# Processing large datasets using Kafka and Spark Streaming

Danilo Bustos Pérez

Software Engineer @ Equifax

@dbustosp

# Agenda

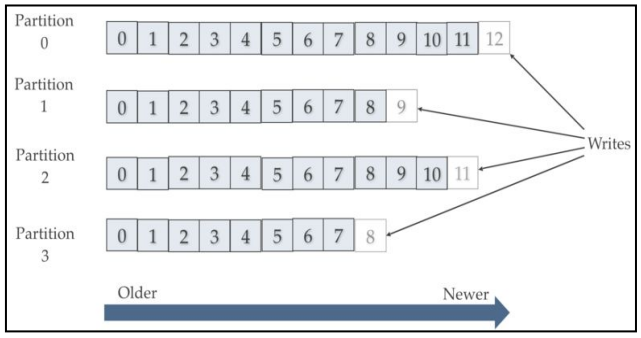
- Stream processing
- Apache Spark fundamentals
- Apache Kafka fundamentals
- Spark Streaming
- Common challenges

# Data Analytics in 2018

- Exponential data growth
  - Internet of things
  - Social Medias
  - Mobiles
  - Others
- Computationally challenge
  - Algorithms must be **time** and **memory** efficient
- Use cases are differents and much more ambitious
  - Running batch processes in a monthly, weekly and daily basis
  - Running Machine Learning algorithms over a huge amount of data
  - **Analytics in a streaming fashion**

# What is stream processing?

- Different programming paradigm that brings computation to **unbounded data**
- It enables users to query **continuous data stream** ⇒ (ms, s, m, h)
- Some key concepts
  - A **stream** represents an unbounded, continuously updating dataset
  - A stream/topic can consist of one or more **partitions**
  - A stream partition is an **ordered**, **replayable** and **fault-tolerant** sequence of immutable data records.
- Frameworks
  - Apache Flink
  - Apache Samza
  - Apache Storm
  - **Apache Spark**




# Spark streaming: Common use cases & Companies

- Use cases
  - Real-time dashboards (products shipped, etc.)
  - Real-time sentiment analysis from different social network
  - Real-time fraud detection (Card fraud or Site Tracking)
- Companies using Spark Streaming
  - Uber: Monitoring real time data
  - Pinterest: Getting insights around pins on real time
  - Netflix: Near real-time recommendations.

