Pyspark

Sunil Miriyala

Table of Contents

[Environment 2](#_Toc47902447)

[IDE: 2](#_Toc47902448)

[Spark 2](#_Toc47902449)

[Python 2](#_Toc47902450)

[Terminal: 2](#_Toc47902451)

[Create New Virtual Environment 2](#_Toc47902452)

[Set Up Env 2](#_Toc47902453)

[Execute 3](#_Toc47902454)

[Standalone: 3](#_Toc47902455)

[Cloud-AWS-EMR 3](#_Toc47902456)

[Spark 3](#_Toc47902457)

[Overview: 3](#_Toc47902458)

[Stack 4](#_Toc47902459)

[Execution Architecture 4](#_Toc47902460)

[YARN: 5](#_Toc47902461)

[Driver: 5](#_Toc47902462)

[Executors: 5](#_Toc47902463)

[Streaming: 5](#_Toc47902464)

[DStream: 6](#_Toc47902465)

[Window Operations: 6](#_Toc47902466)

[Direct Streaming 7](#_Toc47902467)

[RDD (Resilient Distributed Dataset) 8](#_Toc47902468)

[Processing Semantics: 9](#_Toc47902469)

[Spark-SQL: 9](#_Toc47902470)

[Data Formats: 9](#_Toc47902471)

[AVRO 9](#_Toc47902472)

[ORC: 9](#_Toc47902473)

[PARQUET: 9](#_Toc47902474)

# Environment

## IDE:

* Anaconda & Spyder

A screenshot of a social media post

Description automatically generated

* Visual Studio Code
  + Install “Python” extension package
* PyCharm

## Spark

* <https://spark.apache.org/downloads.html>
* Version:
  + Spark Release: 3.0.0
  + Hadoop: 3.2

## Python

* <https://pypi.org/project/pyspark/>
* <https://spark.apache.org/docs/latest/api/python/index.html>

## Terminal:

### Create New Virtual Environment

* <https://docs.python.org/3/library/venv.html>
* pip install pyspark

### Set Up Env

* export MYDIR=<your-dir>/PySpark
* export SPARK\_HOME=$MYDIR/spark-3.0.0-bin-hadoop3.2
* export PATH=$PATH:$MYDIR/spark-3.0.0-bin-hadoop3.2/bin
* export PYTHONPATH=$SPARK\_HOME/python:$SPARK\_HOME/python/lib/py4j-0.10.4-src.zip:$PYTHONPATH
* export PATH=$SPARK\_HOME/python:$PATH

### Execute

* <your-dir>/PySpark/spark-3.0.0-bin-hadoop3.2/bin
* ./pyspark
  + *Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 16:52:21) ….*
  + *Using Python version 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019 16:52:21)*
  + *SparkSession available as 'spark'.*

## Standalone:

* <https://spark.apache.org/docs/latest/spark-standalone.html>
* Master:
  + <your-dir>/PySpark/spark-3.0.0-bin-hadoop3.2/sbin
  + sbin % ./start-master.sh
  + <http://localhost:8080/>
* Slave:
  + ./start-slave.sh localhost:8080
* Launch application
  + <your-dir>/PySpark/spark-3.0.0-bin-hadoop3.2/bin
  + ./spark-shell --master spark://localhost:8080

## Cloud-AWS-EMR

* <http://aws.amazon.com/console/>

# Spark

## Overview:

A high-performance large-scale data processing analytical engine that supports both batch and streaming data. It uses DAG schedulers, query optimizers and physical execution engines to achieve this performance.

It supports running on Hadoop, Kubernetes, Standalone and Cloud.

