(Py)Spark Basics

[(https://github.com/learn-co-curriculum/dsc-spark-basics)](https://github.com/learn-co-curriculum/dsc-spark-basics) [(https://github.com/learn-co-curriculum/dsc-spark-basics/issues/new)](https://github.com/learn-co-curriculum/dsc-spark-basics/issues/new)

Introduction

Before we begin writing PySpark code, let's go over some more of the concepts that underpin Apache Spark.

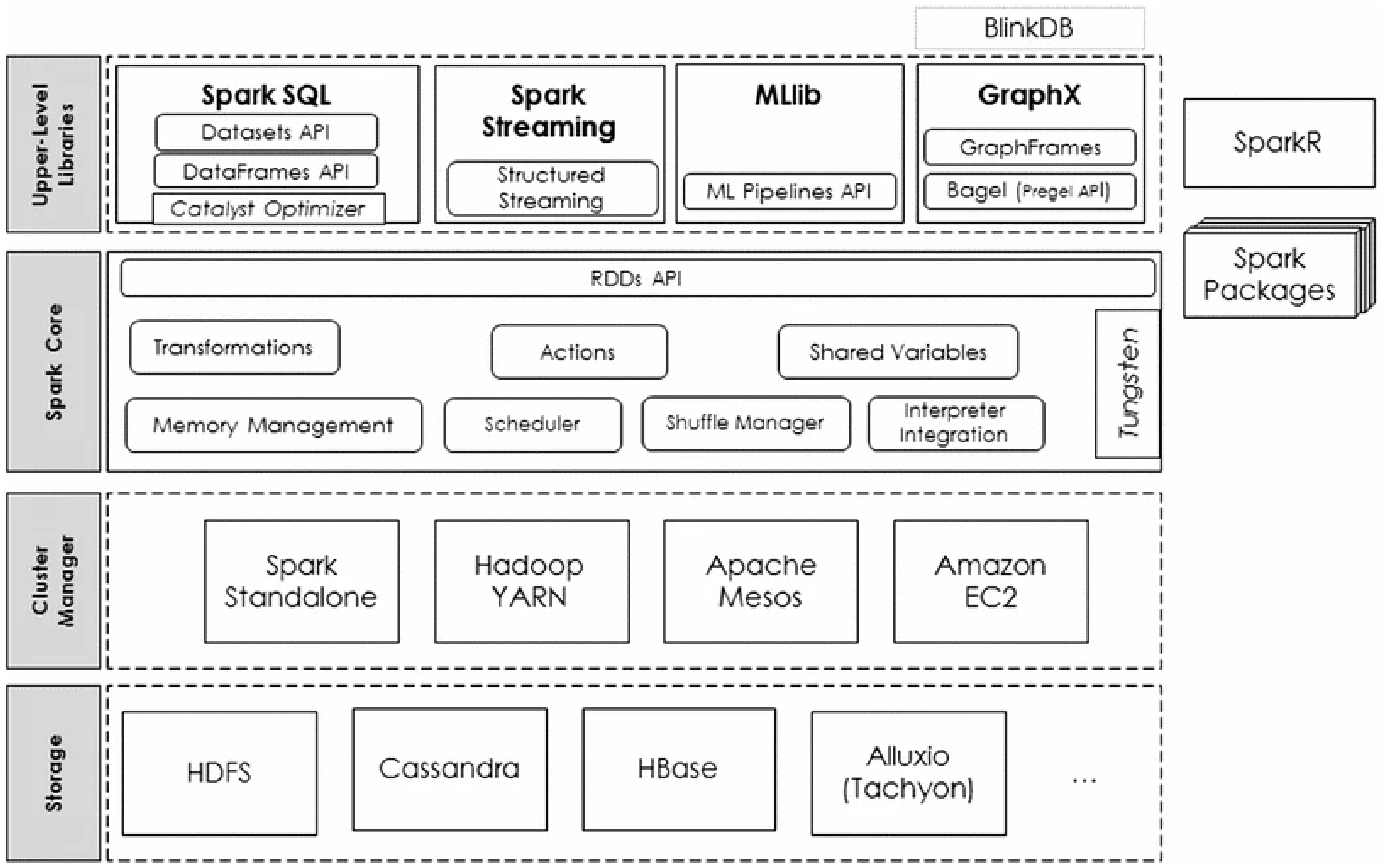
Objectives

You will be able to:

- Describe the high-level architecture of Apache Spark
- Describe the driver, worker, and executor in the context of Spark's parallelism
- Describe the data structures used by Apache Spark and PySpark in particular
- List use cases for Spark

Spark Architecture

The high-level architecture of the Apache Spark stack looks like this:



(Figure from ***Big data analytics on Apache Spark*** ***(https://link.springer.com/article/10.1007/s41060-016-0027-9)***)

We'll start at the bottom and work our way up.

