What is PySpark?

Before we jump into the PySpark tutorial, first, let's understand what is PySpark and how it is related to Python? who uses PySpark and it's advantages.

Introduction

PySpark is a Spark library written in Python to run Python applications 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.

In real-time, PySpark has used a lot in the machine learning & Data scientists community; thanks to vast python machine learning libraries. Spark runs operations on billions and trillions of data on distributed clusters 100 times faster than the traditional python applications.

Who uses PySpark?

PySpark is very well used in Data Science and Machine Learning community as there are many widely used data science libraries written in Python including NumPy, TensorFlow. Also used due to its efficient processing of large datasets. PySpark has been used by many organizations like Walmart, Trivago, Sanofi, Runtastic, and many more.

Features

Following are the main features of PySpark.

- 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

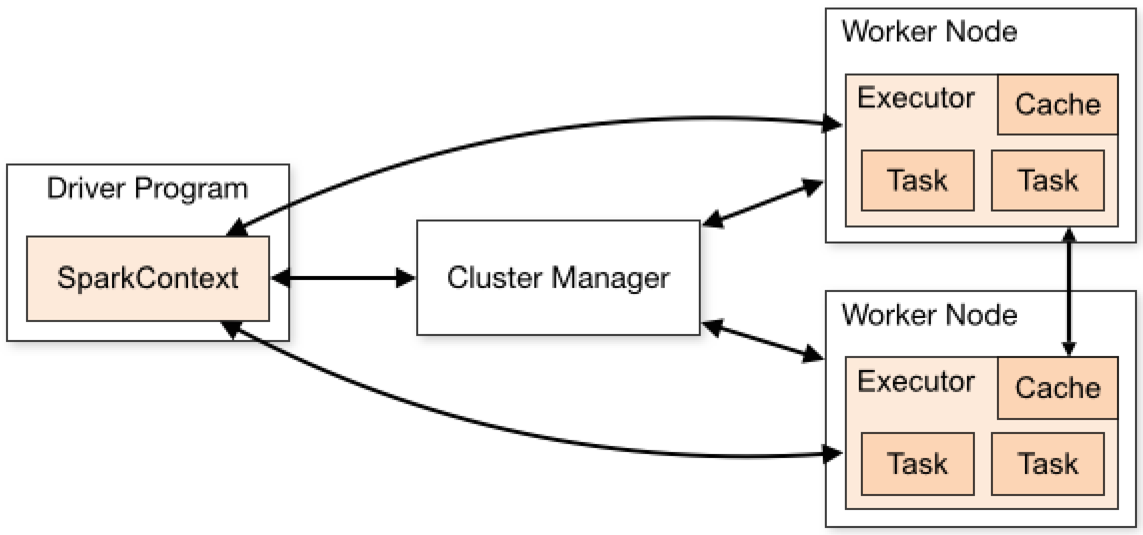
- Cache & persistence
- Inbuild-optimization when using DataFrames
- Supports ANSI SQL

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

Spark supports below cluster managers:

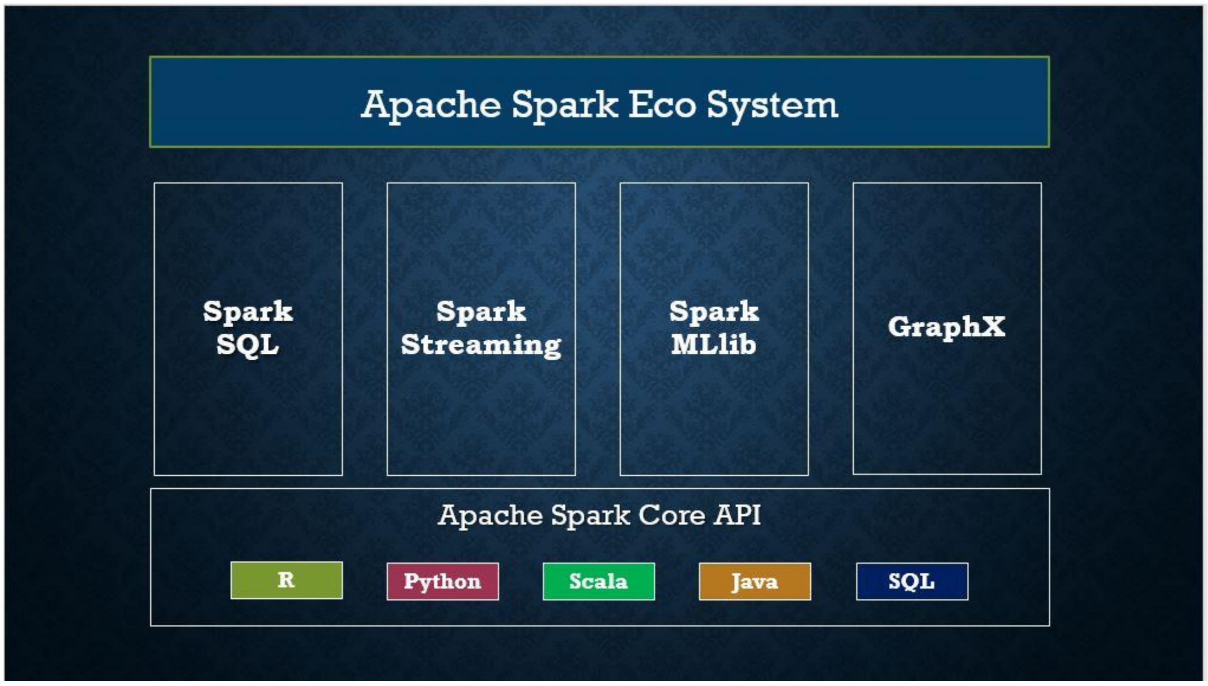
- Standalone – a simple cluster manager included with Spark that makes it easy to set up a cluster.

- 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.

PySpark Modules & Packages

- PySpark RDD (pyspark.RDD)
- PySpark DataFrame and SQL (pyspark.sql)
- PySpark Streaming (pyspark.streaming)
- PySpark MLib (pyspark.ml, pyspark.mllib)
- PySpark GraphFrames (GraphFrames)
- PySpark Resource (pyspark.resource) It's new in PySpark 3.0



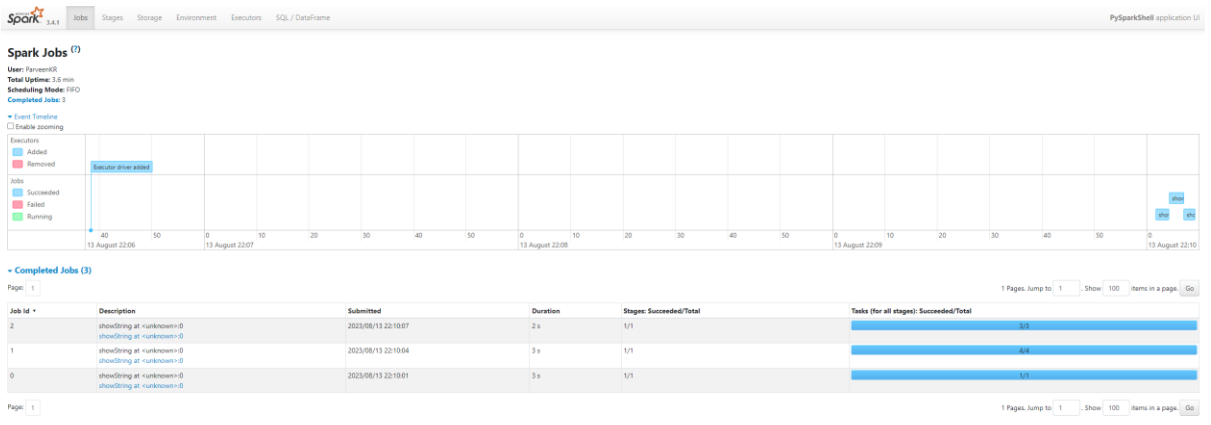
Besides these, if you wanted to use third-party libraries, you can find them at https://spark-packages.org/ . This page is kind of a repository of all Spark third-party libraries.

PySpark Installation

Follow the steps shared separately.

Spark Web UI

Apache Spark provides a suite of Web UIs (Jobs, Stages, Tasks, Storage, Environment, Executors, and SQL) to monitor the status of your Spark application, resource consumption of Spark cluster, and Spark configurations.



Spark Jobs (?)

