PySpark is a Spark library written in Python to run Python application using Apache Spark capabilities, using PySpark we can run applications parallelly on the distributed cluster (multiple nodes).

In other words, PySpark is a Python API for Apache Spark.

Apache Spark is an analytical processing engine for large scale powerful

distributed data processing and machine learning applications.

Spark basically written in Scala and later on due to its industry adaptation

it’s API PySpark released for Python using Py4J. Py4J is a Java library that

is integrated within PySpark and allows python to dynamically interface with JVM objects, hence to run PySpark you also need Java to be installed along with Python, and Apache Spark.

features:

* In-memory computation
* Distributed processing using parallelize
* Can be used with many cluster managers (Spark, Yarn, Mesos e.t.c)
* Fault-tolerant
* Immutable
* Lazy evaluation
* Cache & persistence
* Inbuild-optimization when using DataFrames
* Supports ANSI SQL

Advantages of PySpark:

PySpark is a general-purpose, in-memory, distributed processing engine that allows you to process data efficiently in a distributed fashion.

Applications running on PySpark are 100x faster than traditional systems.

You will get great benefits using PySpark for data ingestion pipelines.

Using PySpark we can process data from Hadoop HDFS, AWS S3, and many file systems.

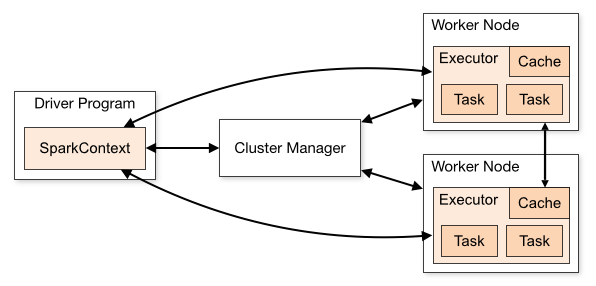
PySpark also is used to process real-time data using Streaming and Kafka.

Using PySpark streaming you can also stream files from the file system and also stream from the socket.

PySpark natively has machine learning and graph libraries.

PySpark Architecture:

Apache Spark works in a master-slave architecture where the master is called “Driver” and slaves are called “Workers”. When you run a Spark application, Spark Driver creates a context that is an entry point to your application, and all operations (transformations and actions) are executed on worker nodes, and the resources are managed by Cluster Manager.



Cluster Manager Types

As of writing this Spark with Python (PySpark) tutoriaSpark supports below cluster managers:

Standalone – a simple cluster manager included with Spark that makes it easy to set up a cluster.

Apache Mesos – Mesons is a Cluster manager that can also run Hadoop MapReduce and PySpark applications.

Hadoop YARN – the resource manager in Hadoop 2. This is mostly used, cluster manager.

Kubernetes – an open-source system for automating deployment, scaling, and management of containerized applications.

local – which is not really a cluster manager but still I wanted to mention as we use “local” for master() in order to run Spark on your laptop/computer.

Modules and Packages:

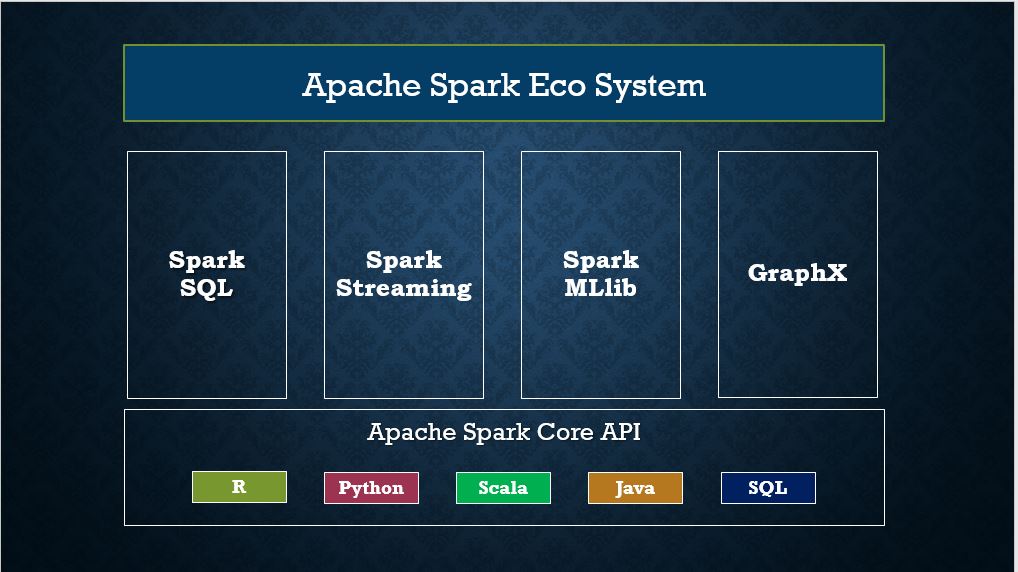
PySpark RDD (pyspark.RDD)

PySpark DataFrame and SQL (pyspark.sql)

PySpark Streaming (pyspark.streaming)

PySpark MLib (pyspark.ml, pyspark.mllib)

PySpark GraphFrames (GraphFrames)



**RDD Creation**

In order to create an RDD, first, you need to create a [SparkSession which is an entry point to the PySpark application](https://sparkbyexamples.com/pyspark/pyspark-what-is-sparksession/). SparkSession can be created using a builder() or newSession() methods of the SparkSession.