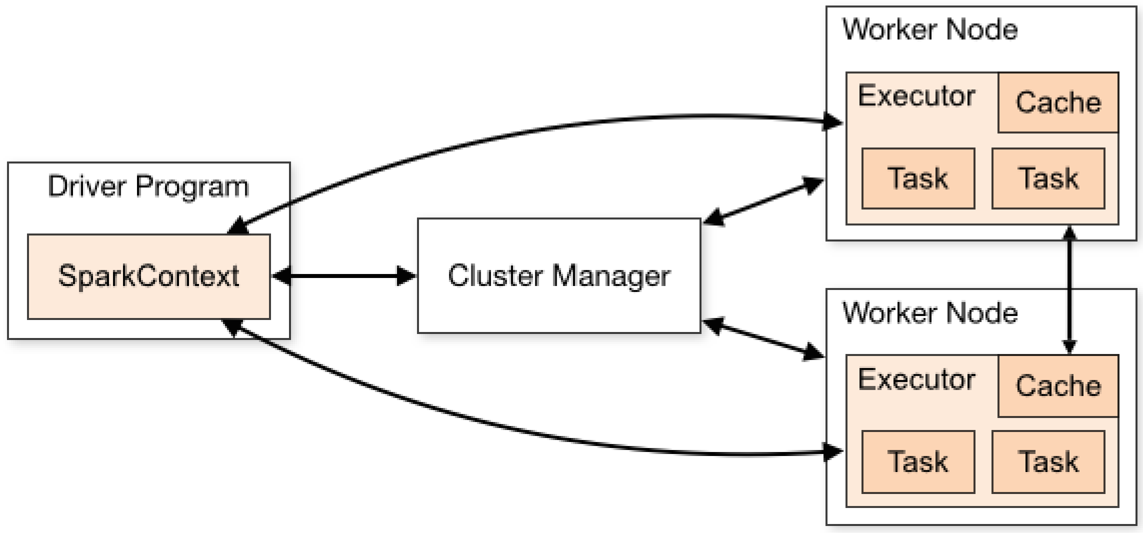
Storage

We won't focus too much on the specifics here, since they are applicable to all sorts of distributed computing systems. The main thing to be aware of is that production-grade Big Data stacks require specialized file systems.

Some storage options that are compatible with Spark are:

- **HDFS (https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html)**
- **Cassandra (https://cassandra.apache.org/_/index.html)**
- **HBase (https://hbase.apache.org/)**
- **Alluxio (https://www.alluxio.io/)**

Cluster Manager



(Figure from **Cluster Mode Overview** (https://spark.apache.org/docs/latest/cluster-overview.html))

As mentioned previously, Big Data tools typically rely on distributed and parallel computing. This is implemented in the Apache Spark stack using a cluster manager.

The main takeaway here should be a basic familiarity with the terminology.

A ***cluster*** is a group of interconnected computers used for distributed and parallel computing. A ***cluster manager*** manages those machines by allocating resources and connecting the driver program and worker nodes. A ***driver*** program maintains information about your application, responds to external programs, and analyzes, distributes, and schedules work across worker nodes. ***Worker*** nodes contain ***executor*** processes that execute the code assigned by the driver.

