1. Explain PySpark in details?

As spark is return in Scala so in order to support python with spark. Spark community has released a tool called PySpark. In python programming language we can also work with RDDs, Using PySpark.

1. For Connecting the ADLS gen2 from Databricks what are information needed?
2. Service principal object ID (aka Client ID)
3. Service principal key
4. Azure active directory tenant id (aka Directory ID)
5. ADLS gen2 account name
6. ADLS gen2 file system name (aka container name)
7. A sample file uploaded to the folder in your ADLS gen2 file system
8. Repartition () vs coalesce ()

In PySpark repartition () is used to increase or decrease the RDD, DataFrame partitions whereas the PySpark coalesce () is used to only decrease the number of partitions in an efficient way.

spark.default.parallelism configuration default value set to the number of all cores on all nodes in a cluster, on local it is set to number of cores on your system.

spark.sql.shuffle.partitions configuration default value is set to 200 and be used when you call shuffle operations like reduceByKey() , groupByKey(), join() and many more. This property is available only in DataFrame API but not in RDD.

1. **RDD Partition and repartition**

rdd = spark.sparkContext.parallelize((0,20))

print("From local[5]"+str(rdd.getNumPartitions()))

rdd1 = spark.sparkContext.parallelize((0,25), 6)

print("parallelize : "+str(rdd1.getNumPartitions()))

rddFromFile = spark.sparkContext.textFile("src/main/resources/test.txt",10)

print("TextFile : "+str(rddFromFile.getNumPartitions()))

output:

From local[5] : 5

Parallelize : 6

TextFile : 10

1. **RDD repartition()**

Spark RDD repartition() method is used to increase or decrease the partitions. The below example decreases the partitions from 10 to 4 by moving data from all partitions.

val rdd2 = rdd1.repartition(4)

print("Repartition size : "+str(rdd2.getNumPartitions()))

rdd2.saveAsTextFile("/tmp/re-partition")

Output :

Partition 1 : 1 6 10 15 19

Partition 2 : 2 3 7 11 16

Partition 3 : 4 8 12 13 17

Partition 4 : 0 5 9 14 18

1. **RDD coalesce()**

Spark RDD coalesce() is used only to reduce the number of partitions. This is optimized or improved version of repartition() where the movement of the data across the partitions is lower using coalesce.

val rdd3 = rdd1.coalesce(4)

print("Repartition size : "+str(rdd3.getNumPartitions()))

rdd3.saveAsTextFile("/tmp/coalesce")

Output:

Partition 1 : 0 1 2

Partition 2 : 3 4 5 6 7 8 9

Partition 4 : 10 11 12

Partition 5 : 13 14 15 16 17 18 19

1. **Window Functions (**<https://sparkbyexamples.com/pyspark/pyspark-window-functions/>**)**

PySpark Window functions are used to calculate results such as the rank, row number e.t.c over a range of input rows. In this article, I’ve explained the concept of window functions, syntax, and finally how to use them with PySpark SQL and PySpark DataFrame API. These come in handy when we need to make aggregate operations in a specific window frame on DataFrame columns.

from pyspark.sql.window import Window

from pyspark.sql.functions import row\_number

windowSpec = Window.partitionBy("department").orderBy("salary")

df.withColumn("row\_number",row\_number().over(windowSpec)) \

.show(truncate=False)

* row\_number()
* rank()
* dense\_rank()
* percent\_rank()
* ntile()

**PySpark Window Analytic functions**

* cume\_dist()
* LAG
* LEAD

**PySpark Window Aggregate Functions**

* **avg**
* **sum**
* **min**
* **max**

1. Difference between lineage graph vs DAG

**Lineage graph**  
As we know, that whenever a series of transformations are performed on an [RDD](https://data-flair.training/blogs/apache-spark-rdd-tutorial/), they are not evaluated immediately, but lazily([Lazy Evaluation](https://data-flair.training/blogs/apache-spark-lazy-evaluation/)). When a new RDD has been created from an existing RDD, that new RDD contains a pointer to the parent RDD. Similarly, all the dependencies between the RDDs will be logged in a graph, rather than the actual data. This graph is called the lineage graph.

Now coming to DAG,

**Directed Acyclic Graph(DAG)**  
DAG in [Apache Spark](https://data-flair.training/blogs/apache-spark-for-beginners/) is a combination of Vertices as well as Edges. In DAG vertices represent the RDDs and the edges represent the Operation to be applied on RDD. Every edge in DAG is directed from earlier to later in a sequence.When we call an[Action](https://data-flair.training/blogs/spark-rdd-operations-transformations-actions/), the created DAG is submitted to DAG Scheduler which further splits the graph into the stages of the task.

1. [Spark difference between reduceByKey vs groupByKey vs aggregateByKey](https://stackoverflow.com/questions/43364432/spark-difference-between-reducebykey-vs-groupbykey-vs-aggregatebykey-vs-combineb)

groupByKey() is just to group your dataset based on a key. It will result in data shuffling when RDD is not already partitioned.

reduceByKey() is something like grouping + aggregation. We can say reduceBykey() equvelent to dataset.group(...).reduce(...). It will shuffle less data unlike groupByKey().

aggregateByKey() is logically same as reduceByKey() but it lets you return result in different type. In another words, it lets you have a input as type x and aggregate result as type y. For example (1,2),(1,4) as input and (1,"six") as output. It also takes *zero-value* that will be applied at the beginning of each key.

1. What is object and classes?

An object is simply a collection of data (variables) and methods (functions) that act on those data. Similarly, a class is a blueprint for that object.

1. Delta Read and write

events = spark.read.json("/databricks-datasets/structured-streaming/events/")

events.write.partitionBy("date").format("delta").save("/mnt/delta/events")

spark.sql("CREATE TABLE events USING DELTA LOCATION '/mnt/delta/events/'")

1. **Create Spark in previous versions of spark 2.0?**

First, we have to create spark conf and spark context in order to interact with spark.

The syntax will be,

//set up the spark configuration and create contexts

**val sparkConf = new SparkConf().setAppName("SparkSessionZipsExample").setMaster("local")**

// your handle to SparkContext to access other context like SQLContext

**val sc = new SparkContext(sparkConf).set("spark.some.config.option", "some-value")**

**val sqlContext = new org.apache.spark.sql.SQLContext(sc)**

1. Create spark session in Spark 2.0?

// Create a SparkSession. No need to create SparkContext

// You automatically get it as part of the SparkSession

val warehouseLocation = "file:${system:user.dir}/spark-warehouse"

val spark = SparkSession

.builder()

.appName("SparkSessionZipsExample")

.config("spark.sql.warehouse.dir", warehouseLocation)

.enableHiveSupport()

.getOrCreate()

1. How to create spark session 3.0 ?

import org.apache.spark.sql.SparkSession

val spark = SparkSession

.builder()

.appName("Spark SQL basic example")

.config("spark.some.config.option", "some-value")

.getOrCreate()

// For implicit conversions like converting RDDs to DataFrames

import spark.implicits.\_

1. Features of spark 3

* adaptive query execution
* dynamic partition pruning
* ANSI SQL compliance
* significant improvements in pandas APIs
* accelerator-aware scheduler

1. **ADLS gen1 session building**: (<https://docs.databricks.com/data/data-sources/azure/azure-datalake.html> )

There are two ways of accessing Azure Data Lake Storage Gen1:

* Mount an Azure Data Lake Storage Gen1 filesystem to DBFS using a service principal and OAuth 2.0.
* Use a service principal directly.

**Step 1:** Read IoT Devices JSON from ADLS Gen1 resource

df2 = spark.read.json("adl://<storage-resource>.azuredatalakestore.net/iot\_devices.json")

display(df2)

**step 2:** Configure authentication for mounting

configs = {"fs.adl.oauth2.access.token.provider.type": "ClientCredential",

"fs.adl.oauth2.client.id": <client-id>,

"fs.adl.oauth2.credential": <service-credential>,

"fs.adl.oauth2.refresh.url": "https://login.microsoftonline.com/<directory-id>/oauth2/token"}

**(OR)**

**Configure credentials**

**spark.conf.set("fs.adl.oauth2.access.token.provider.type", "ClientCredential")**

**spark.conf.set("fs.adl.oauth2.client.id", <client-id>)**

**spark.conf.set("fs.adl.oauth2.credential", <service-credential>)**

**spark.conf.set("fs.adl.oauth2.refresh.url", "https://login.microsoftonline.com/<directory-id>/oauth2/token")**

**Step 3:** Mount filesystem

dbutils.fs.mount(

source = "adl://<storage-resource>.azuredatalakestore.net/",

mount\_point = "/mnt/mymount",

extra\_configs = configs)

**step 4:** List mount

dbutils.fs.ls("/mnt/mymount")

**step 5:**

Unmount filesystem

dbutils.fs.unmount("/mnt/mymount")

1. Adls gen2 session building: (<https://docs.databricks.com/data/data-sources/azure/azure-datalake-gen2.html> )

There are three ways of accessing Azure Data Lake Storage Gen2:

* Mount an Azure Data Lake Storage Gen2 filesystem to DBFS using a service principal and OAuth 2.0.
* Use a service principal directly.
* Use the Azure Data Lake Storage Gen2 storage account access key directly.

**Configure authentication for mounting**

configs = {"fs.azure.account.auth.type": "OAuth",

"fs.azure.account.oauth.provider.type": "org.apache.hadoop.fs.azurebfs.oauth2.ClientCredsTokenProvider",

"fs.azure.account.oauth2.client.id": <client-id>,

"fs.azure.account.oauth2.client.secret": <service-credential>,

"fs.azure.account.oauth2.client.endpoint": "https://login.microsoftonline.com/<directory-id>/oauth2/token"}

**(OR)**

**Configure storage account key**

spark.conf.set("fs.azure.account.key.<storage-account-name>.dfs.core.windows.net", <storage-account-key>)

spark.conf.set("fs.azure.createRemoteFileSystemDuringInitialization", "true")

dbutils.fs.ls("abfss://<file\_system>@<storage-account-name>.dfs.core.windows.net/")

spark.conf.set("fs.azure.createRemoteFileSystemDuringInitialization", "false")

**Mount filesystem**

dbutils.fs.mount(

source = "abfss://<file\_system>@<storage-account-name>.dfs.core.windows.net/",

mount\_point = "/mnt/mymount",

extra\_configs = configs)

**List mount**

dbutils.fs.ls("/mnt/mymount")

**Unmount filesystem**

dbutils.fs.unmount("/mnt/mymount")

1. Spark AVRO read and write:

Read:

val personDF= spark.read.format("avro").load("person.avro")

Write:

val data = Seq(("James ","","Smith",2018,1,"M",3000),

("Michael ","Rose","",2010,3,"M",4000),

("Robert ","","Williams",2010,3,"M",4000),

("Maria ","Anne","Jones",2005,5,"F",4000),

("Jen","Mary","Brown",2010,7,"",-1)

)

val columns = Seq("firstname", "middlename", "lastname", "dob\_year",

"dob\_month", "gender", "salary")

import spark.sqlContext.implicits.\_

val df = data.toDF(columns:\_\*)

df.write.partitionBy("dob\_year","dob\_month")

.format("avro").save("person\_partition.avro")

1. Parquet Read and write:

Read:

val parqDF = spark.read.parquet("/tmp/output/people.parquet")

1. Read and write using delta:

events = spark.read.json("/databricks-datasets/structured-streaming/events/")

events.write.format("delta").save("/mnt/delta/events")

spark.sql("CREATE TABLE events USING DELTA LOCATION '/mnt/delta/events/'")

**Using partition**

events = spark.read.json("/databricks-datasets/structured-streaming/events/")

events.write.partitionBy("date").format("delta").save("/mnt/delta/events")

spark.sql("CREATE TABLE events USING DELTA LOCATION '/mnt/delta/events/'")