3. SPARK STREAMING

Apache Spark - December 2021

EDEM

Escuela de Empresarios

Motivation for Real-Time Processing

- **Data is being created at unprecedented rates**
  - Exponential data growth from mobile, web, social
  - Connected devices: 9B in 2012 to 50B by 2020
  - Over 1 trillion sensors by 2020
  - Datacenter IP traffic growing at CAGR of 25%

- **How can we harness it data in real time?**
  - Value can quickly degrade → Capture Value immediately
  - From reactive analysis to direct operational impact
  - Unlocks new competitive languages
  - Requires a completely new approach…

Uses Cases Across Industries

Credit

Identify fraudulent transactions as soon as they occur.

Transportation

Dynamic Re-routing Of traffic or Vehicle Fleet.

Retail

- Dynamic Inventory Management
- Real-time In-store Offers and recommendations

**Consumer Inter
Mobile**

Optimize user engagement based on user’s current behavior.

Healthcare

Continuously monitor patient vital stats and proactively identify at-risk patients.

Manufacturing

- Identify equipment failures and react instantly
- Perform Proactive maintenance.

Surveillance

Identify threats and intrusions In real-time

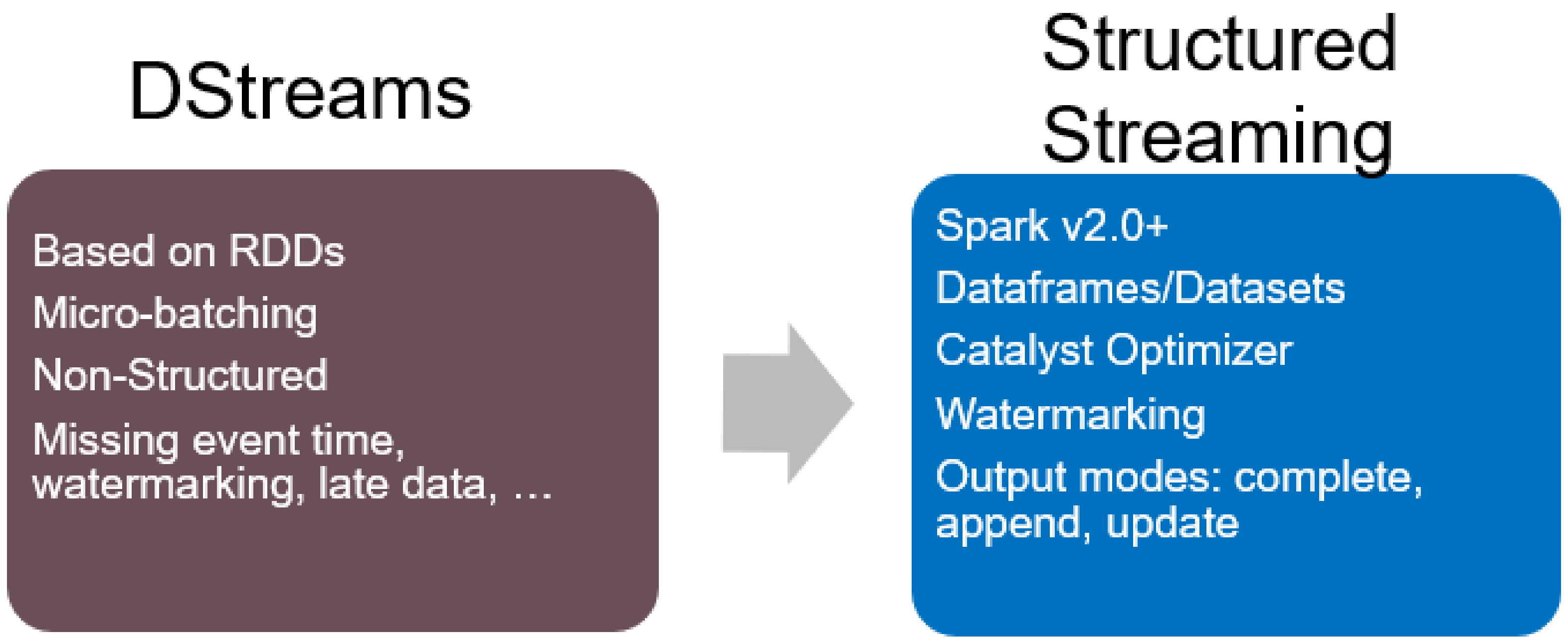
Digital Advertising & Marketing

Optimize and personalize content based on real-time information.

