EVIDENCIA DE LA UNIDAD I

Curso: TECNOLOGIAS Y HERRAMIENTAS BIG DATA

Tarea: Investigación 20%

Ricardo Mezo Bustamante mezo21r@gmail.com

Entrega

No entregado

Calificado

6 días 8 horas restante

El estudiante puede editar esta entrega

Comentarios (0)

Calificación

Calificación:

Hoja de presentación	No contiene todos los datos 0puntos	Datos incompletos 1puntos	Completo 2puntos
Índice	No contiene 0puntos	Muy pequeña 1puntos	Completa 2puntos
Contenido	No cubre los temas 0puntos	La mitad de los temas 8puntos	Completo 15puntos
Archivo PDF	Sin formato 0puntos		Correcto 1puntos

Calificación actual en el libro de calificaciones

20,00

INSTITUTO TECNOLÓGICO SUPERIOR DE SAN ANDRÉS TUXTLA

CARRERA:

INGENIERÍA INFORMÁTICA

MATERIA:

TECNOLOGÍAS Y HERRAMIENTAS DE BIG DATA

HADOOP

DOCENTE:

M.T.I JUAN RAFAEL GONZÁLEZ CADENA

ALUMNO:

RICARDO MEZO BUSTAMANTE

SAN ANDRÉS TUXTLA VERACRUZ, A 13 DE SEPTIEMBRE DEL 2023

Contenido

¿Qué es Apache Hadoop?

Debemos entender que Apache Hadoop es un framework de software que aporta la capacidad de ejecutar aplicaciones distribuidas y escalables, generalmente para el sector del **Big Data**. Así, permite a las aplicaciones hacer uso de miles de nodos de procesamiento y almacenamiento y petabytes de datos.

Hadoop es una de las tecnologías más populares en el ámbito de aplicaciones Big Data. Es usado en multitud de empresas como plataforma central en sus ***Data Lakes*** **(Lagos de datos)**, sobre la que se construyen los casos de uso alrededor de la explotación y el almacenamiento de los datos.

Además, es una plataforma sobre la que desarrollar para sacar partido a los datos de las organizaciones, por ejemplo, mediante técnicas y modelos de *machine learning*.
Entre sus principales ventajas se encuentra el hecho de que está diseñado para ejecutar sobre hardware sencillo, normal y corriente. La potencia del sistema se encuentra en que todos los nodos juntos actúan como un clúster con la capacidad de escalar horizontalmente al agregar más nodos.

Historia

En el año 2006, los dos componentes que formaban parte de Hadoop: MapReduce y HDFS se cedieron a la *Apache Software Foundation* como proyecto **open source**. Esto impulsó su adopción como herramienta Big Data en proyectos en muchas industrias. El proyecto fue desarrollado en el **lenguaje de programación Java**.

La versión 1.0 de Hadoop fue publicada en el año 2012. La versión 2.0 se publicó en el año 2013 añadiendo Yarn como gestor de recursos y desacoplando HDFS de MapReduce. En el año 2017 se publicó Hadoop 3.0 añadiendo mejoras.

Arquitectura de Hadoop

Anteriormente, en los sistemas tradicionales, las tecnologías se han enfocado en traer los datos a los sistemas de almacenamiento. Sin embargo, en los procesos Hadoop, se trata de **acercar el procesamiento** al lugar en donde se encuentran almacenados los datos y así aprovechar técnicas de paralelización, aumentando de manera importante la escalabilidad y el rendimiento de los sistemas que trabajan con grandes cantidades de datos.

La arquitectura de Hadoop y su diseño está basado en la idea de que mover el procesamiento es mucho más rápido, fácil y eficiente que mover grandes cantidades de datos, que pueden producir altas latencias y congestión en la red. El sistema de ficheros distribuido de Hadoop (HDFS) proporciona a las aplicaciones la capacidad de acceder a los datos en el lugar en el que se encuentren almacenados.

Componentes Principales

Para comprender completamente cómo funciona Hadoop debemos entender sus tres componentes principales. Hemos dedicado una entrada a cada uno de ellos:

- **MapReduce**
- **Yarn** (Yet Another Resource Negotiator)
- **HDFS** (Hadoop Distributed File System)

Además de estos componentes, el módulo **Hadoop Common** aporta las utilidades y librerías comunes del proyecto que soportan el resto de componentes y permiten desarrollar las aplicaciones.

Apache Hadoop 1 vs Hadoop 2

HDFS (Hadoop Distributed File System) es el componente principal del ecosistema Hadoop. Esta pieza hace posible almacenar data sets masivos con tipos de datos estructurados, semi-estructurados y no estructurados. Proporciona la división de los datos en bloques que necesita MapReduce para ejecutar sus fases map y reduce. Está optimizado para almacenar grandes cantidades de datos y mantener varias copias en el clúster para garantizar una alta disponibilidad y tolerancia a fallos.

Por último, Yarn (Yet Another Resource Negotiator) es una pieza fundamental en el **ecosistema Hadoop**. Es el framework que permite soportar varios motores de ejecución incluyendo MapReduce, y que proporciona un planificador agnóstico a los trabajos que se encuentran en ejecución en el clúster. También se encarga de proporcionar los recursos computacionales necesarios para los trabajos como memoria o cpu.

Ecosistema Hadoop

Alrededor del framework de Hadoop, compuesto por las tres piezas fundamentales: MapReduce, HDFS y Yarn, han surgido todo un conjunto de tecnologías que la complementan y cumplen funciones específicas. Por ejemplo existen tecnologías que facilitan la ingesta de datos hacia el clúster de Hadoop, otras que aceleran el procesamiento o bien facilitan la búsqueda de datos.
En cuanto a las distribuciones de Hadoop, la distribución más extendida y usada en la actualidad es **Cloudera**. Tras la fusión con Hortonworks se ha convertido en líder de mercado. Se ha encargado de agrupar e integran gran cantidad de estas tecnologías para poder realizar los despliegues de lagos de datos de una manera sencilla y acorde a las necesidades.

Ecosistema Hadoop y Componentes

Tecnologías relacionadas

Los proyectos open source más populares del ecosistema formado alrededor de Apache Hadoop seguramente te sonarán. Se trata de los siguientes:

- **Spark**: Motor de procesamiento en memoria compatible con HDFS. Aumenta la velocidad de MapReduce en 100 veces. Soporta aplicaciones ETL, Machine learning y Streaming de datos así como consultas SQL.
- **Ambari**: Herramienta para gestionar y provisionar clústers de Apache Hadoop y tecnologías relacionadas. Proporciona una interfaz web sencilla y amigable para visualizar y monitorizar el estado del sistema y de todos sus componentes, así como establecer alertas y visualizar estadísticas.
- **Oozie**: Permite ejecutar y planificar en el tiempo trabajos y tareas en Hadoop mediante configuraciones XML.
- **Pig**: Proporciona el lenguaje de programación Pig Latin, con sintaxis parecida a SQL. Transforma los programas en sentencias MapReduce que ejecutan en un clúster Hadoop.

