Tutorial uso Spark con Notebooks y PySpark

Pablo Moreno Quintero

Escuela de Ciencias e Ingeniería, Universidad EAFIT

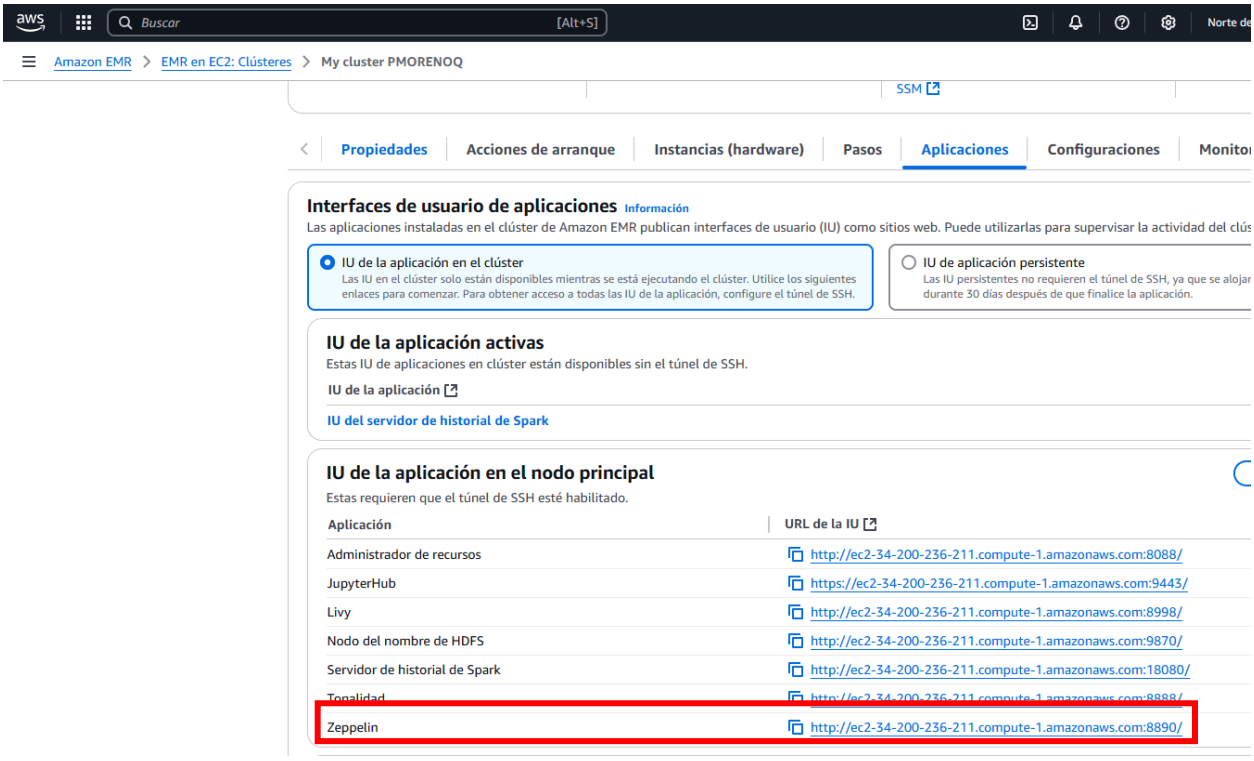
Pregrado en Ingeniería de Sistemas