Introduction

- Spark Streaming provides a scalable, fault tolerant, efficient way of processing streaming data using Spark's simple programming model
- It converts streaming data into "**micro batches**", which enable Spark's batch programming model to be applied in Streaming use cases
- This unified programming model **makes it easy to combine batch and interactive** data processing with streaming

Structured Streaming



DStreams

- Based on RDDs
- Micro-batching
- Non-Structured
- Missing event time, watermarking, late data, …

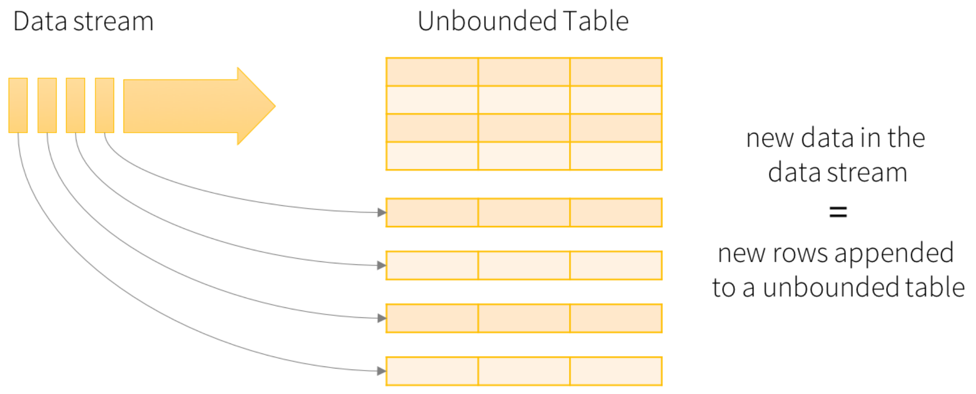
Structured Streaming

- Spark v2.0+
- Dataframes/Datasets
- Catalyst Optimizer
- Watermarking
- Output modes: complete, append, update

Structured Streaming

- Structured Streaming is a **scalable and fault-tolerant** stream processing engine built on the Spark SQL engine

- You can express your streaming computation the same way you would express a batch computation on static data


Data stream
Unbounded Table
new data in the data stream
=
new rows appended to a unbounded table