- **Storm**: Componente encargado de procesar flujos de datos en tiempo real. Su uso suele ir acompañado de Apache Kafka.
- **Tez**: Framework de programación de flujos de datos. Es la evolución de MapReduce que ejecuta sobre Yarn optimizando el código para alcanzar mejoras de hasta 10 veces en el rendimiento. Muchas tecnologías están adoptando Tez como motor de ejecución principal.
- **Zookeeper**: Servicio de coordinación para aplicaciones distribuidas, tolerante a fallos. Generalmente, se despliega en 3 nodos.

Consulta de datos

- **Hive**: Proporciona la interfaz SQL a Hadoop necesaria para convertirse en un *Data Warehouse*. Transforma las consultas SQL en trabajos MapReduce sobre datos estructurados. Hive no es apropiado para realizar consultas de baja latencia.
- **HBase**: Base de datos NoSQL distribuida y escalable para datos estructurados sobre HDFS con formato de columnas y operaciones CRUD (Create, Read, Update, Delete).
- **Hue**: Permite escribir consultas SQL y conectar con los catálogos y con las bases de datos del clúster para analizar datos de forma interactiva.
- **Impala**: Base de datos SQL sobre Hadoop. Proporciona la capacidad de realizar consultas concurrentes y de baja latencia para analítica y ***Business Intelligence*** **(BI)**.

Ingesta de datos

Además de estas tecnologías, debemos comprender el conjunto de tecnologías que se usan para introducir datos en un clúster Hadoop. Este proceso se llama ingesta de datos y generalmente se lleva a cabo por una combinación de los siguientes componentes:

- **Flume**: Actúa como buffer para introducir datos en el lago.
- **Sqoop**: Herramienta especializada para ingestas de datos con fuente en bases de datos externas y con destino HDFS.
- **Kafka**: Sistema muy usado con un modelo pub-sub que divide los datos en temas con bajas latencias. Los consumidores pueden suscribirse para recibir los mensajes.

Alternativas a Hadoop

Para evaluar las alternativas a Hadoop que existen en el mercado debemos tener en cuenta que menudo podemos integrar diversas tecnologías para resolver el problema que se nos plantea.

Apache Hadoop ha formado todo un ecosistema alrededor muy flexible, del que debemos seleccionar solamente los componentes que necesitemos para evitar recargar nuestro sistema con piezas a las que no se va a sacar partido.

Podemos poner el conocido **Hadoop vs Spark** como ejemplo. Ambas tecnologías pueden convivir perfectamente en una solución Big Data completa. HDFS proporcionarán la base sobre la que podrán ejecutar las cargas Spark en el caso de que necesitemos realizar transformaciones de los datos complejas. De esta forma, nos podremos aprovechar de un sistema de almacenamiento distribuido y con las réplicas necesarias para garantizar la alta disponibilidad del dato.

Servicios Cloud

No debemos olvidar que los proveedores cloud como **AWS o Azure** proporcionan servicios gestionados basados en Apache Hadoop. Por ejemplo, con el servicio **HDInsight de Microsoft Azure**, podemos desplegar y escalar clústers de varios tipos predefinidos de una manera sencilla y transparente. Con **EMR, en AWS**, podemos desplegar el ecosistema de Hadoop. Estos servicios tienen muchas ventajas asociadas al ahorro de costes de infraestructura y de gestión y es importante analizarlos.

Además de su uso como servicio, podemos considerar alternativas basadas en la nube como **Databricks**. Este servicio proporciona Spark como servicio, pudiendo desplegar clústers y cargas de trabajo bajo demanda y con escalabilidad automática. Tenemos que tener en cuenta que este tipo de soluciones resuelven el problema del procesamiento de nuestros datos pero no proporcionan el sistema de almacenamiento como HDFS.

Si el objetivo es obtener una plataforma del dato como Data Warehouse totalmente gestionado se pueden evaluar otras soluciones como **Snowflake**. Esto permite abstraer aún más el despliegue al ser software como servicio. Hadoop no se trata de una base de datos NoSQL y tiene unos costes asociados a su complejidad.

¿Cómo integrar Hadoop en tu empresa?

Si deseas integrar Hadoop en tu empresa, ten en cuenta las siguientes consideraciones, pasos y mejores prácticas

1. **Define el uso**: Usa metas pequeñas y alcanzables que ayuden a procesar los datos. Comienza por definir las formas de acceder a los datos que necesitan los usuarios.
2. **Calidad del dato:** Debemos monitorizar Hadoop como cualquier otra herramienta más en la organización.
3. **Uso de frameworks existentes**: No es necesario inventar nuevos métodos, usa frameworks que ya existen en tu organización para supervisar el acceso a los datos como Spring. De esta forma, los desarrolladores podrán centrarse en la lógica de negocio y en implementar nuevas estrategias.
4. **Linaje de datos**: Debemos realizar un seguimiento del linaje de los datos a través de sus metadatos. Existen herramientas que permiten realizar este seguimiento desde el origen hasta el destino.
5. **Modelado de datos**: Modela los datos de acuerdo a los patrones de procesamiento y de acceso.
6. **Seguridad**: Implementa seguridad de acceso basada en Active Directory y LDAP. Refuerza la seguridad del clúster con servicios como Sentry.

Siguientes Pasos, Formación y Cursos de Hadoop

Aquí tienes dos cursos muy recomendados con los que consolidarás conceptos fundamentales para convertirte en experto de estas tecnologías fundamentales para los ingenieros de datos.

¿Para qué sirve Hadoop?

El uso principal de Hadoop es de almacenar y procesar grandes cantidades de datos (Big Data). Proporciona un framework con alta disponibilidad, escalabilidad y tolerancia a fallos, lo que la convierten en una buena solución para convertirse en lagos de datos para organizaciones.

¿Cómo funciona Hadoop?

Hadoop es un sistema distribuido con tres componentes principales: HDFS, MapReduce y Yarn. HDFS proporciona el sistema de ficheros distribuido dividiendo los ficheros de datos en bloques. MapReduce es el modelo de procesamiento dividiendo el trabajo en múltiples tareas independientes y paralelizables. Yarn se encarga de planificar los trabajos y de gestionar los recursos hardware.