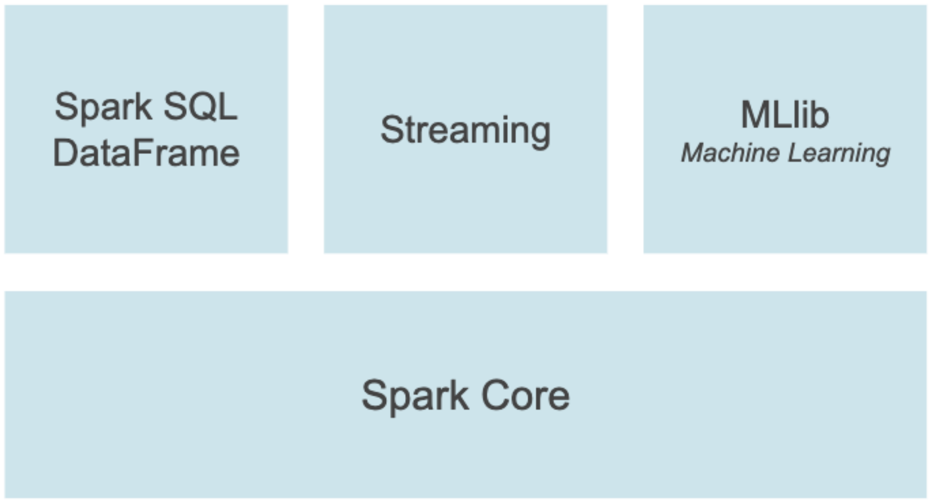
Here are links to some cluster manager options:

- **Hadoop YARN** (https://hadoop.apache.org/docs/stable/hadoop-yarn/hadoop-yarn-site/YARN.html)
- **Apache Mesos** (https://mesos.apache.org/)
- **Amazon EC2** (https://aws.amazon.com/ec2/)
- **Kubernetes** (https://spark.apache.org/docs/latest/running-on-kubernetes.html)

A Note About The Spark Curriculum

Because the curriculum lessons and labs are smaller, proof-of-concept applications of Spark, we will **not** be using a special distributed file storage system like HDFS or a full-fledged cluster manager like YARN. Instead, we will use **Spark Standalone** (https://spark.apache.org/docs/latest/spark-standalone.html) with a local cluster.

Typically a data scientist or data engineer would not be responsible for managing a cluster. In fact, you can refer to the **PySpark documentation** (https://spark.apache.org/docs/latest/api/python/) , which contains a version of the Spark architecture diagram that doesn't even include the storage and cluster manager layers. Instead it just focuses on the Spark Core and upper-level libraries:



Spark Core (Unstructured API)

Advantages Over MapReduce

The Spark Core is where Spark's advantages over MapReduce appear. To quote from ***Big data analytics on Apache Spark*** ***(https://link.springer.com/article/10.1007/s41060-016-0027-9)*** (emphasis added):

> Apache Spark has emerged as the de facto standard for big data analytics after Hadoop's MapReduce. As a framework, it combines a core engine for distributed computing with an advanced programming model for in-memory processing. Although it has the same linear scalability and fault tolerance capabilities as those of MapReduce, it comes with a multistage in-memory programming model comparing to the rigid map-then-reduce disk-based model. With such an advanced model, Apache Spark is much faster and easier to use.
>
> ***Apache Spark leverages the memory of a computing cluster to reduce the dependency on the underlying distributed file system, leading to dramatic performance gains in comparison with Hadoop's MapReduce.***

Recall the difference between data or models *in memory* (e.g. data stored in a Python variable) vs. *on disk* (e.g. a CSV or pickled model file). Almost all of the data work we do in this curriculum is in memory, since this is much faster and more flexible than performing all of the IO operations needed to save everything to disk. Spark uses this same approach.

You can read more about the specific performance gains made by Spark compared to MapReduce **here** **(https://research.ijcaonline.org/volume113/number1/pxc3900531.pdf)** .

Unstructured API

Functionality within the Spark Core is also referred to as the "Unstructured API".

> Note: "API" doesn't necessarily mean an HTTP API accessed over the internet -- in this case it just means the interface of classes and functions that your code can invoke.

The Unstructured API is the older, lower-level interface.

> Note: "lower-level" is literally true in the case of the figure shown at the top of this lesson, but it also generally means that a tool is closer to the underlying machine code executing on a computer. That means that it is usually more configurable than a higher-level tool, but also that it tends to be more difficult to use and is possibly not optimized for specific use cases.

It includes some constructs that resemble MapReduce constructs, such as Accumulators and Broadcast variables, as well as SparkContext and Resilient Distributed Datasets (RDDs). You can find the full PySpark Unstructured API documentation **here** **(https://spark.apache.org/docs/latest/api/python/reference/pyspark.html)** .