User: ParveenKR
Total Uptime: 3.6 min
Scheduling Mode: FIFO
Completed Jobs: 3

Event Timeline
☐ Enable zooming

Completed Jobs (3)

Page: 1 — 1 Pages. Jump to 1 . Show 100 items in a page. Go

Job Id ▾	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
2	showString at <unknown>:0 showString at <unknown>:0	2023/08/13 22:10:07	2 s	1/1	3/3
1	showString at <unknown>:0 showString at <unknown>:0	2023/08/13 22:10:04	3 s	1/1	4/4
0	showString at <unknown>:0 showString at <unknown>:0	2023/08/13 22:10:01	3 s	1/1	1/1

Page: 1 — 1 Pages. Jump to 1 . Show 100 items in a page. Go

PySpark – What is SparkSession?

SparkSession was introduced in version 2.0, It is an entry point to underlying PySpark functionality in order to programmatically create PySpark RDD, DataFrame. It's object spark is default available in pyspark-shell and it can be created programmatically using SparkSession.

1. SparkSession

With Spark 2.0 a new class SparkSession (pyspark.sql import SparkSession) has been introduced. SparkSession is a combined class for all different contexts we used to have prior to 2.0 release (SQLContext and HiveContext e.t.c). Since 2.0 SparkSession can be used in replace with SQLContext, HiveContext, and other contexts defined prior to 2.0.

SparkSession also includes all the APIs available in different contexts –

SparkContext,

SQLContext,

StreamingContext,

HiveContext.

How many SparkSessions can you create in a PySpark application?

You can create as many SparkSession as you want in a PySpark application using either SparkSession.builder() or SparkSession.newSession(). Many Spark session objects are required when you wanted to keep PySpark tables (relational entities) logically separated.

2. SparkSession in PySpark shell

Be default PySpark shell provides "spark" object; which is an instance of SparkSession class. We can directly use this object where required in spark-shell. Start your "pyspark" shell from $SPARK_HOME\bin folder and enter the pyspark command.

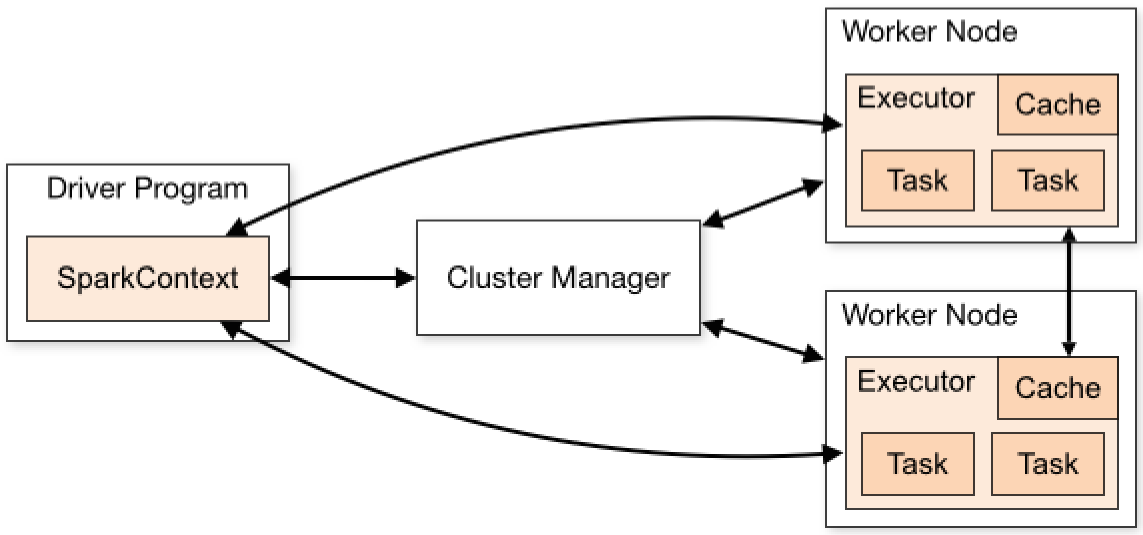
Similar to the PySpark shell, in most of the tools, the environment itself creates a default SparkSession object for us to use so you don't have to worry about creating a SparkSession object.

3. Create SparkSession

In order to create SparkSession programmatically (in .py file) in PySpark, you need to use the builder pattern method builder() as explained below. getOrCreate() method returns an already existing SparkSession; if not exists, it creates a new SparkSession.

