⚡

Spark streaming

1 | Spark Streaming

> ***Spark Streaming is an extension of the core Spark API that allows enables high-throughput, fault-tolerant stream processing of live data streams.***

- 실시간 데이터 처리를 위한 모듈
- 연속적인 데이터 스트림을 스파크가 작동할 수 있는 개별 데이터 컬렉션으로 변환 : 마이크로 배치 모델
- Kafka, Flume, Twitter, ZeroMQ, TCP 소켓 등 다양한 소스에서 데이터 수집 할 수 있다.

1-1 | 특징

1) Event Driven

- 사건이 일어나는 것을 기준으로 해결
- 이벤트가 일어난 범위에서 분석하고 도출해 내려는 형태

  → 수 많은 데이터 조직 가운데 이벤트에 해당하는 것에 반응하는 형태의 구조와 밀접한 관계,

따라서, 실시간 스트리밍은 Event Driven으로 맵핑하면 쉽게 구현 가능

2) 실시간

- 상대적인 개념, 요구 사항에 따라 실시간의 범위를정의 가능
- Service Level Agreement : 실시간 분석 시, 응답 속도가 n 초이내에 정의 된다

3) 낮은 수준의 지연 시간(Low Lstency)

4) 일정한 응답 속도의 보장과 예측 가능한 성공 제공

1-2 | 주요 측면

- 장애 및 낙오에 대한 신속한 복구
- 로드 밸런싱 및 리소스 사용률 향상
- 스트리밍 데이터와 정적 데이터 셋 및 대화형 쿼리 결합
- 고도의 처리 라이브러리와 SQL, 머신러닝, 그래프 처리 등의 네이티브 통합

1-3 | Batch Process VS Streaming Process

	Batch Process	Streaming Process

▼ **Streaming Process**

- 새로운 데이터를 제공하는 한 지속적으로 작동해야 함
- 스트림 처리에서의 시간의 개념
  - 유휴 데이(과거 데이터) : 파일 형식, 데이터베이스 내용 또는 기타 레코드 종류
  - 사용 중인 데이터 : 차량의 센서 또는 GPS 신호 측정과 같이 연속적으로 생성되는 신호 시퀀스

> 💡 **이벤트 시간**
> 이벤트가 생성되었을 때의 시간. 시간 정보는 이벤트를 생성하는 장치의 로컬 시간에 의해 제공된다. 이벤트를 서로 관련시키거나 순서를 정하거나 집계해야 할 때 이러한 타임라인 간의 차별화는 매우 중요해진다.

- 실시간 스트리밍 사례
    - **미디어 추천**

    한 전국 미디어 매체는 뉴스 리포트와 같은 새로운 영상 자료를 추천 시스템에 수집하기 위해 스트리밍 파이프라인을 구축하여 영상 자료가 회사의 미디어 저장소에 수집되는 즉시 사용자의 개인화 추천에 사용할 수 있도록 했다.

    - **고장 탐지**

    한 대규모 하드웨어 제조업체는 복잡한 스트림 처리 파이프라인을 적용하여 장비 측정 항목을 받는다. 시계열 분석을 사용하면 잠재적인 오류가 감지되고 이를 수정하기 위한 조치가 장비에 자동적으로 다시 전송된다.

    - **빠른 대출**

    대출 서비스를 제공하는 한 은행은 여러 데이터 스트림을 스트리밍 애플리케이션으로 결합 함으로써 대출 승인에 걸리는 시간을 몇 시간에서 몇 초로 줄일 수 있었다.

    이 외에도 , 주식 트레이딩, 실시간 추천 광고, 실시간 금융 사기 방지, n분간 주문 상품 Top10 등을 실시간 스트리밍을 통해 구현할 수 있다.