Spark session internally creates a sparkContext variable of SparkContext. You can create multiple SparkSession objects but only one SparkContext per JVM. In case if you want to create another new SparkContext you should stop existing Sparkcontext (using stop()) before creating a new one.

# Import SparkSession

from pyspark.sql import SparkSession

# Create SparkSession

spark = SparkSession.builder \

.master("local[1]") \

.appName("SparkByExamples.com") \

.getOrCreate()

using parallelize()

SparkContext has several functions to use with RDDs.

For example, it’s parallelize() method is used to create an RDD from a list.

# Create RDD from parallelize

dataList = [("Java", 20000), ("Python", 100000), ("Scala", 3000)]

rdd=spark.sparkContext.parallelize(dataList)

using textFile()

RDD can also be created from a text file using textFile() function of the SparkContext.

# Create RDD from external Data source

rdd2 = spark.sparkContext.textFile("/path/test.txt")

Once you have an RDD, you can perform transformation and action operations.

Any operation you perform on RDD runs in parallel.

RDD Operations:

On PySpark RDD, you can perform two kinds of operations.

RDD transformations – Transformations are lazy operations.

RDD actions – operations that trigger computation and return RDD values to the driver.

RDD Transformations

Transformations on Spark RDD returns another RDD and transformations are lazy meaning they don’t execute until you call an action on RDD. Some tansformations on RDD’s are flatMap(), map(), reduceByKey(), filter(), sortByKey() and return new RDD instead of updating the current.

RDD Actions:

RDD Action operation returns the values from an RDD to a driver node.

In other words, any RDD function that returns non RDD[T] is considered as an action.

Some actions on RDDs are count(), collect(), first(), max(), reduce() and more.

DataFrame is a distributed collection of data organized into named columns.

It is conceptually equivalent to a table in a relational database or a data frame in R/Python, but with richer optimizations under the hood.

DataFrames can be constructed from a wide array of sources such as structured data files, tables in Hive, external databases, or existing RDDs.

using createDataFrame():

By using createDataFrame() function of the SparkSession you can create a DataFrame.

data = [('James','','Smith','1991-04-01','M',3000),

('Michael','Rose','','2000-05-19','M',4000),

('Robert','','Williams','1978-09-05','M',4000),

('Maria','Anne','Jones','1967-12-01','F',4000),

('Jen','Mary','Brown','1980-02-17','F',-1)

]

columns = ["firstname","middlename","lastname","dob","gender","salary"]

df = spark.createDataFrame(data=data, schema = columns)

Since DataFrame’s are structure format which contains names and columns, we can get the schema of the DataFrame using df.printSchema()

df.show() shows the 20 elements from the DataFrame.

DataFrame from external data sources

In real-time applications, DataFrames are created from external sources like

files from the local system, HDFS, S3 Azure, HBase, MySQL table e.t.c.

Below is an example of how to read a CSV file from a local system.

df = spark.read.csv("/tmp/resources/zipcodes.csv")

df.printSchema()

In order to use SQL, first, create a temporary table on DataFrame using createOrReplaceTempView() function.

Once created, this table can be accessed throughout the SparkSession using sql() and it will be dropped along with your SparkContext termination.

Use sql() method of the SparkSession object to run the query and this method returns a new DataFrame.

df.createOrReplaceTempView("PERSON\_DATA")

df2 = spark.sql("SELECT \* from PERSON\_DATA")

df2.printSchema()

df2.show()

Let’s see another pyspark example using group by.

groupDF = spark.sql("SELECT gender, count(\*) from PERSON\_DATA group by gender")

groupDF.show()

rdd = spark.sparkContext.textFile("/user/itv001418/test.txt")

rdd.count()

flatMap – flatMap() transformation flattens the RDD after applying the function and returns a new RDD.

On the below example, first, it splits each record by space in an RDD and finally flattens it.

rdd2 = rdd.flatMap(lambda x: x.split(" "))

Resulting RDD consists of a single word on each record.

map – map() transformation is used the apply any complex operations like adding a column, updating a column e.t.c, the output of map transformations would always have the same number of records as input.

rdd3 = rdd2.map(lambda x: (x,1))

In our word count example, we are adding a new column with value 1 for each word, the result of the RDD is PairRDDFunctions which contains key-value pairs, word of type String as Key and 1 of type Int as value.

reduceByKey – reduceByKey() merges the values for each key with the function specified. In our example, it reduces the word string by applying the sum function on value. The result of our RDD contains unique words and their count.

rdd4 = rdd3.reduceByKey(lambda a,b: a+b)

sortByKey – sortByKey() transformation is used to sort RDD elements on key.

In our example, first, we convert RDD[(String,Int]) to RDD[(Int, String]) using map transformation and apply sortByKey which ideally does sort on an integer value.

And finally, foreach with println statements returns all words in RDD and their count as key-value pair

rdd5 = rdd4.map(lambda x: (x[1],x[0])).sortByKey()

#Print rdd5 result to console

print(rdd5.collect())

# Action - first

firstRec = rdd5.first()

print("First Record : "+str(firstRec[0]) + ","+ firstRec[1])

max() – Returns max record.

# Action - max

datMax = rdd5.max()

print("Max Record : "+str(datMax[0]) + ","+ datMax[1])

take() – Returns the record specified as an argument.

# Action - take

data3 = rdd5.take(3)

for f in data3:

print("data3 Key:"+ str(f[0]) +", Value:"+f[1])