Pyspark

Sunil Kumar Miriyala

Table of Contents

[Environment 2](#_Toc48001581)

[GIT 2](#_Toc48001582)

[IDE: 2](#_Toc48001583)

[Spark 2](#_Toc48001584)

[Python 2](#_Toc48001585)

[Terminal: 3](#_Toc48001586)

[Windows: 3](#_Toc48001587)

[Create New Virtual Environment 3](#_Toc48001588)

[Standalone: 3](#_Toc48001589)

[Cloud-AWS-EMR 4](#_Toc48001590)

[Spark 6](#_Toc48001591)

[Overview: 7](#_Toc48001592)

[Stack 7](#_Toc48001593)

[Execution Architecture 7](#_Toc48001594)

[YARN: 8](#_Toc48001595)

[Driver: 8](#_Toc48001596)

[Executors: 8](#_Toc48001597)

[Streaming: 8](#_Toc48001598)

[DStream: 9](#_Toc48001599)

[Window Operations: 9](#_Toc48001600)

[Direct Streaming 10](#_Toc48001601)

[RDD (Resilient Distributed Dataset) 11](#_Toc48001602)

[Processing Semantics: 12](#_Toc48001603)

[Spark-SQL: 12](#_Toc48001604)

[Data Formats: 12](#_Toc48001605)

[AVRO 12](#_Toc48001606)

[ORC 12](#_Toc48001607)

[PARQUET 12](#_Toc48001608)

# Environment

## GIT

<https://github.com/sunmiri/pyspark>

git clone https://github.com/sunmiri/pyspark.git

## IDE:

* PyCharm
* [https://www.jetbrains.com/pycharm/download/](https://www.jetbrains.com/pycharm/download/#section=windows)
* Install Following PlugIns:
  + Git

A screenshot of a computer screen

Description automatically generated

## Spark

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

## Python

* Windows:
  + PyCharm/Visual Studio Code/Spyder
  + RedHat/Centos Linux Sub System
* Mac
  + Visual Studio Code/PyCharm/Spyder
  + Python3
  + <https://pypi.org/project/pyspark/>
  + <https://spark.apache.org/docs/latest/api/python/index.html>

### Create New Virtual Environment

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

## Cloud-AWS-EMR

* <http://aws.amazon.com/console/>
* IAM -> Group
* IAM -> User
* IAM -> User -> Permissions
* EC2 -> KeyPair
* S3 -> Bucket
* Kinesis
* Redshift
* <https://aws.amazon.com/premiumsupport/knowledge-center/emr-pyspark-python-3x/>
* CLI: <https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-install.html>
  + Configure: <https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-quickstart.html>
* S3 🡪 Bucket -> Create a bucket
* A screenshot of a cell phone

  Description automatically generated
* A screenshot of a cell phone

  Description automatically generated
* EC2 -> Key Pair
* A screenshot of a cell phone

  Description automatically generated
* Key Pair Downloaded should be copied to ~/.ssh/
* AWS CLI Configure:
  + <https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-quickstart.html>
  + >aws configure
    - Access key id: <from iam -> users -> your -> security credentials>
    - Secret access key: <from iam -> users -> your -> security credentials>
    - Region: us-east-1
    - Output: json
* Aws Console Portal -> EMR - >
* A screenshot of a social media post

  Description automatically generated
* A screenshot of a cell phone

  Description automatically generated
* Kinesis: <https://docs.aws.amazon.com/streams/latest/dev/kinesis-using-sdk-java-create-stream.html>
* A screenshot of a cell phone

  Description automatically generated

# 

# 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:

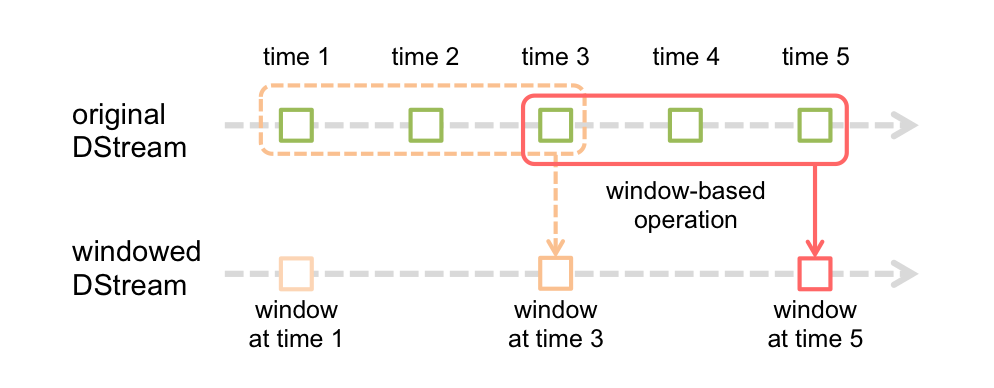
* Discrete Streams, a continuous stream of data. Also referred as sequence of RDDs.

