Data Engineering - Streaming

Data Engineering Course
Lucas Rosa
2024

Data Processing

Introduction

Lucas Porto Rosa

Brazilian, 36 years

Principal Data Engineer at HBDC (Metys & Hugo Boss)

LinkedIn: https://www.linkedin.com/in/lucprosa/

Personal

Moved to Portugal in Oct 2021 with wife and dog

Gaúcho, gremista

Crafter beer, travel, guitar player, etc.

Work Experience

> 10 years of experience working with data as DBA, Business Intelligence analyst and Data Engineer

Education

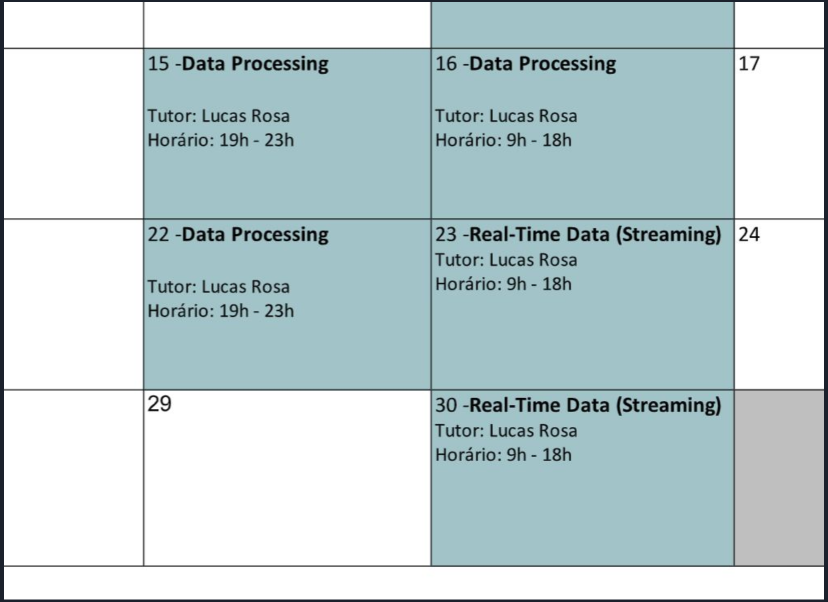
System Development Analysis

MBA Data Science

Tech Certifications

Data Processing

Agenda



	15 -**Data Processing** Tutor: Lucas Rosa Horário: 19h - 23h	16 -**Data Processing** Tutor: Lucas Rosa Horário: 9h - 18h	17
	22 -**Data Processing** Tutor: Lucas Rosa Horário: 19h - 23h	23 -**Real-Time Data (Streaming)** Tutor: Lucas Rosa Horário: 9h - 18h	24
	29	30 -**Real-Time Data (Streaming)** Tutor: Lucas Rosa Horário: 9h - 18h	

- Day 23
  - Data Streaming Introduction
  - Batching & Streaming differences
  - Streaming Use cases
  - Pub-Sub Architecture
  - Spark Streaming Structure Introduction
  -
- Day 30
  - Hands-On
  - Tech challenges
- Day 18
  - Tech challenges (continuation)
  - Doubts/Question

Real-Time Data (Streaming)

- Streaming Processing
  - Diff between batching and streaming
  - Use Cases
  - Technologies
- Pub-Sub Architecture
  - Technologies (Kafka, EventHub, RabbitMQ, EventHub)
  - Topics, queue, schema, checkpoint
  - JSON structure
- Spark Structure Streaming
  - Introduction
  - Syntax (readStream, writeStream, awaitTermination, etc)
  - Checkpoint
  - Hands-on
- Technical Challenge

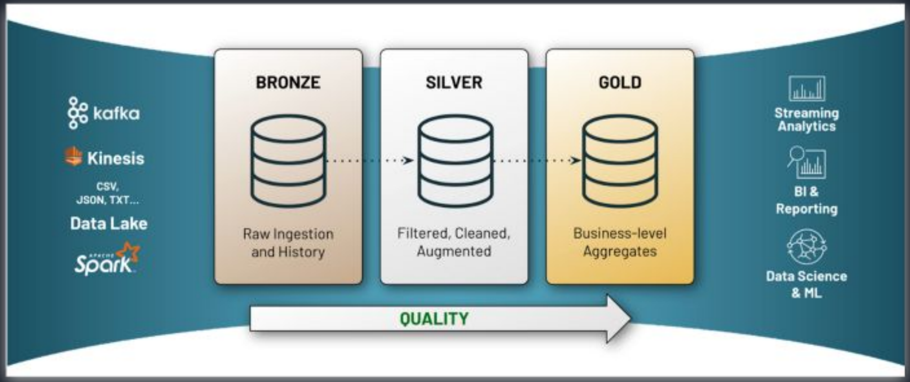
Data Streaming

Data Streaming

Introduction

- Scenario 1
  - Batching process
  - Frequency: daily
  - Output: parquet, table
  - Data refreshness: Day -1
  - Dashboarding, reporting