PRACTICAS 40%

Curso: TECNOLOGIAS Y HERRAMIENTAS BIG DATA
Tarea: Prácticas

Ricardo Mezo Bustamante mezo21r@gmail.com

Entrega

Entregado
Calificado
5 días 11 horas restante
El estudiante puede editar esta entrega

Comentarios (0)

Calificación

Calificación:

Practicas Herramientas de Big Data

Portada	No contiene 0puntos	Contiene 1puntos
Reporte con capturas	No envió 0puntos	Si envió 13puntos
Funcionamiento de código	No funciona 0puntos	Funciona 25puntos
Formato de archivo PDF	Otro formato 0puntos	Con formato 1puntos

Calificación actual en el libro de calificaciones

40.00

Comentarios de retroalimentación

INSTITUTO TECNOLÓGICO SUPERIOR DE SAN ANDRÉS TUXTLA

CARRERA:

INGENIERÍA INFORMÁTICA

MATERIA:

TECNOLOGÍAS Y HERRAMIENTAS DE BIG DATA

PRÁCTICA: INSTALACIÓN DEL ECOSISTEMA APACHE SPARK

DOCENTE:

M.T.I JUAN RAFAEL GONZÁLEZ CADENA

ALUMNO:

RICARDO MEZO BUSTAMANTE

SAN ANDRÉS TUXTLA VERACRUZ, A 23 DE SEPTIEMBRE DEL 2023

INSTALACIÓN Y CONFIGURACIÓN DE APACHE SPARK

Requerimientos de Software:

1. Java
2. Python (Python 3.10.5)
3. Apache spark (spark-3.2.2-bin-hadoop2.7)
4. Hadoop 2.7 (winutils)
5. Paquetes pyspark, findspark y jupyter
6. Winutils (https://github.com/steveloughran/winutils)
7. Conexión a Internet

Instalación y configuración.

Si no se tiene instalado java, se debe de instalar, de preferencia en la siguiente ruta C:\java.

De igual forma con Python, en el caso de que no esté instalado, instalarlo en C:\Python

se descarga el repositorio completo

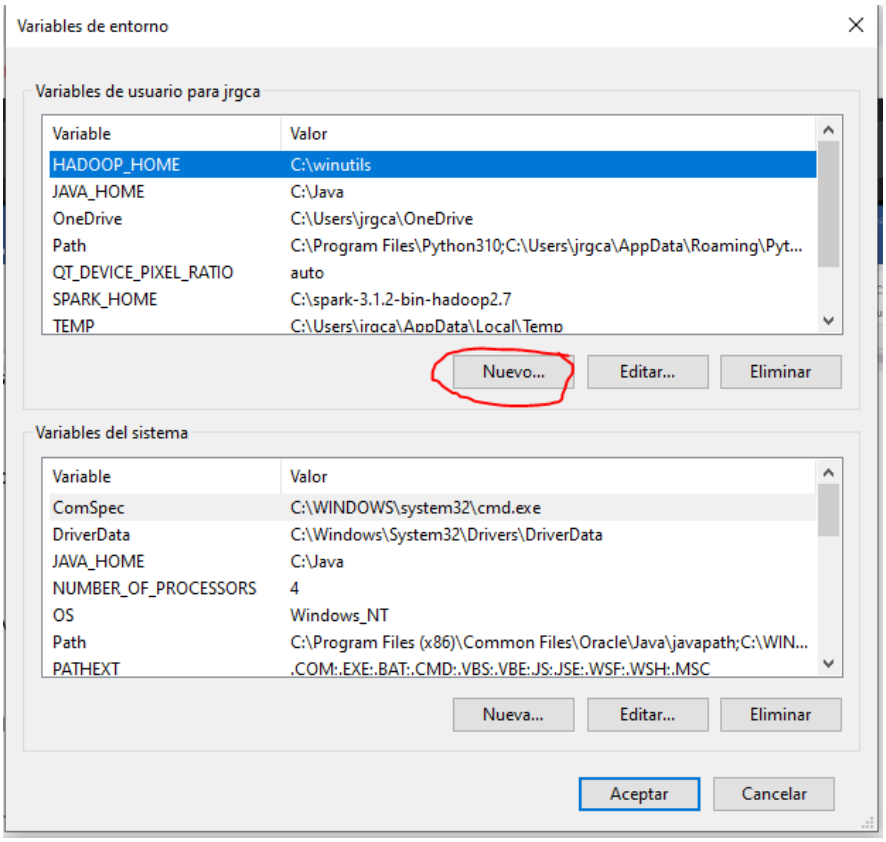
1. Se debe instalar java (de preferencia se instala en c:\java2)
2. Se debe instalar python (de preferencia se instala en c:\Python)
3. Descomprimimos hadoop- spark en la carpeta C:\spark3.2.2binhadoop1.7
4. Creamos una carpeta en C que se llame “winutils” y dentro otra carpeta con el nombre de “bin”… C:\winutils\bin
5. Abrimos winutils con winrar, abrimos la carpeta hadoop 2.7.1, abrimos la carpeta bin Y extraemos el archivo winutils.exe y lo agregamos en una carpeta que se llame C:\winutils\bin
6. Creamos en la unidad C una carpeta con el nombre de “tmp” y dentro creamos la carpeta “hive”

Creamos las variables de entorno

Para crear las variables de entorno debemos de abrir propiedades del sistema-<seleccionamos variables de entono y a continuación crearemos las siguientes variables

- JAVA_HOME
- SPARK_HOME
- HADOOP_HOME
- PYTHON_HOME

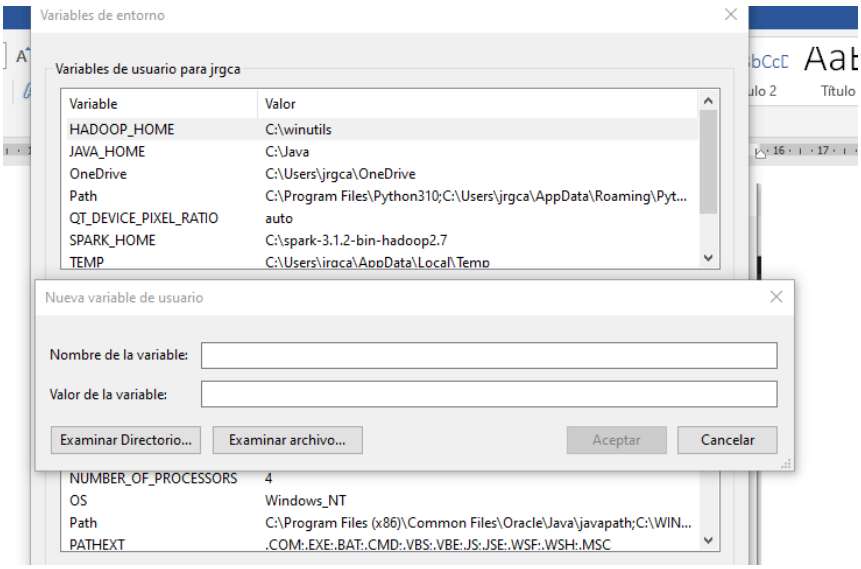
Para eso debemos de seleccionar el botón “Nuevo”



