

Examen Final

Nombre: Nicole Pamela Garcia Cano

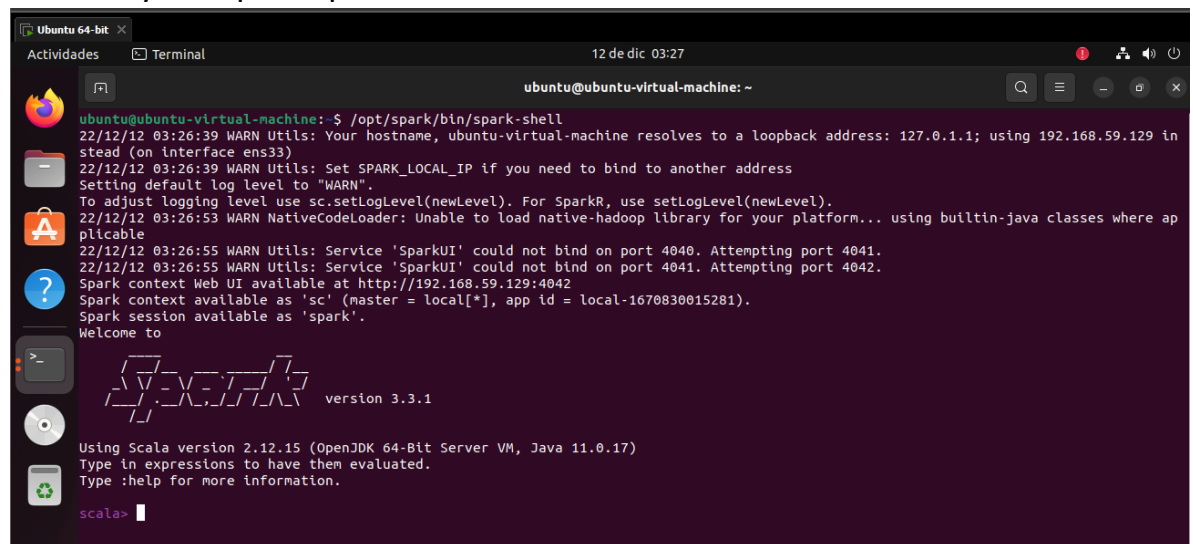
C.I: 9107217 L.P.

1. En una máquina virtual realice la configuración de apache spark, puede guiarse en cualquier tutorial o el proporcionado por el docente.

url: <https://computingforgeeks.com/how-to-install-apache-spark-on-ubuntu-debian/>

Con el shell podrá ejecutar scala por defecto

Instale Python para spark



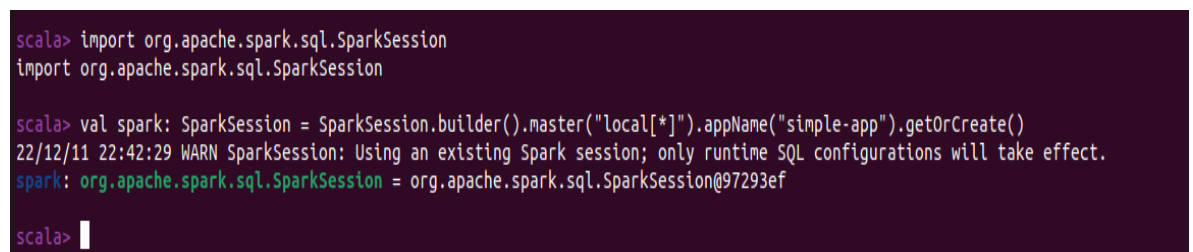
```
ubuntu@ubuntu-virtual-machine: ~  
$ /opt/spark/bin/spark-shell  
22/12/12 03:26:39 WARN Utils: Your hostname, ubuntu-virtual-machine resolves to a loopback address: 127.0.1.1; using 192.168.59.129 in  
stead (on interface ens33)  
22/12/12 03:26:39 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address  
Setting default log level to "WARN".  
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).  
22/12/12 03:26:53 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where ap  
plicable  
22/12/12 03:26:55 WARN Utils: Service 'SparkUI' could not bind on port 4040. Attempting port 4041.  
22/12/12 03:26:55 WARN Utils: Service 'SparkUI' could not bind on port 4041. Attempting port 4042.  
Spark context Web UI available at http://192.168.59.129:4042  
Spark context available as 'sc' (master = local[*], app id = local-1670830015281).  
Spark session available as 'spark'.  
Welcome to  
  
      _ _ _ _ _  
     / _ _ _ _ \  
    / _ _ _ _ \  
   / _ _ _ _ \  
  / _ _ _ _ \  
 / _ _ _ _ \  
/_ _ _ _ _\  
version 3.3.1  
  
Using Scala version 2.12.15 (OpenJDK 64-Bit Server VM, Java 11.0.17)  
Type in expressions to have them evaluated.  
Type :help for more information.  
  
scala>
```