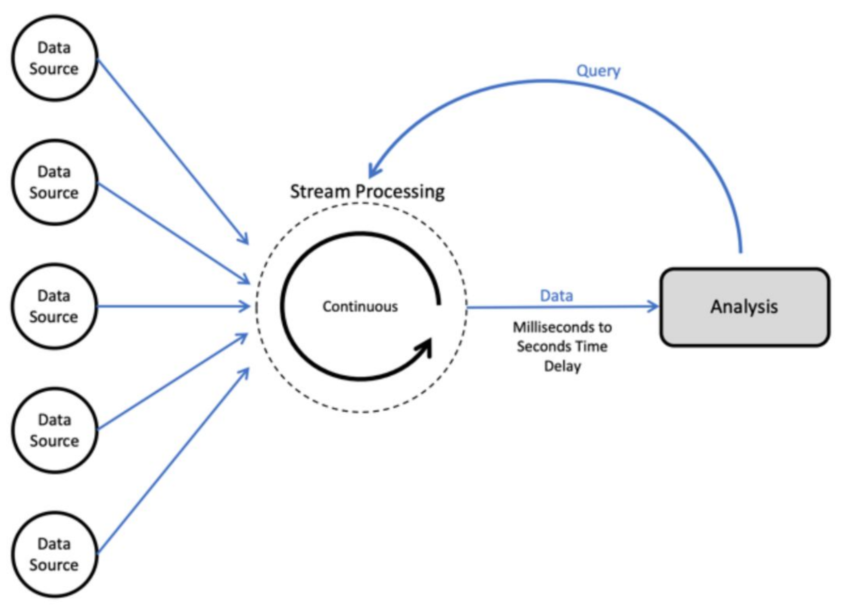
- Scenario 2
  - Data refreshness < 10min
  - Requires real time-aggregations
  - Dashboarding, reporting, application



Data Streaming

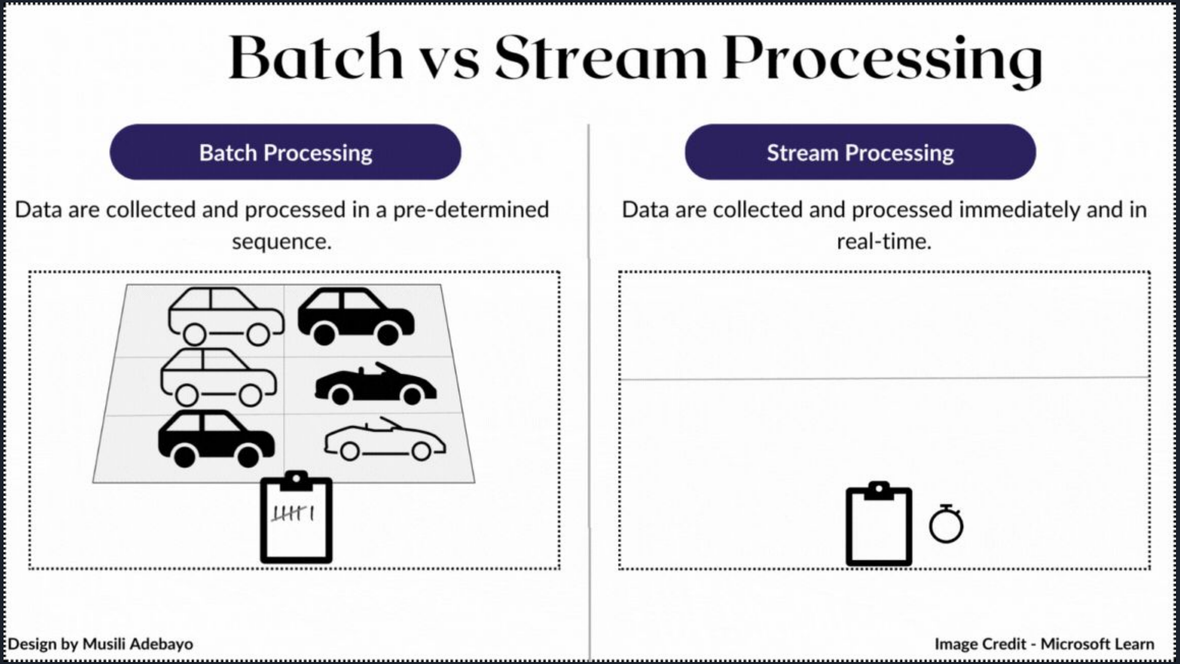
Introduction

- Continuously data processing
- **Data Volume**: Large amount of data
- **Data Latency:** Low latency (by second, minute or hour)
- **Cost**: High cost
- **Use cases**: real-time analytics, fraud detection, anomaly detection



Data Streaming

Batch vs Streaming



Batch vs Stream Processing

Batch Processing

Data are collected and processed in a pre-determined sequence.