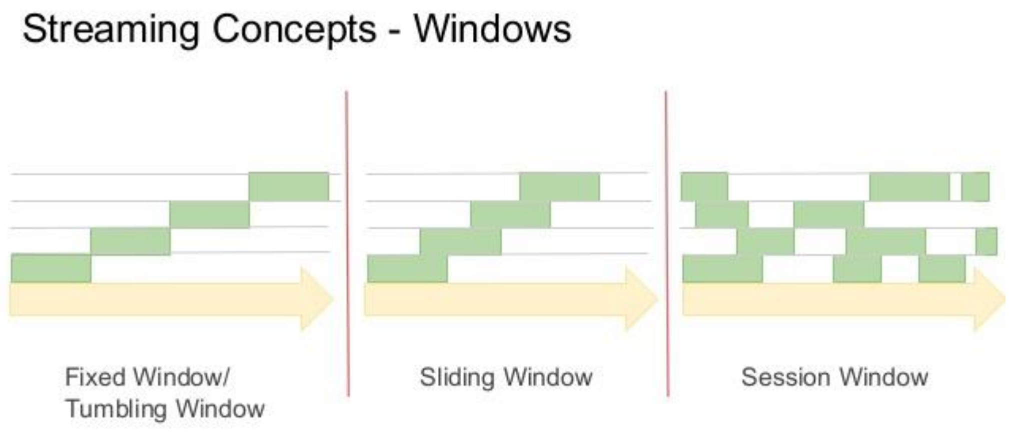
Data stream as an unbounded table

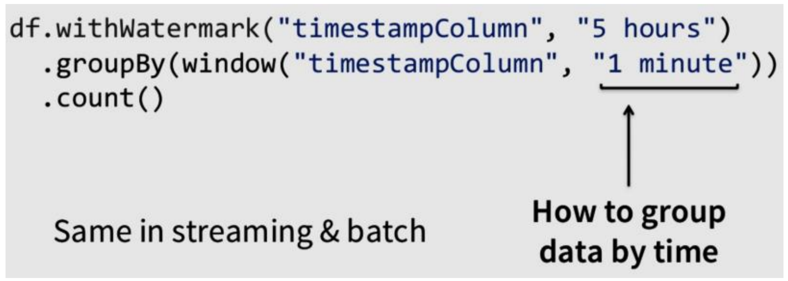
EDEM
Escuela de Empresarios

1. Introduction to Spark Streaming
2. Structured Streaming
3. **Windowing**
4. Kafka Connectors
5. Caching / Persistence
6. Fault Tolerance & Reliability

Windowing

- Windowing is the ability to perform some set-based computation (aggregation) or other operations over subsets of events that fall within some period of time

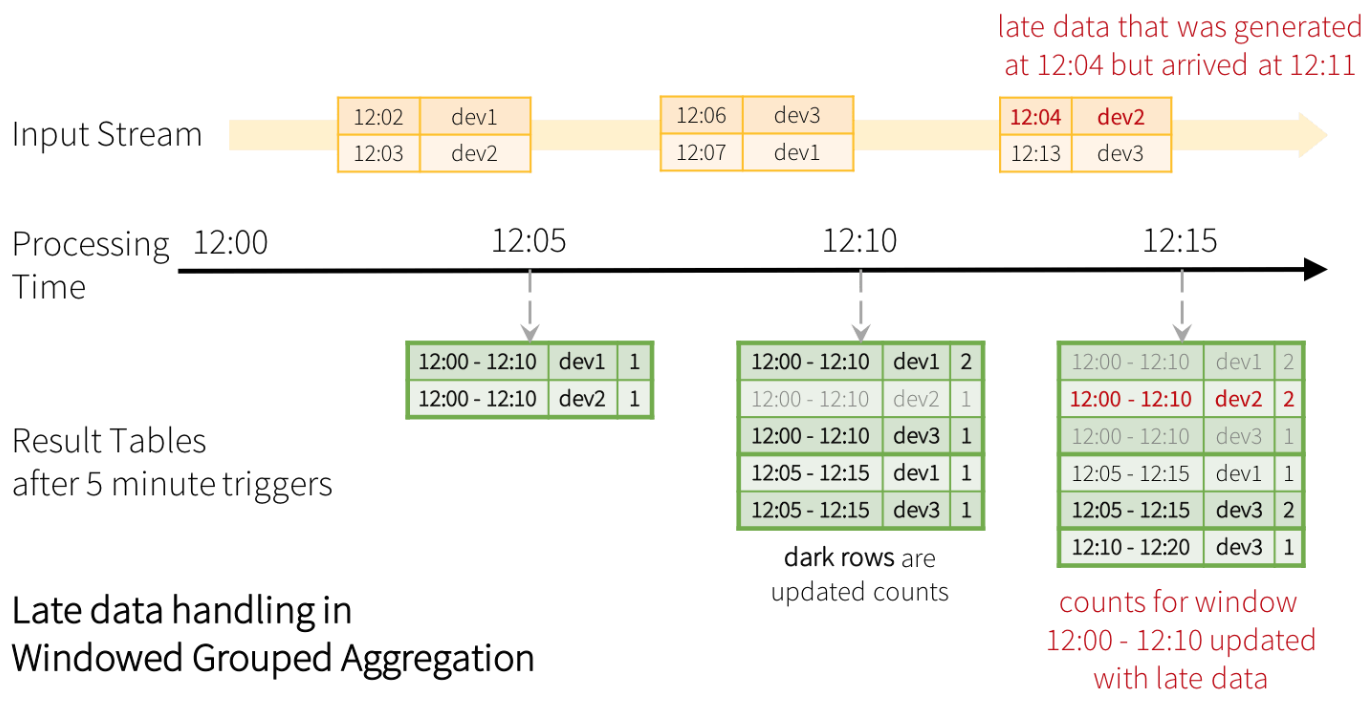




```
df.withWatermark("timestampColumn", "5 hours")
  .groupBy(window("timestampColumn", "1 minute"))
  .count()
```

