

A man with dark hair, wearing a grey button-down shirt, is pointing his right hand towards a wall covered with various charts and graphs. He is holding a tablet in his left hand. The background is slightly blurred, showing more charts and a window.

# Module 01 – Spark Architecture

Copy

# Table of Contents

Don't forget to start WebEx recording  
Do Spark Jeopardy review

Go to: [community.cloud.databricks.com](https://community.cloud.databricks.com) and Logon  
In Left-pane, Click on 'Clusters' or 'Compute' and Terminate old Cluster  
Then click 'Create Cluster' button to create New one

## Session 1-2

**Mod 00** – Intro and Setup

**Mod 01** – Spark Architecture

**Mod 02** – SparkSQL (Read/Write DataFrames/Tables)

**Mod 03** – SparkSQL (Transform) **Hack 00 (Date) / Hack 01 (Air)**

## Session 3-4

**Mod 04** – Complex Data Types

**Hackathon 02 (Fly)**

**Mod 05** – JSON (Optional)

**Mod 06** – Streaming

**Hackathon 03 (Stream)**

**Mod 07** – Architecture-Spark UI

## Session 5-7

**Mod 08** – Catalog-Catalyst-Tungsten

**Mod 09** – Adaptive Query Execution

**Mod 10** – Performance Tuning

**Hackathon 04 (Air)**

**Mod 11** – Machine Learning

**Final Exam**

# Module 01 – Spark Architecture

---

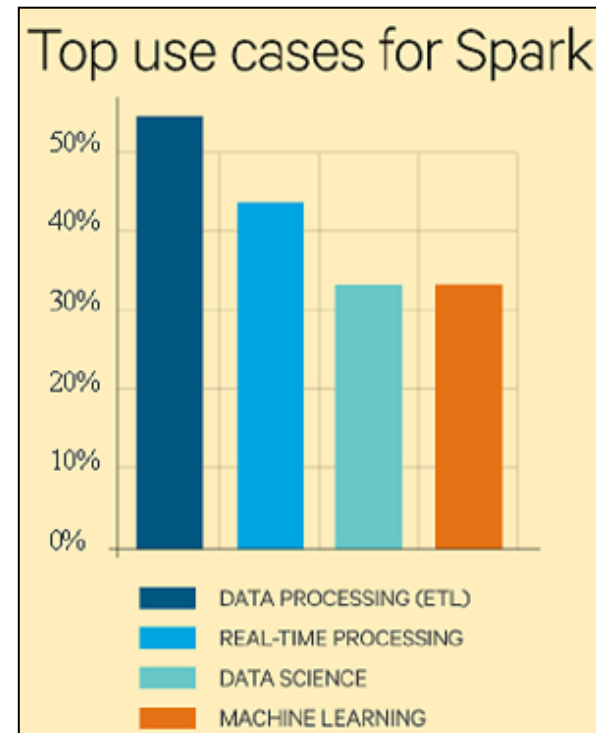
**After completing this module, you should be able name core Spark components and their functions including:**

- What is Apache Spark?
- Spark with Hadoop
- Logging into Spark
- Spark architecture - The Big Picture
  - Programming languages, Unified Stack, Cluster Manager, Data, BI tools
- Resilient Distributed Datasets (RDD)
- Spark cluster components
  - Driver, Spark Context, YARN, Worker nodes and Executors, Files
- Glossary of Spark terminology

# What is Apache Spark?



- Is an Open-Source fast and general Processing engine for large-scale data processing done **in-memory**
- Can run both inside or outside of Hadoop
- Spark does not have its own file system. So must source from local file system, HDFS, HBase, Cassandra, etc
- Supports many application languages including Java, Scala, Python, R, SQL
- Supports various workloads including batch applications, iterative algorithms, visualizations and interactive queries along with Streaming
- Not best choice for certain types of queries (OLTP and small datasets)



# Spark versus Hadoop – Spark with Hadoop

For developers, there is almost no overlap between the two. Hadoop is a framework in which you write MapReduce job by inheriting Java classes. Spark is a library that enables parallel computation via function calls

- What Hadoop gives Spark

- YARN Resource Management
- Distributed File System
- Disaster Recover capabilities
- Data Security (ie: Knox, Ranger)
- Distributed Data Platform

- What Spark give Hadoop

- Machine Learning
- Caching
- Streaming
- Graph processing
- In-memory processing

- Spark can operate independent of Hadoop. Spark does not require: Hadoop, Map Reduce, HDFS, YARN/Tez, Hive, HBase, Mahout, Impala
- However, in many instances, Spark depends on Hadoop's HDFS for input data and can optionally use YARN Resource Manager

# Before we Begin: Open Notebook **Mod-01**

- From Databricks Community image, click on:  
[Workspace](#) > [Shared](#) > [Mod-01-Basics](#)

## Mod 01: Spark Basics

Cmd 3

### Lab 01: Using 'dbutils' command to view Files

Cmd 4

```
1 # Before we begin, confirm all files are loaded
2 # Should have 56 rows if you loaded everything correctly
3 display(dbutils.fs.ls("dbfs:/FileStore/tables"))
```

► (3) Spark Jobs

	path	name	size
1	dbfs:/FileStore/tables/LifeExp.csv	LifeExp.csv	4376
2	dbfs:/FileStore/tables/LifeExp_headers.csv	LifeExp_headers.csv	4400
3	dbfs:/FileStore/tables/advert.csv	advert.csv	10926
4	dbfs:/FileStore/tables/auto.parquet	auto.parquet	17796354
5	dbfs:/FileStore/tables/auto1.parquet	auto1.parquet	1177
6	dbfs:/FileStore/tables/autos.csv	autos.csv	68439217

# Lab 01: Using 'dbutils' to view Files

First, let's confirm we can view the Files we uploaded earlier as well as delete a file using the 'dbutils' command

```
1 # Lab 01a: Before we begin, confirm files are loaded
2
3 display(dbutils.fs.ls("dbfs:/FileStore/tables"))
```

► (3) Spark Jobs

	path	name	size
1	dbfs:/FileStore/tables/Errors/	Errors/	0
2	dbfs:/FileStore/tables/Mod01_Intro-1.ipynb	Mod01_Intro-1.ipynb	1159
3	dbfs:/FileStore/tables/Mod01_Intro.ipynb	Mod01_Intro.ipynb	1159
4	dbfs:/FileStore/tables/beatles.txt	beatles.txt	146
5	dbfs:/FileStore/tables/carriers.csv	carriers.csv	37794
6	dbfs:/FileStore/tables/checkpoint/	checkpoint/	0
7	dbfs:/FileStore/tables/dept.csv	dept.csv	312

```
1 # Lab 01c: To delete a file, type:
2
3 dbutils.fs.rm("dbfs:/FileStore/tables/flights_abbr_1.txt")
```

Out[3]: True

# Lab 02: Language Interpreters

- Databrick's UI is a new web-based notebook which brings data exploration, visualization, sharing and collaboration features to Spark. It support Python, but also a growing list of programming languages such as:
  - Markdown (%md)
  - Python (%py)
  - Scala (%scala)
  - SparkSQL (%sql),
  - Shell (%sh)
  - Hive (%hive)

- Default language is shown in upper left-hand corner of Spark UI
- Default language does not require language defined as first row in Cell
- However if wish to use non-default language, must explicitly code it via %
- Each Language has its own syntax for comments