PySpark SparkContext

pyspark.SparkContext is an entry point to the PySpark functionality that is used to communicate with the cluster and to create an RDD, accumulator, and broadcast variables. In this article, you will learn how to create PySpark SparkContext with examples. Note that you can create only one SparkContext per JVM, in order to create another first you need to stop the existing one using stop() method.



The Spark driver program creates and uses SparkContext to connect to the cluster manager to submit PySpark jobs, and know what resource manager (YARN, Mesos, or Standalone) to communicate to. It is the heart of the PySpark application.

SparkContext in PySpark shell

Be default PySpark shell creates and provides sc object, which is an instance of SparkContext class. We can directly use this object where required without the need of creating.

```
>>sc.appName
```

Similar to the PySpark shell, in most of the tools, notebooks, and Azure Databricks, the environment itself creates a default SparkContext object for us to use so you don't have to worry about creating a PySpark context.

PySpark RDD

What is RDD (Resilient Distributed Dataset)?

RDD (Resilient Distributed Dataset) is a fundamental building block of PySpark which is fault-tolerant, immutable distributed collections of objects. Immutable meaning once you create an RDD you cannot change it. Each record in RDD is divided into logical partitions, which can be computed on different nodes of the cluster.

In other words, RDDs are a collection of objects similar to list in Python, with the difference being RDD is computed on several processes scattered across multiple physical servers also called nodes in a cluster while a Python collection lives and process in just one process.

Additionally, RDDs provide data abstraction of partitioning and distribution of the data designed to run computations in parallel on several nodes, while doing transformations on RDD we don't have to worry about the parallelism as PySpark by default provides.

This Apache PySpark RDD tutorial describes the basic operations available on RDDs, such as map(), filter(), and persist() and many more. In addition, this tutorial also explains Pair RDD functions that operate on RDDs of key-value pairs such as groupByKey() and join() etc.

PySpark RDD Benefits

PySpark is widely adapted in Machine learning and Data science community due to it's advantages compared with traditional python programming.

In-Memory Processing

PySpark loads the data from disk and process in memory and keeps the data in memory, this is the main difference between PySpark and Mapreduce (I/O intensive). In between the transformations, we can also cache/persists the RDD in memory to reuse the previous computations.

Immutability

PySpark RDD's are immutable in nature meaning, once RDDs are created you cannot modify. When we apply transformations on RDD, PySpark creates a new RDD and maintains the RDD Lineage.

Fault Tolerance

PySpark operates on fault-tolerant data stores on HDFS, S3 e.t.c hence any RDD operation fails, it automatically reloads the data from other partitions. Also, When PySpark applications running on a cluster, PySpark task failures are automatically recovered for a certain number of times (as per the configuration) and finish the application seamlessly.

Lazy Evolution

PySpark does not evaluate the RDD transformations as they appear/encountered by Driver instead it keeps the all transformations as it encounters(DAG) and evaluates the all transformation when it sees the first RDD action.

Partitioning

When you create RDD from a data, It by default partitions the elements in a RDD. By default it partitions to the number of cores available.

PySpark RDD Limitations

PySpark RDDs are not much suitable for applications that make updates to the state store such as storage systems for a web application. For these applications, it is more efficient to use systems that perform traditional update logging and data checkpointing, such as databases. The goal of RDD is to provide an efficient programming model for batch analytics and leave these asynchronous applications.

Creating RDD

RDD's are created primarily in two different ways,

- parallelizing an existing collection and
- referencing a dataset in an external storage system (HDFS, S3 and many more).

first let's initialize SparkSession using the builder pattern method defined in SparkSession class. While initializing, we need to provide the master and application name as shown below. In realtime application, you will pass master from spark-submit instead of hardcoding on Spark application.

```
from pyspark.sql import SparkSession

spark:SparkSession = SparkSession.builder()

    .master("local[1]")

    .appName("RDD Example")

    .getOrCreate()
```

master() – If you are running it on the cluster you need to use your master name as an argument to master(). usually, it would be either yarn (Yet Another Resource Negotiator) or mesos depends on your cluster setup.

Use local[x] when running in Standalone mode. x should be an integer value and should be greater than 0; this represents how many partitions it should create when using RDD, DataFrame, and Dataset. Ideally, x value should be the number of CPU cores you have.

appName() – Used to set your application name.

getOrCreate() – This returns a SparkSession object if already exists, and creates a new one if not exist.

Note: Creating SparkSession object, internally creates one SparkContext per JVM.