    👉 [spark streaming 기반 준실시간(NRT)추천 시스템]

    Near Real-Time Netflix Recommendations using Apache Spark (Nitin Sharma and Elliot Chow)

    Nitin Sharma, a Senior Software Engineer at Netflix, and Elliot Chow, a software engineer at Netflix, explain a data-driven company, we use Machine learning based algos and A/B tests to drive all of the content recommendations for our members.

    https://www.youtube.com/watch?v=IGfvVd-v3P8&t=1077s

따라서,

- Batch Process : 특정 기간 동안 이미 정의된 그룹에 존재하는 데이터를 수집하는 경우, 단 기간 분석의 경우
- Streaming Process : 실시간으로 몇 초만에 전환되는 데이터를 분석 가능, 빠른 데이터 처리 속도를 원하는 경우

> 💡 ***이벤트 기반 아키텍처(Event Driven Architecture)***
> 실시간 스트리밍은 이벤트 기반 아키텍처로 맵핑한다면 쉽게 해석이 가능하다.
>
> 이벤트 프로듀서 — — — — — — — — — — — —>이벤트 컨슈머
> 이벤트 처리 로직

---

2. Spark Streaming API

2-1. DStream

: 연속적인 데이터 스트림을 나타냄,

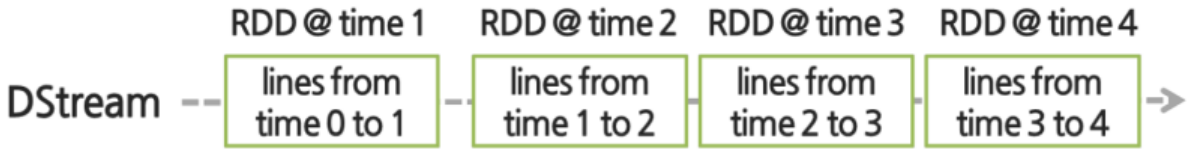
정의된 시간 간격 동안의 이벤트 처리를 우선 수집하여 데이터를 마이크로 단위로 일괄 처리

처리 과정

>>> 지속적으로 Input data가 아날로그 형태로 들어옴 → 실제 처리는 디지털 형태로 처리

: 아날로그 형태의 데이터가 디지털 형태로 분해

> 💡 **형태에 따른 데이터 처리**
> 큰 형태의 데이터 : RDD
> 마이크로 배치(Micro-Batch) 형태의 데이터 : Dstream
> → RDD만으로는 실시간 처리에 어렵기 때문에 Dstream을 통해 RDD와 매핑합니다.
> Dstream을 프로세싱하게 되면 실시간 스트리밍으로 연결



- 각 RDD는 일괄 처리 간격 동안 수집된 이벤트를 나타내고, RDD의 연속 집합은 Dstream으로 수집

  → Dstream은 내부적으로 RDD의 연속적인 시퀀스로 표현

> ex) Word Count : 0~ 1초까지의 시간에 들어온 라인의 개수를 Dstream의 데이터 추상화 객체에 담을 수 있고, 배정된 라인 하나하나를 각각의 RDD로 할당할 수 있음

장/단점

- 장점 : 일반적으로 많이 사용됨(안정적), RDD 코드를 함께 사용해 정적 데이터와 조인 등의 기능 제공
- 단점 : Java, Scala, Python 등의 객체와 함수에 매우 의존적 -> 스트림 처리 엔진의 최적화 기법 적용하기 힘듬 ,처리 시간을 기준으로 동작함-> 이벤트 시간을 기준으로 처리하고 싶은 경우 자체 구현 필요

2-2 | Structured Streaming

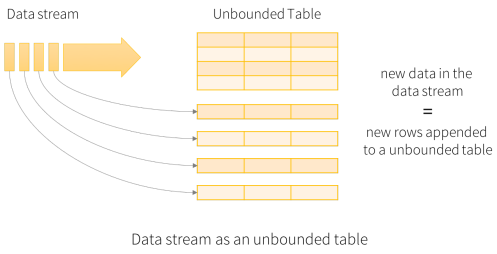
: Spark SQL 추상화 위에 구축된 스트리밍 프로세서 , 구조화된 스트리밍 쿼리는 기본적으로 마이크로 배치 처리 엔진을 사용