Stream Processing

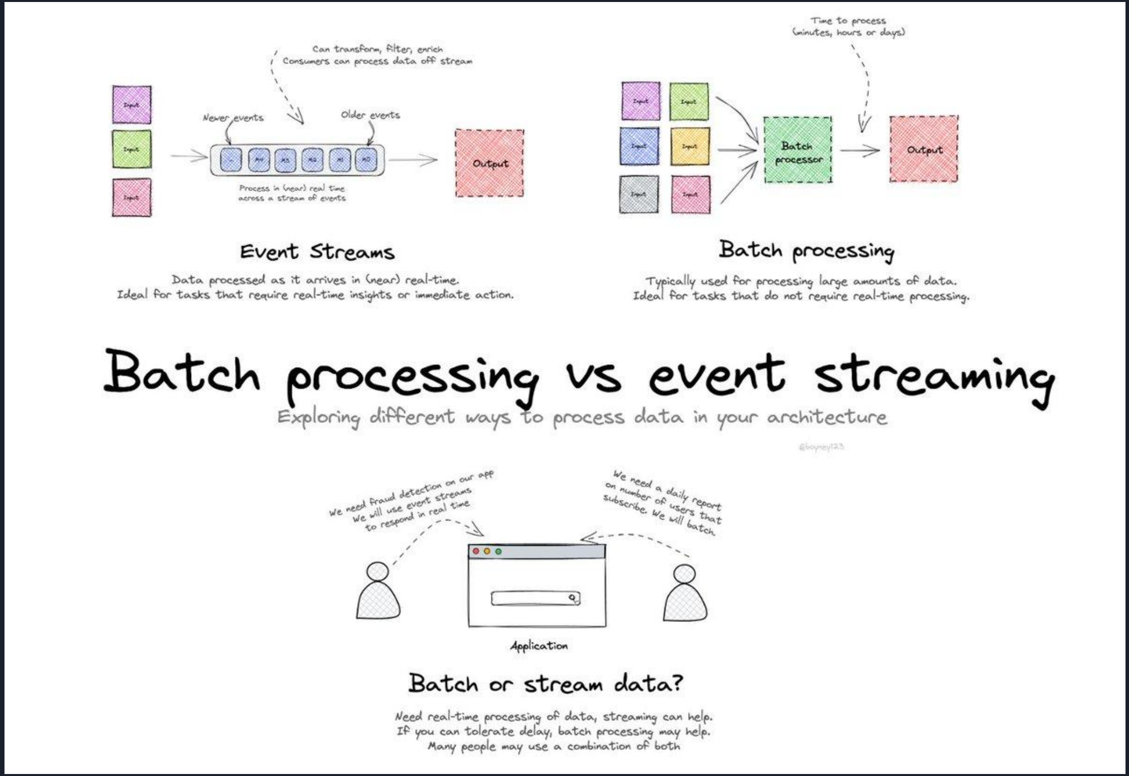
Data are collected and processed immediately and in real-time.

Design by Musili Adebayo

Image Credit - Microsoft Learn

Data Streaming

Batch vs Streaming



Event Streams

Data processed as it arrives in (near) real-time.
Ideal for tasks that require real-time insights or immediate action.

Batch processing

Typically used for processing large amounts of data.
Ideal for tasks that do not require real-time processing.

Batch processing vs event streaming

Exploring different ways to process data in your architecture

Batch or stream data?

Need real-time processing of data, streaming can help.
If you can tolerate delay, batch processing may help.
Many people may use a combination of both

Data Streaming

Use Cases

- IoT applications
- Near real time data analytics
- Product recommendation in real-time
- Event-based processes
- Fraud Detection
- Log Analysis
- Sensor data
- Database migration
- etc

Data Streaming

Technologies

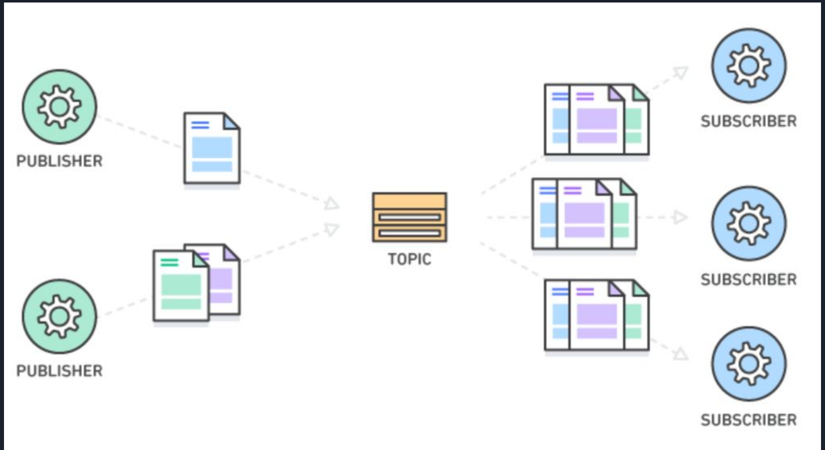
- Technologies/Solutions in market
  - Spark, Flink, Kafka, GCP Dataflow, AWS Kinesis, etc
- Open Source solutions
- Micro batching X real streaming

PubSub Architecture

Data Streaming

PubSub Architecture

- Publisher - Subscriber
- Asynchronous messaging
- Communication between different applications and services
- Scalable architecture (cluster, kubernetes)



- **Publisher** - Who creates the message
- **Subscriber** - Who consumes the data
- **Message or Event** - Data sent by publisher to subscriber (header, payload)
- **Topics** - Store the messages

Reference: https://aws.amazon.com/what-is/pub-sub-messaging/

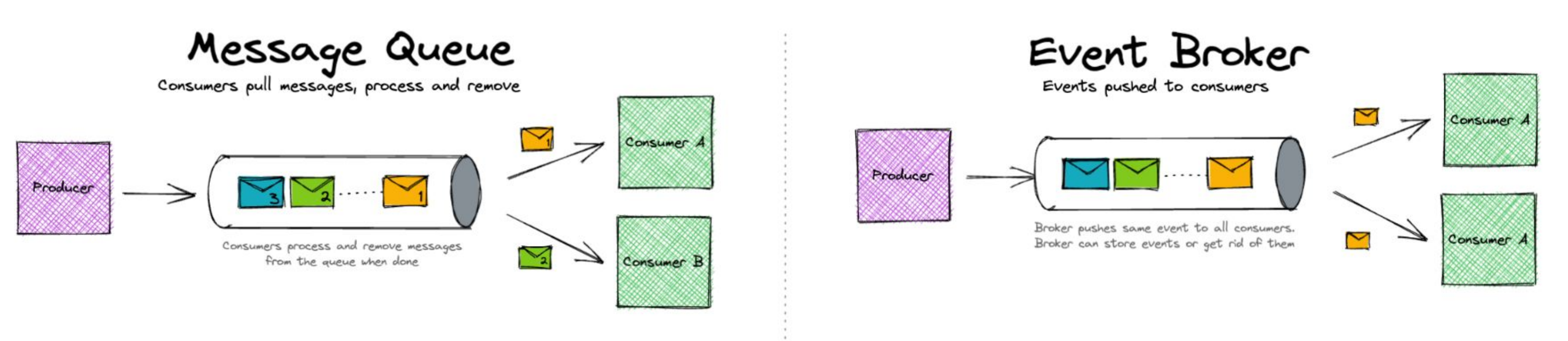
Data Streaming

PubSub x MessageQueue

- Technologies
  - Kafka, EventHub, RabbitMQ, EventHub
- Event broker (Kafka) vs Message Broker/Queue (RabbitMQ)
  - PubSub /Event broker: Communication 1:n (one publisher - many subscribers)
  - Message Queue - Communication 1:1 (one publisher - one subscriber)

Data Streaming

PubSub x MessageQueue


Message Queue
Consumers pull messages, process and remove
Producer
3
2
1
Consumer A
Consumer B
1
2
Consumers process and remove messages
from the queue when done
Event Broker
Events pushed to consumers
Producer
Consumer A
Consumer A
Broker pushes same event to all consumers.
Broker can store events or get rid of them

RabbitMQ

amazon SQS

kafka

amazon KINESIS

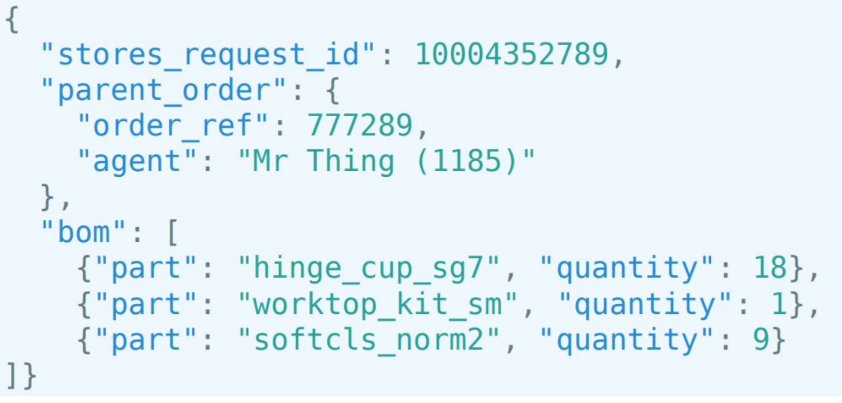
Google Cloud Pub/Sub

PULSAR

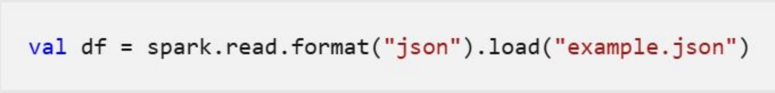
Data Streaming

JSON format

- JSON structure
  - collection of key/values pairs (dictionary)
  - collection of items (list/array)
- Spark reading from Json



```
{
  "stores_request_id": 10004352789,
  "parent_order": {
    "order_ref": 777289,
    "agent": "Mr Thing (1185)"
  },
  "bom": [
    {"part": "hinge_cup_sg7", "quantity": 18},
    {"part": "worktop_kit_sm", "quantity": 1},
    {"part": "softcls_norm2", "quantity": 9}
]}
```



```
val df = spark.read.format("json").load("example.json")
```

Reference:
https://spark.apache.org/docs/3.5.2/sql-data-sources-json.html

Spark Structure Streaming

Data Streaming

Spark Structure Streaming

- Spark Architecture
    - Batching
    - Streaming
- RDDs, DataFrames, parallel and distributed processing, in-memory, caching, etc

- Spark Structure Streaming
    - micro-batches
    - checkpoint
    - stateful operations
    - watermark
    - read & write -> readStream & writeStream
    - etc

Data Streaming

Spark Structure Streaming

Spark Streaming - old version
Spark Structured Streaming - new version

Data Streaming

Spark Structure Streaming

- Introduction
- Diff between spark batching and streaming
- readStream, writeStream, awaitTermination
- Output mode
  - Complete Mode
  - Append Mode
  - Update Mode
- Output format
  - file sink - format(“parquet”)
  - kafka sink - format(“kafka”)
  - foreach sink - foreach()
  - console sink - format(“console”)
  - memory - format(“memory”)