# The Big Picture – Languages (1 of 2)



## Programming Languages – Python, Scala, Java, R

Spark can use [Python](#) (./bin/pyspark), [Scala](#) (./bin/spark-shell), Java and R. Here's example of [Python](#), [Scala](#) and [Java](#) languages to execute a wordcount

### Python (pyspark)

```
text_file = sc.textFile("hdfs://...")
counts = text_file.flatMap(lambda line: line.split(" ")) \
    .map(lambda word: (word, 1)) \
    .reduceByKey(lambda a, b: a + b)
counts.saveAsTextFile("hdfs://...")
```

### Scala (spark-shell)

```
val textFile = sc.textFile("hdfs://...")
val counts = textFile.flatMap(line => line.split(" "))
    .map(word => (word, 1))
    .reduceByKey(_ + _)
counts.saveAsTextFile("hdfs://...")
```

### Java

```
JavaRDD<String> textFile = sc.textFile("hdfs://...");
JavaRDD<String> words = textFile.flatMap(new FlatMapFunction<String, String>() {
    public Iterable<String> call(String s) { return Arrays.asList(s.split(" ")); }
});
JavaPairRDD<String, Integer> pairs = words.mapToPair(new PairFunction<String, String, Integer>() {
    public Tuple2<String, Integer> call(String s) { return new Tuple2<String, Integer>(s, 1); }
});
JavaPairRDD<String, Integer> counts = pairs.reduceByKey(new Function2<Integer, Integer, Integer>() {
    public Integer call(Integer a, Integer b) { return a + b; }
});
counts.saveAsTextFile("hdfs://...");
```

# The Big Picture – Languages (2 of 2)



## Programming Languages – Python, Scala, Java, R

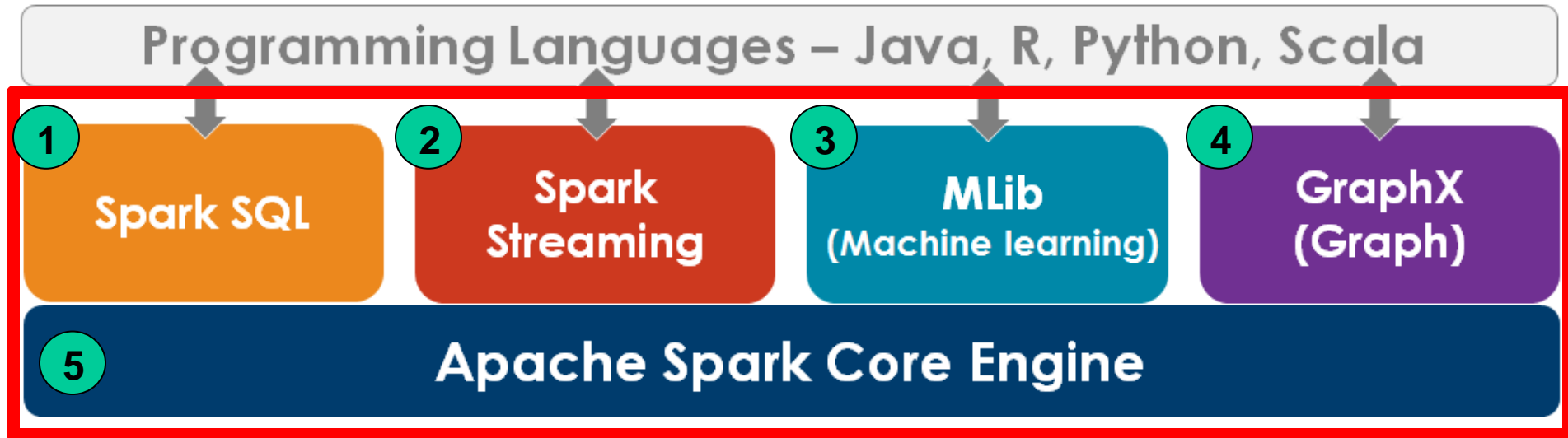
The screenshot shows the RStudio IDE interface. The top menu bar includes File, Edit, Code, View, Plots, Session, Build, Debug, Tools, and Help. The toolbar contains icons for file operations and a search bar. The main editor window displays an R script named 'SparkR1.R' with the following code:

```
1 Sys.setenv(SPARK_HOME="/usr/hdp/current/spark-client/")
2 .libPaths(c(file.path(Sys.getenv("SPARK_HOME"), "R", "lib"), .libPaths()))
3 library(SparkR)
4 sc <- SparkR::sparkR.init(master = "yarn-client")
5
6 sqlContext <- sparkRSQL.init(sc)
7 path <- file.path("file:///usr/hdp/2.4.0.0-169/spark/examples/src/main/resources/people.json")
8 peopleDF <- jsonFile(sqlContext, path)
9 printSchema(peopleDF)
10
11 # Register this DataFrame as a table
12 registerTempTable(peopleDF, "people")
13 teenagers <- sql(sqlContext, "SELECT name, age FROM people WHERE age >= 13 order by age")
14 teenagersLocalDF <- collect(teenagers)
15 print(teenagersLocalDF)
16
17 sparkR.stop()
18
```

The console window at the bottom shows the output of the script, including log messages from YarnScheduler and DAGScheduler, and the final output of the `print(teenagersLocalDF)` command:

```
16/11/21 12:39:54 INFO YarnScheduler: Removed TaskSet 8.0, whose tasks have all completed, from pool
16/11/21 12:39:54 INFO DAGScheduler: ResultStage 8 (dfToCols at NativeMethodAccessorImpl.java:-2) finish
ed in 0.224 s
16/11/21 12:39:54 INFO DAGScheduler: Job 6 finished: dfToCols at NativeMethodAccessorImpl.java:-2, took
0.644229 s
> print(teenagersLocalDF)
  name age
1 Justin 19
2  Andy 30
>
```

# The Big Picture – Add Unified Stack



- 1 **Spark SQL** – Allows you to work with DataFrames/Datasets/Views using structured data. It allows querying with the Apache Hive variant of SQL
- 2 **Spark Streaming** – Enables processing of live data streams (mini-batches and RT)
- 3 **Spark MLib** – Multiple machine learning algorithms such as classification, regression, clustering and collaborative filtering, etc
- 4 **GraphX** – Graph functions such as PageRank and triangle counting
- 5 **Spark Core** – Basic functionality including task scheduling, memory management, fault recover, storage and Resilient Distributed Datasets (RDDs)

# 1 Lab 03: Spark SQL (Tables)

## Spark SQL

- Starting with Spark version 1.3, **DataFrames**, **Tables**, **TempViews** and **DataSets** have been introduced so Spark data can be processed in tabular form and ANSI SQL-like language using keywords like: SELECT, FROM, WHERE, GROUP BY, etc
- Spark SQL is defined as objects which have a Schema (column names and data types)
- And even better, Spark SQL uses an Optimizer that is typically performs faster than a comparable RDD (more on this later)

```
1 %sql
2 SELECT uniquecarrier, avg(depdelay) as AVGdelay FROM flights_abbrev
3 GROUP BY uniquecarrier
4 ORDER BY AVGdelay DESC
```

► (2) Spark Jobs

	uniquecarrier ▲	AVGdelay ▲	
1	WN	27.48697394789579	

# 1 Lab 04: Spark SQL (DataFrames)

## Spark SQL

- Here's an example using the DataFrame API to read and then display

```
1 df = spark.read.csv("/FileStore/tables/header_flights_abbr.csv", header=True)
2 display(df)
```

► (2) Spark Jobs

	month ▲	dayofmonth ▲	dayofweek ▲	deptime ▲	arrtime ▲	uniquecarrier ▲	flightnum ▲
1	1	3	4	2003	2211	WN	335
2	1	3	4	926	1054	WN	1746
3	1	3	4	1940	2121	WN	378
4	1	3	4	1937	2037	WN	509
5	1	3	4	754	940	WN	1144
6	1	3	4	1422	1657	WN	188

## ② Spark Streaming: Mini-batches and Real Time

### Spark Streaming

- Incoming data streams collected for X seconds per node
  - Captured as RDDs
  - Batch sizes as low as half a second
  - Normally 1-60 seconds
- Spark operators process mini-batches
  - Additional processing in Hadoop or in-database
- Competes with Apache Storm



# 3 Machine Learning



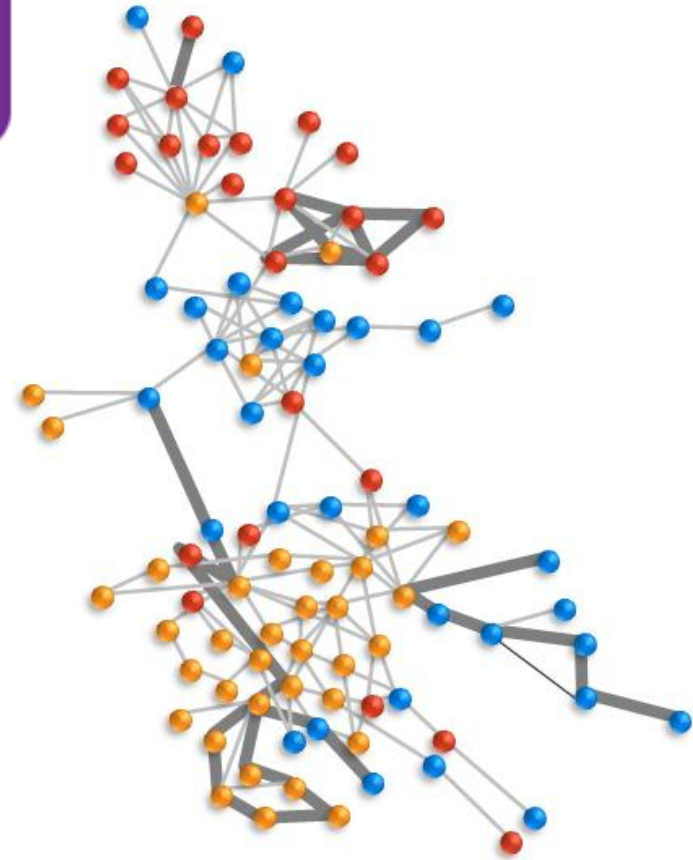
- **Classification and regression**
  - Linear support vector machine
  - Logistic regression
  - Linear least squares, Lasso, ridge regression
  - Decision tree
  - Naive Bayes
- **Collaborative filtering**
  - Alternating least squares
- **Clustering**
  - K-means
- **Dimensionality reduction**
  - Singular value decomposition
  - Principal component analysis
  -
- **Optimization**
  - Stochastic gradient descent
  - Limited-memory BFGS

**In Spark 2.0+, it's called ML**

## 4 GraphX

### GraphX (Graph)

- Analyzes tabular data
  - Nodes: People and things (nouns/keys)
  - Edges: relationships between nodes
- Graph engine, not a Graph database
  - Like Aster® Graph, Apache Giraph
- Algorithms
  - PageRank
  - Connected components
  - Label propagation
  - SVD++
  - Strongly connected components
  - Triangle count



**In Spark 2.0+, it's called SparkGraph**

<https://www.mapr.com/blog/how-get-started-using-apache-spark-graphx-scala>



## 5 Spark Core Engine

### Apache Spark Core Engine

- Is the general purpose Processing Engine for Spark
- Provides in-memory computing capabilities and parallel processing
- Supports various workloads (interactive, batch, streaming)
- Ease of development with native APIs in Java, Scala, Python, R, SQL



# The Big Picture – Add Resource Manager

Programming Languages – Java, R, Python, Scala

Spark SQL

Spark  
Streaming

MLib  
(Machine learning)

GraphX  
(Graph)

Apache Spark Core Engine



Spark Standalone



Spark runs over a variety of Cluster Managers. Spark's built-in Standalone mode is the easiest way to deploy. However if you want to share with other applications, Spark can run on Hadoop's YARN and Apache Mesos

```
spark-submit --class org.apache.hadoop.examples.WordCount --master spark://192.168.100.24:7077 --num-executors 1 --driver-memory 512m --executor-memory 512m --executor-cores 1 /usr/hdp/2.4.0.0-169/hadoop-mapreduce/hadoop-mapreduce-examples-2.7.1.2.4.0.0-169.jar /user/zeppelin/lincoln.txt /user/zeppelin/sf7/
```

```
spark-submit --class org.apache.hadoop.examples.WordCount --master yarn --num-executors 1 --driver-memory 512m --executor-memory 512m --executor-cores 1 /usr/hdp/2.4.0.0-169/hadoop-mapreduce/hadoop-mapreduce-examples-2.7.1.2.4.0.0-169.jar /user/zeppelin/lincoln.txt /user/zeppelin/sf8/
```

Note there is a 4<sup>th</sup> Manager named '**Local**' used for testing/prototyping

# The Big Picture – Add Data

Programming Languages – Java, R, Python, Scala

Spark SQL

Spark  
Streaming

Mlib  
(Machine learning)

GraphX  
(Graph)

Apache Spark Core Engine



Spark Standalone



HDFS

LFS

APACHE  
HBASE



kafka



S3

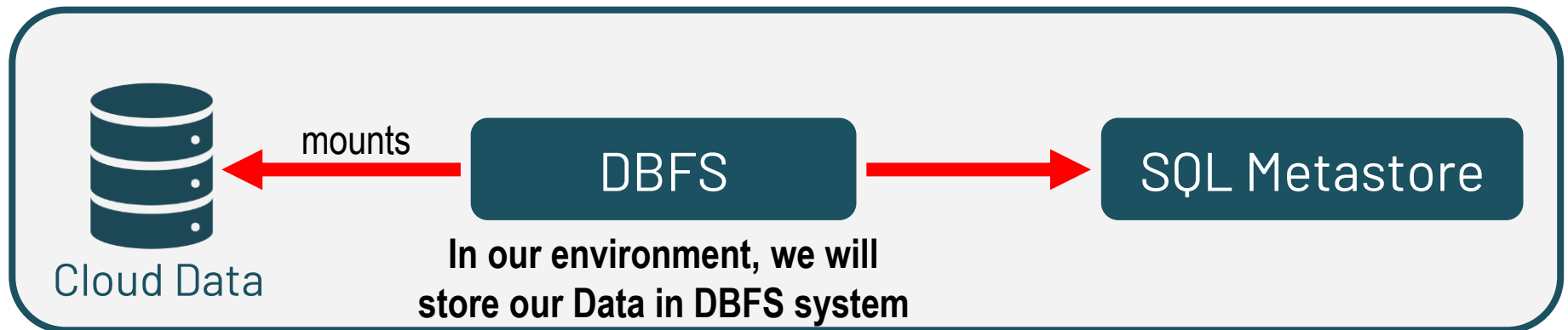
Spark is dependent on data storage as it has none of its own. Spark supports both Hadoop Distributed File System (HDFS) as well as Local File System (LFS). In addition, it supports data from other Projects for ingestion

**We will be using 'dbfs' file system. See next slide**

# Data

DBFS (Databricks File System) is a layer typically over a Cloud-based object store

- It has mounts to your Cloud data storage
- It also hooks into Hive Metastore for your SQL Tables



- [/FileStore](#): Imported data files, generated plots, and uploaded libraries
- [/databricks-datasets](#): Sample public datasets
- [/databricks-results](#): Files generated by downloading the full results of a query
- [/databricks/init](#): init scripts
- [/user/hive/warehouse](#): Data and metadata for non-external Hive table

```
%fs ls /user/hive/warehouse
```

	path
1	dbfs:/user/hive/warehouse/_training_database.db/
2	dbfs:/user/hive/warehouse/accounting.db/
3	dbfs:/user/hive/warehouse/accounts/
4	dbfs:/user/hive/warehouse/ade_brooke_wenig_databricks_com_db.db/

# Data Files Sourced from Databricks dbfs

---

Databricks File System (DBFS) is a distributed file system mounted into a Databricks workspace and available on Databricks clusters. DBFS is an abstraction on top of scalable object storage and offers the following benefits:

- Allows you to mount storage objects so that you can seamlessly access data without requiring credentials
- Allows you to interact with object storage using directory and file semantics instead of storage URLs
- Persists files to object storage, so you won't lose data after you terminate a cluster

<https://docs.databricks.com/data/databricks-file-system.html#special-dbfs-root-location>

# Data Sources – File Formats

Spark supports most common Data file formats including:

1. CSV
2. TSV
3. JSON
4. AVRO
5. Text
6. Parquet
7. **Delta**

Name	Score	ID
...		

**Row-Oriented** data on disk (**CSV**)

Kit	4.2	1	Alex	4.5	2	Terry	4.1	3
-----	-----	---	------	-----	---	-------	-----	---

**Column-Oriented** data on disk (**Parquet**, **Delta**, **ORC**)

Kit	Alex	Terry	4.2	4.5	4.1	1	2	3
-----	------	-------	-----	-----	-----	---	---	---

# Delta File Format

---

Technology designed to be used with Apache Spark to build robust data lakes



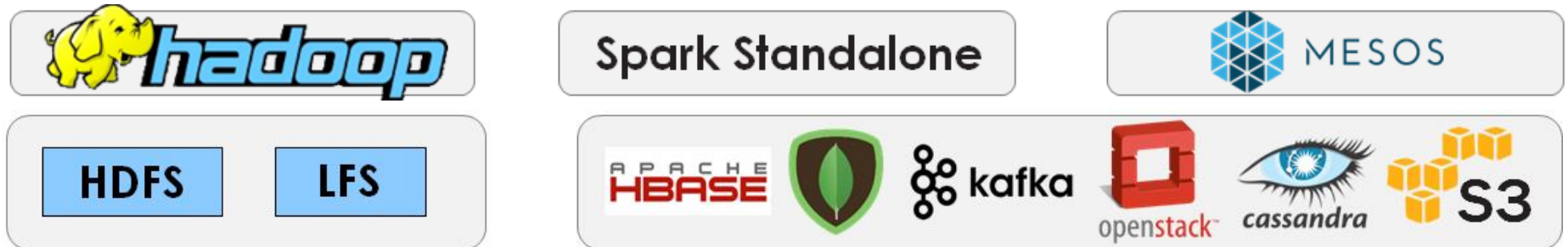
- **ACID transactions**
- **Metadata handling**
- **Data versioning**
- **Batch and Streaming unification**
- **Schema enforcement/Schema evolution**
- **Audit history**
- **Merge**

**'Delta' file format is one component of Delta Lake**

# The Big Picture – Add BI Tools



## Apache Spark Core Engine

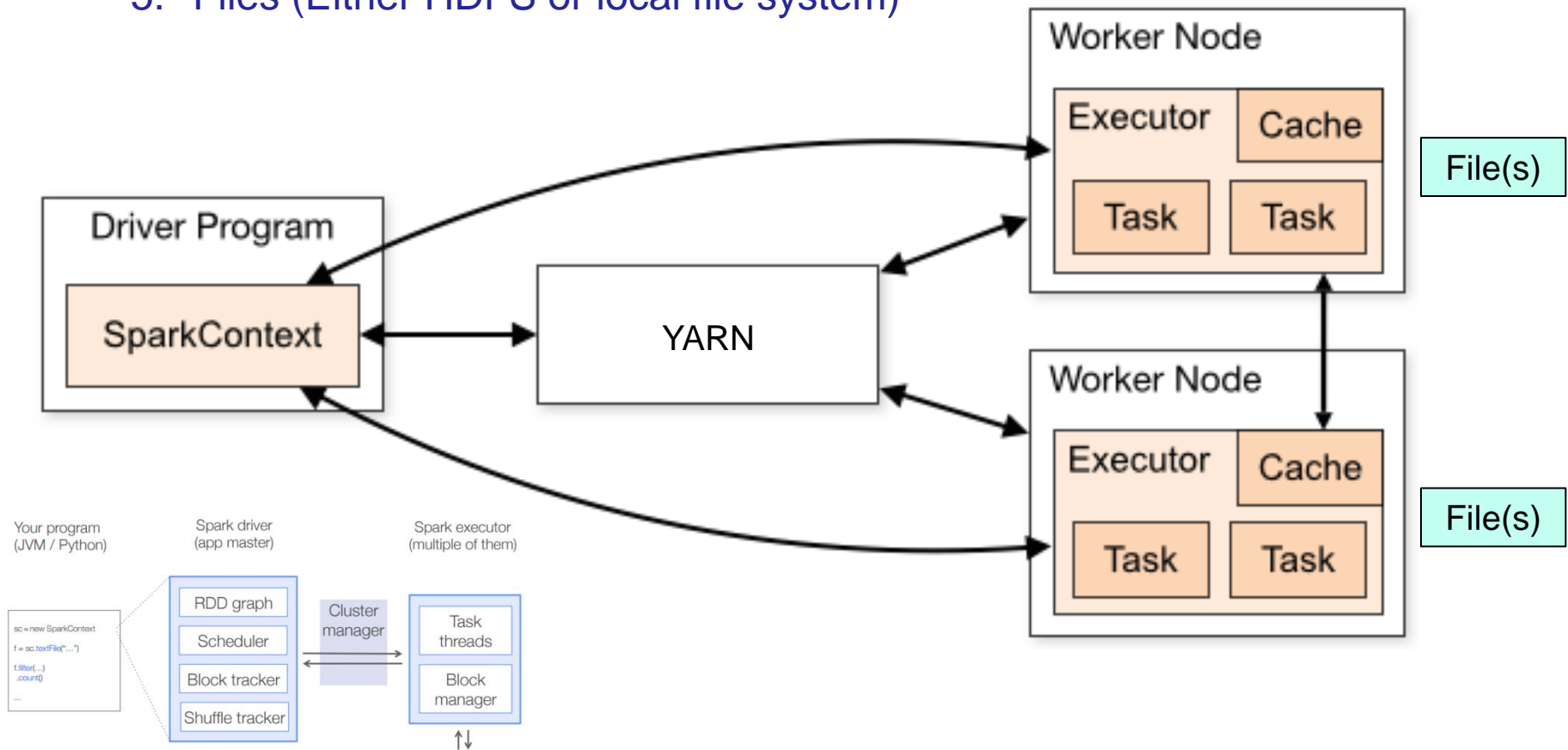


Since Spark SQL is structured data, you can use BI and ETL tools



# Spark Cluster components (using YARN)

1. Driver
2. SparkContext
3. Resource Manager (can be Standalone, Mesos, Hadoop YARN)
4. Worker Nodes and Executors
5. Files (Either HDFS or local file system)

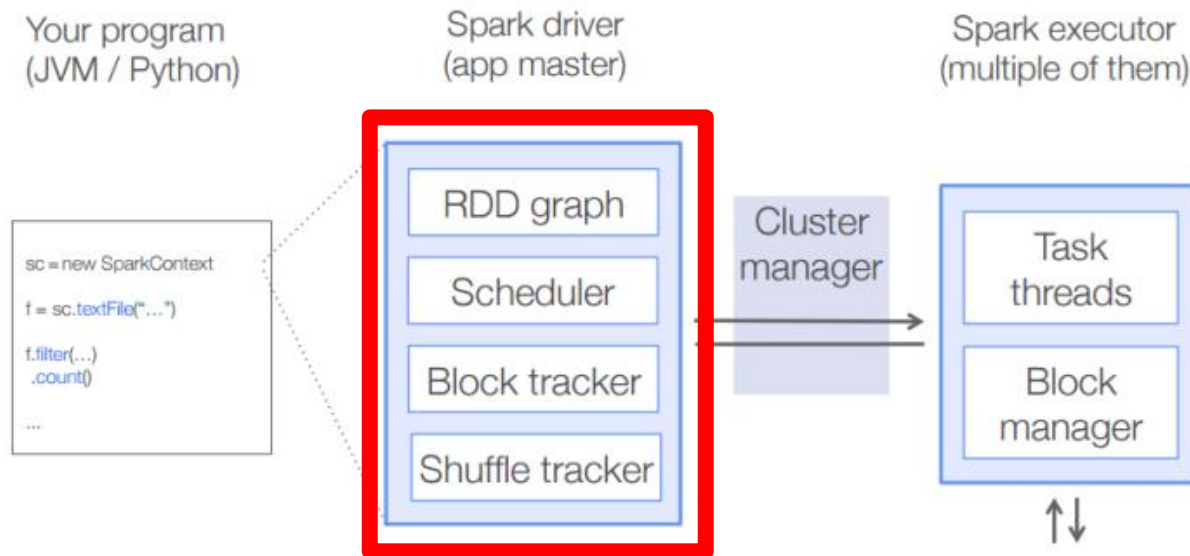


# Driver and Spark Context (and SparkConf)

## 1. Driver

- Is a JVM that maintains connections and submits tasks to Executors on Worker nodes for execution. Responsible for creating Directed Acyclical Graph (DAG) of Transformations and Actions

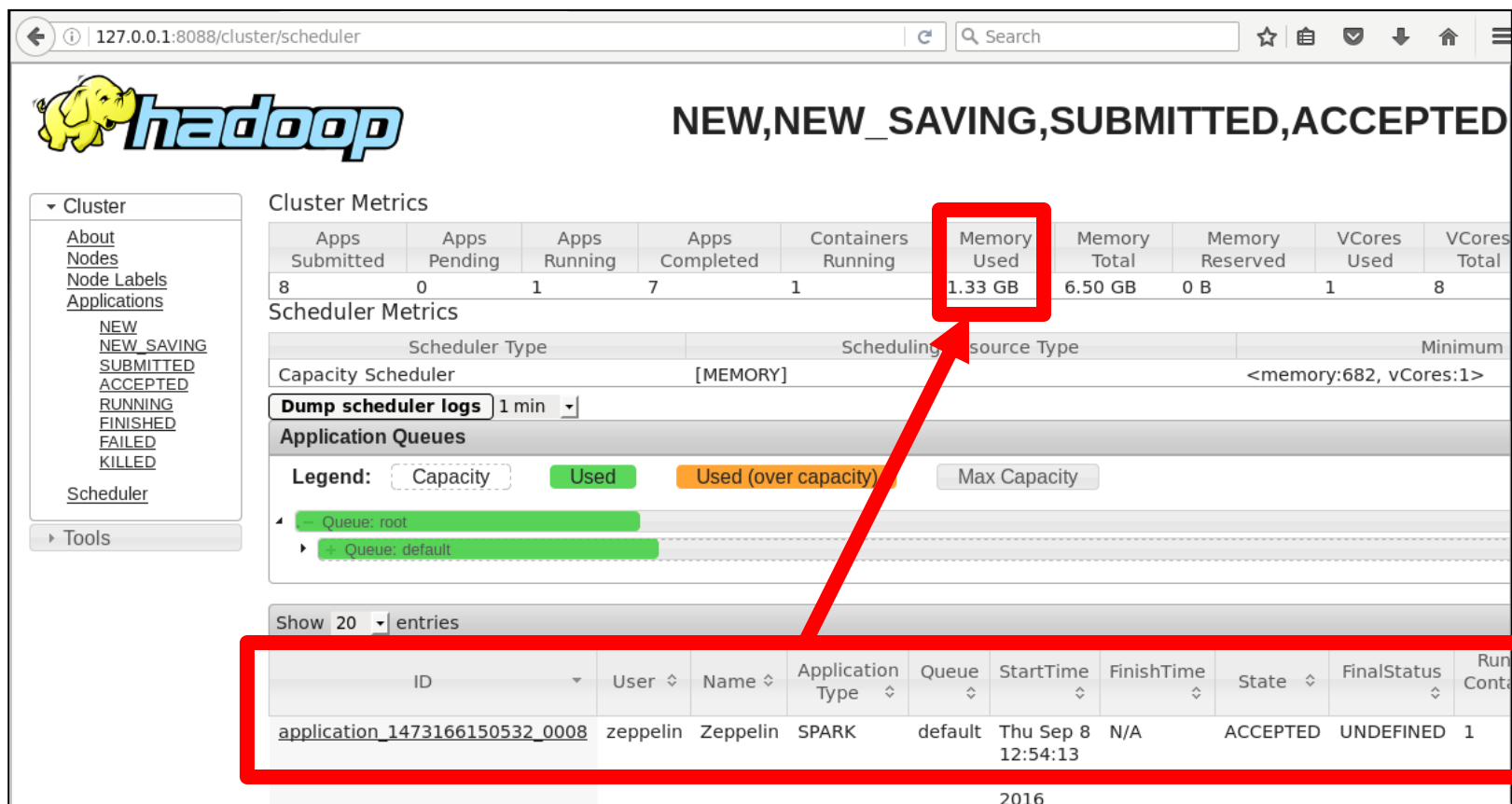
- 2. SparkContext** – Java class that contains the code to process data in a parallel fashion. Coordinates with YARN to schedule resources. Created when user invokes REPL (Read, Evaluate, Print, Loop)



In order to create a **SparkContext** you should first create a **SparkConf**. The **SparkConf** stores configuration parameters that your Spark Driver application will pass to SparkContext

# Resource Manager - YARN

- 3. Resource Manager** – Spark is agnostic to underlying cluster manager as long as it can acquire executor processes. When running in an Hadoop environment, YARN will typically arbitrate all the available cluster resources and thus helps manage the distributed applications



The screenshot displays the Hadoop YARN Resource Manager web interface. The top navigation bar includes the Hadoop logo and the text "NEW,NEW\_SAVING,SUBMITTED,ACCEPTED". The left sidebar contains a "Cluster" section with links for "About", "Nodes", "Node Labels", and "Applications". The main content area is divided into several sections:

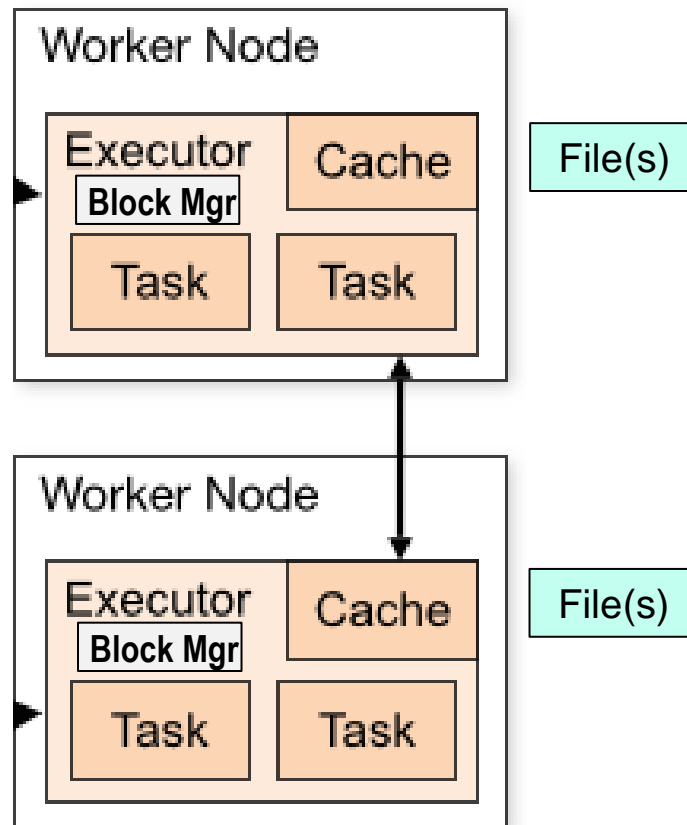
- Cluster Metrics:** A table showing various metrics for the cluster. The "Memory Used" metric is highlighted with a red box and shows a value of 1.33 GB.
- Scheduler Metrics:** A table showing scheduler information. The "Scheduler Type" is "Capacity Scheduler" and the "Scheduling" is "[MEMORY]".
- Application Queues:** A section showing the state of application queues. It includes a legend for "Capacity", "Used", "Used (over capacity)", and "Max Capacity". Below the legend, there are two queues: "Queue: root" and "Queue: default".
- Application List:** A table showing a list of applications. The application "application\_1473166150532\_0008" is highlighted with a red box. It is a Zeppelin SPARK application in the "default" queue, submitted by "zeppelin" on "Thu Sep 8 12:54:13".

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total
8	0	1	7	1	1.33 GB	6.50 GB	0 B	1	8

ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Run Cont
application_1473166150532_0008	zeppelin	Zeppelin	SPARK	default	Thu Sep 8 12:54:13	N/A	ACCEPTED	UNDEFINED	1

# Worker nodes and Executors

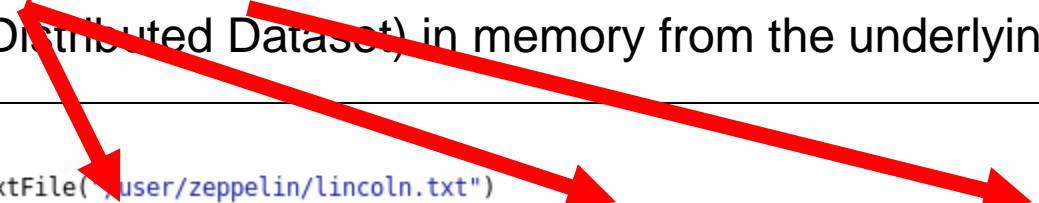
4. **Worker nodes** – Worker nodes are the physical servers that run Executors. Note you may have multiple Executors per Worker node. Workers have a fixed set of resources (CPU and RAM) assigned to it by YARN. Worker nodes must share their resources since multiple applications may be run on same Worker




# Executors

4. **Executors** – Processes which carry out the Tasks. Tasks are composed typically of either **Map** or **Reduce**. Each Executor creates task threads and holds the RDD (Resilient Distributed Dataset) in memory from the underlying file(s)

```
%pyspark
baseRDD = sc.textFile("/user/zeppelin/lincoln.txt")
countkeyRDD = baseRDD.flatMap(lambda line: line.split(" ")).map(lambda word: (word,1)).reduceByKey(lambda v1,v2: v1+v2)
print countkeyRDD.collect()
countkeyRDD.saveAsTextFile("/user/zeppelin/maryfiles")
```



- Each Executor runs a single JVM and has a configurable amount of resources available to it which you can allocate

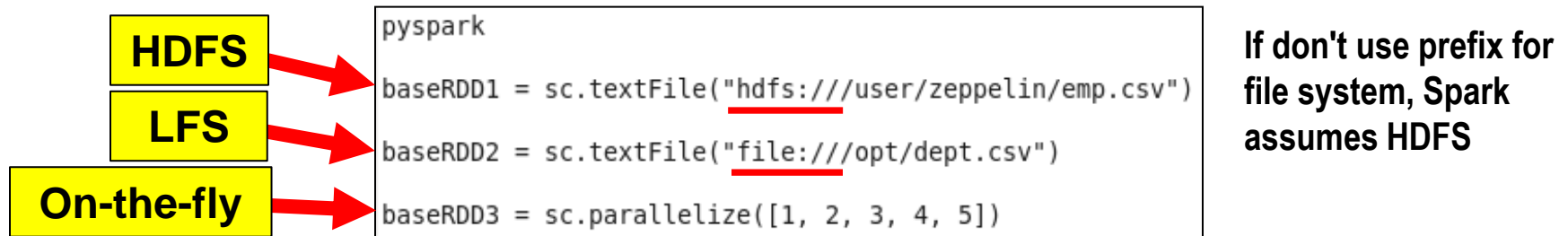


```
spark-submit --master yarn-cluster
--num-executors 10 --executor-memory 1024m --executor-cores 5 --driver-memory 512m
/usr/hdp/2.4.0.0-169/hadoop-mapreduce/hadoop-mapreduce-examples-2.7.1.2.4.0.0-169.jar WordCount
/user/zeppelin/lincoln.txt /user/zeppelin/output/
```

- The Spark processing occurs within the Executor' RAM
- For example, if you submit the Spark query in 1<sup>st</sup> graphic above, the Job will be run in the Executor's RAM. After the result set is returned (either to the Console or in our case, written to a Text File, **the RAM contents are immediately flushed**. It is not retained in Cache unless explicitly coded to do so

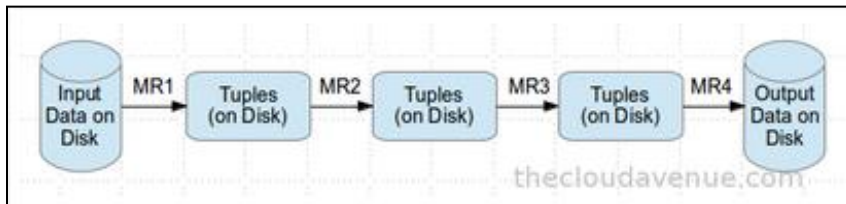
# Resilient Distributed Datasets (RDDs)

- Basic building block of Spark is the concept of a **Resilient Distributed Dataset (RDD)**, which is a fault-tolerant collection of elements that can be operated in parallel. There are two ways to create new RDD: 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

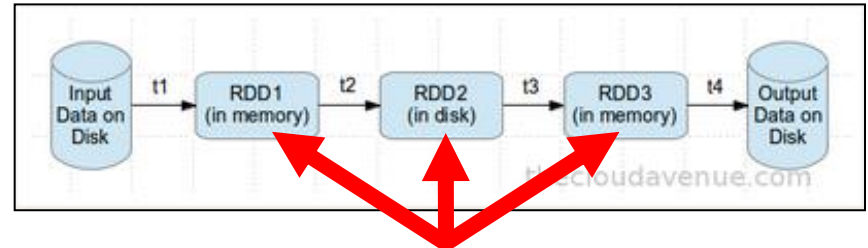


- RDDs use more RAM instead of network and disk I/O so its relatively fast when compared to Hadoop using the Map-Reduce processing engine

## Traditional Map Reduce has extensive Disk I/O



## RDD's can be configured to use RAM or Disk I/O



# Lab 05a/c: Resilient Distributed Datasets

RDD's are a read-only distributed (partitioned) collection of records. It's one of the objects that you perform ETL scripts on (ie: Map, Reduce tasks)

1. Read-only. If want to run an ETL, must create a new RDD based on the previous RDD. In below 2 queries, we count number of words in poem

```
val rdd1=sc.textFile("/user/zeppelin/mary.txt")
val rdd2=rdd1.flatMap(line => line.split(" "))
val rdd3=rdd2.map(word => (word, 1))
val rdd4= rdd3.reduceByKey((x,y) => x+y)
rdd4.collect()
```

After loading file into RDD1, we perform 3 transformations before dumping to Console ...

```
val rdd1=sc.textFile("/user/zeppelin/mary.txt")
val rdd2=rdd1.flatMap(line=>line.split(" ")).map(word=>(word,1)).reduceByKey((x,y)=>x+y)
rdd2.collect()
```

... or can pipeline RDDs together into one

Answer set

```
res23: Array[(String, Int)] = Array((went,1), (Its,1), (fleece,1), (as,1),
(everywhere,1), (go,1), (lamb,2), (little,1), (And,1), (white,1), (The,1),
(was,2), (had,1), (a,1), (that,1), (to,1), (sure,1), (Mary,2), (snow,1))
```

2. Distributed (partitioned) refers to breaking the RDD data into smaller blocks so they can be operated in parallel. At a minimum Spark defaults to 2 partitions

```
val rdd1=sc.textFile("/user/zeppelin/mary.txt")
rdd1.getNumPartitions

rdd1: org.apache.spark.rdd.RDD[String] = MapParti
res27: Int = 2
```

# Spark 3.0+ new features (vs. Spark 2.x)

- **Enhancements to:** Core Engine and SparkSQL (17x faster than Spark 2)
- 3400 JIRA's resolved
- **Enhanced Performance**
  - **Adaptive Query Execution** (Better stats means faster queries)
    - Better Join strategies (via Statistics)
    - Shrink number of Reducers after Over-Partitioning
    - Data Skew minimized
    - Coalesce Shuffle Partitions
  - Dynamic Partition Pruning
  - JOIN Optimizer Hints expanded
- **Other**
  - 32 new built-in functions, Monitoring, Structured Streaming UI
  - Delta Lake, DDL and DML enhancements, SQL compatibility

[https://databricks.com/session\\_na20/deep-dive-into-the-new-features-of-apache-spark-3-0](https://databricks.com/session_na20/deep-dive-into-the-new-features-of-apache-spark-3-0)