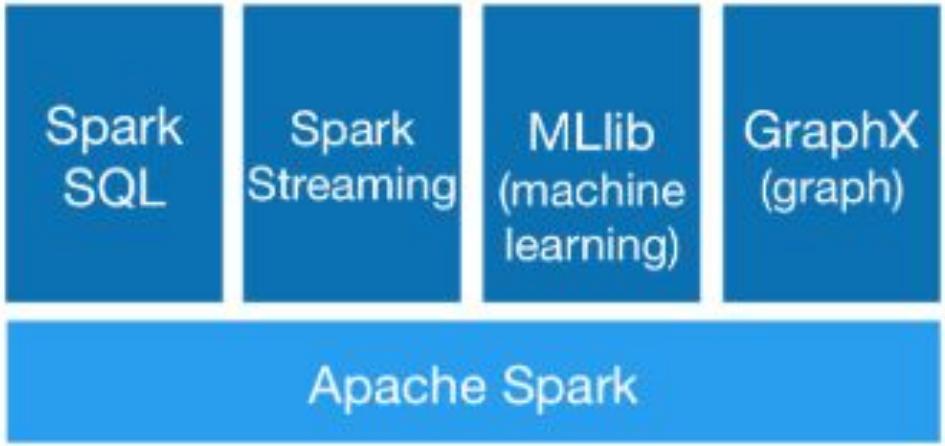
Uber Engineering

NETFLIX

EQUIFAX®

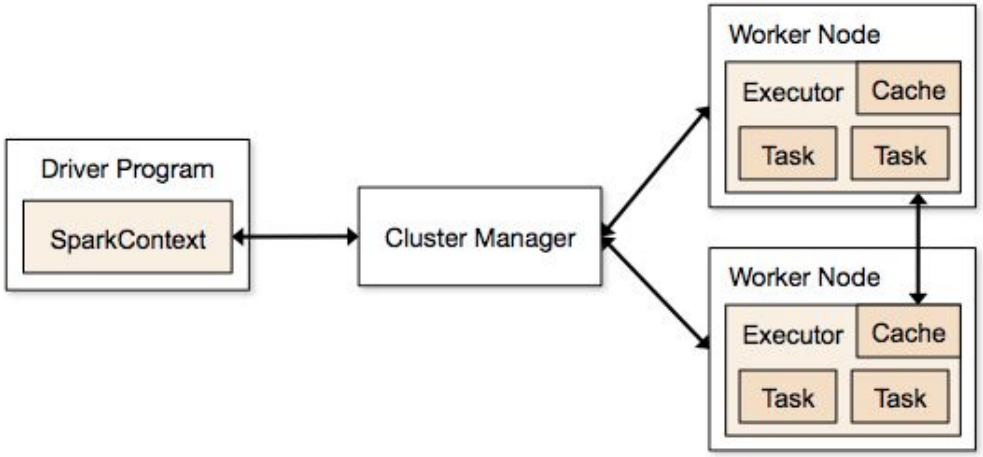
# Apache Spark

- **Open source** distributed computation Framework
- Much faster than Hadoop (MapReduce) ➔ 100x faster
- Ease of use ➔ Java, Scala, Python, R and SQL
- Runs on **Hadoop YARN**, Apache Mesos, Kubernets, Standalone or in the Cloud.
- Access data from different data sources.
- Combine differents libraries in the same application



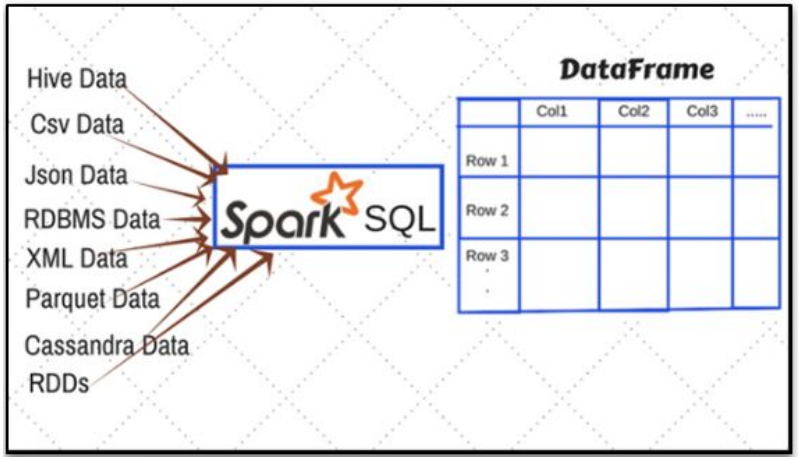

# Spark Fundamentals - Architecture

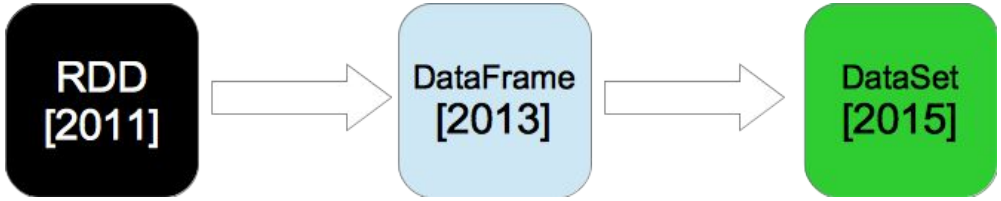
- Follows a **master/slave** architecture with a cluster manager
- Driver and Context resides on master daemon
- **Driver program** ⇒ Starting point and it creates the Spark job
- **Spark Context** ⇒ Entry point for Spark funcionalities