마이크로 배치(Micro-Batch)

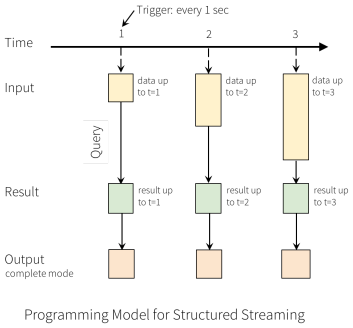
- 데이터 스트림을 개별 단위로 나눈 후, 각 단위 데이터를 스파크 엔진으로 처리
- 데이터를 초 단위의 타임 윈도우로 나눈 후, 스파크 엔진을 실행하는 것을 의미(소량의 데이터에 대한 배치 프로세스가 적용됨.)

> 💡 ***연속형 처리***
> 스파크 스트리밍은 기본적으로 마이크로 배치 방식으로 작업
> 마이크로 배치 방식은 어느 정도 데이터가 쌓이길 기다려서 한꺼번에 효율적으로 작업을 할 수 있지만, **지연 시간이 발생**한다는 단점.
> 반면, 연속형 처리는 레코드 하나 하나를 바로 작업하는 방식이기 때문에 빠른 응답 속도를 자랑함. 하지만 부하가 커서 같은 데이터량을 처리하는데 비용에 대한 부담이 커짐
> → 정확히 한번 실행을 보장(exactly-once guarantees)하는 마이크로 배치 방식보다 100배 빠름. 다만, 최소 1회(at-least-once guarantees)를 보장하기에 데이터 중복의 가능성 있음.

기본 개념



stream as a table

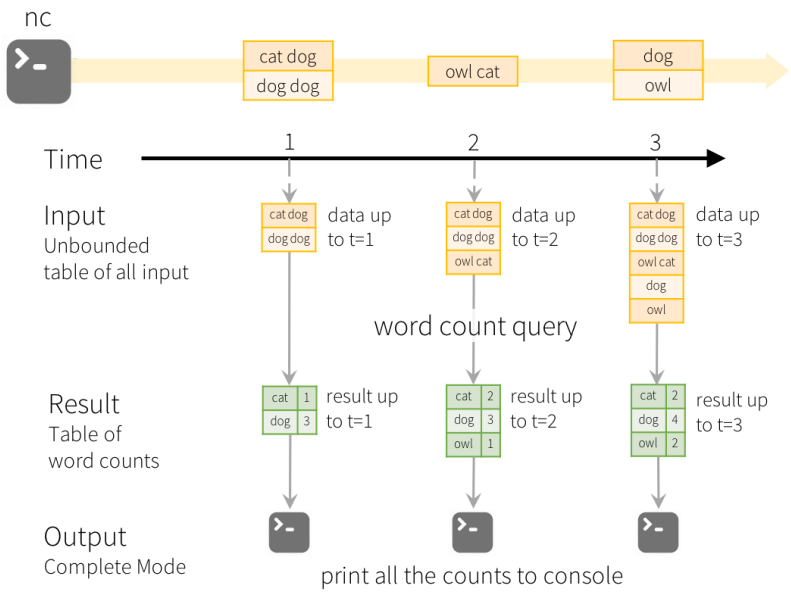


structured streaming model

입/출력 소스

- 입력 : 입력 받는 소스 데이터의 정보
  - HDFS나 S3 등 분산 파일 시스템의 파일
  - 테스트용 소켓 소스
  - 테스트용 증분형 입력 소스(Rate Source)
- 출력 : 외부 스토리지에 기입되는 것
  - CompleteMode : 갱신된 결과 테이블 전체가 외부 스토리지에 기록, 기입 방법은 스토리지 커넥터가 결정
  - Append Mode : 마지막 트리거 이후 결과 테이블에 추가된 새 행만 외부 스토리지에 기록, 결과 테이블의 기존 행이 변경되지 않을 것으로 예상되는 쿼리에만 적용