## Stack

A close up of a sign

Description automatically generated

## Execution Architecture

* 
* Cluster Managers:
  + Allocates resources for the application.
  + Local/Standalone
  + YARN
  + Mesos
  + Kubernetes(K8)
* Executors:
  + Processes that run the actual computations and data.
* Driver program create SparkContext.
* Sends application code to the executors
* SparkContext send tasks to the executors to run
* Isolates one application from another sharing the same set of executors. When application is created, context, it gets a set of executor processes. It stays for length of the execution and runs tasks in multiple threads.
* Driver program controls schedule of tasks on the cluster. It should coexist with the worker for better efficiency.

## YARN:

* Acts as resource allocator and Cluster Manager.
* Alternatives: Kubernetes, Apache Mesos.
* YARN-Cluster: Spark driver runs inside an application master process which is managed by yarn on the cluster. Client is released once app is initialized.
* YARN-Client: Drives runs in the client process and the application master is only used for requesting resources from yarn.
* YARN UI: For application monitoring and debugging performance issues:
* <https://databricks.com/blog/2015/07/08/new-visualizations-for-understanding-apache-spark-streaming-applications.html>

## Driver:

* Runs main programs
* Creates various parallel operations to perform on executors
* All aggregations runs on driver nodes like: collect(), take()

## Executors:

* All Parallel executions happens on executors
* Execution unit inside each function runs here.

## Streaming:

* An extension that supports scalable, high-throughput, fault-tolerant stream processing engine.





## DStream:

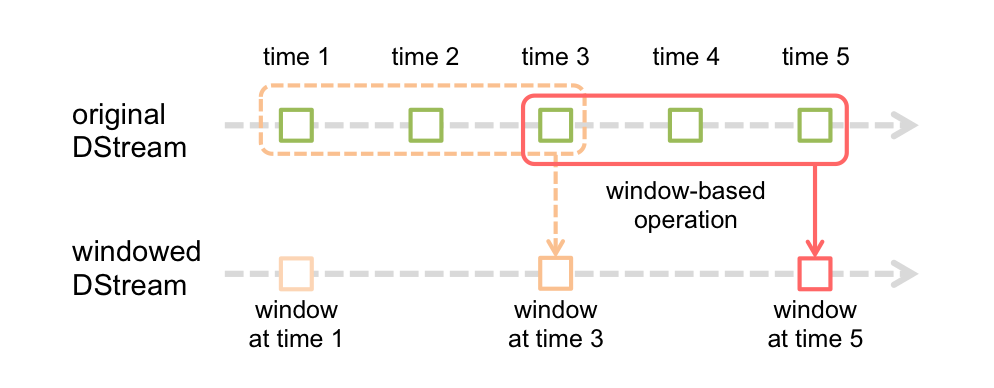
* Discrete Streams, a continuous stream of data. Also referred as sequence of RDDs.

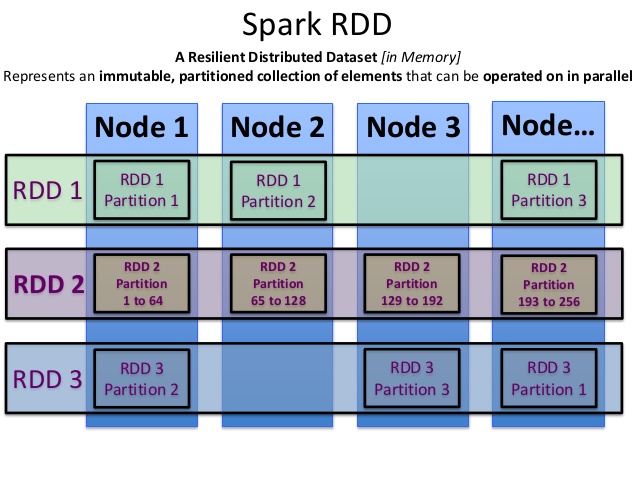


* Input DStreams are DStreams representing the stream of input data received from streaming sources like Kafka.
* Each RDD in a DStream contains data from a certain interval
* Any operation applied on a DStream translates to operations on the underlying RDDs, like split etc.
* Each RDD pushed into the queue will be treated as a batch of data in the DStream, and processed like a stream.
* Input Sources 🡪 DStream 🡪 Receiver 🡪 Spark In-Memory