- Checkpoint
- Watermark
- Trigger interval
- Hands-on

Spark Structured Streaming :https://spark.apache.org/docs/3.5.2/structured-streaming-programming-guide.html

Data Streaming

Spark Structure Streaming

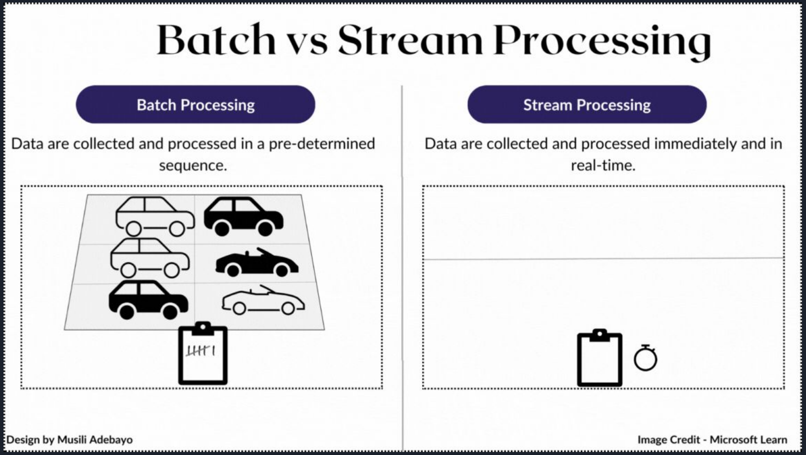
- Processing streaming data as a continuous series of **micro-batches**
- Allow DataFrame and SQL API for streaming transformations
- Fault-tolerant
- Exactly-once event processing

Data Streaming

Spark Structure Streaming

Micro-batches

- Small batches jobs that process the data streams
- Trigger Interval
- Latency in milliseconds



Batch vs Stream Processing

Batch Processing

Data are collected and processed in a pre-determined sequence.

Stream Processing

Data are collected and processed immediately and in real-time.

Design by Musili Adebayo

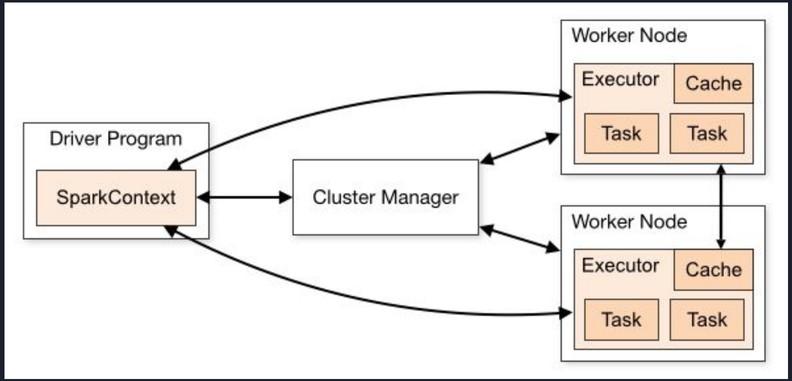
Image Credit - Microsoft Learn

Data Streaming

Spark Structure Streaming

Spark Streaming Flow (Spark Architecture)

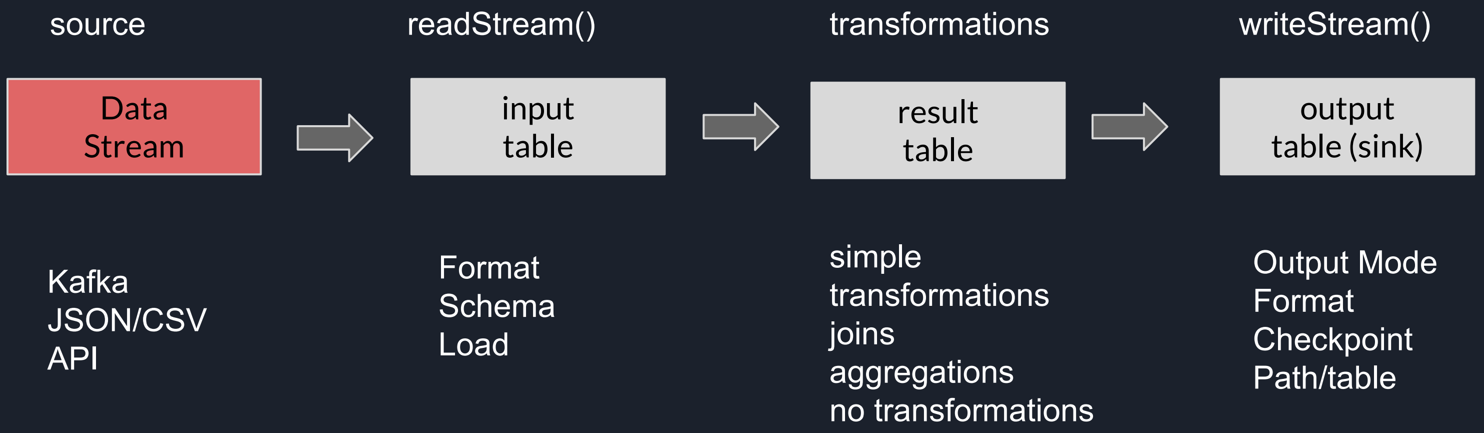
- Input source
- One or more receiver processes (consumer) that pull data from the input source
- Tasks that process the data
- Output sink
- A driver process that manages the long-running job