Watermarking

- Watermarking is a moving threshold in event-time that trails behind the maximum event-time seen by the query in the processed data

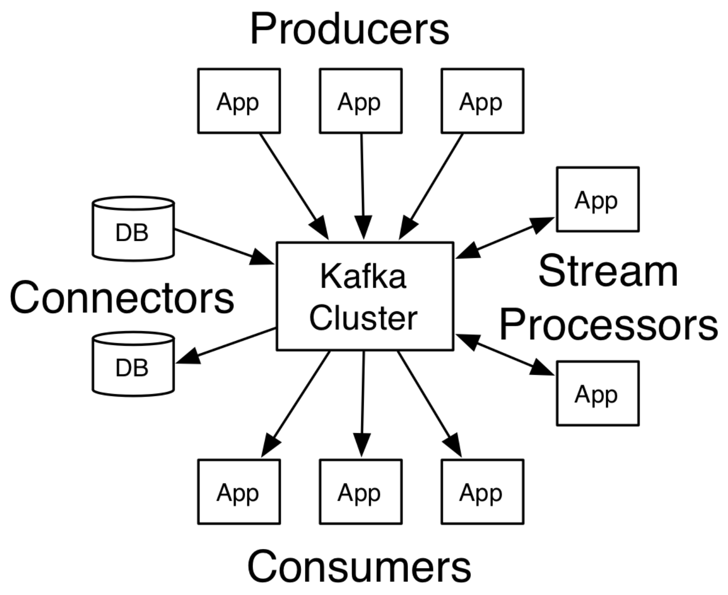


Output Modes

- **Complete Mode** The entire updated Result Table will be written to the external storage.
- **Append Mode** Only new rows appended in the Result Table since the last trigger will be written to the external storage.
- **Update Mode** Only rows that were updated in the Result Table since the last trigger will be written to the external storage (available since Spark 2.1.1)
- Not all output modes are feasible with all queries; check here

Apache Kafka Architecture

- Apache Kafka is a distributed streaming platform with three key capabilities
  - Publish and subscribe to streams of records
  - Store streams of records in a fault-tolerant durable way
  - Process streams of records as they occur



Kafka Connectors

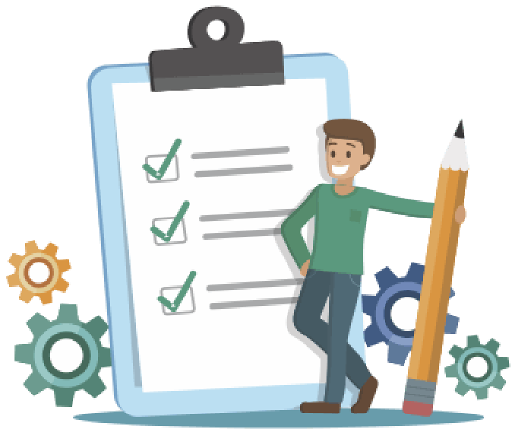
- Structured Streaming integration for Kafka 0.10 to read data from and write data to Kafka.

```
// Subscribe to 1 topic
val df = spark
 .readStream    //read for batch queries
.format("kafka")
 .option("kafka.bootstrap.servers", "host1:port1,host2:port2")
.option("subscribe", "topic1")
 .load()

df.selectExpr("CAST(key AS STRING)", "CAST(value AS STRING)") .as[(String, String)]
```

Kafka Connectors

```scala
// Write key-value data from a DataFrame to a specific Kafka topic specified in
an option
val ds = df
  .selectExpr("CAST(key AS STRING)", "CAST(value AS STRING)")
  .writeStream
  .format("kafka")
  .option("kafka.bootstrap.servers", "host1:port1,host2:port2")
  .option("topic", "topic1")
  .start()

// Write key-value data from a DataFrame to Kafka using a topic specified in the
data
val ds = df
  .selectExpr("topic", "CAST(key AS STRING)", "CAST(value AS STRING)")
  .writeStream
  .format("kafka")
  .option("kafka.bootstrap.servers", "host1:port1,host2:port2")
  .start()
```

Hands-on



- Open “03.Spark_Streaming_Kafka.ipynb” in Google Colab:
    - Execute example 1
    - Try Exercises 1, 2 and 3

EDEM
Escuela de Empresarios