2. Realice el siguiente código, documente su funcionamiento en apache spark

Sesiones

- **val spark: SparkSession = SparkSession.builder().master("local[*]").appName("simple-app").getOrCreate()**

Para poder ejecutar este commando debemos primero usar la libreria



```
scala> import org.apache.spark.sql.SparkSession  
import org.apache.spark.sql.SparkSession  
  
scala> val spark: SparkSession = SparkSession.builder().master("local[*]").appName("simple-app").getOrCreate()  
22/12/11 22:42:29 WARN SparkSession: Using an existing Spark session; only runtime SQL configurations will take effect.  
spark: org.apache.spark.sql.SparkSession = org.apache.spark.sql.SparkSession@97293ef  
  
scala>
```

val spark = SparkSession.builder()

Una SparkSession es el objeto principal o la base a partir de la cual cuelga toda la funcionalidad de Apache Spark para trabajar con SparkSQL, los DataFrame y DataSet tiene este constructor.

Al que se le pueden añadir una serie de parámetros o de nuevas funciones para añadir un nombre, indicar la cantidad de memoria y otros muchos aspectos de configuración; los métodos siguientes son:

master() – si se está ejecutando en el clúster, debe usar su nombre maestro como argumento se utiliza local[x] cuando se ejecuta en modo independiente. x debe ser un valor entero y debe ser mayor que 0; esto representa cuántas particiones debe crear al usar RDD, DataFrame y Dataset. Idealmente, el valor x debería ser la cantidad de núcleos de CPU que tiene.

appName() – establece un nombre para la aplicación Spark que se muestra en la interfaz de usuario web de Spark. Si no se establece un nombre de aplicación, establece un nombre aleatorio.

getOrCreate() – Esto devuelve un objeto SparkSession si ya existe. Crea uno nuevo si no existe.

- **val dataSet: Dataset[String] = spark.read.textFile("textfile.csv")**

Para poder ejecutar este commando debemos primero usar la librería

```
scala> import org.apache.spark.sql.Dataset
import org.apache.spark.sql.Dataset

scala> val dataSet: Dataset[String] = spark.read.textFile("textfile.csv")
org.apache.spark.sql.AnalysisException: Path does not exist: file:/home/ubuntu/textfile.csv
    at org.apache.spark.sql.errors.QueryCompilationErrors$.dataPathNotExistError(QueryCompilationErrors.scala:1011)
```

Spark SQL permite spark.read().csv("file_name") leer un archivo o directorio de archivos en formato CSV.

- **val df: DataFrame = dataSet.toDF()**

Para poder ejecutar este commando debemos primero usar la librería

```
scala> import org.apache.spark.sql.DataFrame
import org.apache.spark.sql.DataFrame

scala> val df: DataFrame = dataSet.toDF()
```

Una vez que tengamos un RDD, usemos toDF() para crear DataFrame en Spark. De forma predeterminada, crea nombres de columna como "_1" y "_2", ya que tenemos dos columnas para cada fila

Streaming

- **val streamingContext: StreamingContext = new StreamingContext(sparkContext, Seconds(20))**