- Update Mode : 마지막 트리거 이후, 결과 테이블에서 업데이트된 행만 외부 스토리지에 기록, (* 마지막 트리거 이후 변경된 행만 출력한다는 점에서 Complete Mode와 다름, 쿼리에 집약이 포함되지 않으면 Append Mode와 동일함)



Model of the Quick Example

❗ TCP 소켓에서 수신된 텍스트 데이터의 실행 워드 수를 유지한다고 가정

→ 정적 데이터 프레임으로서 소켓 연결을 통해 새로운 데이터를 지속적으로 확인하고 이전 데이터와 결합하는 증분 쿼리 실행

테이블 전체를 읽는 것이 아니라 **최신 정보를 읽다가 스트리밍 데이터 소스에서 사용 가능한 데이터를 결과를 점진적으로 업데이트**

트리거

: 스트리밍 데이터 처리 타이밍

- 쿼리는 고정된 배치 간격을 가진 마이크로패키지 쿼리 또는 연속 처리 쿼리로 실행
- 처리 시간(고정된 주기로만 신규 데이터 탐색) 기반의 트리거 지원
- 작은 크기 파일이 여러 개 생성 될 수 있음.

Structure Streaming 특징

- 새로운 모델이 있을 때 , 결과 테이블을 업데이트할 책임
- Kafka, Socket 등으로 데이터를 입력받는 것 가능

- dstream보다 최근에 도입
  (* dstream은 RDD로, Structure Streaming은 DataFrame 형태로 결과값이 도출되므로 데이터 처리에 용이)

- 스파크의 구조적 API를 기반으로 하는 고수준 스트리밍 API
- 구조적 처리를 할 수 있는 모든 환경에서 Scala, Java, Python, R, SQL 등을 통해 사용 가능
- DStream처럼 고수준 연산 기반의 선언형 API 제공
- 최적화 기술 자동으로 수행
- 이벤트 시간 처리 지원(+Watermark)

---

3 | 이벤트시간 윈도우, 워터마크

3-1 | 이벤트 시간 윈도우 (event-time window)

📌 이벤트 시간 윈도우란, 행의 **이벤트 시간**이 속하는 윈도우별로 집계함 (그룹화 함)

▼ + 이벤트 시간 처리 개념

> 이벤트가 생성된 시간을 기준으로 정보를 분석해 늦게 도착한 이벤트까지 처리할 수 있게 해줌

[구분]

- 이벤트 시간 : 이벤트가 실제로 발생한 시간
  - 데이터에 기록된 시간
  - 데이터 마다 이벤트 시간을 비교할 때, 지연되거나 무작위로 도착한 이벤트가 있으므로 스트리밍 처리 시 이를 제어할 수 있어야 된다.
- 처리시간 : 이벤트가 시스템에 도착하거나 처리된 시간
  - 스트림 처리 시스템이 데이터를 실제 수신한 시간
  - 처리 시간은 세부 구현과 관련된 내용과 관련됨
  - 처리시간은 이벤트 시간처럼 외부 시스템에서 제공하는 것이 아니라 스트리밍 시스템이 제공하는 속성이므로 순서가 뒤섞이지 않음