Edwin Nelson Montoya Munera

23 de noviembre de 2024

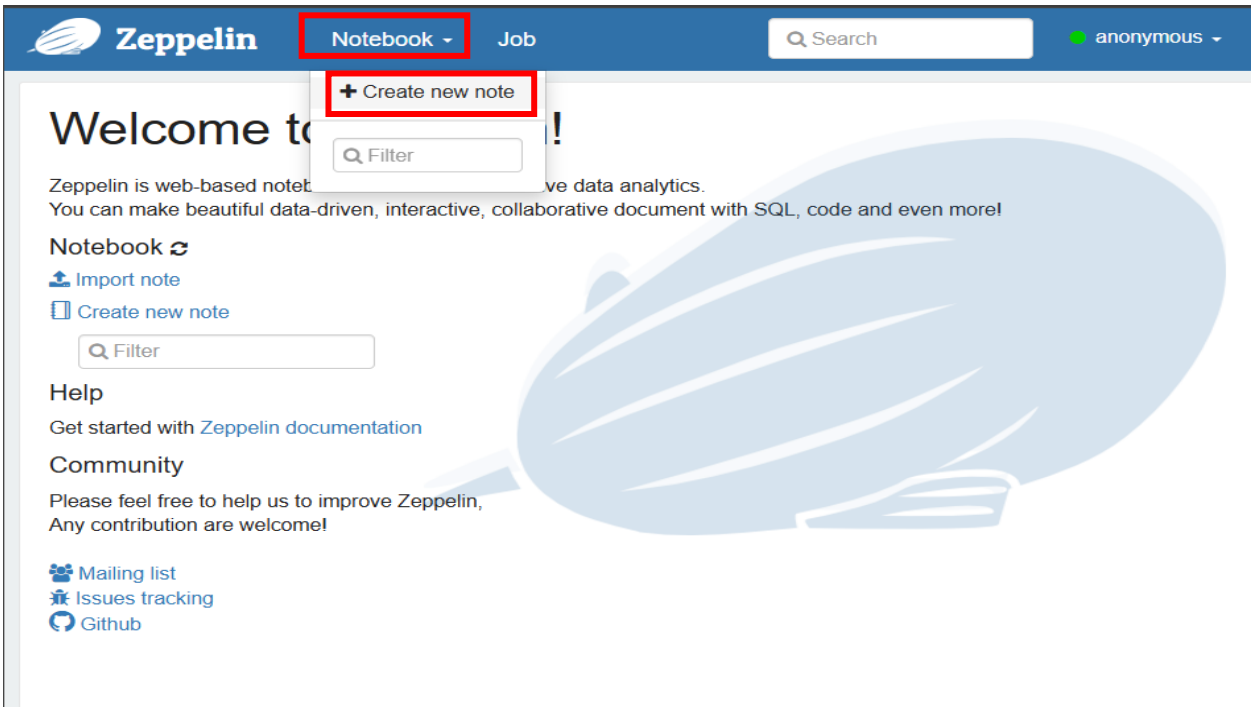
1. Uso de Zepellin notebook

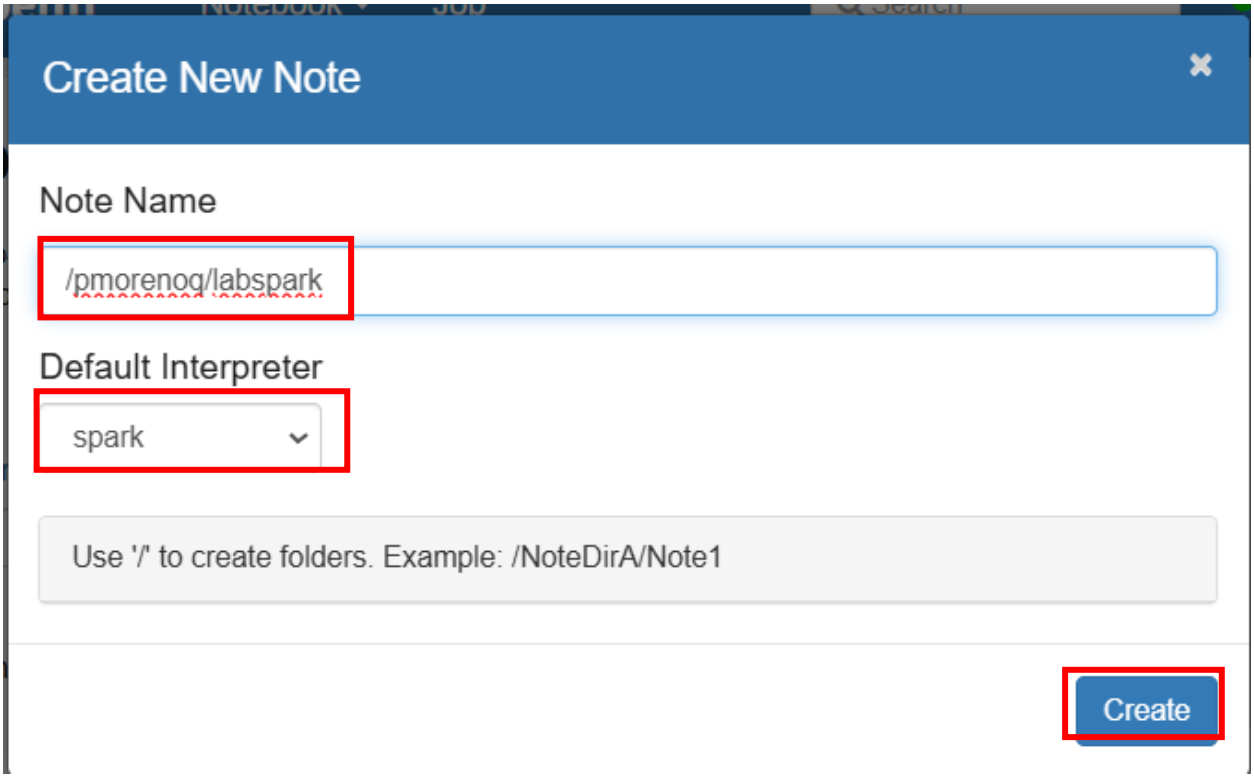
Acceder al servicio de Zepellin mediante el nodo máster



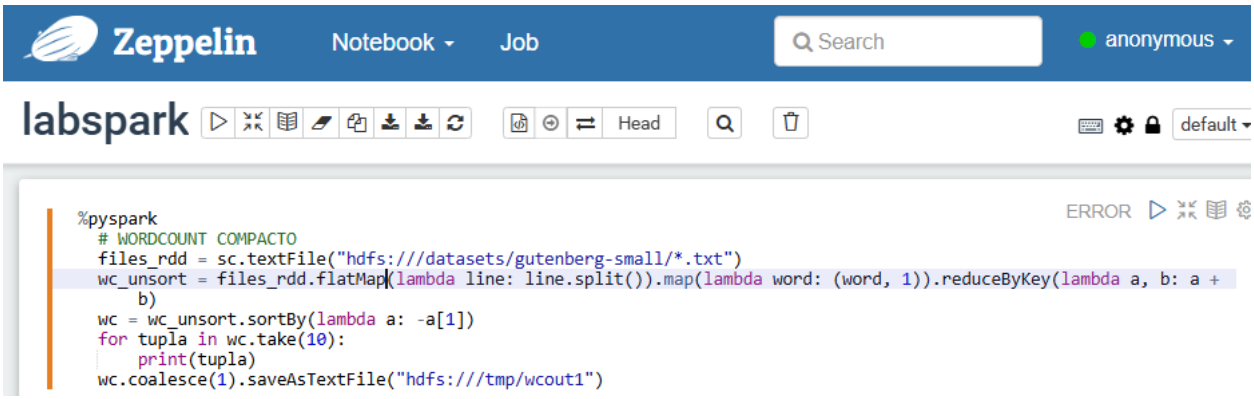
Al estar ingresando mediante EMR no es necesario ingresar credenciales

En la interfaz seleccionamos notebook y creado una nota





Ingresamos el siguiente código para correr algún programa

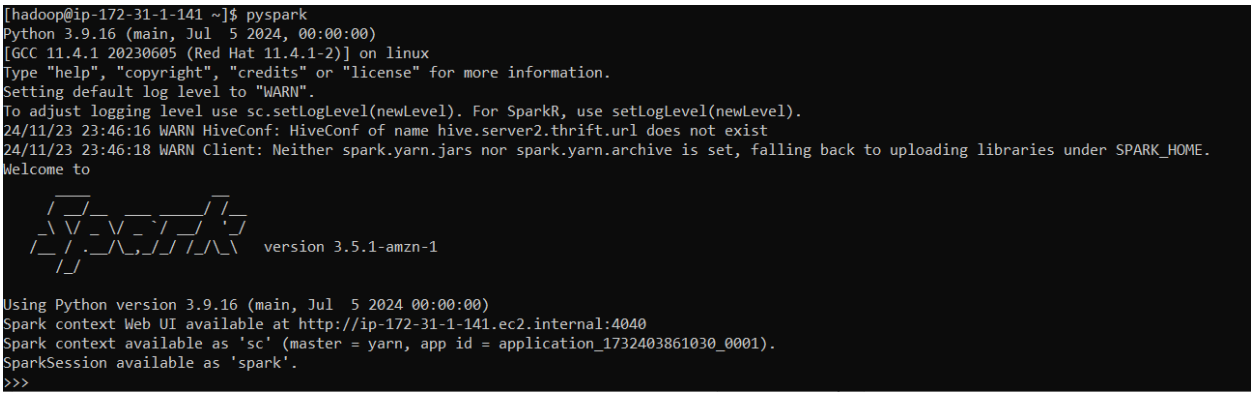


*Algunos de los códigos requieren ser ejecutados en JupyterNotebooks previos a poder ser usados

2. Realizar operaciones mediante pyspark

Establecer conexión ssh con el nodo máster

Entrar al entorno de pyspark



Allí podremos realizar la configuración para spark necesaria con los siguientes comandos_

$ pyspark

>>> files_rdd = sc.textFile("hdfs:///datasets/gutenberg-small/*.txt")

>>> files_rdd = sc.textFile("s3://<nombre bucket>/gutenberg-small/*.txt")

>>> wc_unsort = files_rdd.flatMap(lambda line: line.split()).map(lambda word: (word, 1)).reduceByKey(lambda a, b: a + b)

>>> wc = wc_unsort.sortBy(lambda a: -a[1])

>>> for tupla in wc.take(10):

>>> print(tupla)