Primero, importamos los nombres de las clases de Spark Streaming y algunas conversiones implícitas de StreamingContext para agregar métodos útiles a otras clases

```
scala> import org.apache.spark._
import org.apache.spark._

scala> import org.apache.spark.streaming._
import org.apache.spark.streaming._

scala> import org.apache.spark.streaming.StreamingContext._
import org.apache.spark.streaming.StreamingContext._

scala> val streamingContext: StreamingContext = new StreamingContext(sparkContext, Seconds(20))
<console>:41: error: not found: value sparkContext
      val streamingContext: StreamingContext = new StreamingContext(sparkContext, Seconds(20))
                                                                    ^
```

Creamos un StreamingContext local con dos subprocesos de ejecución y un intervalo de lote de 20 segundos.

- **val lines: ReceiverInputDStream[String] = streamingContext.socketTextStream("localhost", 9999)**

Este lines representa el flujo de datos que se recibirán del servidor de datos. Cada registro en este DStream es una línea de texto.

```
scala> import org.apache.spark.streaming.dstream.InputDStream
import org.apache.spark.streaming.dstream.InputDStream

scala> val lines: ReceiverInputDStream[String] = streamingContext.socketTextStream("localhost", 9999)
```

Este código representa la transmisión de datos desde una fuente TCP, especificada como nombre de host (p. ej. localhost) y puerto (p. ej. 9999).

RDD

- **val cadenas = Array("Docentes", "inteligenciaArtificial", "quefinal")**

Las colecciones paralelas se crean llamando SparkContext.parallelize() método los elementos de la colección se copian para formar un conjunto de datos distribuido que se puede operar en paralelo.

- **val cadenasRDD = sc.parallelize(cadenas)**

Una vez creado, el conjunto de datos distribuido se puede operar en paralelo un parámetro importante para las colecciones paralelas es la cantidad de particiones en las que se cortará el conjunto de datos.

- `cadenasRDD.collect()`

PySpark RDD/DataFrame `collect()` es una operación de acción que se usa para recuperar todos los elementos del conjunto de datos (de todos los nodos) al nodo controlador

- `val filtro = cadenasRDD.filter(line => line.contains("quefinal"))`

Ahora usemos una transformación la `filter` transformación para devolver un nuevo RDD con un subconjunto de los elementos del archivo.

- `val fileNotFound = sc.textFile("/7añljdsjd/alkls/", 6)`

`sparkContext.textFile()` El método se utiliza para leer un archivo de texto de HDFS, S3 y cualquier sistema de archivos compatible con Hadoop. Este método toma la ruta como argumento y, opcionalmente, toma varias particiones como segundo argumento.

- `fileNotFound.collect()`

```
scala> val cadenas = Array("Docentes","inteligenciaArtificial","quefinal")
cadenas: Array[String] = Array(Docentes, inteligenciaArtificial, quefinal)

scala> val cadenasRDD = sc . parallelize (cadenas)
cadenasRDD: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[0] at parallelize at <console>:29

scala> cadenasRDD.collect()
res0: Array[String] = Array(Docentes, inteligenciaArtificial, quefinal)

scala> file.collect()
<console>:28: error: not found: value file
file.collect()
^

scala> val filtro = cadenasRDD.filter(line => line.contains("quefinal"))
<console>:1: error: illegal character '\u201c'
val filtro = cadenasRDD.filter(line => line.contains("quefinal"))
^

<console>:1: error: illegal character '\u201d'
val filtro = cadenasRDD.filter(line => line.contains("quefinal"))
^

scala> val filtro = cadenasRDD.filter(line => line.contains("quefinal"))
filtro: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[1] at filter at <console>:28

scala> val fileNotFound = sc.textFile("/7añljdsjd/alkls/", 6)
fileNotFound: org.apache.spark.rdd.RDD[String] = /7añljdsjd/alkls/ MapPartitionsRDD[3] at textFile at <console>:28

scala> fileNotFound.collect()
org.apache.hadoop.mapred.InvalidInputException: Input path does not exist: file:/7añljdsjd/alkls
at org.apache.hadoop.mapred.FileInputFormat.validate(FileInputFormat.java:394)
```