Data Streaming

Spark Structure Streaming

- End-to-End Flow



source

Data Stream

Kafka
JSON/CSV
API

readStream()

input table

Format
Schema
Load

transformations

result table

simple transformations
joins
aggregations
no transformations

writeStream()

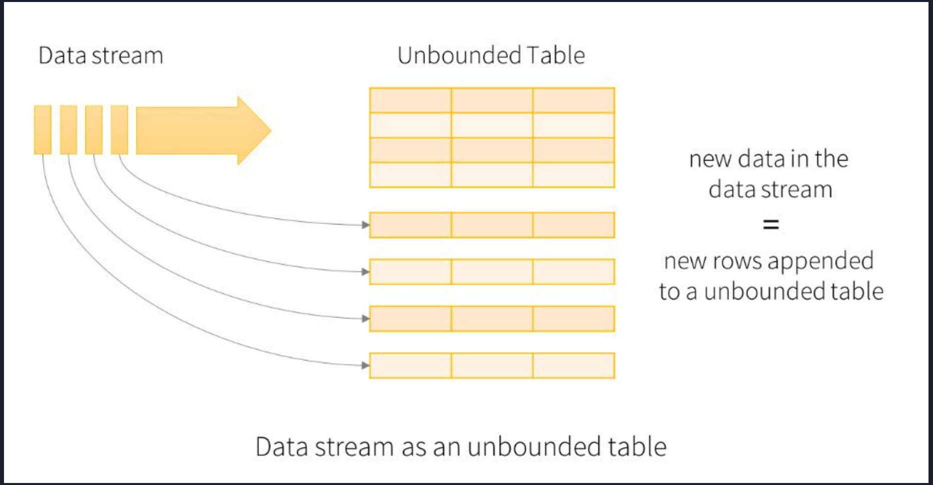
output table (sink)

Output Mode
Format
Checkpoint
Path/table

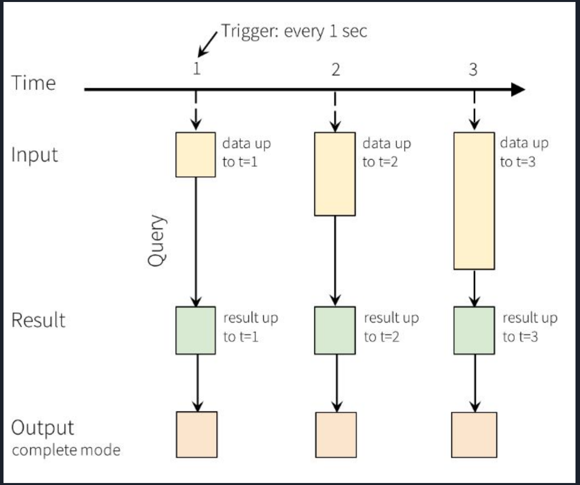
Data Streaming

Spark Structure Streaming

- Input table/Unbounded Table


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

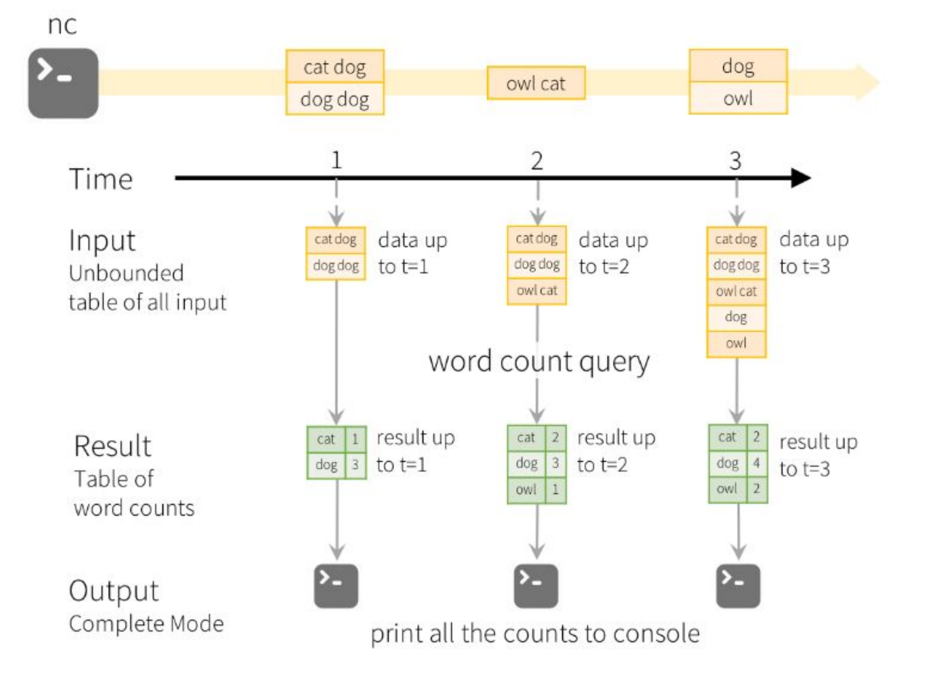
Data stream as an unbounded table



Spark Structured Streaming :https://spark.apache.org/docs/3.5.2/structured-streaming-programming-guide.html