SparkContext

SparkContext is the entry point for using the Unstructured API. You'll notice it is inside the "Driver Program" rectangle in the cluster manager figure above. We will cover more details of how SparkContext is used with PySpark in a future lesson. You can also read more from the PySpark documentation **here** **(https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.SparkContext.html)** .

Resilient Distributed Datasets (RDDs)

Resilient Distributed Datasets (RDDs) are the fundamental data structure used by the Spark Core and accessible via the Unstructured API. Once again, we will cover more details in a future lesson, and you can read more from the PySpark documentation **here** **(https://spark.apache.org/docs/latest/api/python/reference/pyspark.html#rdd-apis)** .

Upper-Level Libraries (Structured API)

The upper-level libraries, also known as the Structured API, is where Spark gets really exciting. They are higher-level, easier to use, and optimized for particular tasks.

For data analysis and manipulation, the Structured API offers Spark SQL, a `pandas` API, and Spark Streaming. For machine learning the Structured API offers MLlib.

Spark SQL

Spark SQL has data structures called DataFrame and Dataset.

A Spark SQL ***DataFrame*** is similar to a `pandas` DataFrame in that it keeps track of column names and types, which improves efficiency and makes the data easier to work with. It is not the same as the DataFrame used in the `pandas` API, although it is possible to convert between them if necessary. You can find more documentation **here (https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame)** .

A Spark SQL ***Dataset*** works similar to a DataFrame except it has an additional Row construct. Datasets are not usable in PySpark (only in Scala and Java) at this time, although you may see references to them in the main Spark documentation.

Rather than a SparkContext like is used for the Unstructured API, the entry point to Spark SQL is a ***SparkSession***. You can find more documentation **here (https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql.html)** .

Pandas API

The `pandas` API allows you to use familiar `pandas` class and function names, with the power of Spark! **The PySpark maintainers recommend (https://spark.apache.org/docs/latest/api/python/user_guide/pandas_on_spark/faq.html#should-i-use-pyspark-s-dataframe-api-or-pandas-api-on-spark)** that anyone who already knows how to use `pandas` uses this API. You can find the API reference **here (https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/index.html)** and user guide **here (https://spark.apache.org/docs/latest/api/python/user_guide/pandas_on_spark/index.html)** .

Spark Streaming

Streaming data is outside the scope of this curriculum, but it's useful to know that Spark has functionality for it. You can find the PySpark documentation for Spark Streaming **here (https://spark.apache.org/docs/latest/api/python/reference/pyspark.streaming.html)** .

MLlib

MLlib allows you to perform many of the same machine learning tasks as scikit-learn, including transforming data, building and evaluating supervised and unsupervised machine learning models, and even building pipelines. There is also an Alternating Least Squares (ALS) implementation, which we will apply to a recommender system!

You can find the PySpark documentation for MLlib **here (https://spark.apache.org/docs/latest/api/python/reference/pyspark.ml.html)** .

Additional Resources

- **Big data analytics on Apache Spark (https://link.springer.com/article/10.1007/s41060-016-0027-9)** (2016) is an excellent review article. It should take 90-120 minutes to read, and we highly encourage you to take the time if you're interested in using Spark.
- **Intro to Apache Spark (http://stanford.edu/%7Erezab/sparkclass/slides/itas_workshop.pdf)** (2014) is a 194-slide presentation that goes into more detail about Spark with many code examples. Note: it appears that links in the slide deck starting with `cdn.liber118.com` are no longer working, but the GitHub links are still functional.

Summary

At a high level, Spark's architecture consists of:

- Storage
- Cluster Manager
- Spark Core (Unstructured API)
- Upper-Level Libraries (Structured API)

The Cluster Manager divides and shares the physical resources of a cluster of machines, utilizing a driver program that specifies tasks for executors within worker nodes.

The Spark Core (Unstructured API) is accessed using SparkContext, and utilizes the RDD data structure.

The upper-level libraries (Structured API) include code for specific use cases, including data analysis and manipulation (Spark SQL, `pandas` API, Spark Streaming) and machine learning (MLlib). Spark SQL is accessed using SparkSession and introduces two additional data structures (DataFrame and Dataset).

Now that we've covered the concepts, let's dive into some specific implementations!