En la ventana que aparece escribiremos, por ejemplo

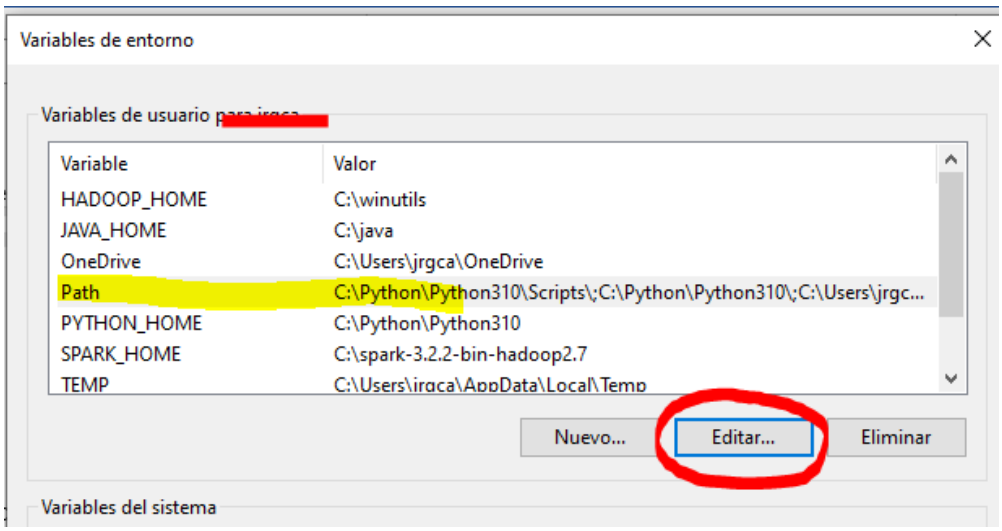
Nombre de la variable: HADOOP_HOME

Valor de la variable: C:\winutils (se debe poner las rutas donde se encuentren las carpetas “bin”)

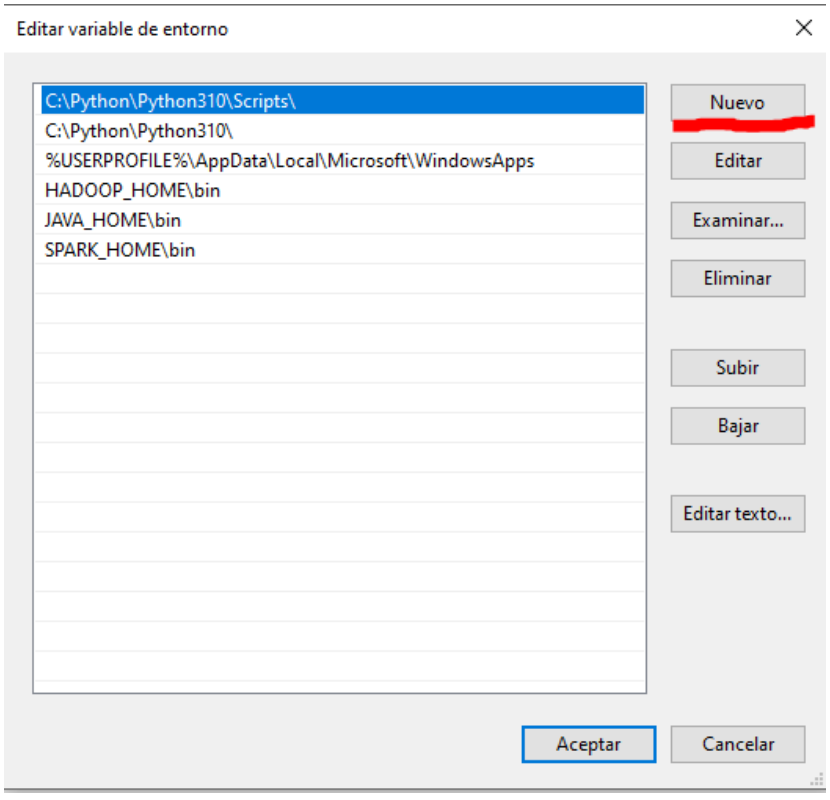


Esto debemos de hacer para todas las variables que vamos a agregar.

A continuación se debe editar el “PATH”, esto lo hacemos de la siguiente forma

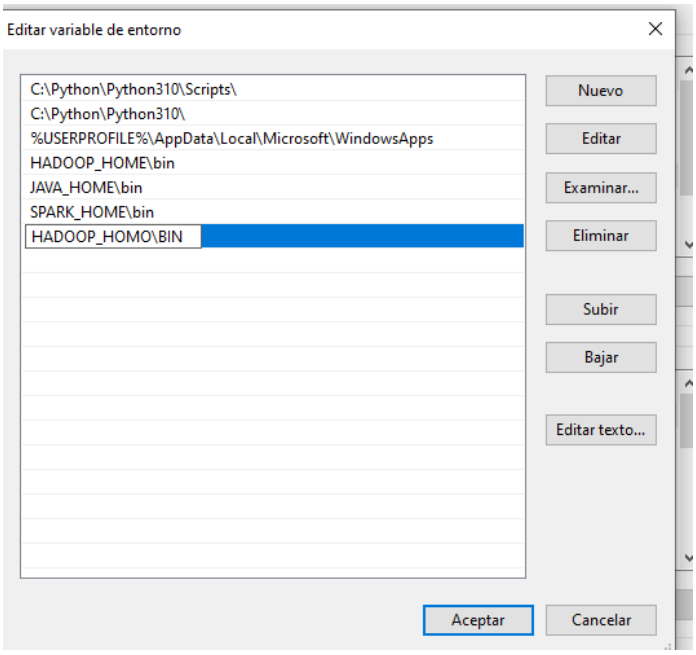


Seleccionamos “Path” y a continuación se hace clic en “Nuevo”



A continuación escribimos la variable de usuario que definimos y le agregamos “bin” por ejemplo:

HADOOP_HOME\bin



Al finalizar de agregar los datos en el “Path” presionamos “Aceptar”

Modificar la carpeta C:\tmp\Hive

Abrimos una terminal (CMD) y entramos a la carpeta donde esta winutils:

C:\winutils\bin

Escribimos el comando chmod 777 para cambiar los permisos de administrador de la siguiente forma

winutils chmod 777 C:\tmp\hive *(presionamos enter)*

Instalación de las librerías pyspark y findspark

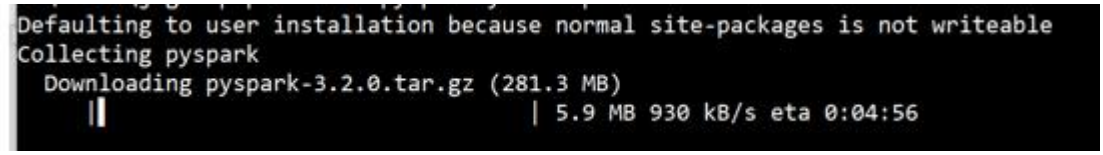
Para instalar estas librerías debemos de agregar lo siguiente en el "Path"(variables de usuario):

C:\Python\Python310\Scripts

Esto es necesario porque se utilizará el comando "pip" de Python que sirve para ejecutar ciertas ordenes, en este caso vamos a utilizar la orden "install", una vez hecho esto ejecutamos lo siguiente en la consola:

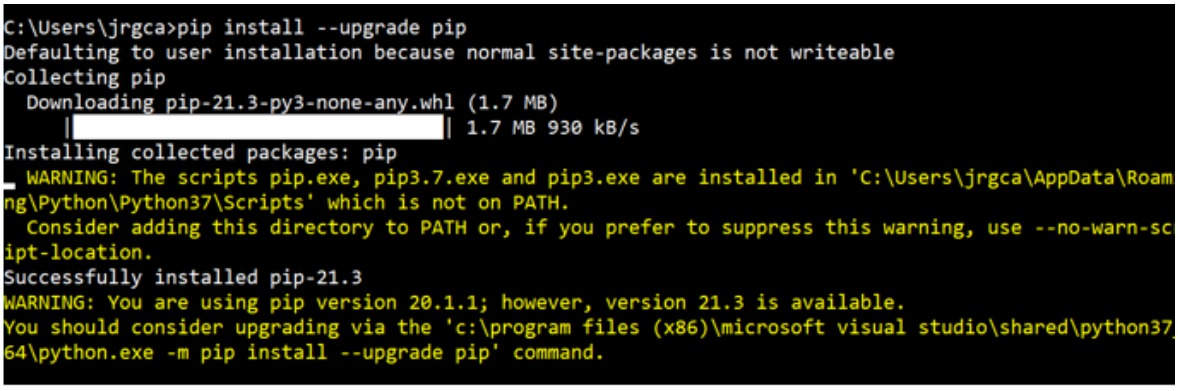
C:\pip install pyspark findspark

NOTA: en el caso de que aparezca el siguiente mensaje se debe a que hay que actualizar ya sea Python o solo "pip"



Para actualizar "pip" debemos de escribir lo siguiente en la consola:

C:\pip install –upgrade pip



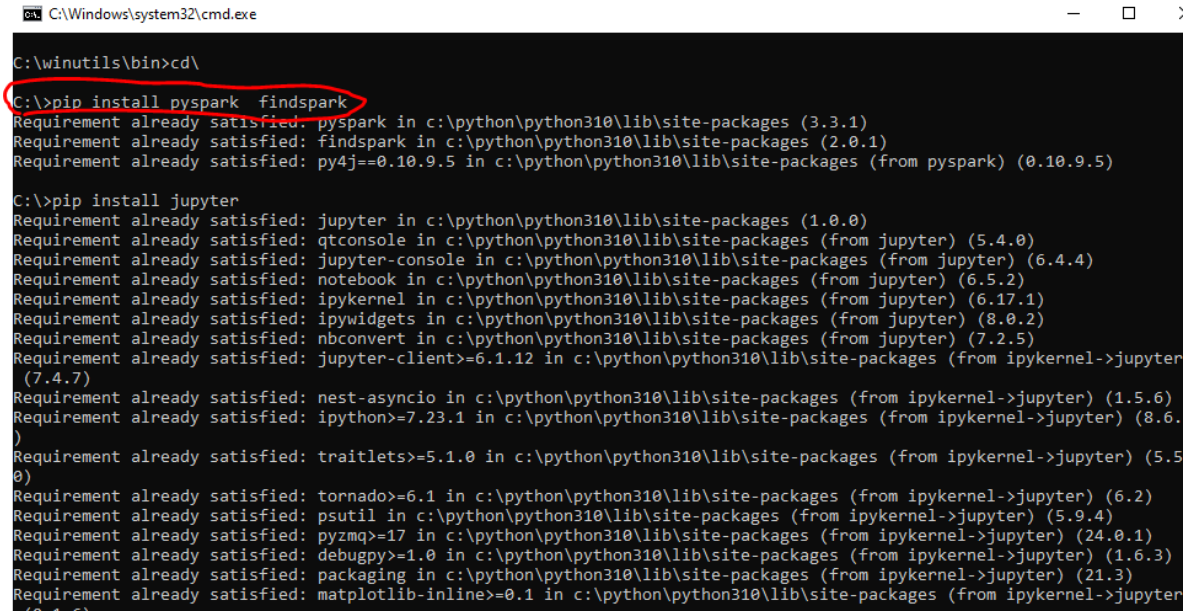
Después de escribir ***c:\pip install pyspark findspark*** presionamos "enter" empezara la descarga de paquetes, en el caso de que no acepte la instrucción como se puso, debemos de hacerlo por separado: ***c:\pip install pyspark*** y ***c:\pip install findspark.***

Para comprobar la instalación debemos de valernos de un editor de código para este caso utilizaremos “Jupyter”

Instalación de Jupyter

En la terminal (cmd) escribimos lo siguiente:

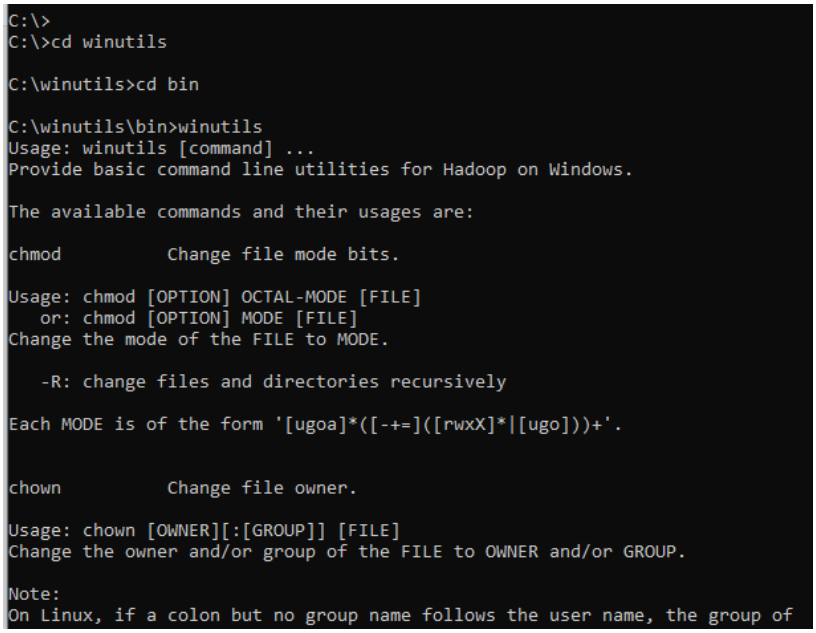
pip install jupyter



Una vez terminado el proceso, ya podemos usarlo

Verificamos que se instaló todo correctamente.

Desde la consola ejecutamos winutils.exe para iniciar “hadoop”



```
        The cpu limit is an integral value of percentage * 100. The memory
        limit is an integral number of memory in MB.
        The limit will not be set if 0 or negative value is passed in as
        parameter(s).

      task createAsUser [TASKNAME] [USERNAME] [PIDFILE] [COMMAND_LINE]
        Creates a new task jobobject with taskname as the user provided

      task isAlive [TASKNAME]
        Checks if task job object is alive

      task kill [TASKNAME]
        Kills task job object

      task processList [TASKNAME]
        Prints to stdout a list of processes in the task
        along with their resource usage. One process per line
        and comma separated info per process
        ProcessId,VirtualMemoryCommitted(bytes),
        WorkingSetSize(bytes),CpuTime(Millisec,Kernel+User)
service         Service operations.

    Usage: service
    Starts the nodemanager Windows Secure Container Executor helper service.
    The service must run as a high privileged account (LocalSystem)
    and is used by the nodemanager WSCE to spawn secure containers on Windows.


C:\winutils\bin>
```

Ejecutamos “jupyter”

Escribimos ***jupyter notebook***

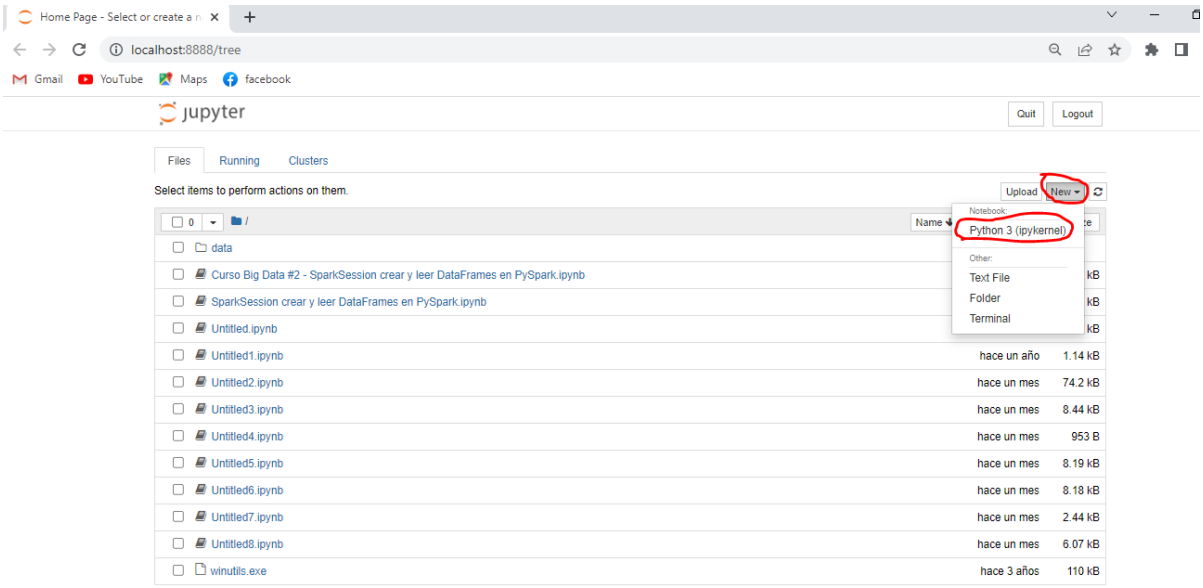
```
C:\winutils\bin>jupyter notebook
```

NOTA: los comando se pueden ejecutar desde cualquier directorio del sistema (C:\jupyter notebook)

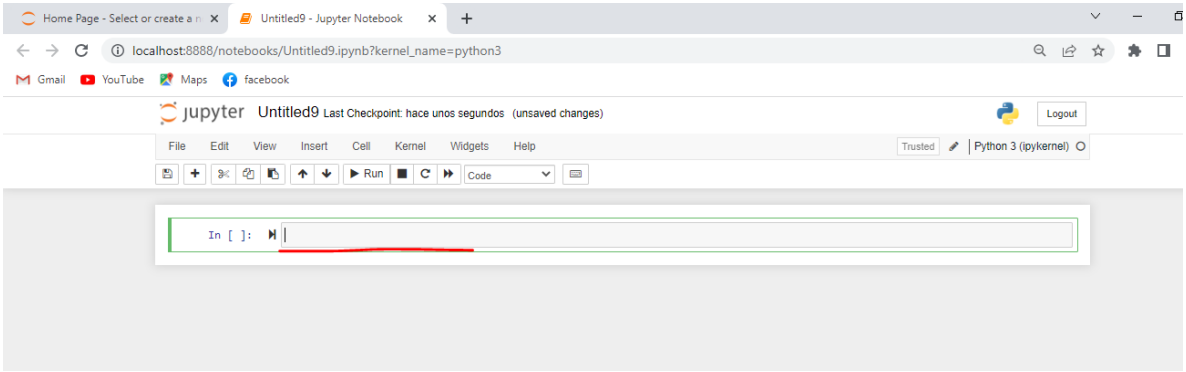
```
C:\winutils\bin>jupyter notebook
[I 11:00:57.613 NotebookApp] Serving notebooks from local directory: C:\winutils\bin
[I 11:00:57.613 NotebookApp] Jupyter Notebook 6.5.2 is running at:
[I 11:00:57.614 NotebookApp] http://localhost:8888/?token=08557af4da744ea2111043d15fe2ce5ca9bd0c900e1487ef
[I 11:00:57.614 NotebookApp]  or http://127.0.0.1:8888/?token=08557af4da744ea2111043d15fe2ce5ca9bd0c900e1487ef
[I 11:00:57.615 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 11:00:57.669 NotebookApp]

    To access the notebook, open this file in a browser:
        file:///C:/Users/jrgca/AppData/Roaming/jupyter/runtime/nbserver-1432-open.html
    Or copy and paste one of these URLs:
        http://localhost:8888/?token=08557af4da744ea2111043d15fe2ce5ca9bd0c900e1487ef
     or http://127.0.0.1:8888/?token=08557af4da744ea2111043d15fe2ce5ca9bd0c900e1487ef
```

Una vez ejecutado se abrirá el navegador que tengamos pregerminado en la computadora y mostrará lo siguiente.



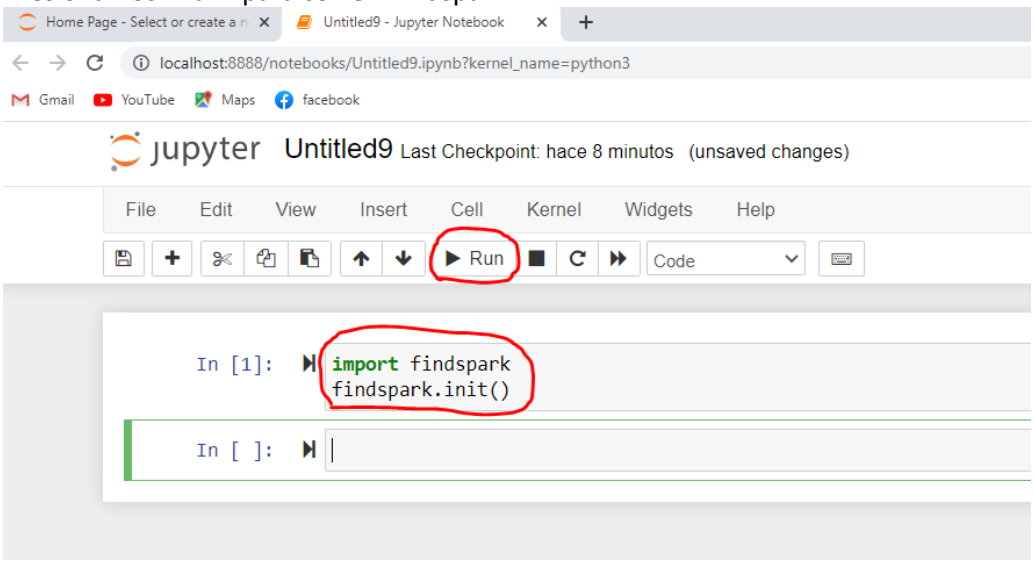
Debemos de hacer clic en “New” y en el menú desplegable seleccionar “Python 3” y a continuación mostrará lo siguiente:



Desde ese momento ya podemos escribir código.

Inicializar Spark

Primero iniciamos findspark escribiendo lo siguiente
import findspark
findspark.init()
Presionamos “Run” para correr “findspark”



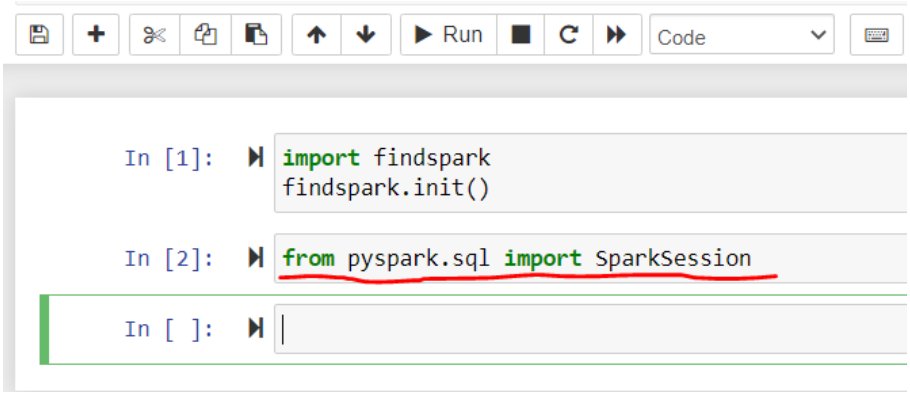
Si no hubo errores mostrará un nuevo bloque de código.

Creamos una sesión en spark

Escribimos

from pyspark.sql import SparkSession

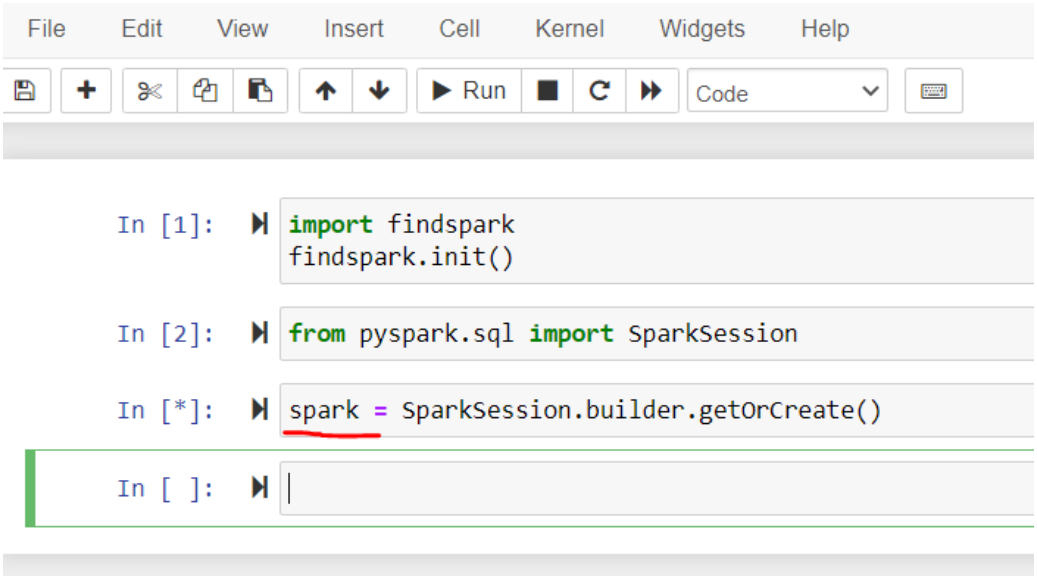
y presionamos Run



A continuación creamos la sesión “spark” escribimos:

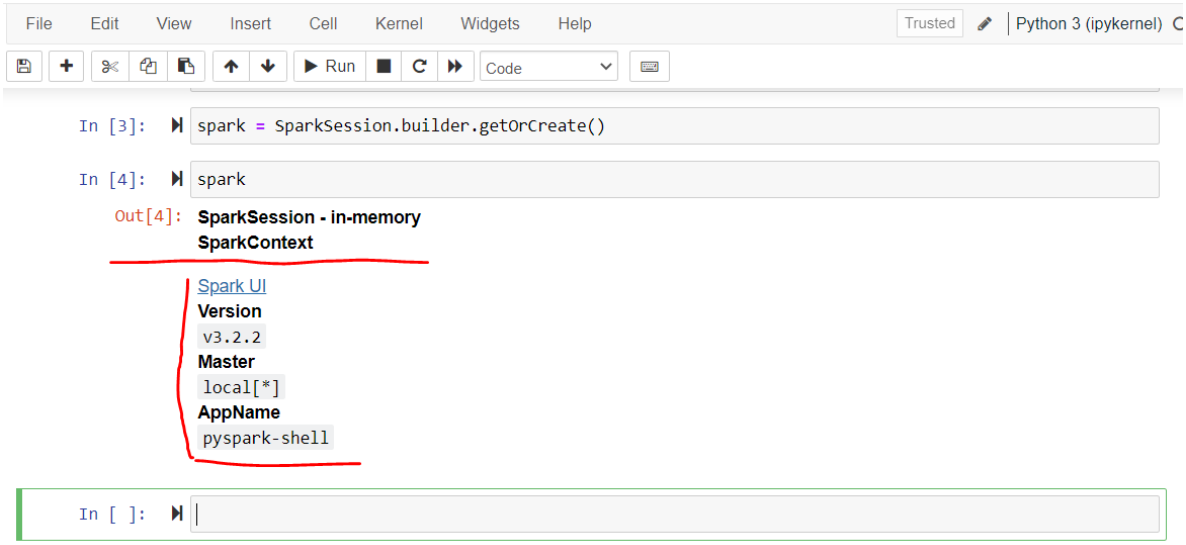
spark = SparkSession.builder.getOrCreate()

presionamos Run



Para finalizar probamos la sesión escribiendo solamente

Spark



EXAMEN 40%

Tarea: Examen practico

Ricardo Mezo Bustamante mezo21r@gmail.com

Entrega

Entregado

Calificado

11 horas antes de la entrega

El estudiante puede editar esta entrega

Calificación

Calificación:

Practicas

Hoja de presentación	No contien todos los datos 0puntos	Datos incompletos 1puntos	Completo 2puntos
Practicas	No contiene 0puntos	Parcialmente 20puntos	Todas 38puntos

Calificación actual en el libro de calificaciones

40.00

EXAMEN: SparkSession crear y leer DataFrames en PySpark

```
import findspark
findspark.init()
```

In [2]:
```
from pyspark.sql import SparkSession
```

1. Crear la SparkSession

In [3]:
```
spark = SparkSession.builder.appName('firstSession')\
    .config('spark.master',  'local[4]')\
    .config('spark.executor.memory',  '1g')\
    .config("spark.sql.shuffle.partitions",  1)\
    .config('spark.driver.memory','1g')\
    .getOrCreate()
```

In [4]:
```
spark
```

Out[4]:

SparkSession - in-memory

SparkContext

Spark UI

Version	`v3.1.1`
Master	`local[4]`
AppName	`firstSession`

Se puede obtener y cambiar la configuración inicial dada

In [5]:
```
spark.conf.get('spark.sql.shuffle.partitions')
```

Out[5]:
```
'1'
```

In [6]:
```
spark.conf.set("spark.sql.shuffle.partitions",  2)
```

In [7]:
```
spark.conf.get('spark.sql.shuffle.partitions')
```

Out[7]: '2'

```
[('spark.sql.shuffle.partitions', '1'), ('spark.app.name', 'firstSession'),
   ('spark.driver.host', 'DESKTOP-1EE24UT'), ('spark.master', 'local[4]'),
```

Out[7]: '2'

```
 ('spark.executor.id', 'driver'),
 ('spark.driver.memory', '1g'),
 ('spark.driver.port', '65081'),
```

In [28]:
```
spark.sparkContext.getConf().getAll()
```

Out[28]:
```
 ('spark.sql.warehouse.dir',
  'file:/C:/Users/rodgo/errodringer/cursoBigData/02_PrimerosPasosConPySpark/code/spark-warehouse'),
 ('spark.executor.memory', '1g'),
 ('spark.rdd.compress', 'True'),
 ('spark.serializer.objectStreamReset', '100'),
 ('spark.app.startTime', '1618732647355'),
('spark.submit.pyFiles', ''),
('spark.submit.deployMode', 'client'),
('spark.default.parallelism', '4'),
('spark.ui.showConsoleProgress', 'true')]
```

Con el comando:

`spark.stop()`

detenemos la aplicación. Lo veremos al final, ahora no interesa parar

```
# spark.stop()
```

2. Crear tabla

In [9]:

2.1 A partir de una lista

In [9]:
```
columnas = ["id", "nombre","l"]
lista = [(1, "Errodringer", "a"), (2, "Paco", "b"), (3, "Hola", "c"), (4, "Adios",
lista
```

Out[9]:
```
[(1, 'Errodringer', 'a'),
 (2, 'Paco', 'b'),
 (3, 'Hola', 'c'),
 (4, 'Adios', 'd')]
```

In [10]:
```
df_1 = spark.createDataFrame(lista, schema=columnas)
```

Número total de registros (filas)

In [11]:
```
df_1.count()
```

Out[11]:
```
4
```

Mostramos *n* registros, indicando este parámetro como entrada en la función:

```
show(n)
```

In [12]:
```
df_1.show(2)
```

```
+---+-----------+---+
| id|     nombre|  l|
+---+-----------+---+
|  1|Errodringer|  a|
|  2|       Paco|  b|
+---+-----------+---+
only showing top 2 rows
```

Columnas del DataFrame

In [13]:
```
df_1.columns
```

Out[13]:
```
['id', 'nombre', 'l']
```

Schema del DataFrame

In [14]:
```
df_1.printSchema()
```

```
root
 |-- id: long (nullable =true)
 |-- nombre: string (nullable = true)
 |-- l: string (nullable = true)
```

In [15]:
```
df_1.describe().show()
```

```
+-------+------------------+------+----+
|summary|                id|nombre|   l|
+-------+------------------+------+----+
|  count|                 4|     4|   4|
|   mean|               2.5|  null|null|
| stddev|1.2909944487358056|  null|null|
|    min|                 1| Adios|   a|
|    max|                 4|  Paco|   d|
+-------+------------------+------+----+
```

In [16]:
```
from pyspark.sql.types import StructType, StructField, IntegerType, StringType

schema_1 = StructType([
    StructField("id", IntegerType(), True),
    StructField("name", StringType(), True),
    StructField("l", StringType(), True)])

df_11 = spark.createDataFrame(lista, schema=schema_1)
```

In [17]:
```
df_11.printSchema()
```

```
root
 |-- id: integer (nullable = true)
 |-- name: string (nullable = true)
 |-- l: string (nullable = true)
```

In [18]:
```
df_11.show()
```

```
+---+-----------+---+
| id|       name|  l|
+---+-----------+---+
|  1|Errodringer|  a|
|  2|       Paco|  b|
|  3|       Hola|  c|
|  4|      Adios|  d|
+---+-----------+---+
```