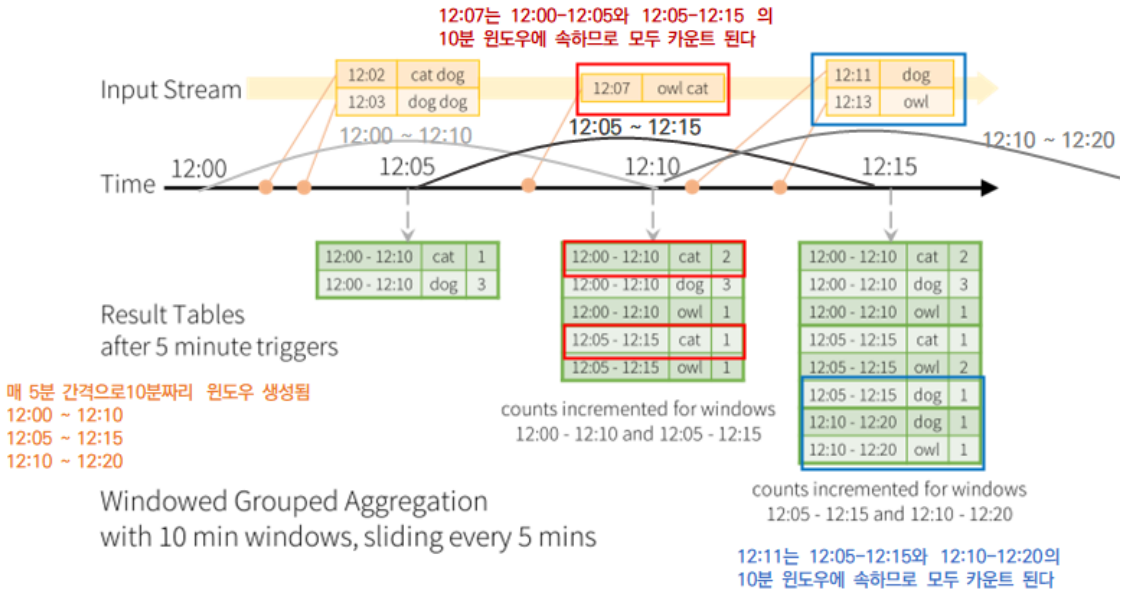
한국에 데이터센터가 있다고 가정하자.
일본에서 발생한 이벤트와 미국에서 발생한 이벤트가 같은 시간에 다른 장소에 발생했다면,
일본에서 이벤트가 미국의 이벤트보다 먼저 데이터 센터에 도착한다.

이를 처리 시간으로 데이터 분석하면 일본의 이벤트가 미국 이벤트 보다 먼저 발생한 것으로 나타나므로 정상적이지 않다.

이를 이벤트 시간 기준으로 분석하면 같은 시간 이벤트로 처리 가능하다.

즉, 처리 시스템의 이벤트 순서는 이벤트 시간 순으로 정렬되어 있다고 보장 할 수 없으므로, 이벤트 시간 기준의 처리가 필요하다.
처리시스템에 도착한 시간 대신 스트림 데이터가 가진 시간 정보를 참조한다.