## Window Operations:

* Every time the window slides over a source DStream, the source RDDs that fall within the window are combined and operated upon to produce the RDDs of the windowed DStream

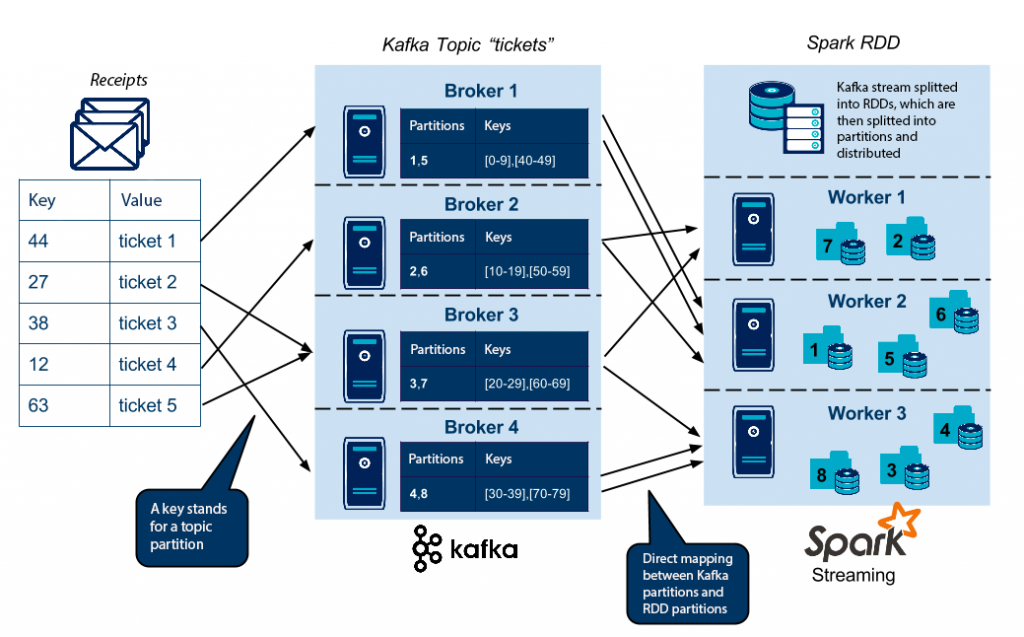




Direct Streaming:

A screenshot of a cell phone

Description automatically generated



## RDD (Resilient Distributed Dataset)

* An RDD is a fault-tolerant collection of elements that can be operated on in parallel.
* You can create them parallelizing an existing collection in your driver program, or referencing a dataset in an external storage system, such as a shared filesystem, HDFS, HBase, or any data source offering a Hadoop InputFormat

## Processing Semantics:

* At most once: Each record will be either processed once or not processed at all.
* At least once: Each record will be processed one or more times. This is stronger than at-most once as it ensure that no data will be lost. But there may be duplicates.
* Exactly once: Each record will be processed exactly once - no data will be lost and no data will be processed multiple times. This is obviously the strongest guarantee of the three

## Spark-SQL:

* Useful for querying structured data using DataFrame API or SQL syntax. Supports filtering, joins, grouping and more. Support HiveQL syntax, UDF’s and direct access to hive data stores.
* Uses internally Cost Based Optimization for executing queries. It uses columnar storage and code generation that assist in running performant queries. It will launch spark jobs and use the power of cluster and scale to high volume data queries.

A picture containing drawing, woman

Description automatically generated

## Data Formats:

### AVRO

* Avro, as a component, supports a rich set of primitive data types including: numeric, binary data and strings; and a number of complex types including arrays, maps, enumerations and records. A sort order can also be defined for the data.

### ORC:

### PARQUET: