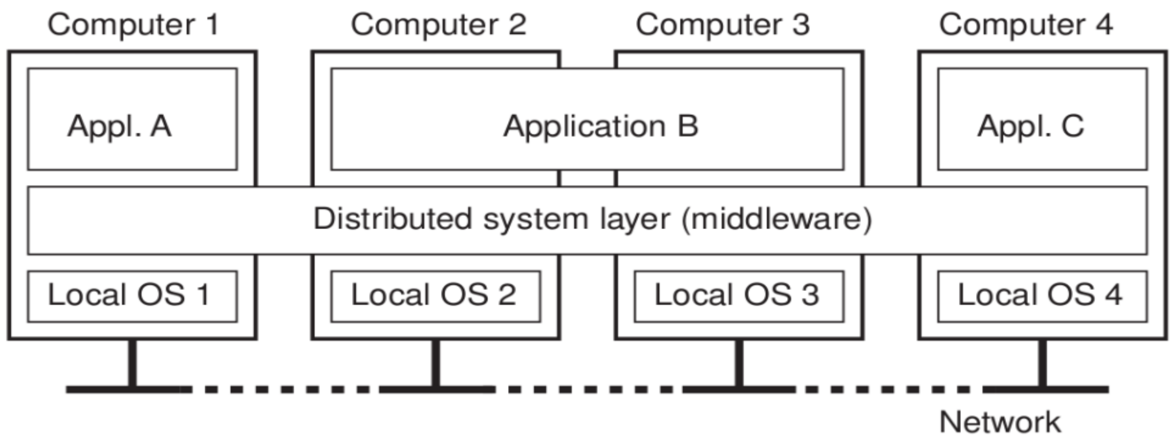
SPARK

Distributed Systems: The Big Picture

A distributed system is one in which components located at networked communication and coordinate their actions only by-passing messages or it is a system with multiple components located on different machines that communicate and coordinate actions in order to appear as a single coherent system for the end users so that they get benefit out of it.



How a Distributed System Works?

Hardware and software architectures are used to maintain a distributed system. Everything must be interconnected CPUs via the network and processes via the communication system.

Benefits and challenges of distributed systems

1. **Horizontal Scalability:** Since computing happens independently on each node, it is easy and generally inexpensive to add additional nodes and functionality as necessary.

2. **Reliability:** Most distributed systems are fault tolerant as they can be made up of hundreds of nodes that work together. The system generally doesn't experience any disruptions if a single machine fails.

3. **Performance:** Distributed systems are extremely efficient because workloads can be broken up and sent to multiple machines.

Getting Started with Spark 2

Spark is a framework which provides a number of inter-connected platforms, systems and standards for Big Data projects.

Spark can access data on many different storage solutions. The data structure that Spark uses is called **Resilient Distributed Dataset**, or **RDD**.

Spark has been found to run **100** times **faster** in-memory, and 10 times faster on disk. It's also been used to sort **100 TB** of data 3 times faster than Hadoop MapReduce on one-tenth of the machines.

Benefits of Apache Spark

- Speed
- Ease of Use
- A Unified Engine
- Support

Why we need spark?

Apache **Spark** is a data processing framework that can quickly perform processing tasks on very large data sets, and can also distribute data processing tasks across multiple computers, either on **its** own or in tandem with other distributed computing tools.

There are five main components of Apache Spark

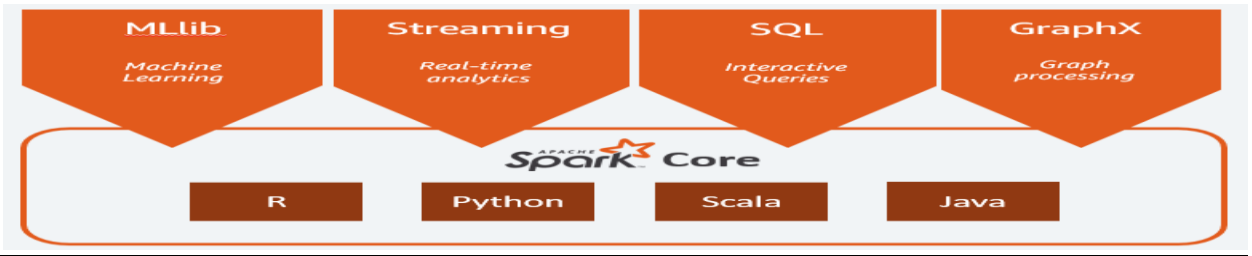
1. **Apache Spark Core**: The basis of the whole project. Spark Core is responsible for necessary functions such as scheduling, task dispatching, input and output operations, fault recovery, etc.

2. **Spark Streaming**: This component enables the processing of live data streams. Data can originate from many different sources like Kafka, etc.

3. **Spark SQL:** Spark uses this component to gather information about the structured data and how the data is processed.

4. **Machine Learning Library (MLlib):** This library consists of many machine learning algorithms.

5. **GraphX:** A set of APIs used for facilitating graph analytics tasks.



Structured Streaming in Apache Spark 2

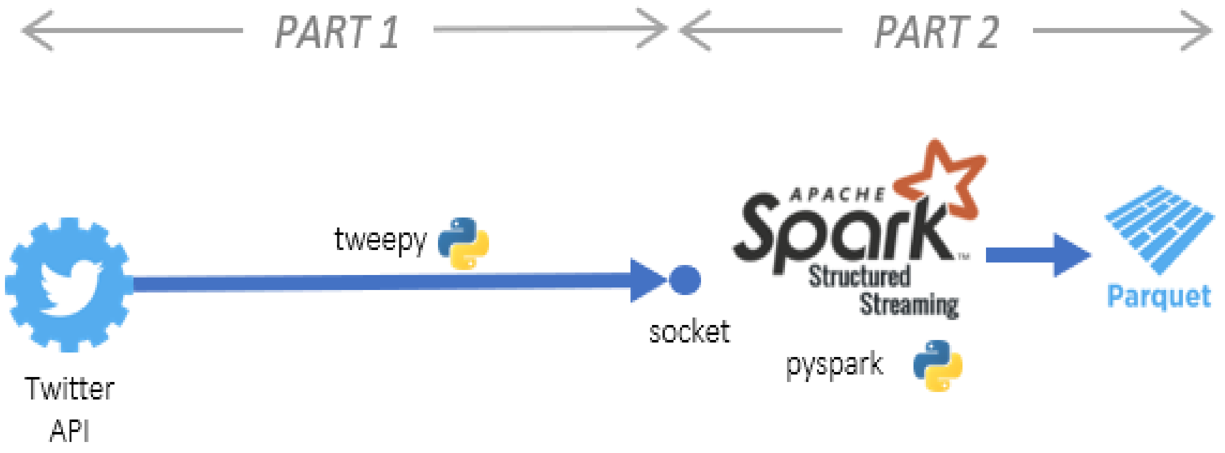
Structured Streaming is a scalable and fault-tolerant stream processing engine built on the Spark SQL engine. You can express your streaming computation the same way you would express a batch computation on static data.
Stream processing applications work with continuously updated data and react to changes in real-time.
Structured Streaming in Apache Spark 2, you'll focus on using the tabular data frame API to work with streaming, unbounded datasets using the same APIs that work with bounded batch data.

Use of Structured Streaming

Spark Streaming is an extension of the core Spark API that allows data engineers and data scientists to process real-time data from various sources including Kafka, Flume, and Amazon Kinesis. This processed data can be pushed out to file systems, databases, and live dashboards.

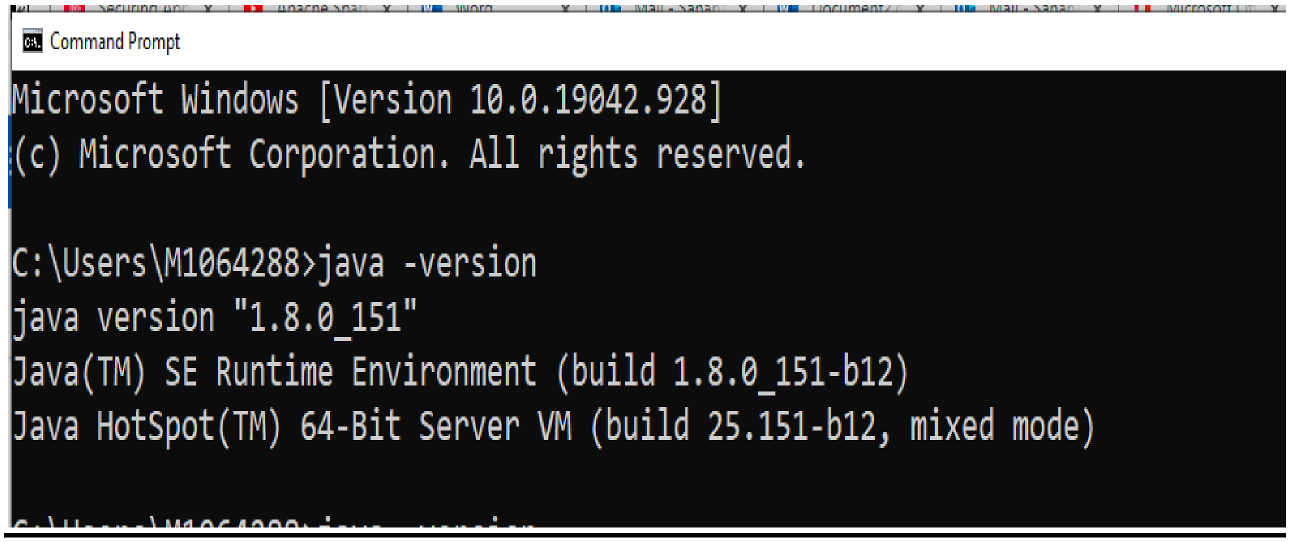
Example:

Sentiment analysis on streaming Twitter data using Spark Structured Streaming & Python



Installation of spark

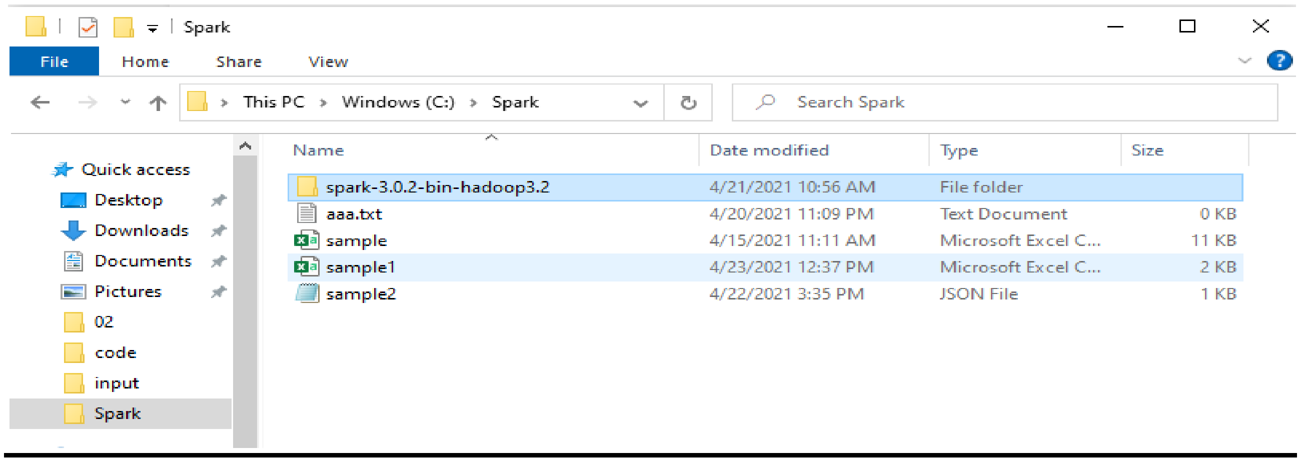
Step 1: In order to install spark, we need to have java on our system, if not first need to download java. Check if the java is present by giving a command java -version on command prompt.



```
Microsoft Windows [Version 10.0.19042.928]
(c) Microsoft Corporation. All rights reserved.

C:\Users\M1064288>java -version
java version "1.8.0_151"
Java(TM) SE Runtime Environment (build 1.8.0_151-b12)
Java HotSpot(TM) 64-Bit Server VM (build 25.151-b12, mixed mode)
```

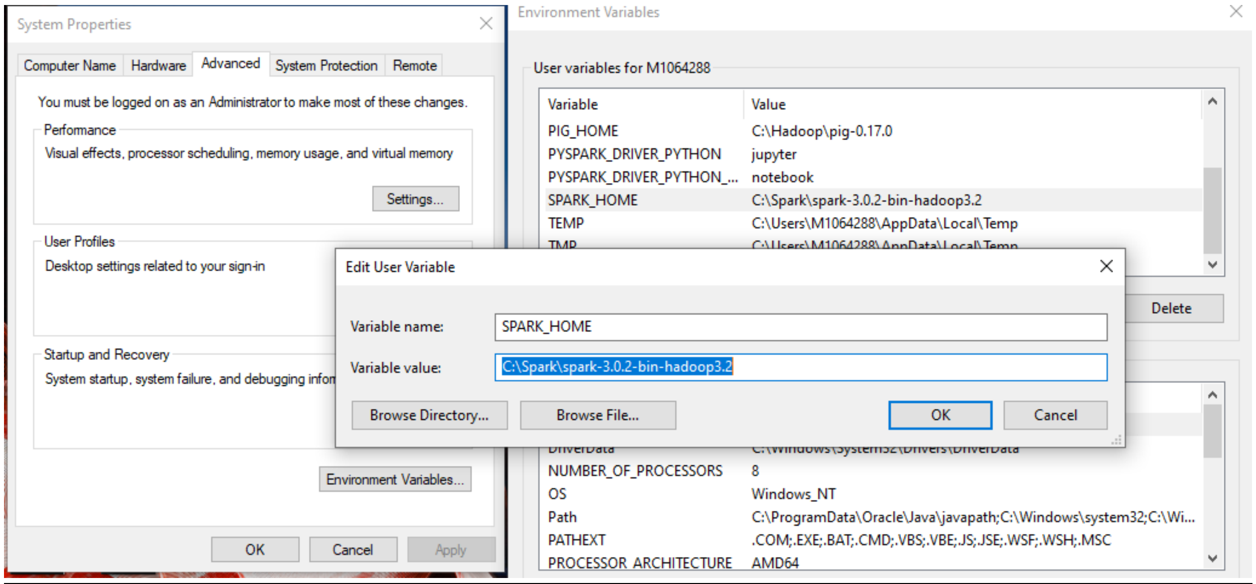
Step 2: To download the spark for windows click on the link https://mirrors.estointernet.in/apache/spark/spark-3.0.2/spark-3.0.2-bin-hadoop3.2.tgz

Step 3: After downloading extract the files to a folder.



Name	Date modified	Type	Size
spark-3.0.2-bin-hadoop3.2	4/21/2021 10:56 AM	File folder	
aaa.txt	4/20/2021 11:09 PM	Text Document	0 KB
sample	4/15/2021 11:11 AM	Microsoft Excel C...	11 KB
sample1	4/23/2021 12:37 PM	Microsoft Excel C...	2 KB
sample2	4/22/2021 3:35 PM	JSON File	1 KB

Step 4: Set environment variables for spark. If the Hadoop environment variable is not set earlier then need to create it as well because spark runs on top of Hadoop.



You must be logged on as an Administrator to make most of these changes.

Visual effects, processor scheduling, memory usage, and virtual memory

Desktop settings related to your sign-in

System startup, system failure, and debugging infor

User variables for M1064288

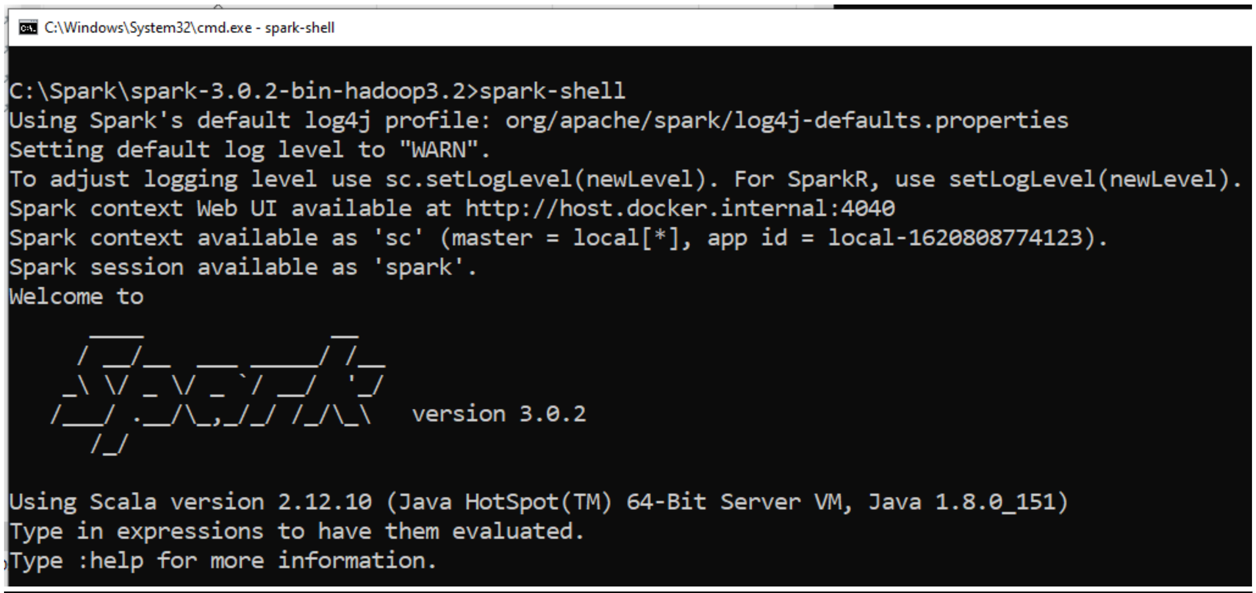
Variable	Value
PIG_HOME	C:\Hadoop\pig-0.17.0
PYSPARK_DRIVER_PYTHON	jupyter
PYSPARK_DRIVER_PYTHON_...	notebook
SPARK_HOME	C:\Spark\spark-3.0.2-bin-hadoop3.2
TEMP	C:\Users\M1064288\AppData\Local\Temp
TMP	C:\Users\M1064288\AppData\Local\Temp

Variable name: SPARK_HOME

Variable value: C:\Spark\spark-3.0.2-bin-hadoop3.2

Variable	Value
DriverData	C:\Windows\System32\Drivers\DriverData
NUMBER_OF_PROCESSORS	8
OS	Windows_NT
Path	C:\ProgramData\Oracle\Java\javapath;C:\Windows\system32;C:\Wi...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
PROCESSOR_ARCHITECTURE	AMD64

Step 5: Check if the installation is proper by providing spark-shell command in command prompt.



```
C:\Spark\spark-3.0.2-bin-hadoop3.2>spark-shell
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Spark context Web UI available at http://host.docker.internal:4040
Spark context available as 'sc' (master = local[*], app id = local-1620808774123).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 3.0.2
      /_/

Using Scala version 2.12.10 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_151)
Type in expressions to have them evaluated.
Type :help for more information.
```

Step 6: The installation of spark is complete. The version of spark installed is 3.0.2.

Ability to work with spark core using RDD's

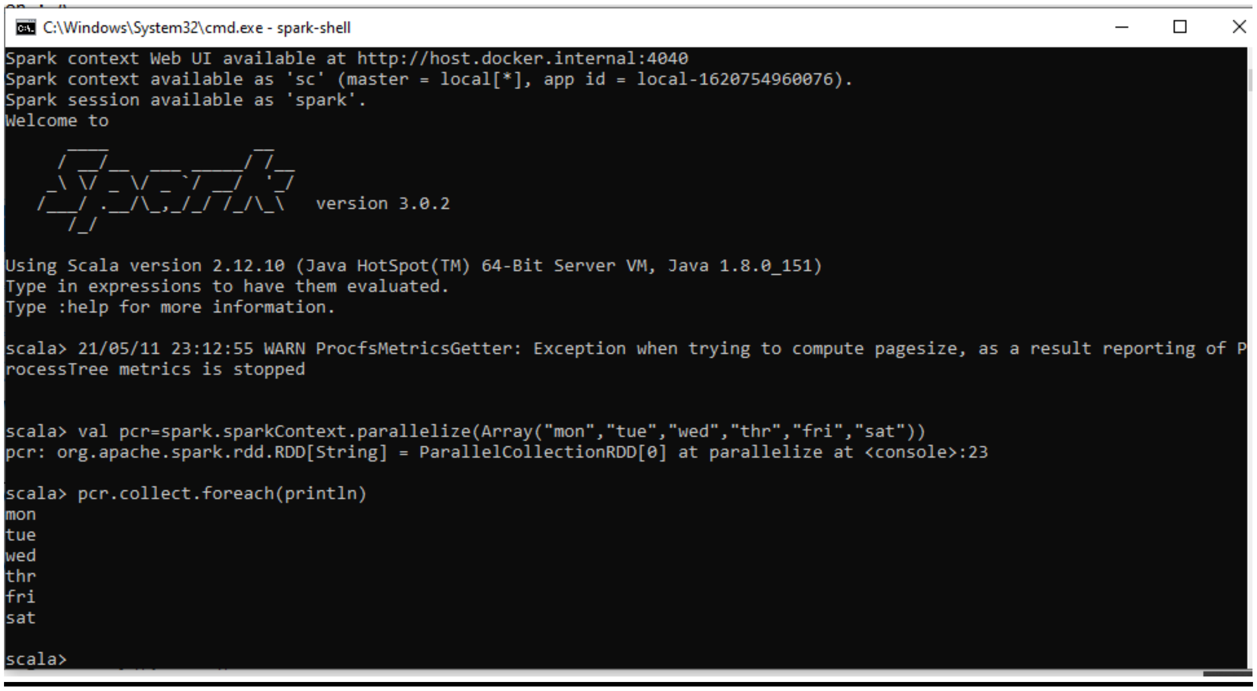
RDD (Resilient Distributed Dataset)

RDD is the data structure used in Spark.

Features:
1. **In memory**: The data is stored in ram so it provides faster way to access data
2. **Fault tolerance**: data can be retrieved if it is lost.
3. **Multiple executable nodes**: to prevent data loss data is stored in multiple data blocks.
4. **Lazy evaluation:** only when action is triggered you get output data.
5. **Immutable data:** cannot modify data, if we want to modify then will use transformation.

Creation of RDD: It can be created in three ways

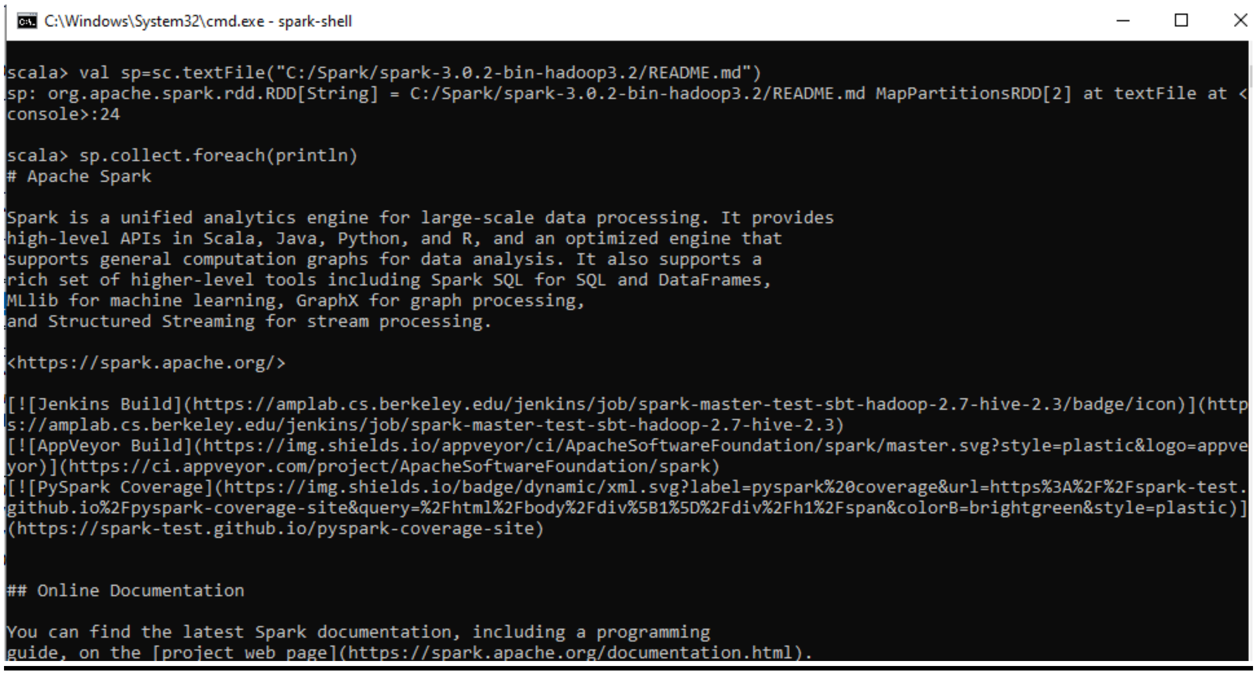
1. Parallelized Collection: RDD can be created by taking an existing collection in the program and passing it to spark context parallelize () method.



C:\Windows\System32\cmd.exe - spark-shell

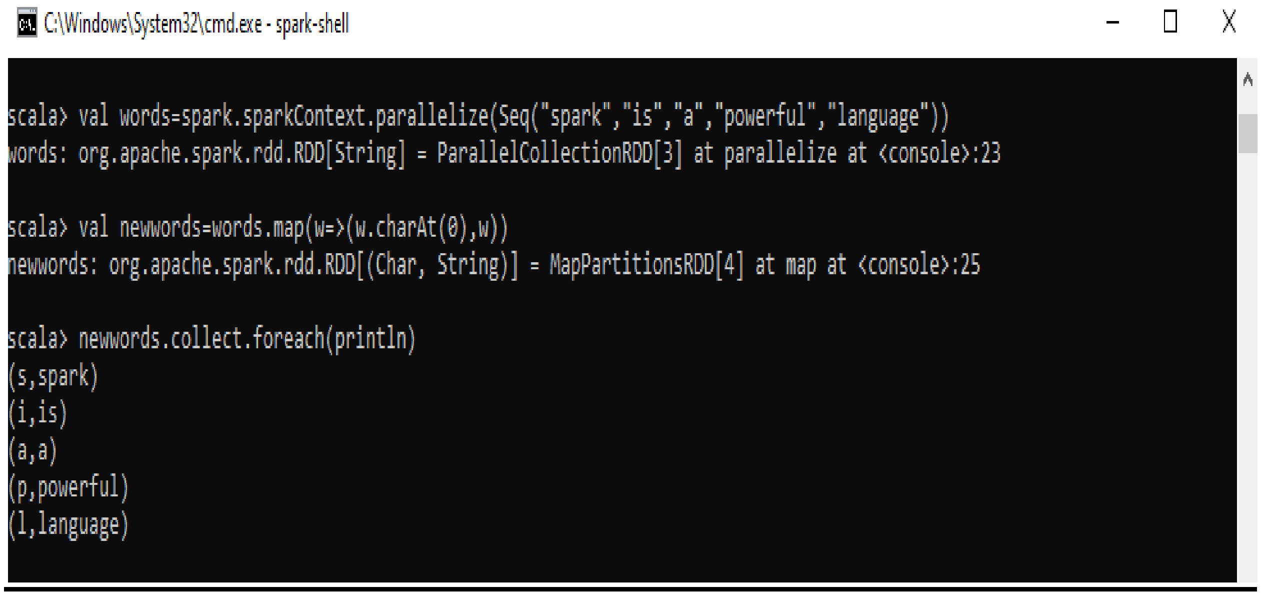
```
Spark context Web UI available at http://host.docker.internal:4040
Spark context available as 'sc' (master = local[*], app id = local-1620754960076).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 3.0.2
      /_/

Using Scala version 2.12.10 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_151)
Type in expressions to have them evaluated.
Type :help for more information.

scala> 21/05/11 23:12:55 WARN ProcfsMetricsGetter: Exception when trying to compute pagesize, as a result reporting of P
rocessTree metrics is stopped


scala> val pcr=spark.sparkContext.parallelize(Array("mon","tue","wed","thr","fri","sat"))
pcr: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[0] at parallelize at <console>:23

scala> pcr.collect.foreach(println)
mon
tue
wed
thr
fri
sat

scala>
```

2.External Storage(Hive,text,Hbase etc.): RDD can be created by taking data from external storage and passing it to spark context ‘s object(sc) .


C:\Windows\System32\cmd.exe - spark-shell

```
scala> val sp=sc.textFile("C:/Spark/spark-3.0.2-bin-hadoop3.2/README.md")
sp: org.apache.spark.rdd.RDD[String] = C:/Spark/spark-3.0.2-bin-hadoop3.2/README.md MapPartitionsRDD[2] at textFile at <
console>:24

scala> sp.collect.foreach(println)
# Apache Spark

Spark is a unified analytics engine for large-scale data processing. It provides
high-level APIs in Scala, Java, Python, and R, and an optimized engine that
supports general computation graphs for data analysis. It also supports a
rich set of higher-level tools including Spark SQL for SQL and DataFrames,
MLlib for machine learning, GraphX for graph processing,
and Structured Streaming for stream processing.

<https://spark.apache.org/>

[![Jenkins Build](https://amplab.cs.berkeley.edu/jenkins/job/spark-master-test-sbt-hadoop-2.7-hive-2.3/badge/icon)](http
s://amplab.cs.berkeley.edu/jenkins/job/spark-master-test-sbt-hadoop-2.7-hive-2.3)
[![AppVeyor Build](https://img.shields.io/appveyor/ci/ApacheSoftwareFoundation/spark/master.svg?style=plastic&logo=appve
yor)](https://ci.appveyor.com/project/ApacheSoftwareFoundation/spark)
[![PySpark Coverage](https://img.shields.io/badge/dynamic/xml.svg?label=pyspark%20coverage&url=https%3A%2F%2Fspark-test.
github.io%2Fpyspark-coverage-site&query=%2Fhtml%2Fbody%2Fdiv%5B1%5D%2Fdiv%2Fh1%2Fspan&colorB=brightgreen&style=plastic)]
(https://spark-test.github.io/pyspark-coverage-site)


## Online Documentation

You can find the latest Spark documentation, including a programming
guide, on the [project web page](https://spark.apache.org/documentation.html).
```

3.**Existing RDD:** RDD can be created using the already created RDD using the spark Context .


C:\Windows\System32\cmd.exe - spark-shell

```
scala> val words=spark.sparkContext.parallelize(Seq("spark","is","a","powerful","language"))
words: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[3] at parallelize at <console>:23

scala> val newwords=words.map(w=>(w.charAt(0),w))
newwords: org.apache.spark.rdd.RDD[(Char, String)] = MapPartitionsRDD[4] at map at <console>:25

scala> newwords.collect.foreach(println)
(s,spark)
(i,is)
(a,a)
(p,powerful)
(l,language)
```

Ability to analyze data using RDD's and Dataframe

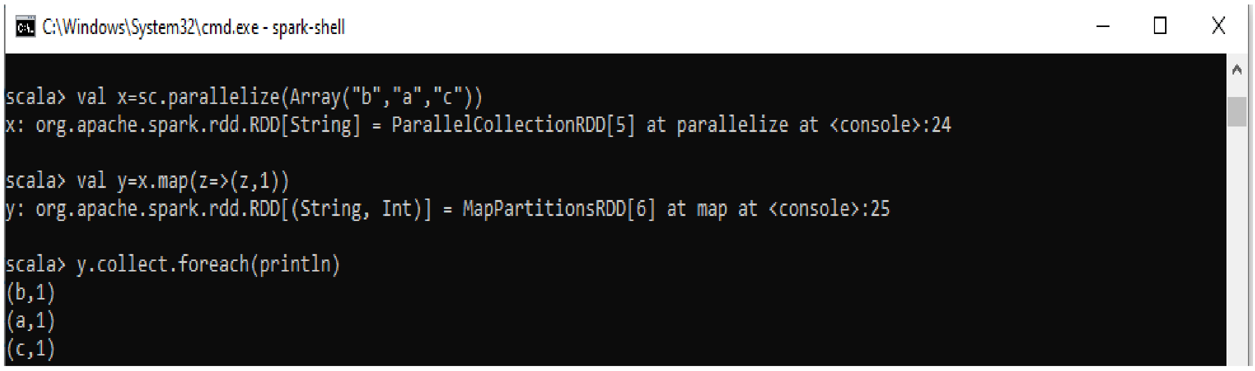
Transformation and action in RDD

RDD Transformations are Spark operations when executed on RDD, it results in a single or multiple new RDD's. There are two types of transformations,

1. Narrow Transformations
2. Wide Transformations

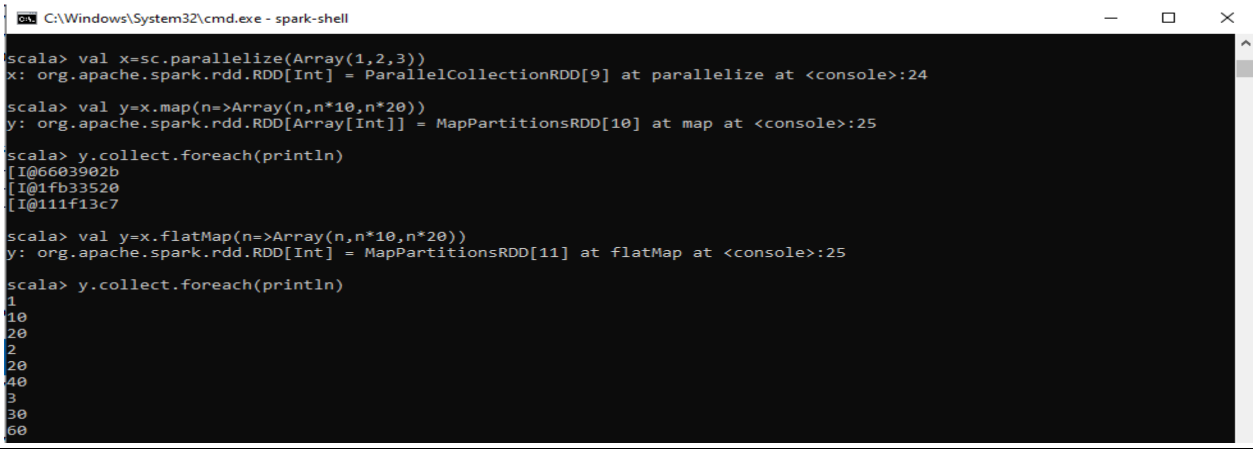
Narrow Transformations: It computes data that live on a single partition. Functions such as map(), flatMap(), filter(), union() are some examples of narrow transformation.

map(): This transformation function is used for mapping some values to RDD



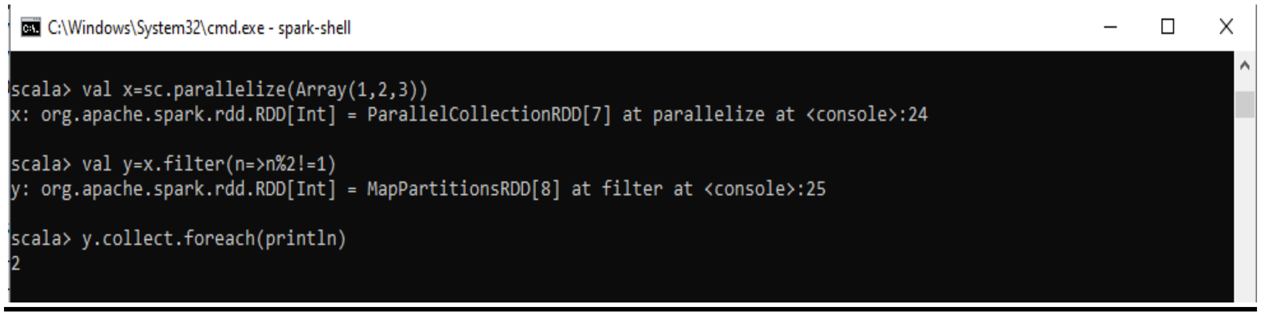
```
scala> val x=sc.parallelize(Array("b","a","c"))
x: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[5] at parallelize at <console>:24

scala> val y=x.map(z=>(z,1))
y: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[6] at map at <console>:25

scala> y.collect.foreach(println)
(b,1)
(a,1)
(c,1)
```

flatMap(): This transformation function flattens the data in the RDD



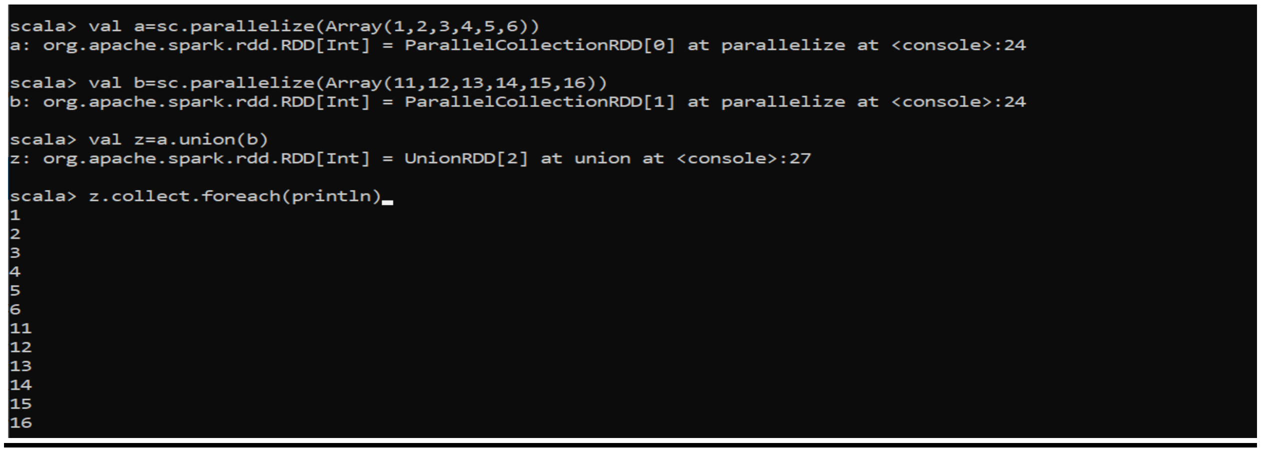
```
scala> val x=sc.parallelize(Array(1,2,3))
x: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[9] at parallelize at <console>:24

scala> val y=x.map(n=>Array(n,n*10,n*20))
y: org.apache.spark.rdd.RDD[Array[Int]] = MapPartitionsRDD[10] at map at <console>:25

scala> y.collect.foreach(println)
[I@6603902b
[I@1fb33520
[I@111f13c7

scala> val y=x.flatMap(n=>Array(n,n*10,n*20))
y: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[11] at flatMap at <console>:25

scala> y.collect.foreach(println)
1
10
20
2
20
40
3
30
60
```

filter(): This transformation function is used to filter certain data in RDD.



```
scala> val x=sc.parallelize(Array(1,2,3))
x: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[7] at parallelize at <console>:24

scala> val y=x.filter(n=>n%2!=1)
y: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[8] at filter at <console>:25

scala> y.collect.foreach(println)
2
```

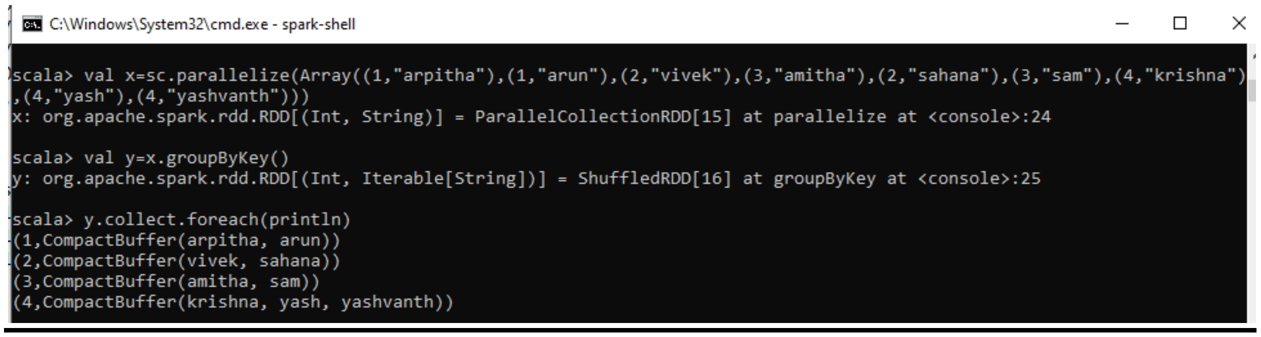
union(): This transformation function is used to merge two RDD's.



```
scala> val a=sc.parallelize(Array(1,2,3,4,5,6))
a: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[0] at parallelize at <console>:24

scala> val b=sc.parallelize(Array(11,12,13,14,15,16))
b: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[1] at parallelize at <console>:24

scala> val z=a.union(b)
z: org.apache.spark.rdd.RDD[Int] = UnionRDD[2] at union at <console>:27

scala> z.collect.foreach(println)
1
2
3
4
5
6
11
12
13
14
15
16
```

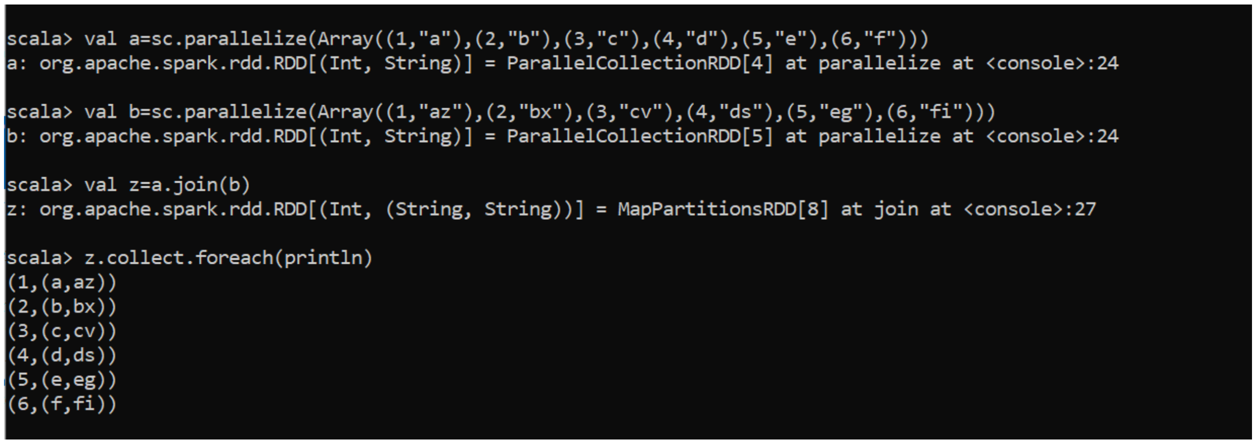
Wide Transformations: It computes data that live on a multiple partition. Functions such as groupByKey(), join(), repartition() are some examples of wide transformation.

groupByKey(): This transformation function is used to combine the values in RDD based on their key value.



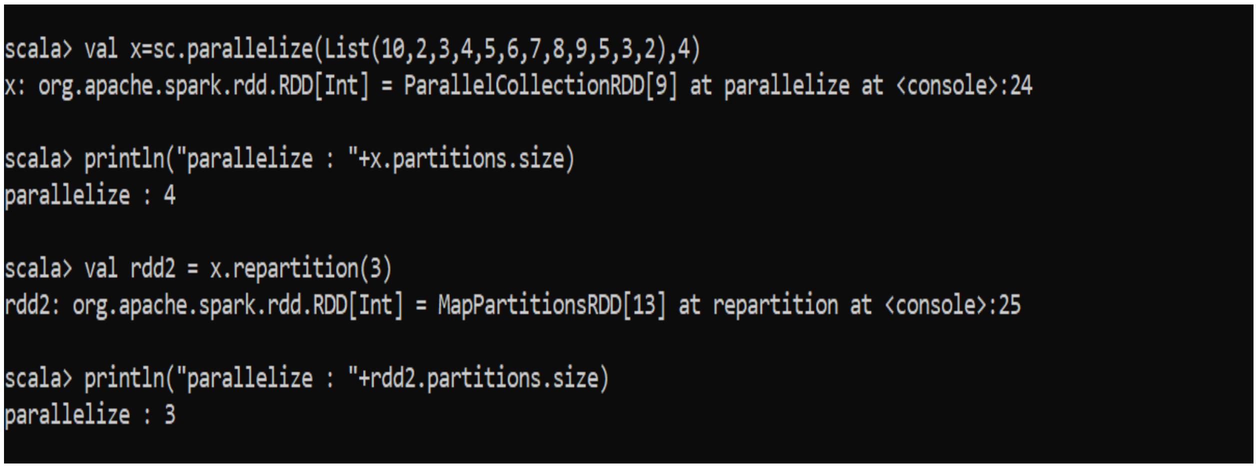
```
scala> val x=sc.parallelize(Array((1,"arpitha"),(1,"arun"),(2,"vivek"),(3,"amitha"),(2,"sahana"),(3,"sam"),(4,"krishna")
,(4,"yash"),(4,"yashvanth")))
x: org.apache.spark.rdd.RDD[(Int, String)] = ParallelCollectionRDD[15] at parallelize at <console>:24

scala> val y=x.groupByKey()
y: org.apache.spark.rdd.RDD[(Int, Iterable[String])] = ShuffledRDD[16] at groupByKey at <console>:25

scala> y.collect.foreach(println)
(1,CompactBuffer(arpitha, arun))
(2,CompactBuffer(vivek, sahana))
(3,CompactBuffer(amitha, sam))
(4,CompactBuffer(krishna, yash, yashvanth))
```

join(): This transformation function is used to join the values of two RDD based on key.



```
scala> val a=sc.parallelize(Array((1,"a"),(2,"b"),(3,"c"),(4,"d"),(5,"e"),(6,"f")))
a: org.apache.spark.rdd.RDD[(Int, String)] = ParallelCollectionRDD[4] at parallelize at <console>:24

scala> val b=sc.parallelize(Array((1,"az"),(2,"bx"),(3,"cv"),(4,"ds"),(5,"eg"),(6,"fi")))
b: org.apache.spark.rdd.RDD[(Int, String)] = ParallelCollectionRDD[5] at parallelize at <console>:24

scala> val z=a.join(b)
z: org.apache.spark.rdd.RDD[(Int, (String, String))] = MapPartitionsRDD[8] at join at <console>:27

scala> z.collect.foreach(println)
(1,(a,az))
(2,(b,bx))
(3,(c,cv))
(4,(d,ds))
(5,(e,eg))
(6,(f,fi))
```

repartition(): This transformation function is used to repartition the RDD that had partitioned already.



```
scala> val x=sc.parallelize(List(10,2,3,4,5,6,7,8,9,5,3,2),4)
x: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[9] at parallelize at <console>:24

scala> println("parallelize : "+x.partitions.size)
parallelize : 4

scala> val rdd2 = x.repartition(3)
rdd2: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[13] at repartition at <console>:25

scala> println("parallelize : "+rdd2.partitions.size)
parallelize : 3
```

Similarly, there are lot of transformations such as groupBy(), sample(), distinct(),keyBy() etc. that performs their respective functions in transforming the RDD’s.

Actions

RDD actions are operations that return the values. Some of the actions methods are,

aggregate(): This action method is used for the aggregating values of RDD.

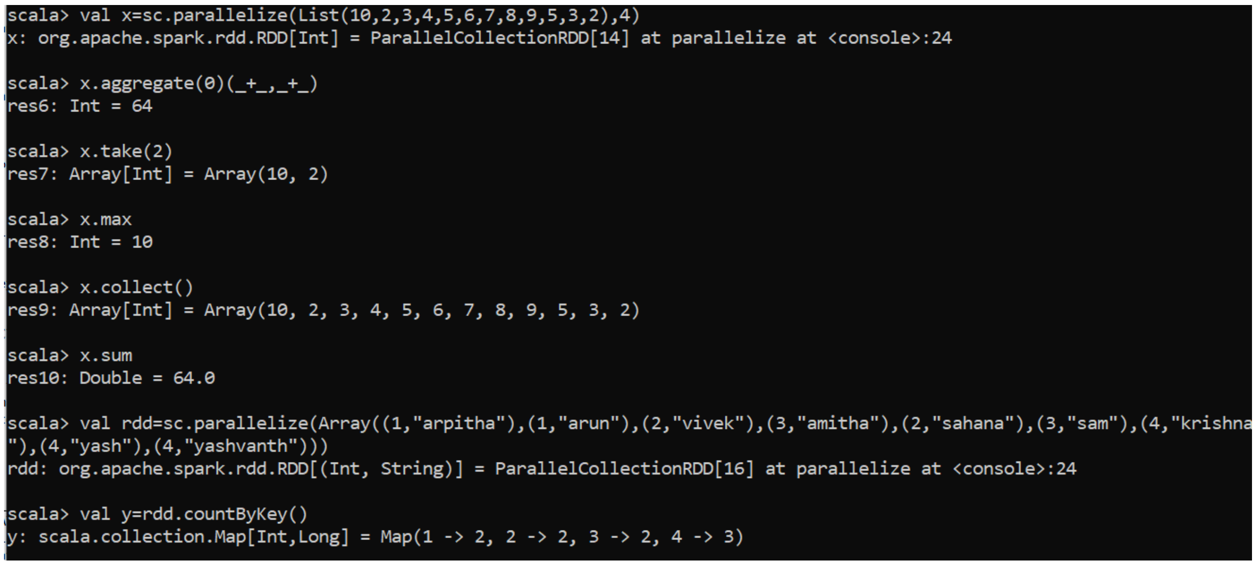
take(): This action method is used for displaying number of values from RDD that are mentioned as parameter in method.

max(): This action method is used for the returning the maximum value present of RDD.

collect(): This action method is used for the returning all the values present of RDD.

sum(): This action method is used for the returning the sum of values present of RDD.

countByKey(): This action method is used for the returning the map that contains key associated with number of values with that key in RDD.



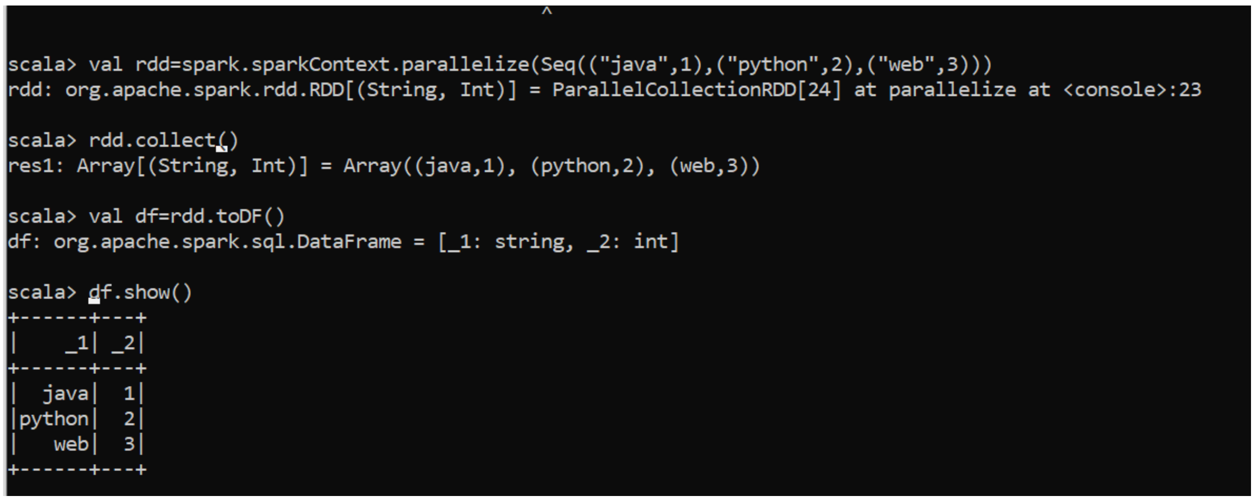
```
scala> val x=sc.parallelize(List(10,2,3,4,5,6,7,8,9,5,3,2),4)
x: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[14] at parallelize at <console>:24

scala> x.aggregate(0)(_+_,_+_)
res6: Int = 64

scala> x.take(2)
res7: Array[Int] = Array(10, 2)

scala> x.max
res8: Int = 10

scala> x.collect()
res9: Array[Int] = Array(10, 2, 3, 4, 5, 6, 7, 8, 9, 5, 3, 2)

scala> x.sum
res10: Double = 64.0

scala> val rdd=sc.parallelize(Array((1,"arpitha"),(1,"arun"),(2,"vivek"),(3,"amitha"),(2,"sahana"),(3,"sam"),(4,"krishna
"),(4,"yash"),(4,"yashvanth")))
rdd: org.apache.spark.rdd.RDD[(Int, String)] = ParallelCollectionRDD[16] at parallelize at <console>:24

scala> val y=rdd.countByKey()
y: scala.collection.Map[Int,Long] = Map(1 -> 2, 2 -> 2, 3 -> 2, 4 -> 3)
```

Dataframe

A Spark DataFrame is considered as a distributed collection of data which is organized under named columns.
The DataFrame API is a part of the Spark SQL module. The API provides an easy way to work with data within the Spark SQL framework while integrating with general-purpose languages like Java, Python, and Scala.

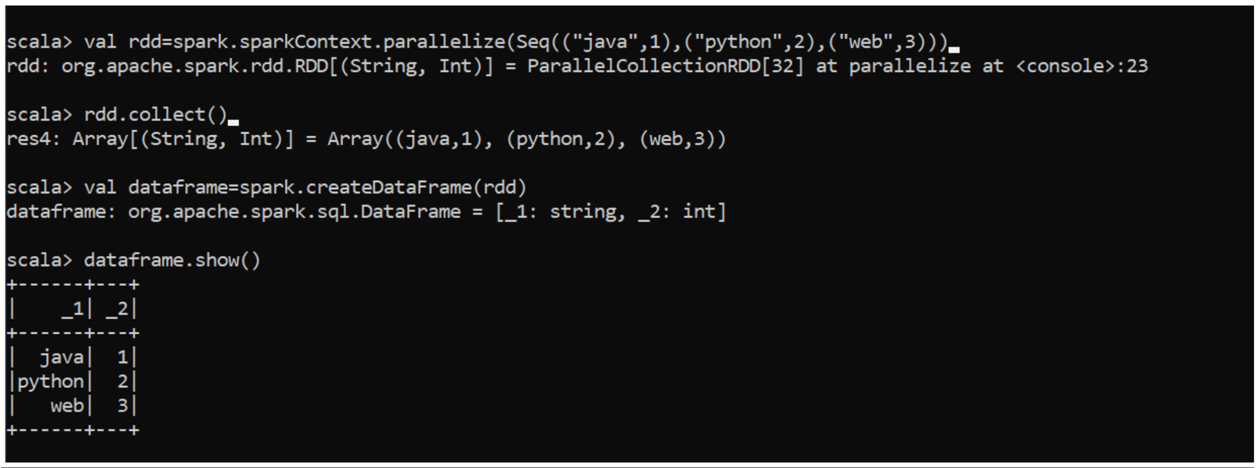
Creation of DataFrame: DataFrame can be created in multiple ways,

1. toDF(): Existing rdd can be converted to dataframe using toDF()



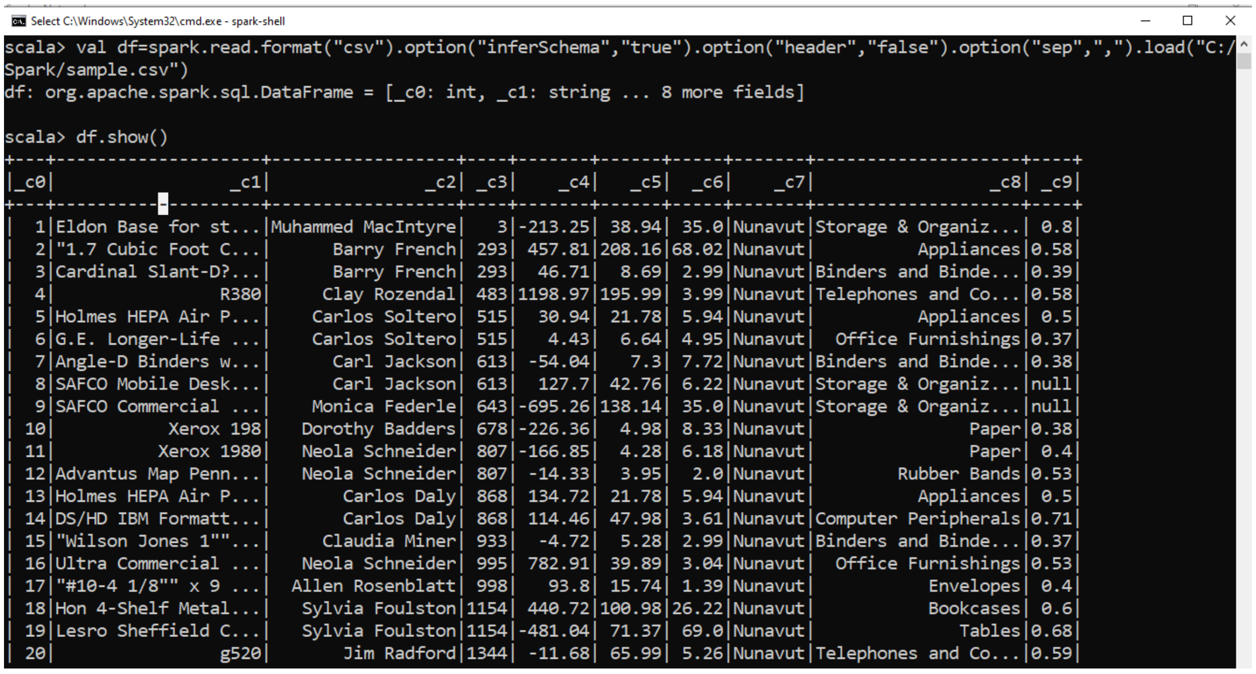
```
scala> val rdd=spark.sparkContext.parallelize(Seq(("java",1),("python",2),("web",3)))
rdd: org.apache.spark.rdd.RDD[(String, Int)] = ParallelCollectionRDD[24] at parallelize at <console>:23

scala> rdd.collect()
res1: Array[(String, Int)] = Array((java,1), (python,2), (web,3))

scala> val df=rdd.toDF()
df: org.apache.spark.sql.DataFrame = [_1: string, _2: int]

scala> df.show()
+------+---+
|    _1| _2|
+------+---+
|  java|  1|
|python|  2|
|   web|  3|
+------+---+
```

2.createDataFrame(): rdd can be converted to dataframe using createDataFrame() along with spark session object.



```
scala> val rdd=spark.sparkContext.parallelize(Seq(("java",1),("python",2),("web",3)))
rdd: org.apache.spark.rdd.RDD[(String, Int)] = ParallelCollectionRDD[32] at parallelize at <console>:23

scala> rdd.collect()
res4: Array[(String, Int)] = Array((java,1), (python,2), (web,3))

scala> val dataframe=spark.createDataFrame(rdd)
dataframe: org.apache.spark.sql.DataFrame = [_1: string, _2: int]

scala> dataframe.show()
+------+---+
|    _1| _2|
+------+---+
|  java|  1|
|python|  2|
|   web|  3|
+------+---+
```

3. From external sources(text,hive,csv,json): To create dataframes using external source, the options associated with the kind of the source should be known.
Syntax: spark.read.format(filetype).options().....load(file location)

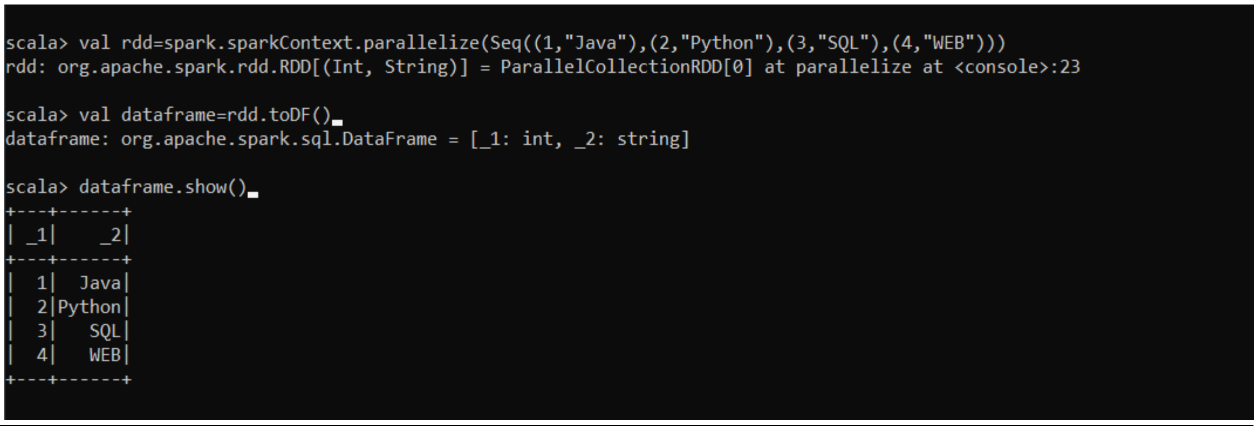


```
scala> val df=spark.read.format("csv").option("inferSchema","true").option("header","false").option("sep",",").load("C:/
Spark/sample.csv")
df: org.apache.spark.sql.DataFrame = [_c0: int, _c1: string ... 8 more fields]

scala> df.show()
+---+--------------------+------------------+----+-------+------+-----+-------+--------------------+----+
|_c0|                 _c1|               _c2| _c3|    _c4|   _c5|  _c6|    _c7|                 _c8| _c9|
+---+--------------------+------------------+----+-------+------+-----+-------+--------------------+----+
|  1|Eldon Base for st...|Muhammed MacIntyre|   3|-213.25| 38.94| 35.0|Nunavut|Storage & Organiz...| 0.8|
|  2|"1.7 Cubic Foot C...|      Barry French| 293| 457.81|208.16|68.02|Nunavut|          Appliances|0.58|
|  3|Cardinal Slant-D?...|      Barry French| 293|  46.71|  8.69| 2.99|Nunavut|Binders and Binde...|0.39|
|  4|                R380|     Clay Rozendal| 483|1198.97|195.99| 3.99|Nunavut|Telephones and Co...|0.58|
|  5|Holmes HEPA Air P...|    Carlos Soltero| 515|  30.94| 21.78| 5.94|Nunavut|          Appliances| 0.5|
|  6|G.E. Longer-Life ...|    Carlos Soltero| 515|   4.43|  6.64| 4.95|Nunavut|  Office Furnishings|0.37|
|  7|Angle-D Binders w...|      Carl Jackson| 613| -54.04|   7.3| 7.72|Nunavut|Binders and Binde...|0.38|
|  8|SAFCO Mobile Desk...|      Carl Jackson| 613|  127.7| 42.76| 6.22|Nunavut|Storage & Organiz...|null|
|  9|SAFCO Commercial ...|    Monica Federle| 643|-695.26|138.14| 35.0|Nunavut|Storage & Organiz...|null|
| 10|           Xerox 198|   Dorothy Badders| 678|-226.36|  4.98| 8.33|Nunavut|               Paper|0.38|
| 11|          Xerox 1980|   Neola Schneider| 807|-166.85|  4.28| 6.18|Nunavut|               Paper| 0.4|
| 12|Advantus Map Penn...|   Neola Schneider| 807| -14.33|  3.95|  2.0|Nunavut|        Rubber Bands|0.53|
| 13|Holmes HEPA Air P...|       Carlos Daly| 868| 134.72| 21.78| 5.94|Nunavut|          Appliances| 0.5|
| 14|DS/HD IBM Formatt...|       Carlos Daly| 868| 114.46| 47.98| 3.61|Nunavut|Computer Peripherals|0.71|
| 15|"Wilson Jones 1""...|     Claudia Miner| 933|  -4.72|  5.28| 2.99|Nunavut|Binders and Binde...|0.37|
| 16|Ultra Commercial ...|   Neola Schneider| 995| 782.91| 39.89| 3.04|Nunavut|  Office Furnishings|0.53|
| 17|"#10-4 1/8"" x 9 ...|  Allen Rosenblatt| 998|   93.8| 15.74| 1.39|Nunavut|           Envelopes| 0.4|
| 18|Hon 4-Shelf Metal...|   Sylvia Foulston|1154| 440.72|100.98|26.22|Nunavut|           Bookcases| 0.6|
| 19|Lesro Sheffield C...|   Sylvia Foulston|1154|-481.04| 71.37| 69.0|Nunavut|              Tables|0.68|
| 20|                g520|       Jim Radford|1344| -11.68| 65.99| 5.26|Nunavut|Telephones and Co...|0.59|
```

Analyzing DataFrame :

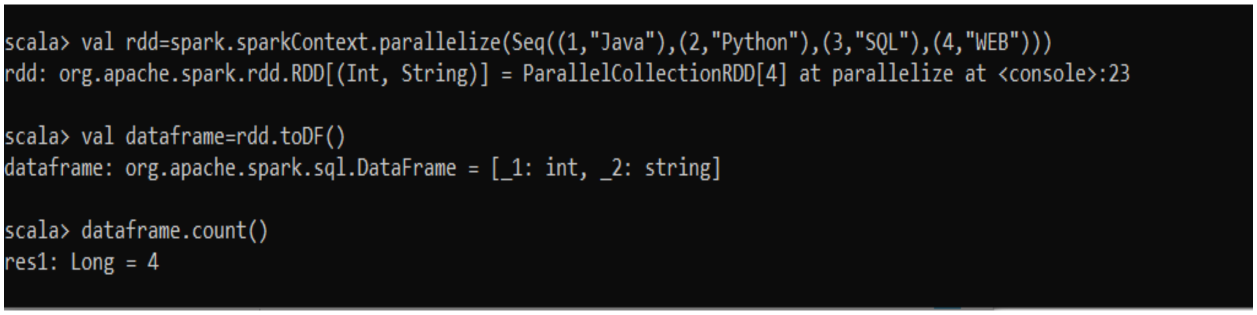
Dataframes can be analyzed using a set to methods.

1.show(): The show() is used to analyze what are the data present in the dataframe.



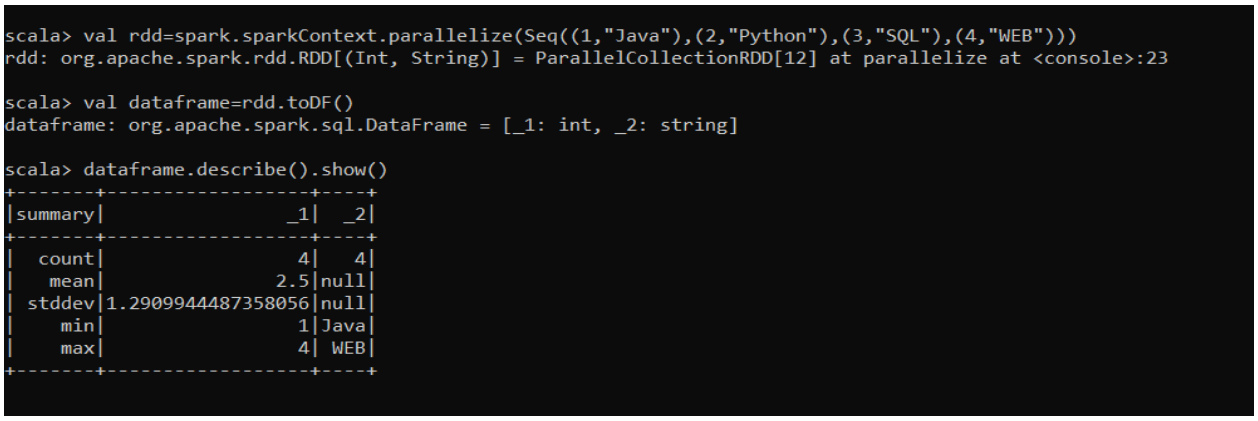
```
scala> val rdd=spark.sparkContext.parallelize(Seq((1,"Java"),(2,"Python"),(3,"SQL"),(4,"WEB")))
rdd: org.apache.spark.rdd.RDD[(Int, String)] = ParallelCollectionRDD[0] at parallelize at <console>:23

scala> val dataframe=rdd.toDF()
dataframe: org.apache.spark.sql.DataFrame = [_1: int, _2: string]

scala> dataframe.show()
+---+------+
| _1|    _2|
+---+------+
|  1|  Java|
|  2|Python|
|  3|   SQL|
|  4|   WEB|
+---+------+
```

2.count(): The count() is used to analyze the number of data present in the dataframe.



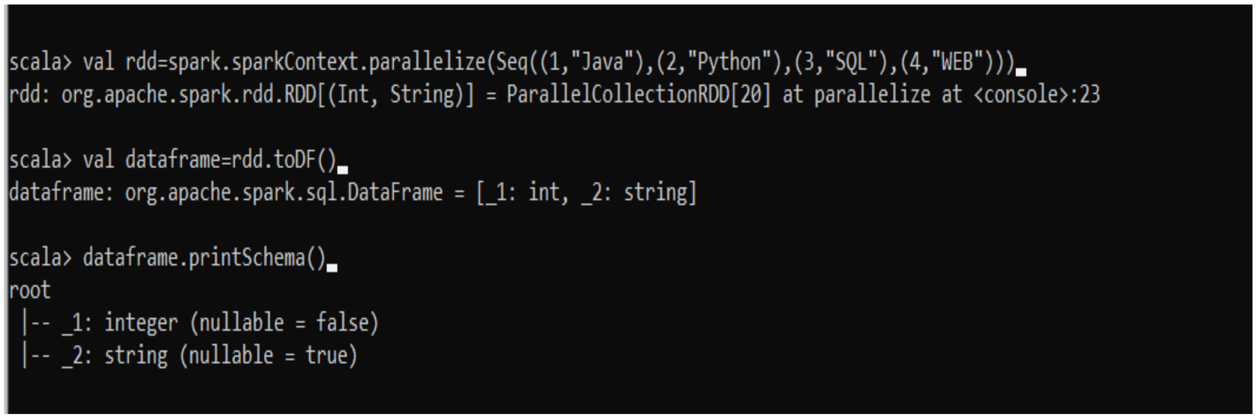
```
scala> val rdd=spark.sparkContext.parallelize(Seq((1,"Java"),(2,"Python"),(3,"SQL"),(4,"WEB")))
rdd: org.apache.spark.rdd.RDD[(Int, String)] = ParallelCollectionRDD[4] at parallelize at <console>:23

scala> val dataframe=rdd.toDF()
dataframe: org.apache.spark.sql.DataFrame = [_1: int, _2: string]

scala> dataframe.count()
res1: Long = 4
```

3.describe(): The describe() is used to analyze the statistical details on the data present in the dataframe.



```
scala> val rdd=spark.sparkContext.parallelize(Seq((1,"Java"),(2,"Python"),(3,"SQL"),(4,"WEB")))
rdd: org.apache.spark.rdd.RDD[(Int, String)] = ParallelCollectionRDD[12] at parallelize at <console>:23

scala> val dataframe=rdd.toDF()
dataframe: org.apache.spark.sql.DataFrame = [_1: int, _2: string]

scala> dataframe.describe().show()
+-------+------------------+----+
|summary|                _1|  _2|
+-------+------------------+----+
|  count|                 4|   4|
|   mean|               2.5|null|
| stddev|1.2909944487358056|null|
|    min|                 1|Java|
|    max|                 4| WEB|
+-------+------------------+----+
```

4.printSchema(): The printSchema() is used to analyze the schema of the rows and columns of the dataframe.



```
scala> val rdd=spark.sparkContext.parallelize(Seq((1,"Java"),(2,"Python"),(3,"SQL"),(4,"WEB")))
rdd: org.apache.spark.rdd.RDD[(Int, String)] = ParallelCollectionRDD[20] at parallelize at <console>:23

scala> val dataframe=rdd.toDF()
dataframe: org.apache.spark.sql.DataFrame = [_1: int, _2: string]

scala> dataframe.printSchema()
root
 |-- _1: integer (nullable = false)
 |-- _2: string (nullable = true)
```

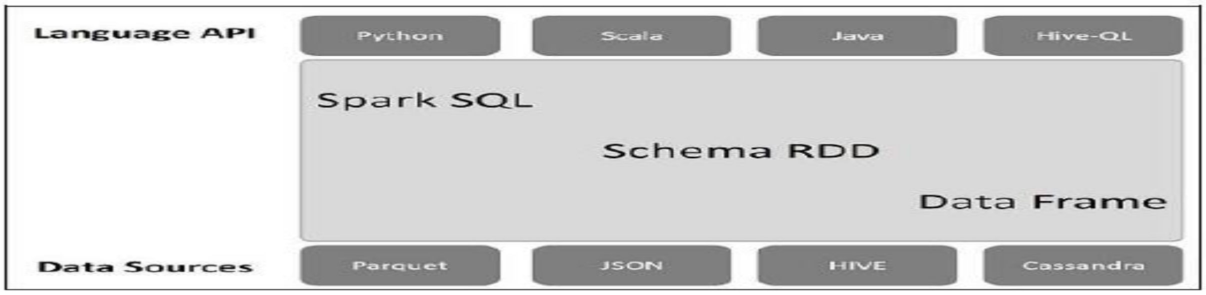
Ability to data analysis using Spark SQL

Features of Spark SQL

1. **Integrated** − Spark SQL lets you query structured data as a distributed dataset (RDD) in Spark. This tight integration makes it easy to run SQL queries alongside complex analytic algorithms.
2. **Unified Data Access** − Load and query data from a variety of sources. Schema-RDDs provide a single interface for efficiently working with structured data, including Apache Hive tables, parquet files and JSON files.
3. **Hive Compatibility** − Run unmodified Hive queries on existing warehouses.
4. **Standard Connectivity** − Spark SQL includes a server mode with industry standard JDBC and ODBC connectivity.

Ability to data analysis using Spark SQL

Spark SQL Architecture



Language API − Spark is compatible with different languages and Spark SQL. Generally, Spark SQL works on schemas, tables, and records. Therefore, we can use the Schema RDD as temporary table.

Data Sources − Usually the Data source for spark-core is a text file, Avro file, etc. However, the Data Sources for Spark SQL is different. Those are Parquet file, JSON document, HIVE tables, and Cassandra database.

To open spark sql shell

>>>spark-sql -S

here -S is used to skip the debug messages in silent mode.

To display all the databases following command is used

>>>show databases;

At the beginning the default database has been created. So the output will be default.

To create database

>>>CREATE DATABASE mysparkdb LOCATION '/home/ mindtree/ mysparkdb/';

If we don't specify the location then it will automatically sets the default location. By default the database is stored in HDFS.

To view the database name and location

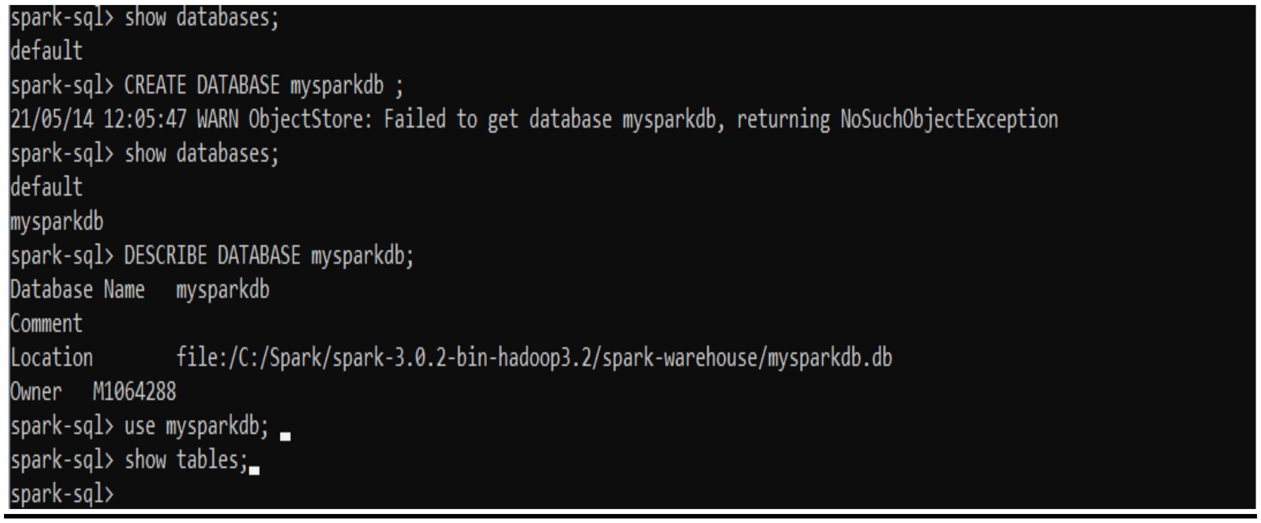
>>>DESCRIBE DATABASE mysparkdb;

To change the default setting at the session level

>>>SET spark.sql.warehouse.dir;

To use the database

>>use mysparkdb;



```
spark-sql> show databases;
default
spark-sql> CREATE DATABASE mysparkdb ;
21/05/14 12:05:47 WARN ObjectStore: Failed to get database mysparkdb, returning NoSuchObjectException
spark-sql> show databases;
default
mysparkdb
spark-sql> DESCRIBE DATABASE mysparkdb;
Database Name   mysparkdb
Comment
Location        file:/C:/Spark/spark-3.0.2-bin-hadoop3.2/spark-warehouse/mysparkdb.db
Owner   M1064288
spark-sql> use mysparkdb;
spark-sql> show tables;
spark-sql>
```

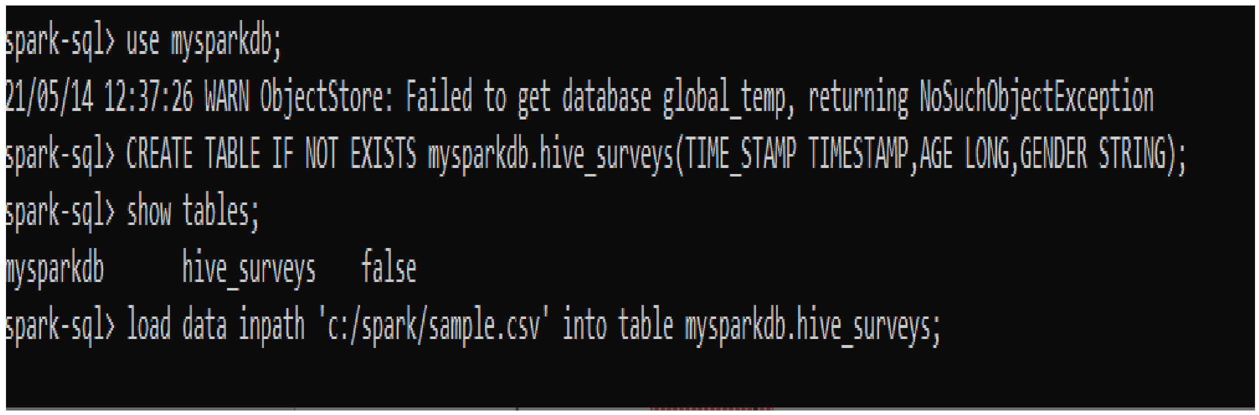
To create table having a name hive_surveys and columns time_stamp,age,gender
>>>CREATE TABLE IF NOTEXISTS mysparkdb.hive_surveys(TIME_STAMP TIMESTAMP, AGE LONG, GENDER STRING);

To view existing tables
>>>show TABLES;

To load the data present in csv file to the table
>>>LOAD DATA INPATH 'c:/spark.sample.csv' into TABLE mysparkdb.hive_surveys;

To view top 5 rows from hive_surveys table
>>>SELECT TIME_STAMP, AGE, GENDER from mysparkdb.hive_surveys limit 5;

To view schema of the table
>>>describe table extended mysparkdb.hive_surveys;



```
spark-sql> use mysparkdb;
21/05/14 12:37:26 WARN ObjectStore: Failed to get database global_temp, returning NoSuchObjectException
spark-sql> CREATE TABLE IF NOT EXISTS mysparkdb.hive_surveys(TIME_STAMP TIMESTAMP,AGE LONG,GENDER STRING);
spark-sql> show tables;
mysparkdb       hive_surveys    false
spark-sql> load data inpath 'c:/spark/sample.csv' into table mysparkdb.hive_surveys;
```

Ability to work with Spark Structured Streaming

Basic of streaming

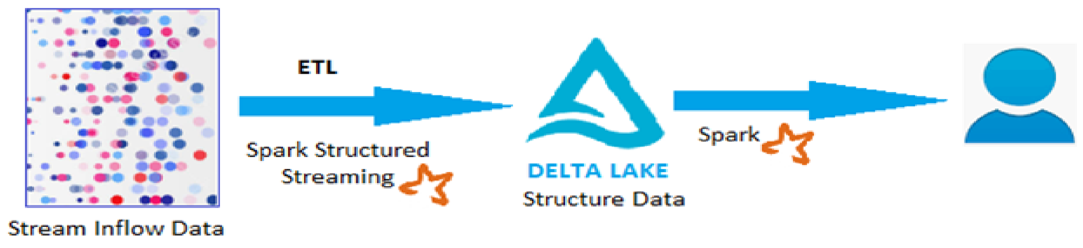
Streaming is a continuous inflow of data from sources. The data may be in different shapes — structured, unstructured or semi-structured.

Stream Inflow Data
Insight

Different Scenarios and respective data pipeline creation

In real life there might be different use cases and solution will be based on the types and requirements.

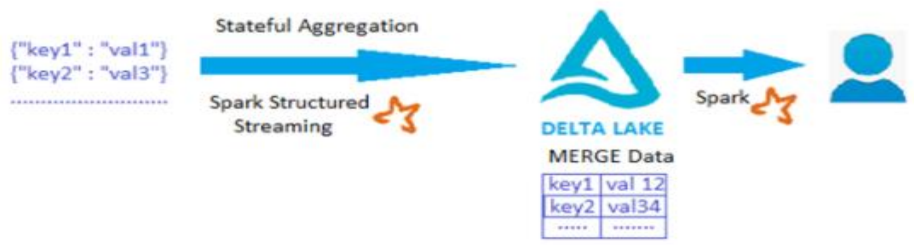
1. **Common etl processing**
 This is the very basic use case where Unstructured or Semi-Structured Stream inflow data has to be processed and store in a Structured manner for further analysis and reporting, and the expectation of the output processed data is not real time. We can consider Latency = Few Minutes.



In this use case we can use Structured Streaming processing with Default Trigger to create our ETL pipeline and store the processed structured data into Delta Lake, or we can keep the data in Parquet or ORC.

2. **Dealing with key-value data**

Another common use case, where input is a key-value pair stream and finally we need to collect the aggregated values (count, sum etc.) on each keys. Here the output data will be used for periodic reporting job where Latency = Few Minutes. In that case instead of Cassandra we can use Delta Lake with MERGE operation, which will be combined with structured streaming Stateful operation. The foreachBatch will enable to track updates from each processing.



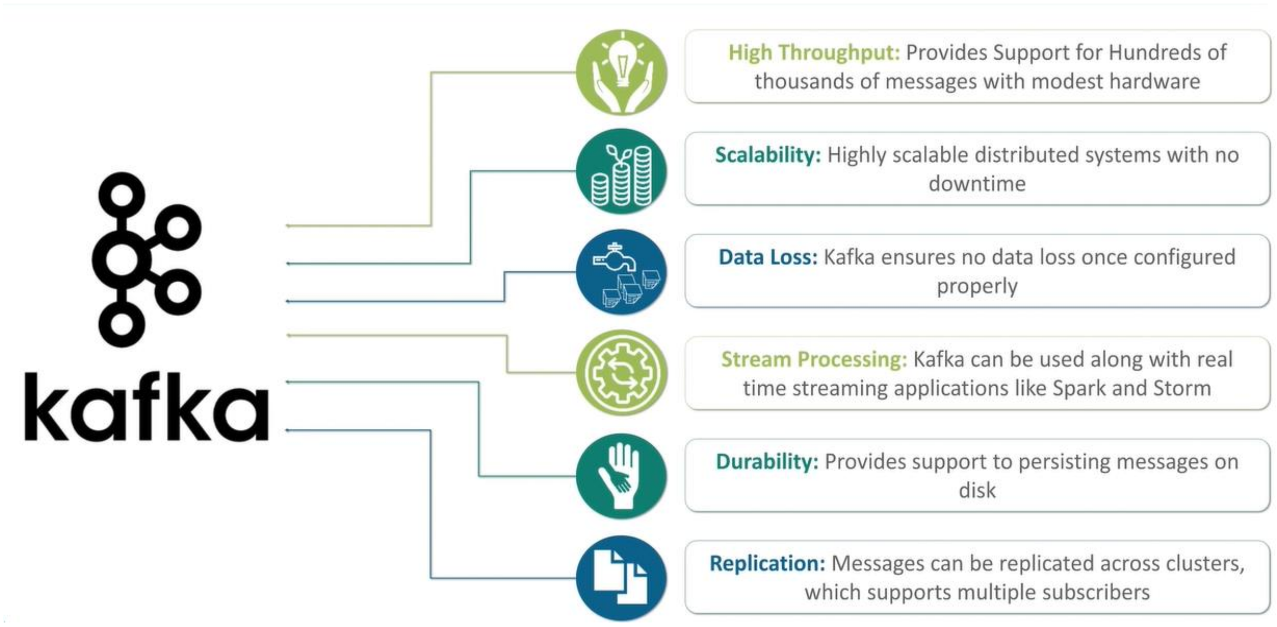
Able to integrate Apache Kafka with Structured Streaming

Introduction to Kafka: Apache kafka is a distributed **publish-subscribe** messaging system. Kafka is fast, scalable, durable, fault-tolerant and distributed by design. Kafka is a open source software which provides a framework for storing, reading and analyzing streaming data. It is used to handle high volume of data and enables us to pass messages from one end-point to another.

Kafka Terminologies

1. **Producer:** A producer can be any application who can publish messages to a topic.
2. **Consumer:** A consumer can be any application that subscribes to a topic and consumes the messages.
3. **Partition:** Topics are broken up into ordered commit logs called partitions.
4. **Broker:** A kafka broker allows consumers to fetch messages by topic, partition and offset. Kafka brokers create a kafka cluster by sharing information between each directly or indirectly using Zookeeper.
5. **Topic:** A topic is a feed name to which records/messages are published.
6. **Zookeeper:** Zookeeper is used to managing and coordinating kafka broker.

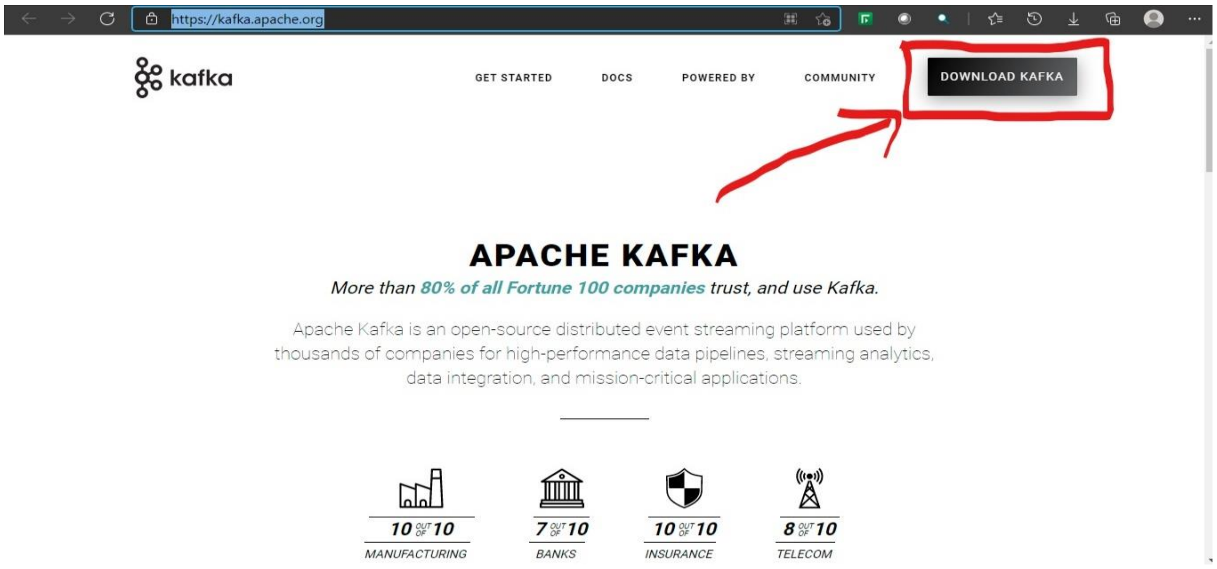
Kafka Features


kafka
High Throughput: Provides Support for Hundreds of thousands of messages with modest hardware
Scalability: Highly scalable distributed systems with no downtime
Data Loss: Kafka ensures no data loss once configured properly
Stream Processing: Kafka can be used along with real time streaming applications like Spark and Storm
Durability: Provides support to persisting messages on disk
Replication: Messages can be replicated across clusters, which supports multiple subscribers

Installing Kafka

Step 1: Open the link : https://kafka.apache.org (Apache Kafka)

Step 2: Click on Download Kafka



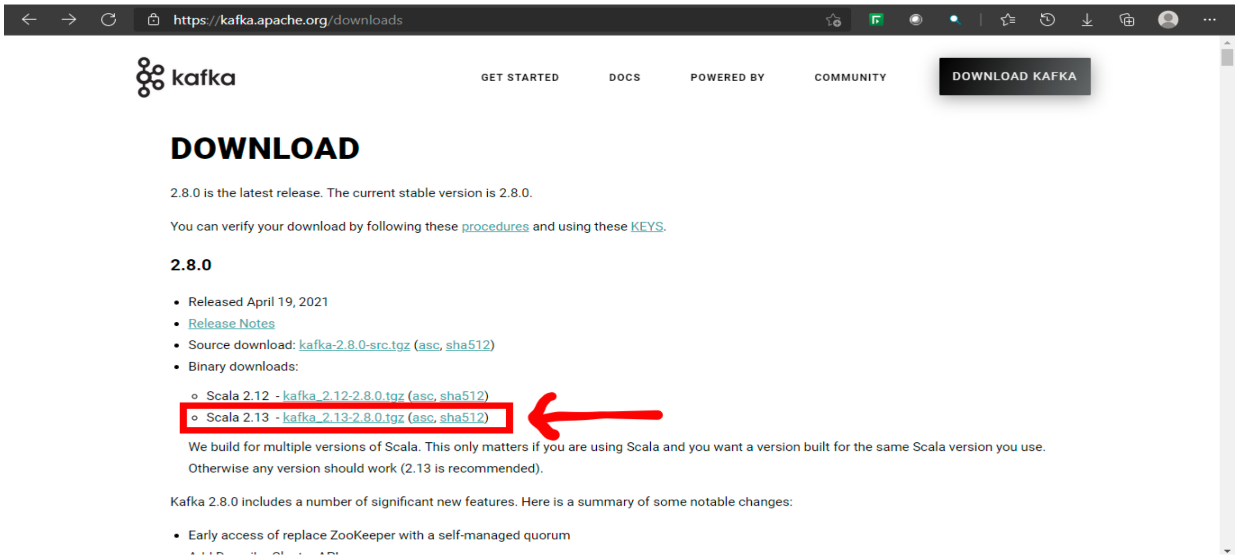
APACHE KAFKA

More than 80% of all Fortune 100 companies trust, and use Kafka.

Apache Kafka is an open-source distributed event streaming platform used by thousands of companies for high-performance data pipelines, streaming analytics, data integration, and mission-critical applications.

10 OUT OF 10 MANUFACTURING
7 OUT OF 10 BANKS
10 OUT OF 10 INSURANCE
8 OUT OF 10 TELECOM

Step 3: From the latest version, download binary file.



DOWNLOAD

2.8.0 is the latest release. The current stable version is 2.8.0.

You can verify your download by following these procedures and using these KEYS.

2.8.0

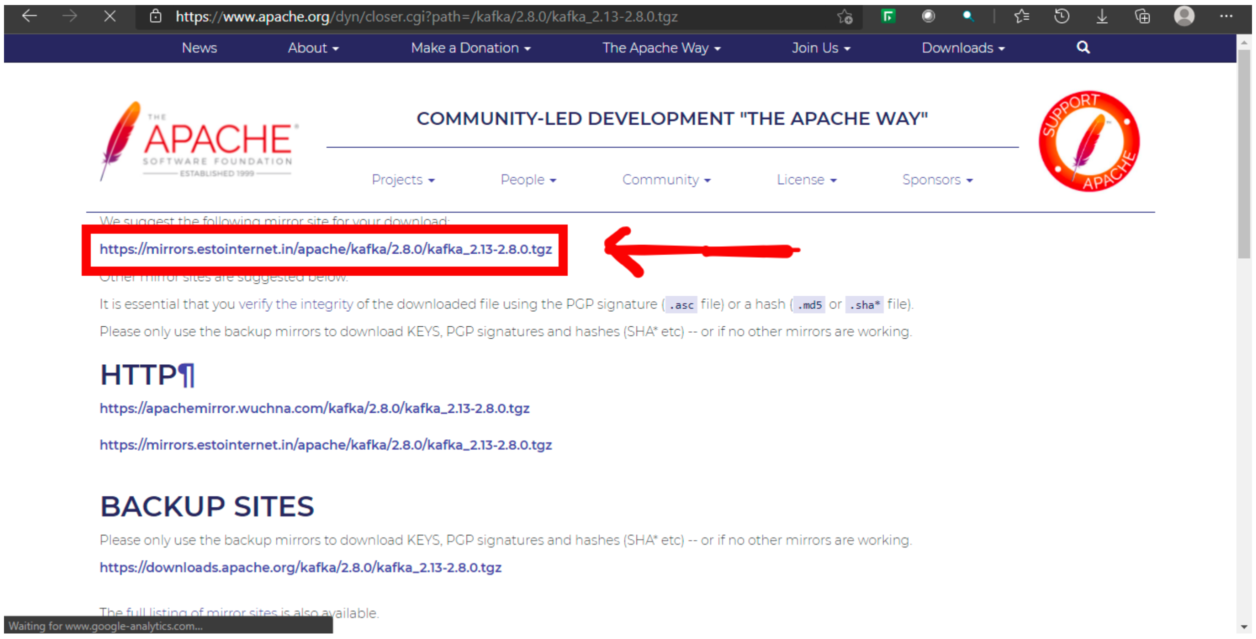
- Released April 19, 2021
- Release Notes
- Source download: kafka-2.8.0-src.tgz (asc, sha512)
- Binary downloads:
  - Scala 2.12 - kafka_2.12-2.8.0.tgz (asc, sha512)
  - Scala 2.13 - kafka_2.13-2.8.0.tgz (asc, sha512)

  We build for multiple versions of Scala. This only matters if you are using Scala and you want a version built for the same Scala version you use. Otherwise any version should work (2.13 is recommended).

Kafka 2.8.0 includes a number of significant new features. Here is a summary of some notable changes:

- Early access of replace ZooKeeper with a self-managed quorum

Step 4: Download the **.tgz** compressed file.


https://www.apache.org/dyn/closer.cgi?path=/kafka/2.8.0/kafka_2.13-2.8.0.tgz
News
About
Make a Donation
The Apache Way
Join Us
Downloads
THE APACHE SOFTWARE FOUNDATION
ESTABLISHED 1999
COMMUNITY-LED DEVELOPMENT "THE APACHE WAY"
Projects
People
Community
License
Sponsors
SUPPORT APACHE
Waiting for www.google-analytics.com...

We suggest the following mirror site for your download:

https://mirrors.estointernet.in/apache/kafka/2.8.0/kafka_2.13-2.8.0.tgz

Other mirror sites are suggested below.

It is essential that you verify the integrity of the downloaded file using the PGP signature (`.asc` file) or a hash (`.md5` or `.sha*` file).

Please only use the backup mirrors to download KEYS, PGP signatures and hashes (SHA* etc) -- or if no other mirrors are working.

HTTP¶

https://apachemirror.wuchna.com/kafka/2.8.0/kafka_2.13-2.8.0.tgz

https://mirrors.estointernet.in/apache/kafka/2.8.0/kafka_2.13-2.8.0.tgz

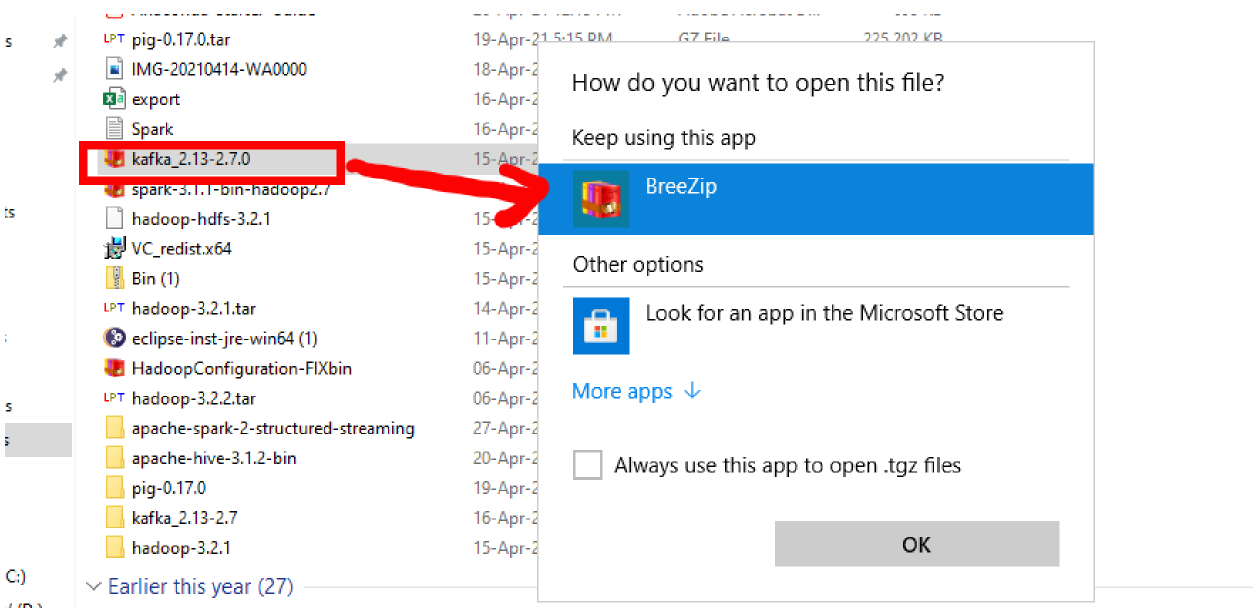
BACKUP SITES

Please only use the backup mirrors to download KEYS, PGP signatures and hashes (SHA* etc) -- or if no other mirrors are working.

https://downloads.apache.org/kafka/2.8.0/kafka_2.13-2.8.0.tgz

The full listing of mirror sites is also available.

Step 5: After downloaded, De-compress the kafka_2.13-2.7.0.tgz file using any software.



pig-0.17.0.tar 19-Apr-21 5:15 PM GZ File 225,202 KB
IMG-20210414-WA0000 18-Apr-2
export 16-Apr-2
Spark 16-Apr-2
kafka_2.13-2.7.0 15-Apr-2
spark-3.1.1-bin-hadoop2.7
hadoop-hdfs-3.2.1 15-Apr-2
VC_redist.x64 15-Apr-2
Bin (1) 15-Apr-2
hadoop-3.2.1.tar 14-Apr-2
eclipse-inst-jre-win64 (1) 11-Apr-2
HadoopConfiguration-FIXbin 06-Apr-2
hadoop-3.2.2.tar 06-Apr-2
apache-spark-2-structured-streaming 27-Apr-2
apache-hive-3.1.2-bin 20-Apr-2
pig-0.17.0 19-Apr-2
kafka_2.13-2.7 16-Apr-2
hadoop-3.2.1 15-Apr-2
Earlier this year (27)

How do you want to open this file?

Keep using this app

BreeZip

Other options

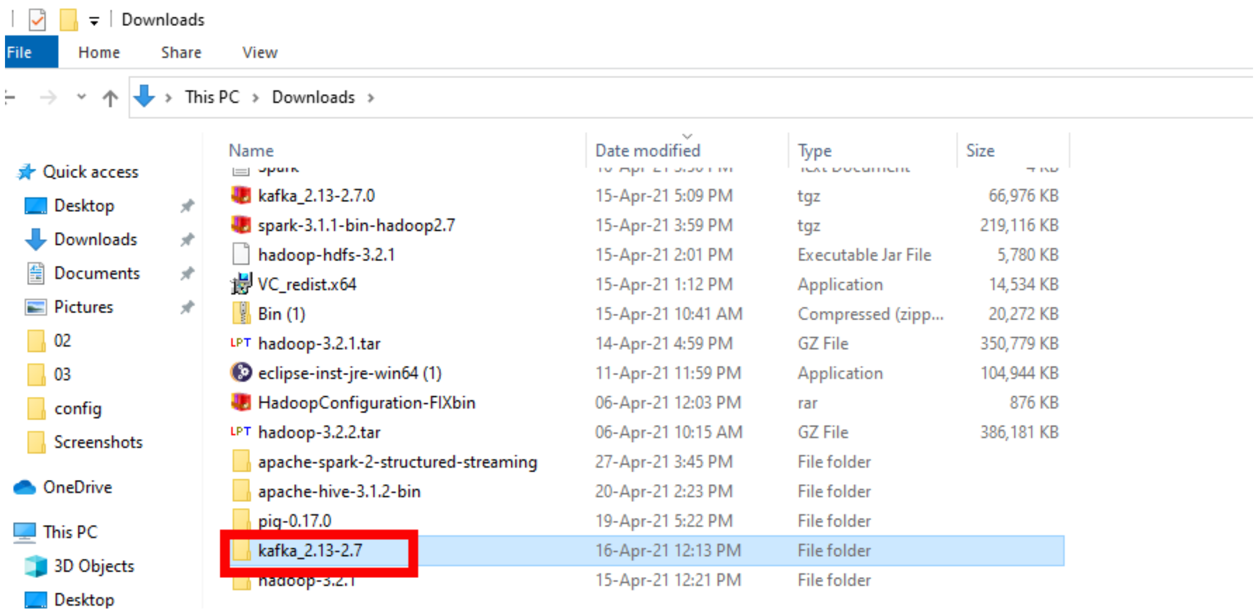
Look for an app in the Microsoft Store

More apps ↓

☐ Always use this app to open .tgz files

OK

Step 6: Then you will get a folder with same name kafka_2.13-2.7.



Downloads

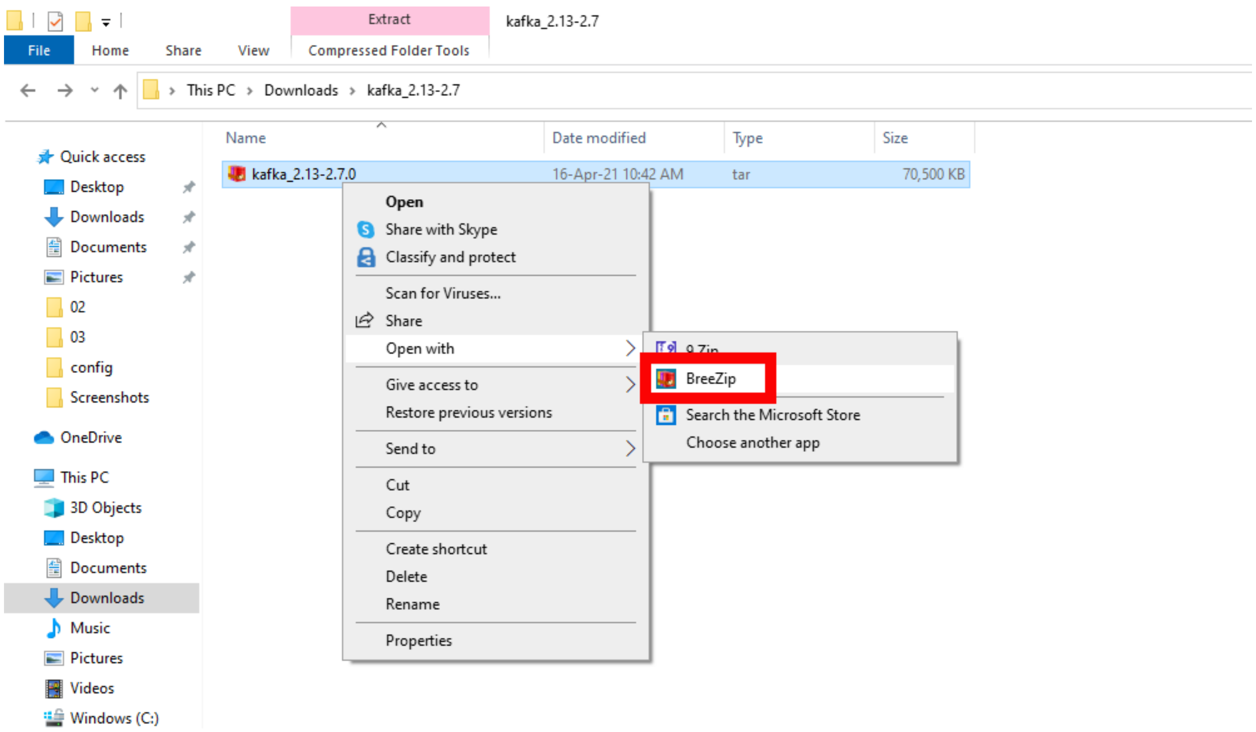
File Home Share View

This PC › Downloads ›

Quick access, Desktop, Downloads, Documents, Pictures, 02, 03, config, Screenshots, OneDrive, This PC, 3D Objects, Desktop

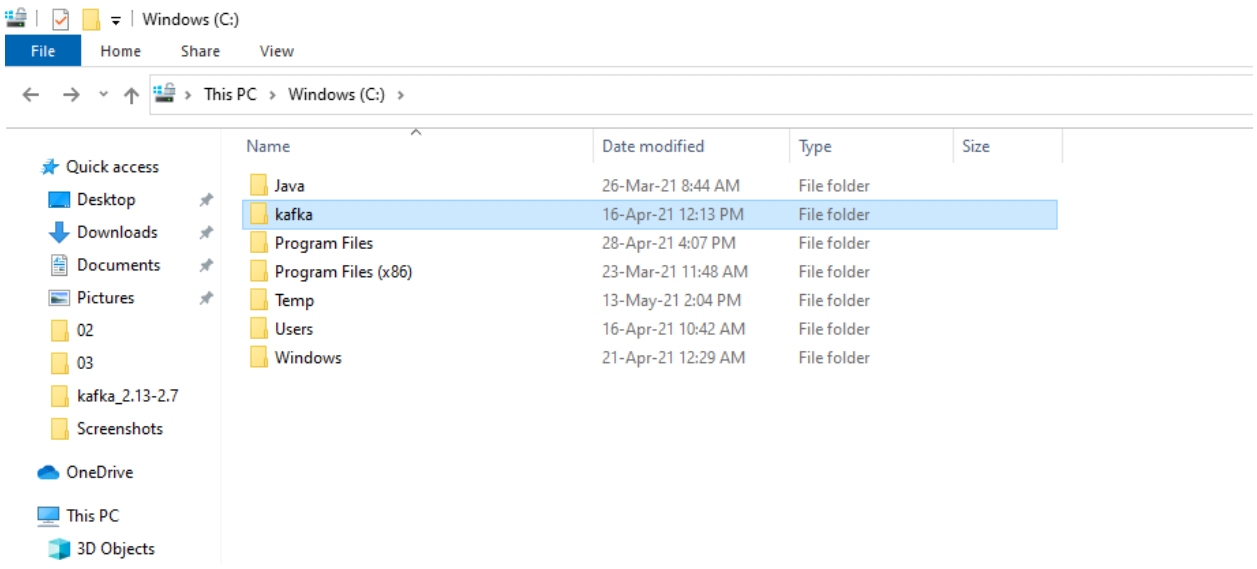
Name	Date modified	Type	Size
kafka_2.13-2.7.0	15-Apr-21 5:09 PM	tgz	66,976 KB
spark-3.1.1-bin-hadoop2.7	15-Apr-21 3:59 PM	tgz	219,116 KB
hadoop-hdfs-3.2.1	15-Apr-21 2:01 PM	Executable Jar File	5,780 KB
VC_redist.x64	15-Apr-21 1:12 PM	Application	14,534 KB
Bin (1)	15-Apr-21 10:41 AM	Compressed (zipp...	20,272 KB
hadoop-3.2.1.tar	14-Apr-21 4:59 PM	GZ File	350,779 KB
eclipse-inst-jre-win64 (1)	11-Apr-21 11:59 PM	Application	104,944 KB
HadoopConfiguration-FIXbin	06-Apr-21 12:03 PM	rar	876 KB
hadoop-3.2.2.tar	06-Apr-21 10:15 AM	GZ File	386,181 KB
apache-spark-2-structured-streaming	27-Apr-21 3:45 PM	File folder	
apache-hive-3.1.2-bin	20-Apr-21 2:23 PM	File folder	
pig-0.17.0	19-Apr-21 5:22 PM	File folder	
kafka_2.13-2.7	16-Apr-21 12:13 PM	File folder	
hadoop-3.2.1	15-Apr-21 12:21 PM	File folder	

Step 7: Open it, there will be another .tar file**.** Decompress that file also.



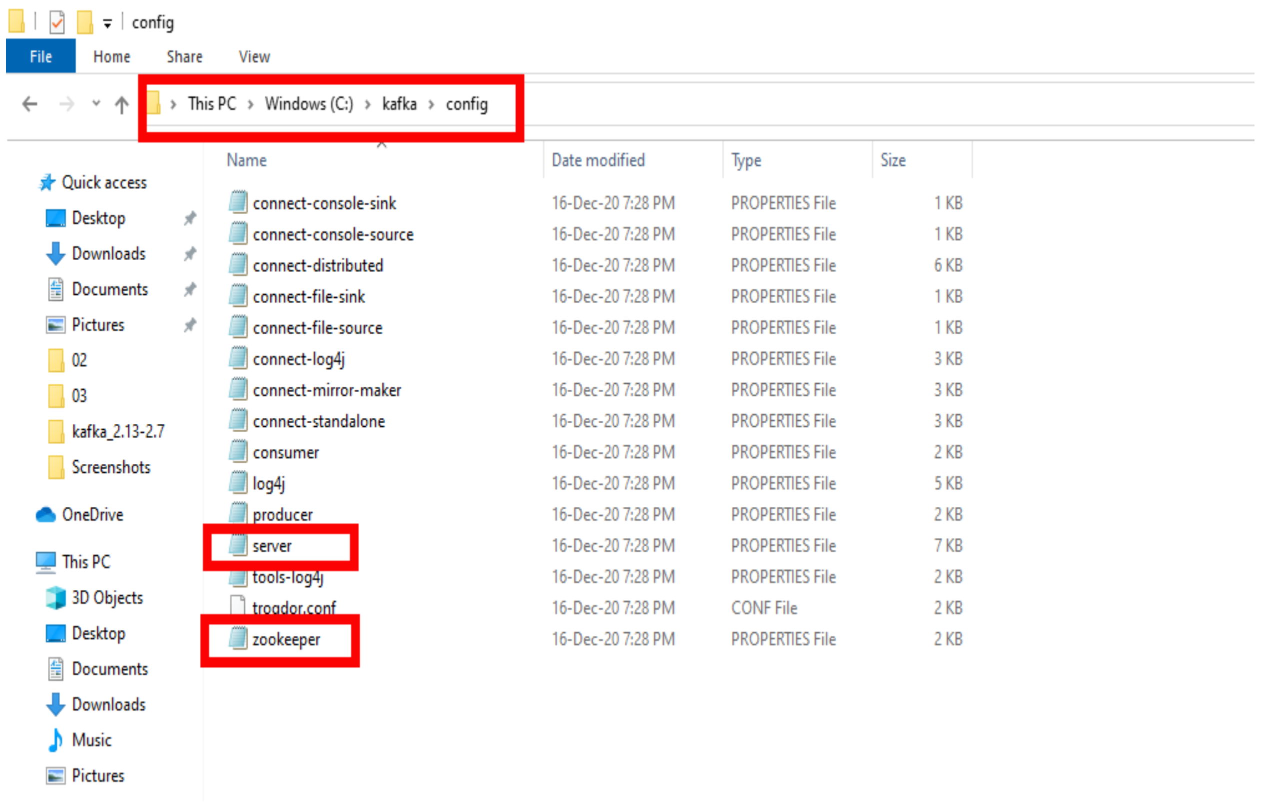
Name	Date modified	Type	Size
kafka_2.13-2.7.0	16-Apr-21 10:42 AM	tar	70,500 KB

Step 8: Now a new folder will be generated with name kafka_2.13-2.7. Rename the folder as kafka an move it to the C disk.



Name	Date modified	Type	Size
Java	26-Mar-21 8:44 AM	File folder	
kafka	16-Apr-21 12:13 PM	File folder	
Program Files	28-Apr-21 4:07 PM	File folder	
Program Files (x86)	23-Mar-21 11:48 AM	File folder	
Temp	13-May-21 2:04 PM	File folder	
Users	16-Apr-21 10:42 AM	File folder	
Windows	21-Apr-21 12:29 AM	File folder	

Step 9: Then open the kafka folder, go to config folder, and update server.properties file &zookeeper.properties file.



This PC › Windows (C:) › kafka › config

Name	Date modified	Type	Size
connect-console-sink	16-Dec-20 7:28 PM	PROPERTIES File	1 KB
connect-console-source	16-Dec-20 7:28 PM	PROPERTIES File	1 KB
connect-distributed	16-Dec-20 7:28 PM	PROPERTIES File	6 KB
connect-file-sink	16-Dec-20 7:28 PM	PROPERTIES File	1 KB
connect-file-source	16-Dec-20 7:28 PM	PROPERTIES File	1 KB
connect-log4j	16-Dec-20 7:28 PM	PROPERTIES File	3 KB
connect-mirror-maker	16-Dec-20 7:28 PM	PROPERTIES File	3 KB
connect-standalone	16-Dec-20 7:28 PM	PROPERTIES File	3 KB
consumer	16-Dec-20 7:28 PM	PROPERTIES File	2 KB
log4j	16-Dec-20 7:28 PM	PROPERTIES File	5 KB
producer	16-Dec-20 7:28 PM	PROPERTIES File	2 KB
server	16-Dec-20 7:28 PM	PROPERTIES File	7 KB
tools-log4j	16-Dec-20 7:28 PM	PROPERTIES File	2 KB
trogdor.conf	16-Dec-20 7:28 PM	CONF File	2 KB
zookeeper	16-Dec-20 7:28 PM	PROPERTIES File	2 KB

Step 10: Open server.properties file and change log.dirs value as follows and save it.

log.dirs = C:\kafka\kafka-logs

Step 11: Open zookeeper.properties and change dataDir value as follows and save it.

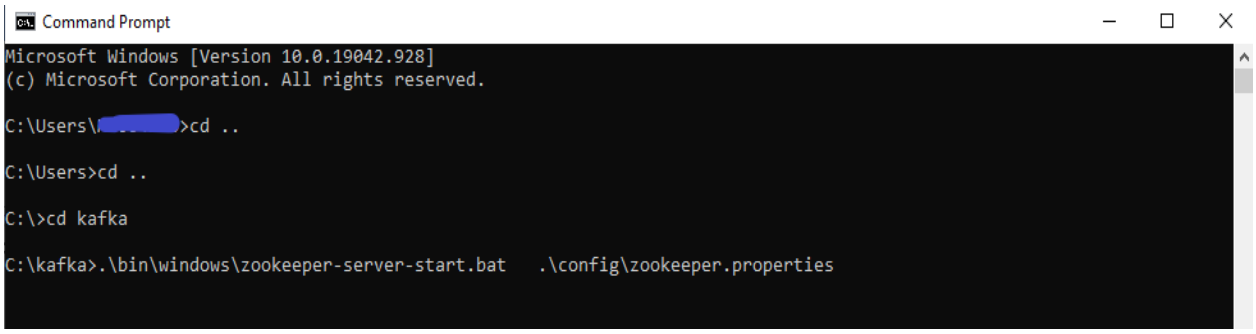
dataDir=C:/kafka/zookeeper-data

Structured Streaming with Kafka

i. for invoking zookeeper

a. Open command window and go to kafka directory.

b.command:-.\bin\windows\zookeeper-server-start.bat.\config\zookeeper.properties

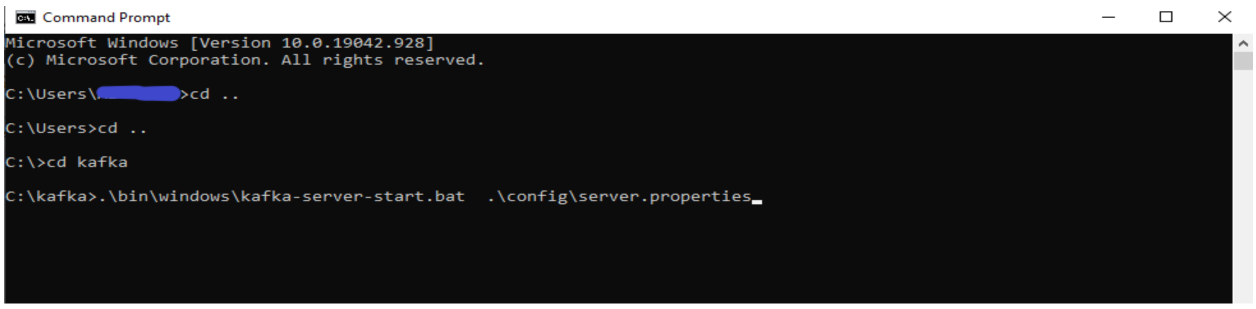


```
Microsoft Windows [Version 10.0.19042.928]
(c) Microsoft Corporation. All rights reserved.

C:\Users\[redacted]>cd ..

C:\Users>cd ..

C:\>cd kafka

C:\kafka>.\bin\windows\zookeeper-server-start.bat   .\config\zookeeper.properties
```

ii. for invoking servers

a. Open command window and go to kafka directory.

b. command: .\bin\windows\kafka-server-start.bat .\config\server.properties

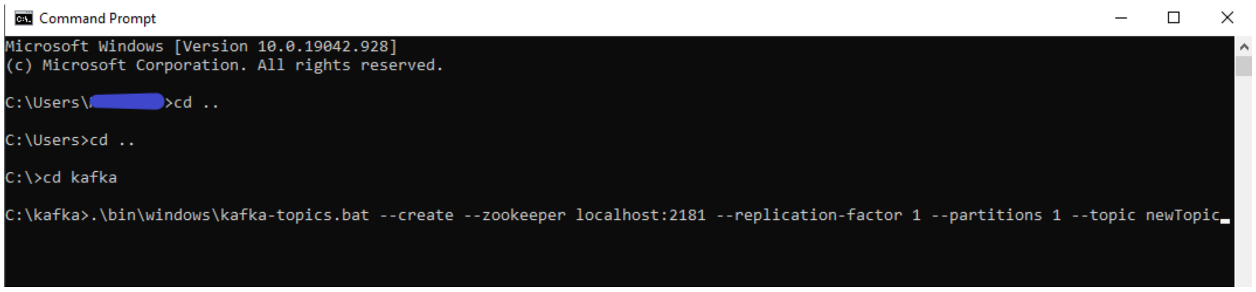


```
Microsoft Windows [Version 10.0.19042.928]
(c) Microsoft Corporation. All rights reserved.

C:\Users\[redacted]>cd ..

C:\Users>cd ..

C:\>cd kafka

C:\kafka>.\bin\windows\kafka-server-start.bat  .\config\server.properties
```

iii. creating topic

a. Open command window and go to kafka directory.

b. command: .\bin\windows\kafka-topics.bat –create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic newTopic

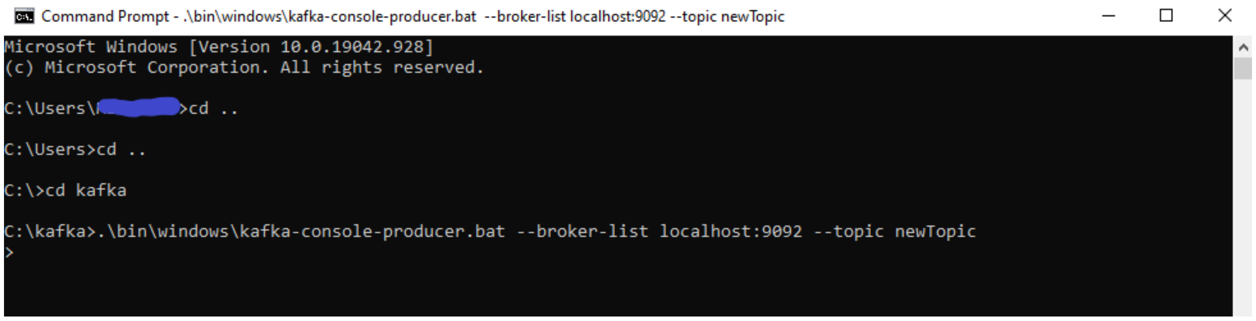


```
Microsoft Windows [Version 10.0.19042.928]
(c) Microsoft Corporation. All rights reserved.

C:\Users\[redacted]>cd ..

C:\Users>cd ..

C:\>cd kafka

C:\kafka>.\bin\windows\kafka-topics.bat --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic newTopic
```

iv. creating producer

a. Open command window and go to kafka directory.

b. command for producer:.\bin\windows\kafka-console-producer.bat --broker-list localhost:9092 --topic newTopic

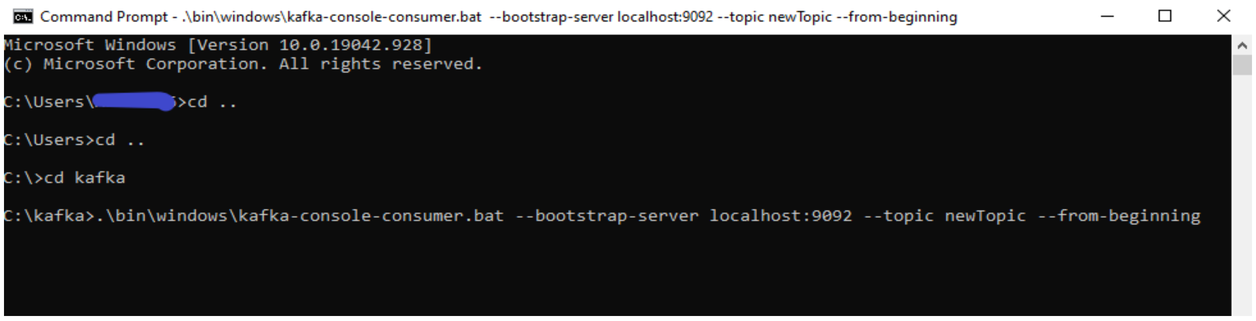


```
Command Prompt - .\bin\windows\kafka-console-producer.bat --broker-list localhost:9092 --topic newTopic
Microsoft Windows [Version 10.0.19042.928]
(c) Microsoft Corporation. All rights reserved.

C:\Users\[redacted]>cd ..

C:\Users>cd ..

C:\>cd kafka

C:\kafka>.\bin\windows\kafka-console-producer.bat --broker-list localhost:9092 --topic newTopic
>
```

v. creating consumer

a. Open command window and go to kafka directory.

b. command for consumer: .\bin\windows\kafka-console-consumer.bat--bootstrap-server localhost:9092 --topic newTopic --from-beginning



```
Command Prompt - .\bin\windows\kafka-console-consumer.bat --bootstrap-server localhost:9092 --topic newTopic --from-beginning
Microsoft Windows [Version 10.0.19042.928]
(c) Microsoft Corporation. All rights reserved.

C:\Users\[redacted]>cd ..

C:\Users>cd ..

C:\>cd kafka

C:\kafka>.\bin\windows\kafka-console-consumer.bat --bootstrap-server localhost:9092 --topic newTopic --from-beginning
```

vi. Example Twitter streaming python script todisplay hashtags. Save it with name kafkaHashtagProducer.py

```
# Setup APIs for tweepy and pykafka to get stream tweets to Kafka

import sys
import tweepy
from tweepy import OAuthHandler
from tweepy import Stream
from tweepy.streaming import StreamListener
import json
import pykafka
```

```
class TweetsListener(StreamListener):

def __init__(self, kafkaProducer):

print ("Tweets producer initialized")

#sends tweets to kafka producer
self.producer = kafkaProducer

#gets each tweet in the form of json and displays the tweets having hashtag
def on_data(self, data):

try:
      json_data = json.loads(data)
      words = json_data["text"].split()
      hashtagList = list(filter(lambda x: x.lower().startswith("#"), words))

if(len(hashtagList)!=0):
for hashtag in hashtagList:
print(hashtag)
self.producer.produce(bytes(hashtag,"utf-8"))

#Prints error if any error occurs
except KeyError as e:
print("Error on_data: %s" % str(e))

return True

  def on_error(self, status):

print(status)
return True


#Connecting to twitter using the Twitter API details
def connect_to_twitter(kafkaProducer, tracks):

    api_key = "N3BhfveA3Q4mXCtyRRphCrC7v"
api_secret = "6UFnndGzuMwKKIldE2kw9syulo3Hb4gVwQ14umDP43jTdv9x1s"
```

```
access_token = "368413415-5FaufzgFTozESYU00xp6nxrxoiUUp1IIhThVHHOp"
access_token_secret = "c1JV2hFifKz2WFiLxLnpfrBPqmgeBLH5kBk5mqeq8PF9B"

auth = OAuthHandler(api_key, api_secret)
    auth.set_access_token(access_token, access_token_secret)

    twitter_stream = Stream(auth, TweetsListener(kafkaProducer))
    twitter_stream.filter(track=tracks, languages=["en"])

if __name__ == "__main__":

if len(sys.argv) <5:
print("Usage: python kafkaHashtagProducer.py <host><port><topic_name><tracks>",
file=sys.stderr)
exit(-1)

    host = sys.argv[1]
    port = sys.argv[2]
    topic = sys.argv[3]
    tracks = sys.argv[4:]

#Initializing kafka client using hostname and port number
    kafkaClient = pykafka.KafkaClient(host + ":" + port)


    kafkaProducer = kafkaClient.topics[bytes(topic,"utf-8")].get_producer()

    connect_to_twitter(kafkaProducer, tracks)
```

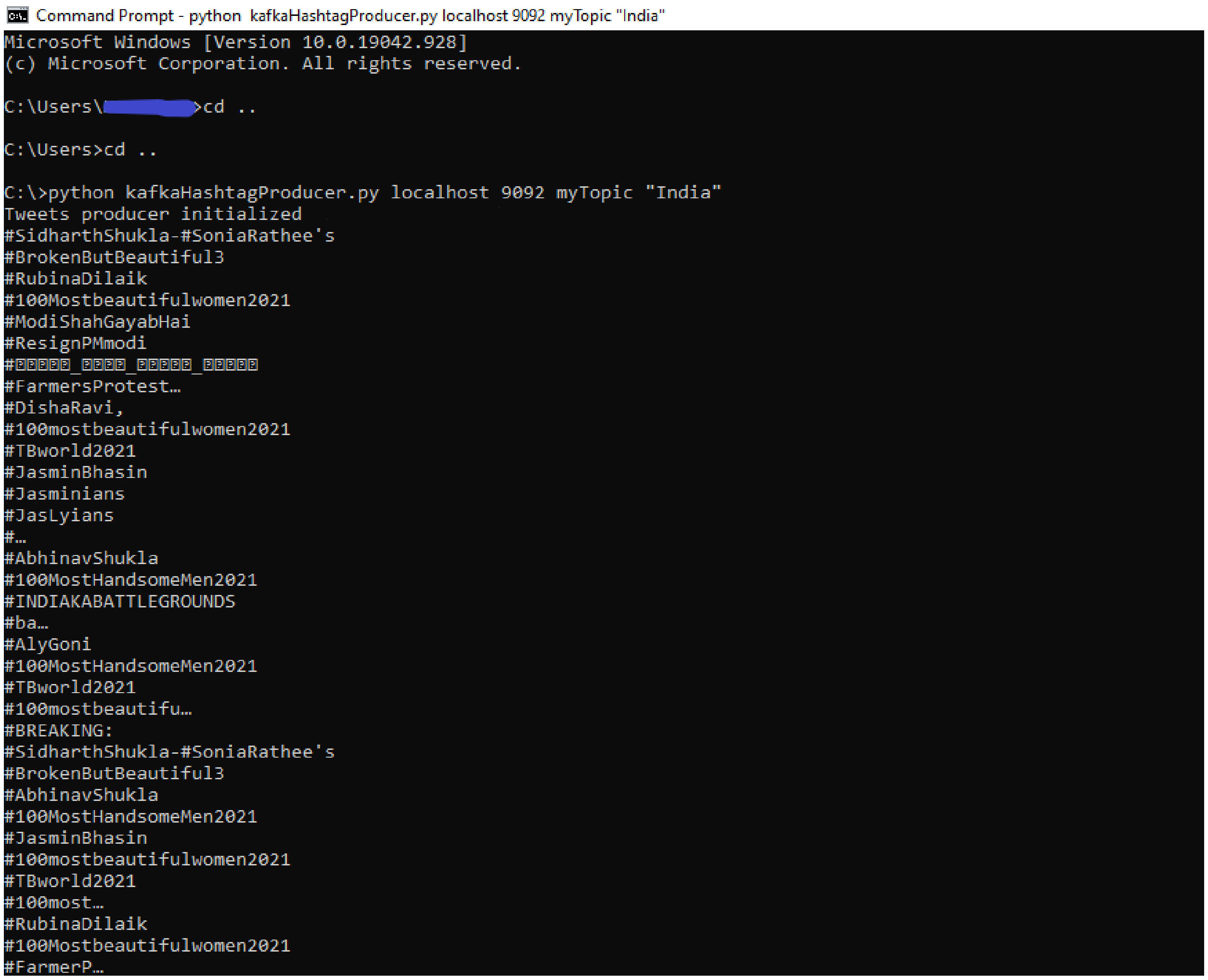
vii. Run the python script kafkaHashtagProducer.py in the command window

a. Open command prompt and change directory to where the script file kafkaHashtagProducer.py stored.

b. Command format for running the kafka program:

python kafkaHashtagProducer.py<host><port><topic><"keyword">

Ex: python kafkaHashtagProducer.py localhost 9092 myTopic "India"



Command Prompt - python kafkaHashtagProducer.py localhost 9092 myTopic "India"

```
Microsoft Windows [Version 10.0.19042.928]
(c) Microsoft Corporation. All rights reserved.

C:\Users\[redacted]>cd ..

C:\Users>cd ..

C:\>python kafkaHashtagProducer.py localhost 9092 myTopic "India"
Tweets producer initialized
#SidharthShukla-#SoniaRathee's
#BrokenButBeautiful3
#RubinaDilaik
#100Mostbeautifulwomen2021
#ModiShahGayabHai
#ResignPMmodi
#?????_????_?????_?????
#FarmersProtest…
#DishaRavi,
#100mostbeautifulwomen2021
#TBworld2021
#JasminBhasin
#Jasminians
#JasLyians
#…
#AbhinavShukla
#100MostHandsomeMen2021
#INDIAKABATTLEGROUNDS
#ba…
#AlyGoni
#100MostHandsomeMen2021
#TBworld2021
#100mostbeautifu…
#BREAKING:
#SidharthShukla-#SoniaRathee's
#BrokenButBeautiful3
#AbhinavShukla
#100MostHandsomeMen2021
#JasminBhasin
#100mostbeautifulwomen2021
#TBworld2021
#100most…
#RubinaDilaik
#100Mostbeautifulwomen2021
#FarmerP…
```

Able to Tune Performance

What is Spark performance tuning?

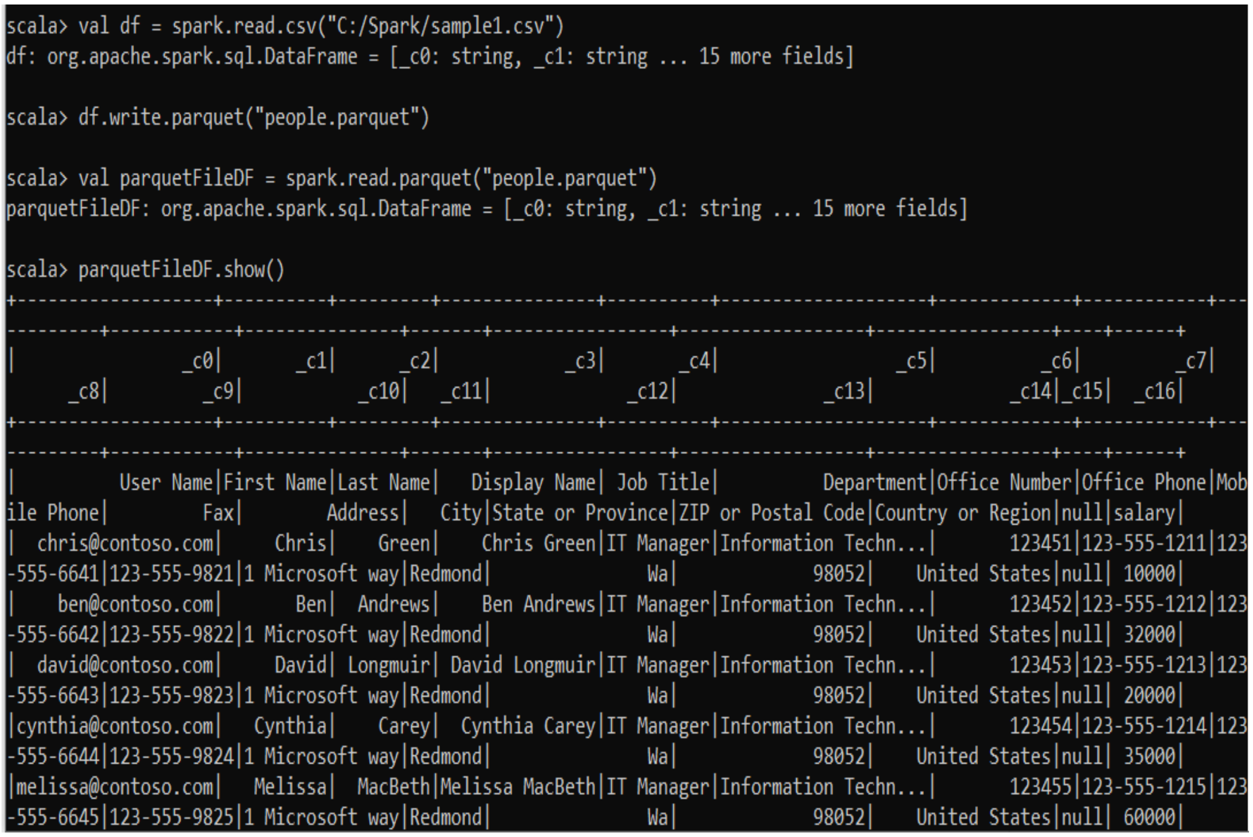
Spark Performance Tuning refers to the process of adjusting settings to record for memory, cores, and instances used by the system. The bottleneck for these spark optimization computations can be CPU, memory or any resource in the cluster.

There are 5 ways to improve the Spark performance.

1. **Serialization**

- In order to reduce memory usage, we have to store spark RDDs in serialized form.
- Spark uses java serializer by default.
- Spark can use 'Kryo' serializer which is in compact binary format and offers processing 10 times faster than Java serializer.

s

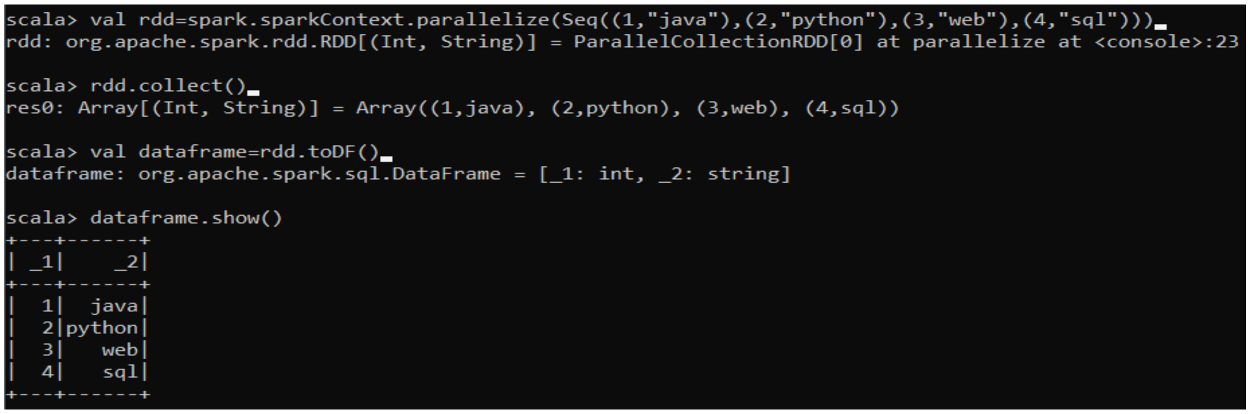


```
scala> val df = spark.read.csv("C:/Spark/sample1.csv")
df: org.apache.spark.sql.DataFrame = [_c0: string, _c1: string ... 15 more fields]

scala> df.write.parquet("people.parquet")

scala> val parquetFileDF = spark.read.parquet("people.parquet")
parquetFileDF: org.apache.spark.sql.DataFrame = [_c0: string, _c1: string ... 15 more fields]

scala> parquetFileDF.show()
+-------------------+----------+---------+---------------+----------+--------------------+-------------+------------+---
---------+------------+---------------+-------+-----------------+------------------+-----------------+----+------+
|                _c0|       _c1|      _c2|            _c3|       _c4|                 _c5|          _c6|         _c7|
      _c8|         _c9|           _c10|   _c11|             _c12|              _c13|             _c14|_c15|  _c16|
+-------------------+----------+---------+---------------+----------+--------------------+-------------+------------+---
---------+------------+---------------+-------+-----------------+------------------+-----------------+----+------+
|          User Name|First Name|Last Name|   Display Name| Job Title|          Department|Office Number|Office Phone|Mob
ile Phone|         Fax|        Address|   City|State or Province|ZIP or Postal Code|Country or Region|null|salary|
|  chris@contoso.com|     Chris|    Green|    Chris Green|IT Manager|Information Techn...|       123451|123-555-1211|123
-555-6641|123-555-9821|1 Microsoft way|Redmond|               Wa|             98052|    United States|null| 10000|
|    ben@contoso.com|       Ben|  Andrews|    Ben Andrews|IT Manager|Information Techn...|       123452|123-555-1212|123
-555-6642|123-555-9822|1 Microsoft way|Redmond|               Wa|             98052|    United States|null| 32000|
|  david@contoso.com|     David| Longmuir| David Longmuir|IT Manager|Information Techn...|       123453|123-555-1213|123
-555-6643|123-555-9823|1 Microsoft way|Redmond|               Wa|             98052|    United States|null| 20000|
|cynthia@contoso.com|   Cynthia|    Carey|  Cynthia Carey|IT Manager|Information Techn...|       123454|123-555-1214|123
-555-6644|123-555-9824|1 Microsoft way|Redmond|               Wa|             98052|    United States|null| 35000|
|melissa@contoso.com|   Melissa|  MacBeth|Melissa MacBeth|IT Manager|Information Techn...|       123455|123-555-1215|123
-555-6645|123-555-9825|1 Microsoft way|Redmond|               Wa|             98052|    United States|null| 60000|
```

2. API selection

Spark has API such as RDD,DtaFrame and Dataset.

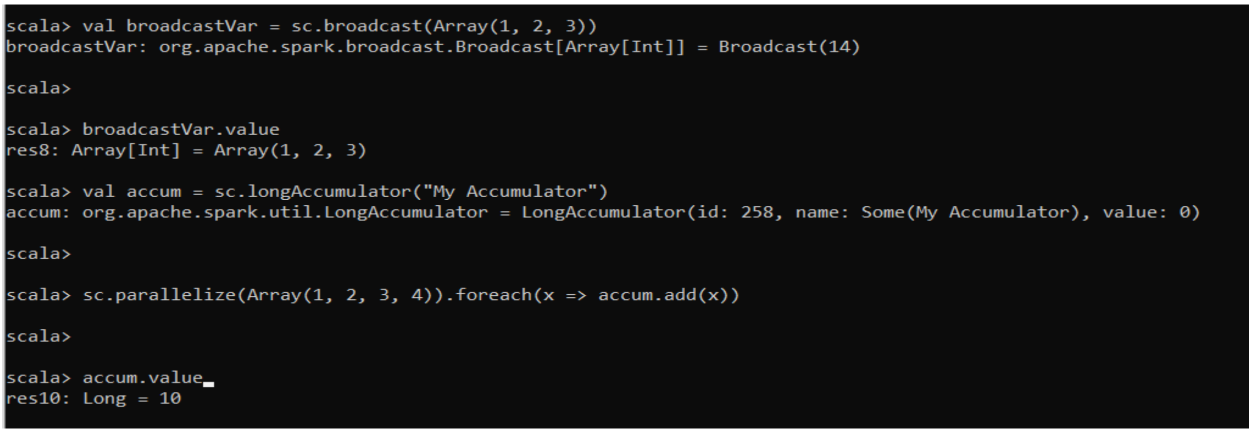
- RDD is used for low level operations and has less optimization techniques.
- Dataframe is best choice in most cases due to its catalyst optimizer which creates a query plan result in better performance.
  Using Dataframes rather than RDD increases performance



```
scala> val rdd=spark.sparkContext.parallelize(Seq((1,"java"),(2,"python"),(3,"web"),(4,"sql")))
rdd: org.apache.spark.rdd.RDD[(Int, String)] = ParallelCollectionRDD[0] at parallelize at <console>:23

scala> rdd.collect()
res0: Array[(Int, String)] = Array((1,java), (2,python), (3,web), (4,sql))

scala> val dataframe=rdd.toDF()
dataframe: org.apache.spark.sql.DataFrame = [_1: int, _2: string]

scala> dataframe.show()
+---+------+
| _1|    _2|
+---+------+
|  1|  java|
|  2|python|
|  3|   web|
|  4|   sql|
+---+------+
```

3. Advance Variable

- Spark have 2 advanced variables – Accumulator and Broadcast
- Broadcasting variable will make your small data set available on each node and that node & data will treated locally for the process.
- Broadcasting plays an important role while tuning spark jobs.



```
scala> val broadcastVar = sc.broadcast(Array(1, 2, 3))
broadcastVar: org.apache.spark.broadcast.Broadcast[Array[Int]] = Broadcast(14)

scala>

scala> broadcastVar.value
res8: Array[Int] = Array(1, 2, 3)

scala> val accum = sc.longAccumulator("My Accumulator")
accum: org.apache.spark.util.LongAccumulator = LongAccumulator(id: 258, name: Some(My Accumulator), value: 0)

scala>

scala> sc.parallelize(Array(1, 2, 3, 4)).foreach(x => accum.add(x))

scala>

scala> accum.value
res10: Long = 10
```

4. Cache and Persist

- Spark provides its own caching mechanism like persist() and cache().
- persist() and cache() mechanism will store data set into memory whenever there is requirement.
- Data is stored where you have a small data set and that data set is being used multiple times in your program.

5. ByKey Operation

- ByKey operations generate lot of shuffle. Shuffles are heavy operations because they consume lot of memory.
- While coding in Spark, try to avoid shuffle operations.
- High shuffling gives OutOfMemory error.
- Use reduceByKey instead of groupBykey.
- Partition the data properly.

6. File Format Selection

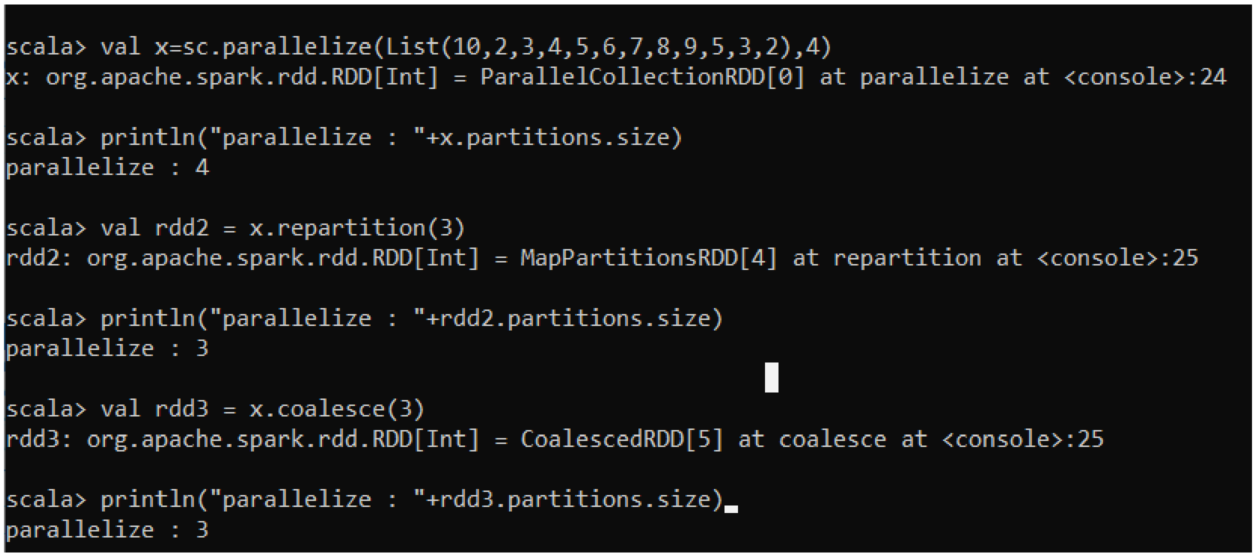
- Spark supports many file formats such as CSV, JSON, XML, PARQUET, ORC, AVRO and more.
- Spark jobs can be optimized by choosing the parquet file with snappy compression which gives the high performance and best analysis.
- Parquet file is native to Spark which carries the metadata along with its footer.

7. Garbage Collection Tuning

- We know that Spark job is running in JVM, JVM garbage collection can be a problem when you have a large collection of unused objects.
- The first step in GC tuning is to collect statistics by s=choosing – verbose while submitting spark jobs.
- In an ideal situation we try to keep GC overheads <10% of heap memory.

8. Level of parallelism

- Every partition or task requires a single core for processing.
- We can repartition by using Repartition and Coalesce.
- Repartition gives equal number of partitions with high shuffling.
- Coalesce actually reduces number of partitions with less shuffling.



```
scala> val x=sc.parallelize(List(10,2,3,4,5,6,7,8,9,5,3,2),4)
x: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[0] at parallelize at <console>:24

scala> println("parallelize : "+x.partitions.size)
parallelize : 4

scala> val rdd2 = x.repartition(3)
rdd2: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[4] at repartition at <console>:25

scala> println("parallelize : "+rdd2.partitions.size)
parallelize : 3

scala> val rdd3 = x.coalesce(3)
rdd3: org.apache.spark.rdd.RDD[Int] = CoalescedRDD[5] at coalesce at <console>:25

scala> println("parallelize : "+rdd3.partitions.size)
parallelize : 3
```