1) 예시 - 슬라이딩 윈도우



슬라이딩 윈도우 예제 - 매 5분 간격으로 10분짜리 윈도우 생성됨

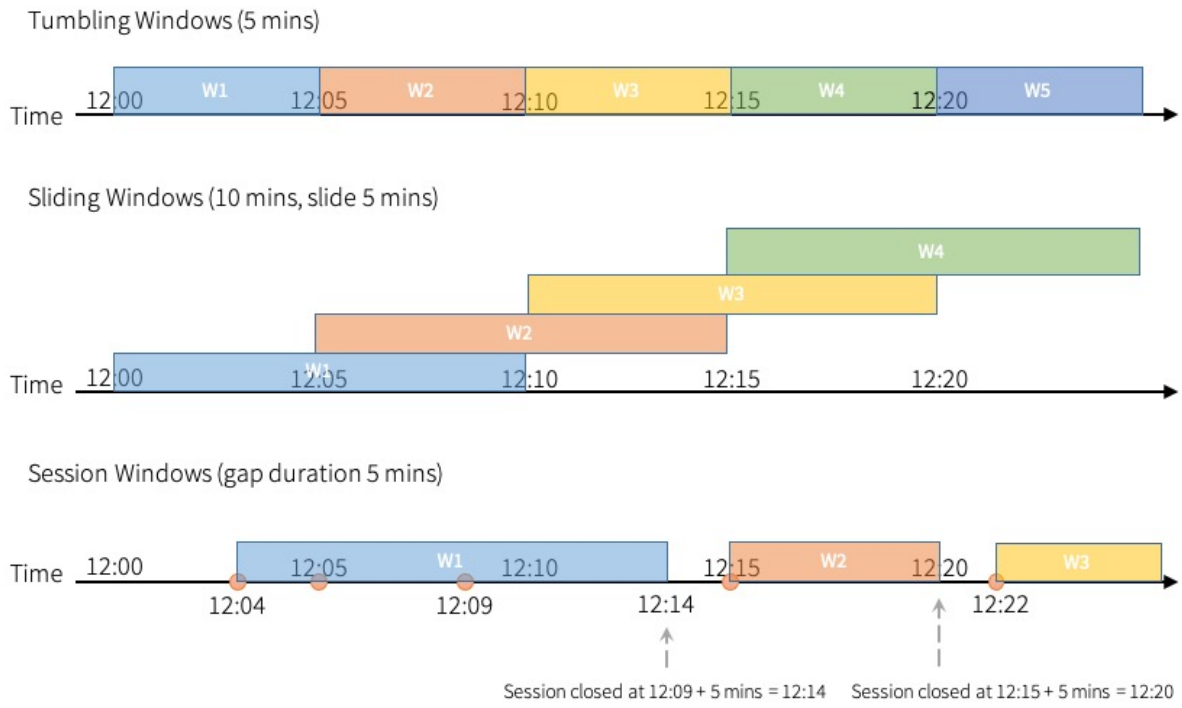
> **매 5분 간격으로10분짜리 윈도우 생성됨**

- 12:00 ~ 12:10
  12:05 ~ 12:15
  12:10 ~ 12:20

> **카운트는 그룹핑 키(word)와 윈도우(이벤트 시간)에 모두 인덱싱됨**

- 12:07는 12:00-12:05와 12:05-12:15 의 10분 윈도우에 속하므로 모두 카운트 된다
- 12:11는 12:05-12:15와 12:10-12:20의 10분 윈도우에 속하므로 모두 카운트 된다

2) 시간 윈도우 종류



시간 윈도우 종류

1. **텀블링 윈도우** : 고정된 크기로 겹치지 않는 연속된 시간 간격. 하나의 입력은 하나의 윈도우에만 바인딩됨
2. **슬라이딩 윈도우** : 고정된 크기의 윈도우를 갖고 있지만, 슬라이드의 기간이 창의 지속 기간 보다 작으면 겹칠 수 있다. 이럴 경우 하나의 입력은 여러개의 윈도우에 바인딩 된다.
3. **세션 윈도우** : 윈도우 크기가 입력크기에 따라 동적으로 변함.
    - 세션 윈도우는 입력이 들어왔을 때 시작되며, 입력이 주어지는 갭 기간 동안 확장된다.
    - 세션윈도우는 마지막 입력 값을 받고, 갭 기간동안 더 이상 입력값이 없다면 닫힌다.

3) 코드

텀블링/슬라이딩 윈도우는 `window` 함수를, 세션 윈도우는 `session_window` 함수가 이용된다.

