

```
(
    df
    .withWatermark("timestamp", "10 seconds")
    .groupby("llave", "timestamp").count()
    .writeStream
    .outputMode("append")
    .option("checkpointLocation", "checkpoint/")
    .format("parquet")
    .start("file.parquet")
)
```

```
(
    df
    .groupby("llave", "timestamp").count()
    .writeStream
    .outputMode("append")
    .option("checkpointLocation", "checkpoint/")
    .format("parquet")
    .start("file.parquet")
)
```

Ejercicio práctico