Data Streaming

Spark Structure Streaming



Word Count example

- add animal as input
- "time" is the trigger interval (micro-batches)
- "Input" - Shows all input data
- "Result" - Apply transformations/query
- "Output" - Write into the console

Spark Structured Streaming :https://spark.apache.org/docs/3.5.2/structured-streaming-programming-guide.html

Data Streaming

Spark Structure Streaming

How to guarantee no data loss?

- **Fault-Tolerant storage**
- **Exactly-Once semantic -** Message is guarantee to be processed just once
- **Replayable source** - If some error occurs on Spark, the source must be able to send the message again
- **Reliable receivers/consumer** - Store state in fault-tolerant storage like HDFS or use Spark Streaming checkpoints
- **Write-Ahead Log** - First write received event into checkpoint
- **Checkpoint for the driver** - Store DAG state into fault-tolerant storage
- **Idempotent Sinks** - The target destination should be intelligent enough to handle possible data duplication and ignore it

Data Streaming

Spark Streaming

Reprocessing strategy

Reprocessing strategy	**Description**	**Characteristic**
At least once	Each message is guaranteed to be processed, but it may get processed more than once.	Possible duplication
At most once	Each message may or may not be processed. If a message is processed, it's only processed once.	Possible data lost
Exactly once	Each message is guaranteed to be processed once and only once.	No duplication, no data loss

Data Streaming

Spark Streaming

Stateful Operations

- State Store
  - Handle stateful operations across the micro-batches (stores state in aggregations)
- Stateful operations
  - aggregations
  - dropDuplicates
  - joins, stream-stream joins
  - mapGroupsWithState

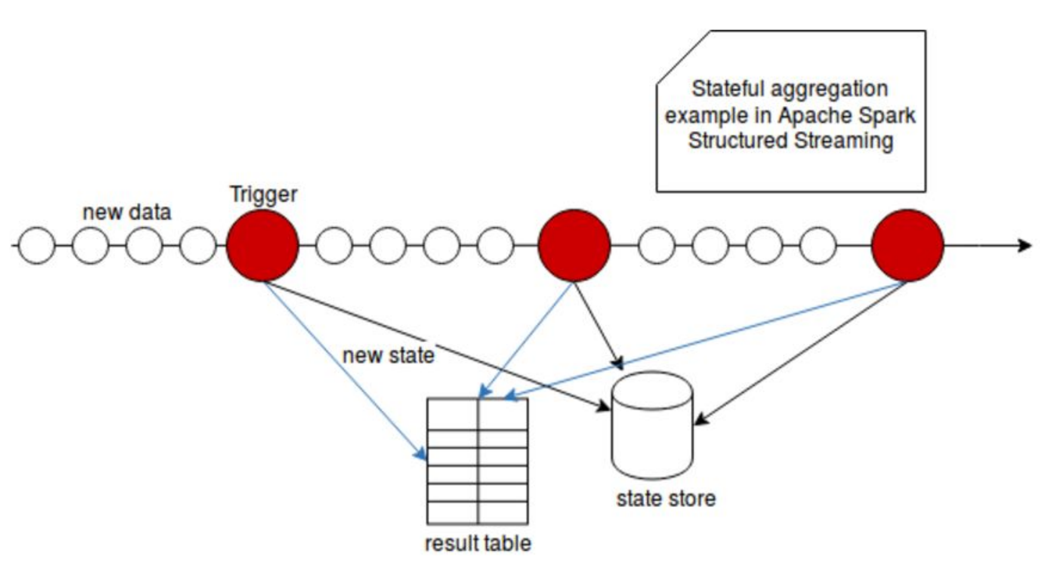
Where to store the state?

- HDFS state store provider - JVM memory
- RocksDB state store implementation - Optimized state manager (for bigger stateful operations)

Data Streaming

Spark Streaming

State Store


Stateful aggregation example in Apache Spark Structured Streaming
Trigger
new data
new state
result table
state store

- new events coming
- micro-batches
- state store

Data Streaming

Spark Structure Streaming

- readStream()
    - File source
    - Kafka source
    - Socket source (for testing)
    - Rate source (for testing)
    - Rate Per Micro-Batch source (for testing)

Spark Structured Streaming :https://spark.apache.org/docs/3.5.2/structured-streaming-programming-guide.html

Data Streaming

Spark Structure Streaming

- writeStream()
  - File source (parquet, csv, json)
  - Kafka source
  - Socket source (for testing)
  - Memory (for testing)
  - foreach, foreachBatch (custom logic)
  - Multiple Sources using foreachBatch

Spark Structured Streaming :https://spark.apache.org/docs/3.5.2/structured-streaming-programming-guide.html

Data Streaming

Spark Streaming

Output / Write Mode

- Append mode (default)
  - Only new rows added to the Result Table since the last trigger will be outputted to the sink. This is supported for only those queries where rows added to the Result Table is never going to change. Hence, this mode guarantees that each row will be output only once (assuming fault-tolerant sink). For example, queries with only select, where, map, flatMap, filter, join, etc. will support Append mode.
- Complete mode
  - The whole Result Table will be outputted to the sink after every trigger. This is supported for aggregation queries.
- Update mode
  - Only the rows in the Result Table that were updated since the last trigger will be outputted to the sink.