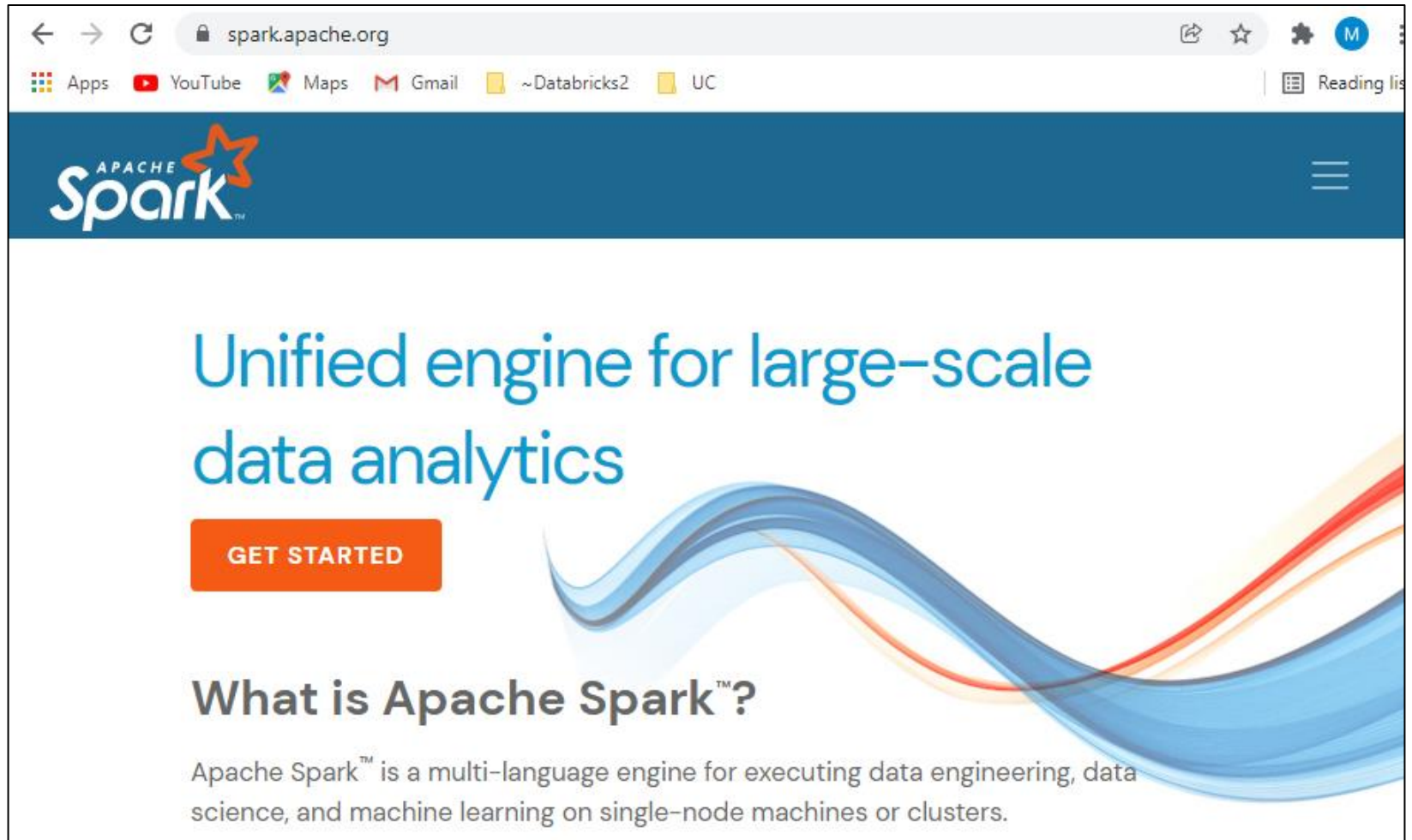


# Glossary - Spark terminology

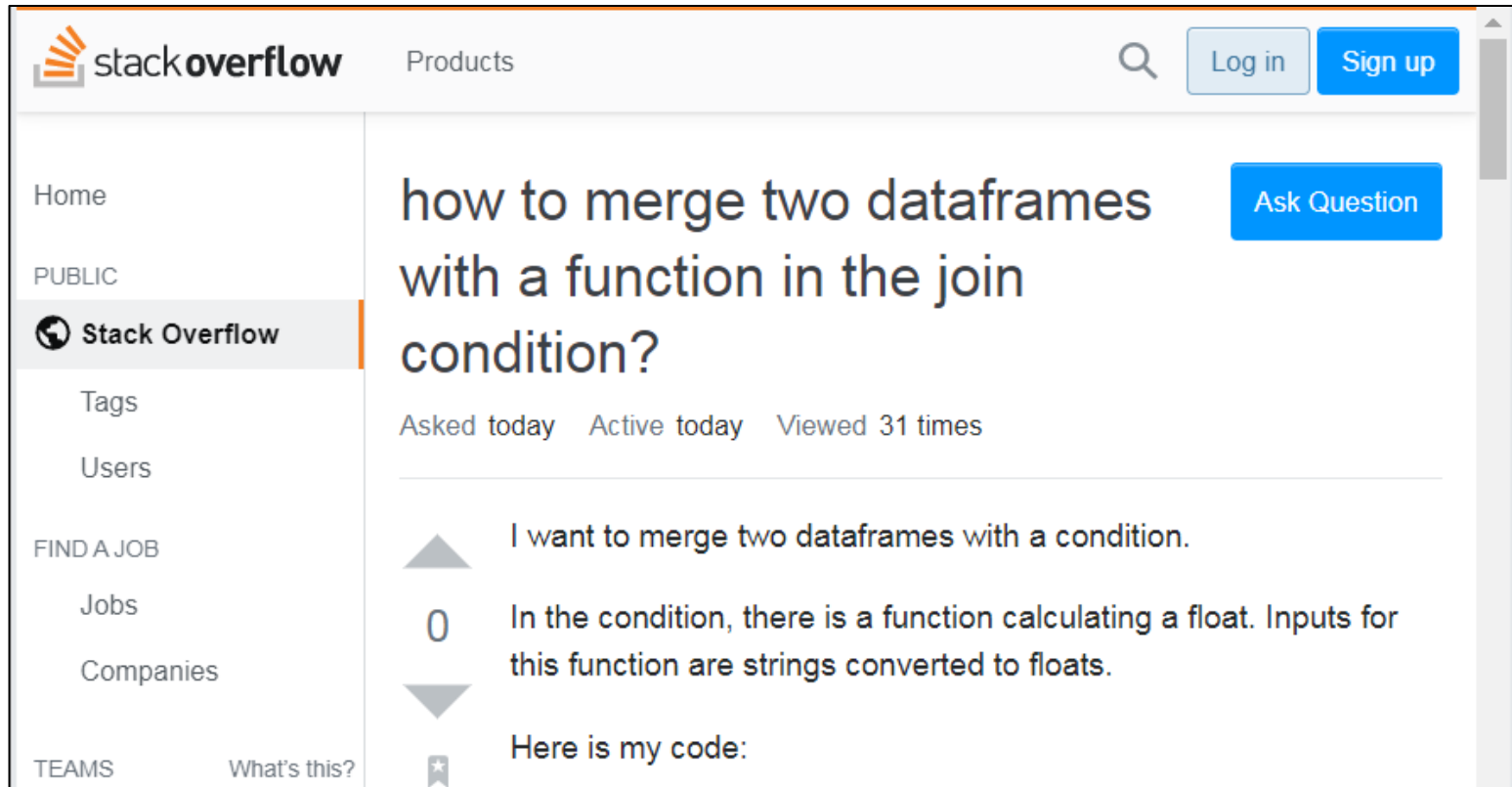
Term	Meaning
Application	User program built on Spark. Consists of a <i>driver program</i> and <i>executors</i> on the cluster.
Application jar	A jar containing the user's Spark application. In some cases users will want to create an "uber jar" containing their application along with its dependencies. The user's jar should never include Hadoop or Spark libraries, however, these will be added at runtime.
Driver program	The process running the <code>main()</code> function of the application and creating the <code>SparkContext</code>
Cluster manager	An external service for acquiring resources on the cluster (e.g. standalone manager, Mesos, YARN)
Deploy mode	Distinguishes where the driver process runs. In "cluster" mode, the framework launches the driver inside of the cluster. In "client" mode, the submitter launches the driver outside of the cluster.
Worker node	Any node that can run application code in the cluster
Executor	A process launched for an application on a worker node, that runs tasks and keeps data in memory or disk storage across them. Each application has its own executors.
Task	A unit of work that will be sent to one executor
Job	A parallel computation consisting of multiple tasks that gets spawned in response to a Spark action (e.g. <code>save</code> , <code>collect</code> ); you'll see this term used in the driver's logs.
Stage	Each job gets divided into smaller sets of tasks called <i>stages</i> that depend on each other (similar to the map and reduce stages in MapReduce); you'll see this term used in the driver's logs.

<http://spark.apache.org/docs/latest/cluster-overview.html>

Want to Learn More? Go to: [spark.apache.org](https://spark.apache.org)



# For Troubleshooting, Go to: **Stackoverflow**



The screenshot shows the Stack Overflow homepage. The header includes the Stack Overflow logo, a search bar, and links for 'Log in' and 'Sign up'. The left sidebar contains navigation links: Home, PUBLIC, Stack Overflow (selected), Tags, Users, FIND A JOB, Jobs, Companies, TEAMS, and What's this?. The main content area displays a question titled 'how to merge two dataframes with a function in the join condition?'. Below the title, it shows 'Asked today', 'Active today', and 'Viewed 31 times'. The question body contains the text: 'I want to merge two dataframes with a condition. In the condition, there is a function calculating a float. Inputs for this function are strings converted to floats. Here is my code:'. An 'Ask Question' button is visible in the top right corner of the question area.

stackoverflow Products

Log in Sign up

Home

PUBLIC

Stack Overflow

Tags

Users

FIND A JOB

Jobs

Companies

TEAMS What's this?

## how to merge two dataframes with a function in the join condition?

Ask Question

Asked today Active today Viewed 31 times

I want to merge two dataframes with a condition.

0 In the condition, there is a function calculating a float. Inputs for this function are strings converted to floats.

Here is my code:

# In Review: Spark Architecture

**After completing this module, you should familiar with the following terms and concepts**

- Spark is an in-memory processing engine dependent on external storage
- You login to Spark using REPL (sometimes called Spark shells)
- Spark consists of:
  - Programming languages (Scala, Python, etc.),
  - Unified Stack including: Spark SQL, Streaming, MLib and GraphX
  - Resource Manager
  - Data
  - BI and ETL tools
- What Resilient Distributed Datasets(RDD) are
- Spark Cluster components include:
  - Driver
  - SparkContext
  - YARN
  - Worker nodes and Executors

```
hdfs dfs -du -h -s /user/zeppelin/job.csv (see RF 1 size and RF 3 size)
```