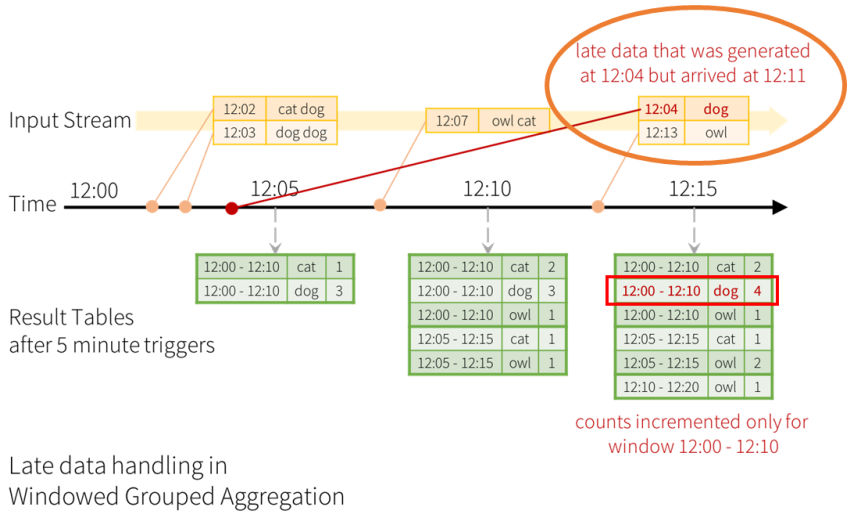
```
words = ...  # streaming DataFrame of schema { timestamp: Timestamp, word: String }

# Group the data by window and word and compute the count of each group
windowedCounts = words \
    .withWatermark("timestamp", "10 minutes") \
    .groupBy(
        window(words.timestamp, "10 minutes", "5 minutes"),
        words.word) \
    .count()
```

```
events = ...  # streaming DataFrame of schema { timestamp: Timestamp, userId: String }
```

```
# Group the data by session window and userId, and compute the count of each group
sessionizedCounts = events \
    .withWatermark("timestamp", "10 minutes") \
    .groupBy(
        session_window(events.timestamp, "5 minutes"),
        events.userId) \
    .count()
```

3-2 | 지연데이터 워터마크 처리

1) 예시



이벤트 하나가 늦게 도착한 경우(=지연데이터)

> 만약, 12:11에 12:04에 생성된 데이터가 늦게 수신된다고 하면, 해당 윈도우 시간(12:00 ~ 12:10)에 집계되어야 한다.

⇒ 지연 데이터가 이전 윈도우의 집계를 정확히 업데이트 할 수 있게, 오랜 기간동안 이전의 부분 집계에 대한 중간 상태를 유지해야 한다.

> 하지만, 해당 쿼리가 며칠 동안 걸쳐 장기간 실행된다면, 시스템에 누적되는 메모리가 과부하가 올 것이다. → **워터마킹**으로 제한을 둔다

⇒ 즉, 워터마킹이란 spark 엔진이 현재 이벤트 시간을 자동으로 추적하고 이전 상태들을 정리할 수 있도록 한 기능이다.

2) 코드

```
words = ...  # streaming DataFrame of schema { timestamp: Timestamp, word: String }

# Group the data by window and word and compute the count of each group
windowedCounts = words \
    .withWatermark("timestamp", "10 minutes") \
    .groupBy(
        window(words.timestamp, "10 minutes", "5 minutes"),
        words.word) \
    .count()
```

- 지연되는 데이터를 허용하는 시간 임계값 : 10분

  ⇒ 즉, 10분 이내의 이벤트만 받아들인다. 만약 워터 마크를 지정하지 않으면 전체 윈도우 데이터를 영원히 유지하며 결과를 갱신하기 때문에 시스템 부하가 크다.

▼ 참고

https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html

https://www.databricks.com/glossary/what-is-spark-streaming#:~:text=Spark Streaming is an extension,%2C databases%2C and live dashboards.

https://memgraph.com/blog/batch-processing-vs-stream-processing#toc-2

https://eyeballs.tistory.com/83

<Stream Processing with Apache Spark : 스파크를 활용한 실시간 처리> 제러드 마스, 프랑수아 가랄로 지음

<Spark : The Definitive Guide 스파크 완벽가이드>빌 체임버스, 마테이 자하리아 지음