Reference: ttps://spark.apache.org/docs/3.5.1/structured-streaming-programming-guide.html#output-modes

Data Streaming

Spark Streaming

Trigger Interval

- Fixed interval micro-batches
- Sets the interval between micro-batches (default is 500ms)
- `.trigger(processingTime='10 seconds')`

Watermark

- Handle late data
- Threshold based on the event time column that defines how old the data can be to be considered as part of the process
- Events odlers than the watermark are deleted from the memory
- *.withWatermark("timestamp", "10 minutes")*

Data Streaming

Spark Streaming

Checkpoint

- Stores the last state of the data streaming
- Tracks the information that identifies the query, including:
    - state information
    - processed records
- If deleted, the next run will read the data from the beginning (possibly generating duplicates)
- .option('checkpointLocation', 'content/output/checkpoint')

Hands-On

Data Processing

Apache Spark

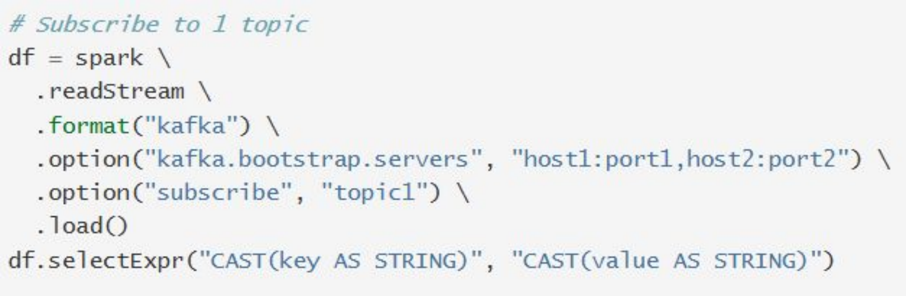
HANDS-ON

https://github.com/lucprosa/dataeng-basic-course/tree/main/spark_streaming/examples

Data Streaming

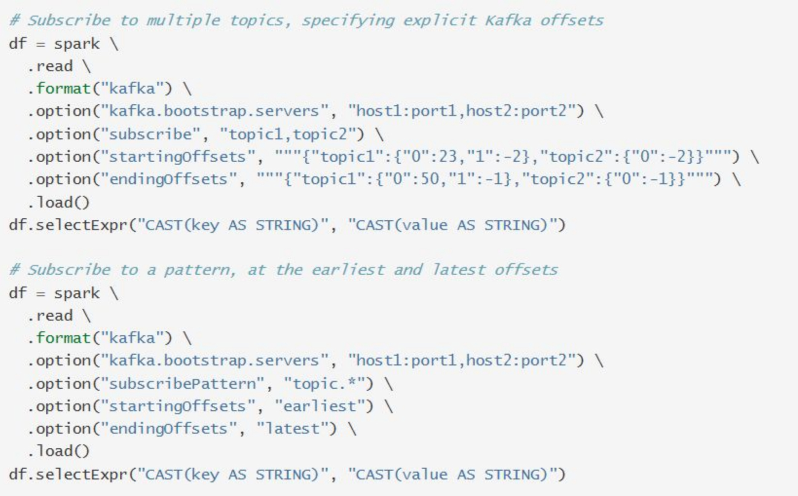
Spark Structure Streaming

How to read from Kafka?



```
# Subscribe to 1 topic
df = spark \
  .readStream \
  .format("kafka") \
  .option("kafka.bootstrap.servers", "host1:port1,host2:port2") \
  .option("subscribe", "topic1") \
  .load()
df.selectExpr("CAST(key AS STRING)", "CAST(value AS STRING)")
```

- Need to convert JSON values
- startingOffsets (earliest, latest)



```
# Subscribe to multiple topics, specifying explicit Kafka offsets
df = spark \
  .read \
  .format("kafka") \
  .option("kafka.bootstrap.servers", "host1:port1,host2:port2") \
  .option("subscribe", "topic1,topic2") \
  .option("startingOffsets", """{"topic1":{"0":23,"1":-2},"topic2":{"0":-2}}""") \
  .option("endingOffsets", """{"topic1":{"0":50,"1":-1},"topic2":{"0":-1}}""") \
  .load()
df.selectExpr("CAST(key AS STRING)", "CAST(value AS STRING)")

# Subscribe to a pattern, at the earliest and latest offsets
df = spark \
  .read \
  .format("kafka") \
  .option("kafka.bootstrap.servers", "host1:port1,host2:port2") \
  .option("subscribePattern", "topic.*") \
  .option("startingOffsets", "earliest") \
  .option("endingOffsets", "latest") \
  .load()
df.selectExpr("CAST(key AS STRING)", "CAST(value AS STRING)")
```

Spark Structured Streaming :https://spark.apache.org/docs/3.5.1/structured-streaming-kafka-integration.html

Data Engineering - Streaming