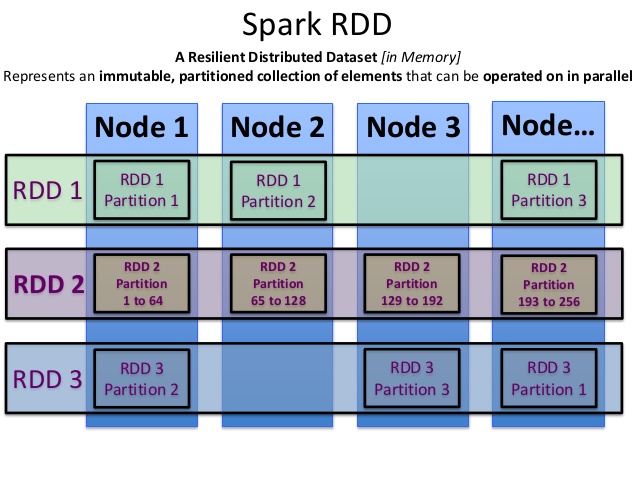


* Input DStreams are DStreams representing the stream of input data received from streaming sources like Kafka.
* Each RDD in a DStream contains data from a certain interval
* Any operation applied on a DStream translates to operations on the underlying RDDs, like split etc.
* Each RDD pushed into the 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

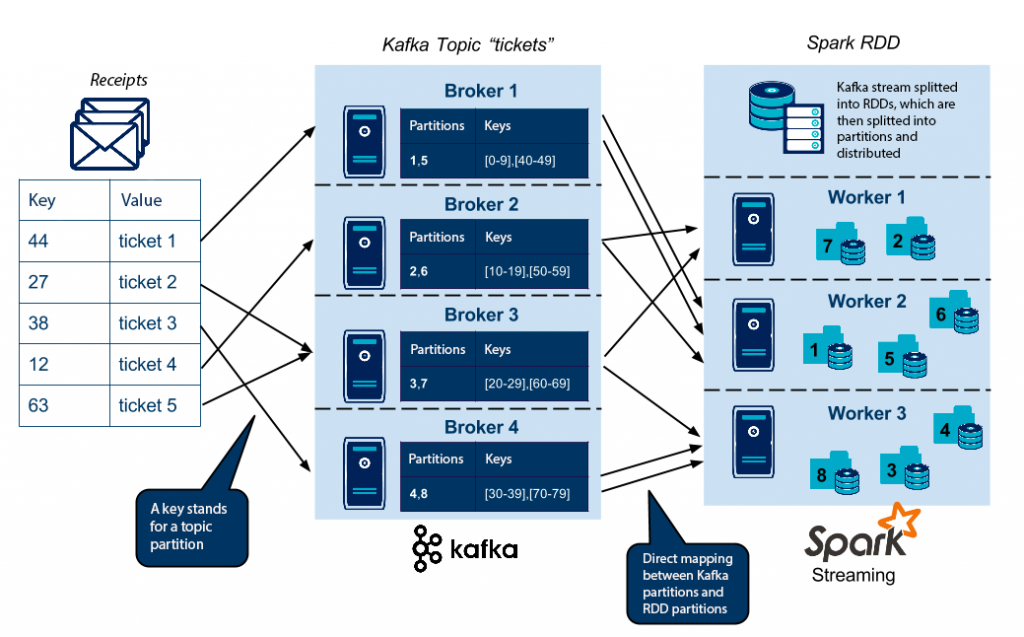




Direct Streaming:

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

### Kinesis Checkpointing

* Each Kinesis input DStream periodically stores the current position of the stream in the backing DynamoDB table. This allows the system to recover from failures and continue processing where the DStream left off.
* Checkpointing too frequently will cause excess load on the AWS checkpoint storage layer and may lead to AWS throttling. The provided example handles this throttling with a random-backoff-retry strategy.
* If no Kinesis checkpoint info exists when the input DStream starts, it will start either from the oldest record available (KinesisInitialPositions.TrimHorizon), or from the latest tip (KinesisInitialPositions.Latest), or (except Python) from the position denoted by the provided UTC timestamp (KinesisInitialPositions.AtTimestamp(Date timestamp)). This is configurable.
  + KinesisInitialPositions.Latest could lead to missed records if data is added to the stream while no input DStreams are running (and no checkpoint info is being stored).
  + KinesisInitialPositions.TrimHorizon may lead to duplicate processing of records where the impact is dependent on checkpoint frequency and processing idempotency.

### Kinesis retry configuration

* spark.streaming.kinesis.retry.waitTime : Wait time between Kinesis retries as a duration string. When reading from Amazon Kinesis, users may hit ProvisionedThroughputExceededException’s, when consuming faster than 5 transactions/second or, exceeding the maximum read rate of 2 MiB/second. This configuration can be tweaked to increase the sleep between fetches when a fetch fails to reduce these exceptions. Default is “100ms”.
* spark.streaming.kinesis.retry.maxAttempts : Max number of retries for Kinesis fetches. This config can also be used to tackle the Kinesis ProvisionedThroughputExceededException’s in scenarios mentioned above. It can be increased to have more number of retries for Kinesis reads. Default is 3.