# Spark Fundamentals - Collections

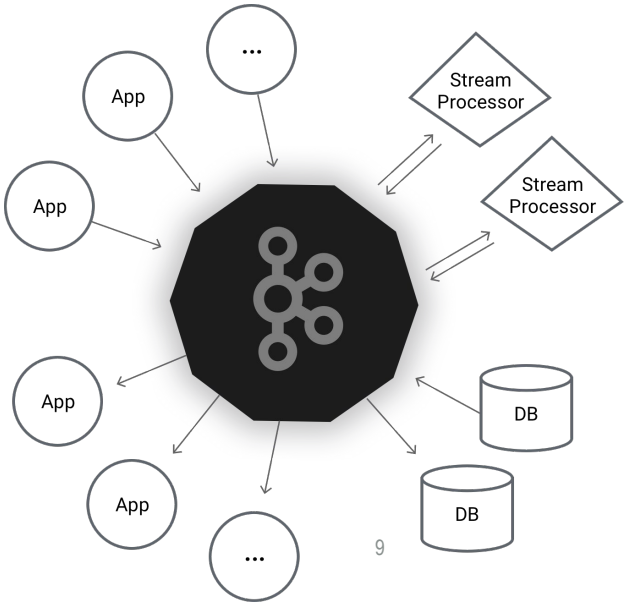
- RDD
  - Resilient Distributed Dataset
  - **Fault-tolerant**
- DataFrame
  - Dataset organized into named **columns**
- DataSet
  - Distributed collections of data
  - **RDD + SparkSQL**



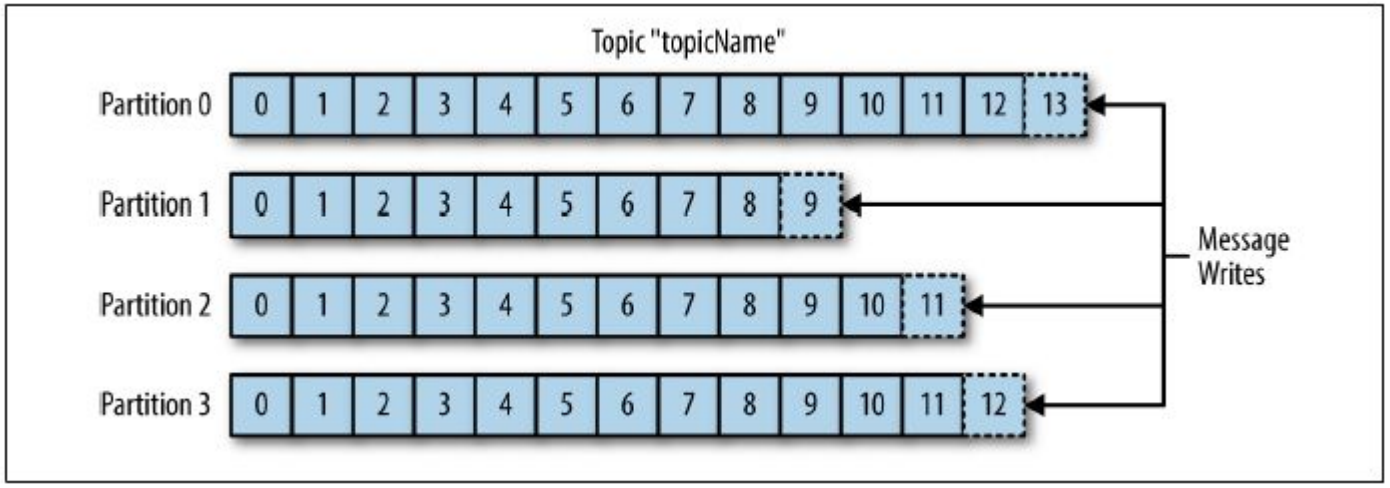
# Kafka

- Messaging system ⇒ Distributed streaming platform
  - Publish & Subscribe to a stream of records (message queue)
  - Store a streams of records in a fault-tolerant durable way
  - Process streams of records as they occur
- Why is Kafka good?
  - …
- Common use cases
  - Activity tracking (User profile)
  - Messaging (Sending emails)
  - Metrics and Logging
  - Stream processing (Twitter with Spark)



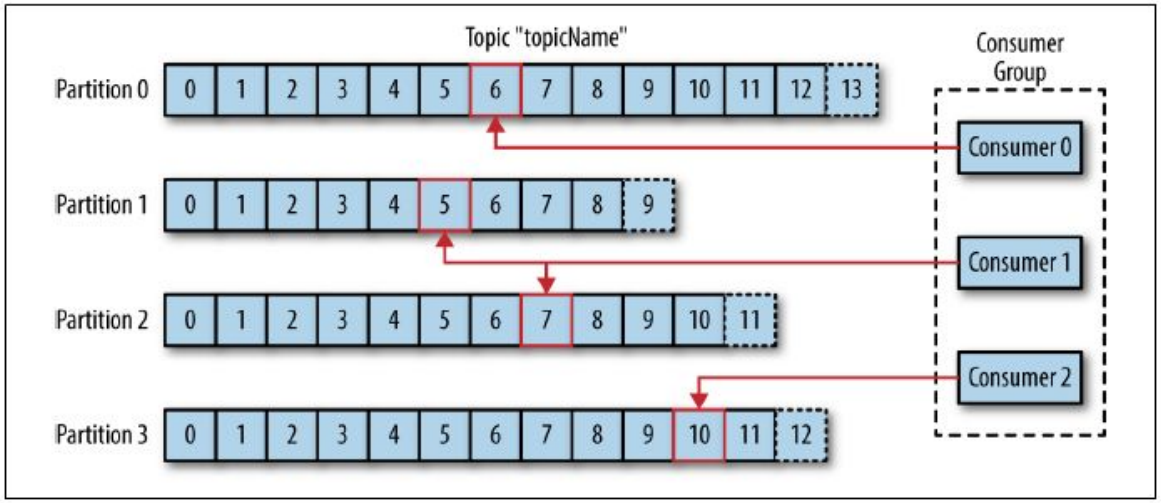
# Kafka Fundamentals: Topics & Partitions

- Messages are categorized into **topics**
  - Topics are additionally broken down into a number of partitions
  - Messages are written into an **append-only** fashion and are read from beginning to end
- Partitions are the way to provide **redundancy** and **scalabiliy**
  - Each partition can be hosted on a different server
  - Topic can be scaled horizontally across multiple servers



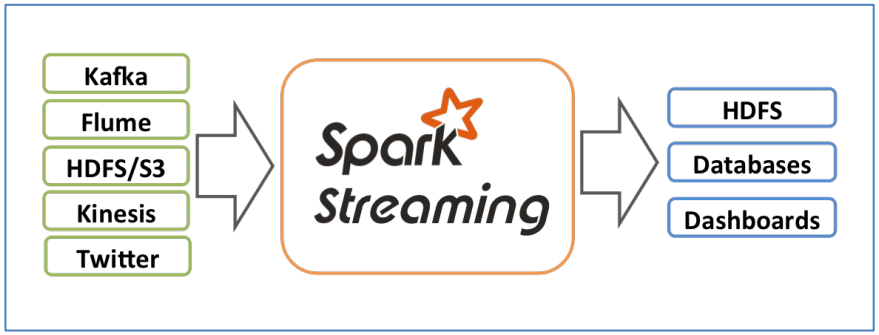
# Kafka Fundamentals: Consumer

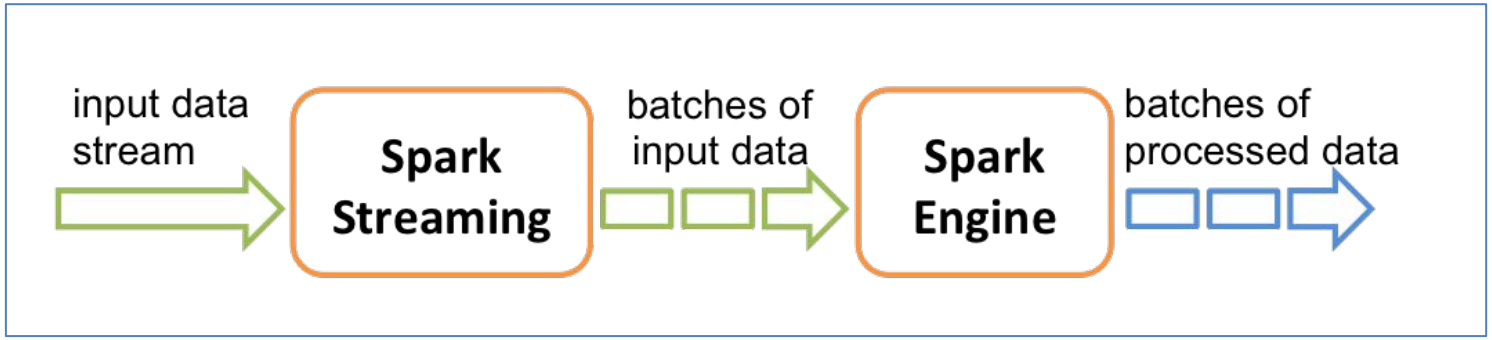
- Consumers
  - Subscribe 1+ topics and read messages in the same order were produced
  - Keep track of the messages already consumed (offset)
- Consumers work as part of a Consumer Group
  - This assures that each partition is only consumed by one member
  - This way Consumers can horizontally scale to read topics with large number of messages
  - Consumer fail, remain member of the group will rebalance the partitions being consumed



# Spark Streaming

- Spark API extension that enables scalable, high-throughput, fault-tolerant stream processing of live data streams.



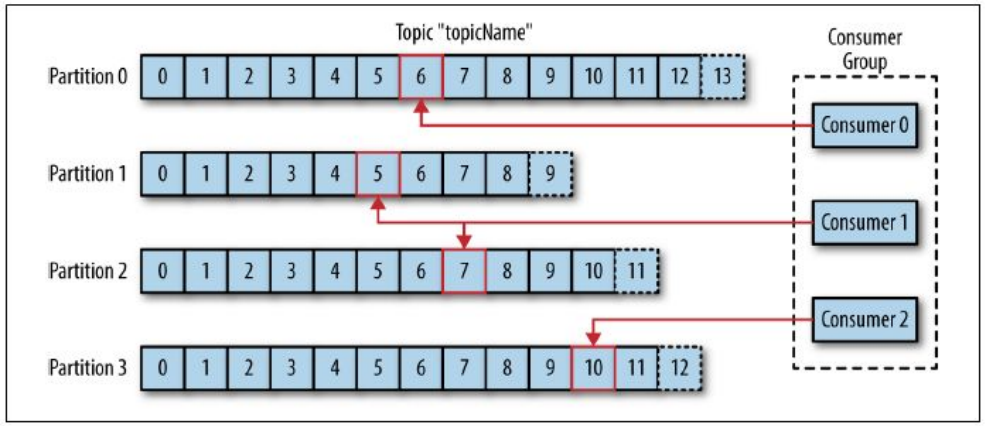
# Spark Streaming – Discretized Stream

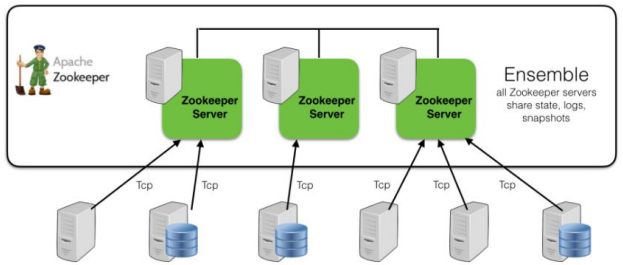
- Discretized streams (**DStream**)
  - Basic abstraction provided by Spark Streaming
  - Represent continuous stream of data
  - Created from Kafka, Flume and Kinesis.
- Internally a DStream is represented by a **continuous series of RDDs**
  - Containing data from a certain interval
- Operations on a Dstream translate to operations on the underlying RDDs, computed by the Spark engine.

```scala
val conf = new SparkConf()
  .setAppName("Spark Streaming Example")
  .setMaster("local[*]")

val sc = new SparkContext(conf)
val ssc = new StreamingContext(sc, Seconds(windowBatch))
val kafkaParams = getKafkaParams(config)

val dStream =
  KafkaUtils
    .createDirectStream[String, GenericRecord](ssc, PreferConsistent, Subscribe[String, GenericRecord](Array(topic), kafkaParams))

dStream.foreachRDD { rdd =>
  if (!rdd.isEmpty()) {
    val offsetRanges = rdd.asInstanceOf[HasOffsetRanges].offsetRanges
    println("off: " + offsetRanges)
  }
}
```

# Common challenges – Handling offsets

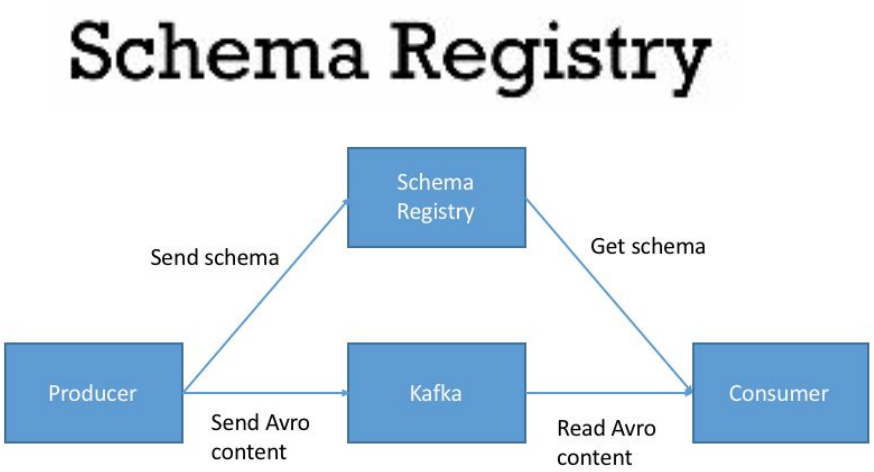
- is allowed to **loose data**? is allowed to have **data duplicated**?
- **Handling offsets**
  - Zookeeper
    - Coordination service used by a cluster, maintaining shared data with robust syncronization techniques.
  - HBase
    - Distribuided non-relational database




# Common challenges – Schema changes

- Schemas change over the time
- Producers/Consumers will must use the same Serialization/Deserialization classes
- Avro
  - It is a **data serialization framework**
  - **Backward** compatibility
- **Schema Registry**
  - Serving layer for metadata
  - It provides a RESTful interface for storing and retrieving Avro schemas

# Summary

- Casos de uso han ido evolucionando con el correr del tiempo
  - **Procesamiento Streaming**
- **Spark Streaming**
  - Internamente utiliza Spark + Kafka
- Existen muchas compañías con casos de uso exitosos ocupando el Stack de tecnología





Schema Registry

# Thank you!

Email

dfbustosp@gmail.com

Twitter/Stackoverflow/Github

@dbustosp