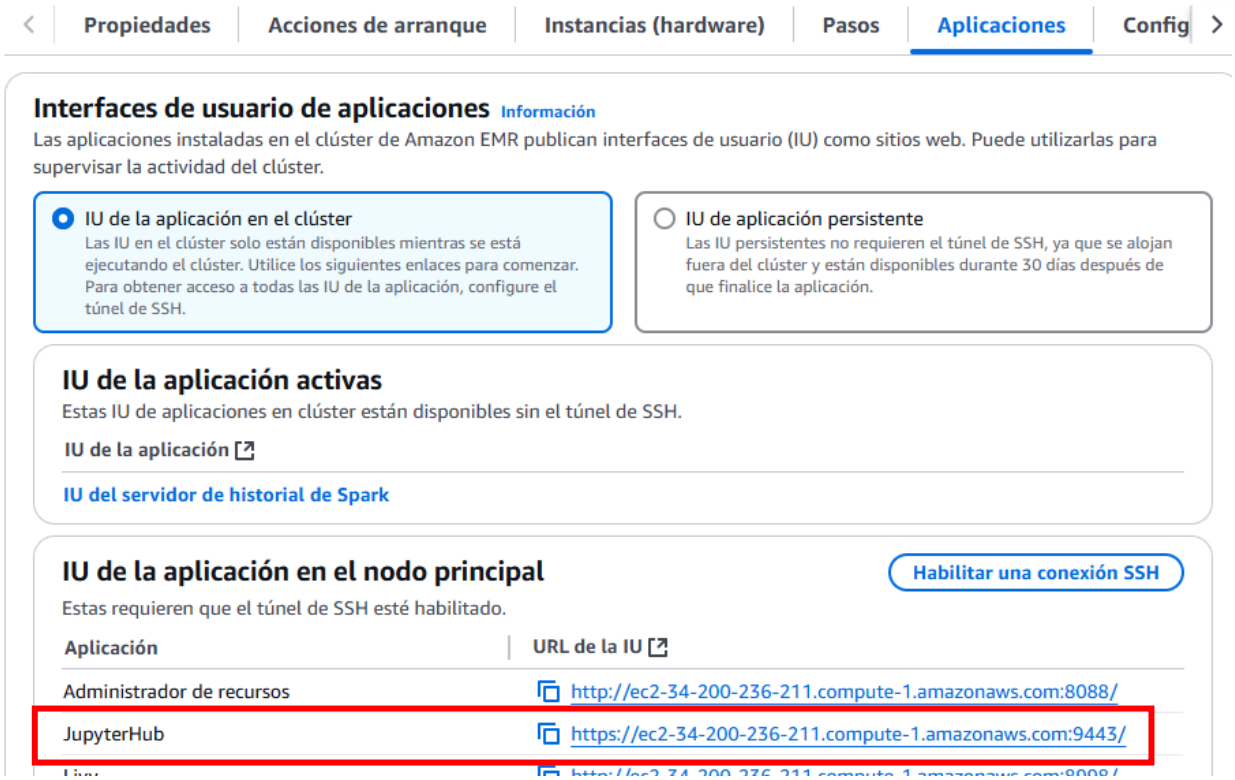
>>> wc.saveAsTextFile("hdfs:///tmp/wcout1")

$ pyspark

>>> wc.coalesce(1).saveAsTextFile("hdfs:///tmp/wcout2")

3. Procesamiento de datos usando Pyspark

Ingresamos a JupyterHub desde las aplicaciones de nuestro clúster



Cuando nos requiera credenciales entramos con las predeterminadas:

Usuario: jovyan

Contraseña: jupyter

Creamos un nuevo archivo de pyspark

Creamos un archivo data-processing con la siguiente información

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

In [3]:
```
spark=SparkSession.builder.appName('data_processing').getOrCreate()
```

In [ ]:
```
df=spark.read.csv('s3://<bucket/dir>/sample_data.csv',inferSchema=True,header=True)
```

In [ ]:
```
df.columns
```

In [ ]:
```
len(df.columns)
```

In [ ]:
```
df.count()
```

In [ ]:
```
print((df.count(),len(df.columns)))
```

In [ ]:
```
#printSchema
df.printSchema()
```

In [ ]:
```
#fisrt few rows of dataframe
df.show(5)
```

In [ ]:
```
#select only 2 columns
df.select('age','mobile').show(5)
```

In [ ]:
```
#info about dataframe
df.describe().show()
```

In [ ]:
```
from pyspark.sql.types import StringType,DoubleType,IntegerType
```

In [ ]:
```
#with column
df.withColumn("age_after_10_yrs",(df["age"]+10)).show(10,False)
```

In [ ]:
```
df.withColumn('age_double',df['age'].cast(DoubleType())).show(10,False)
```

In [ ]:
```
#with column
df.withColumn("age_after_10_yrs",(df["age"]+10)).show(10,False)
```

In [ ]:
```
#filter the records
df.filter(df['mobile']=='Vivo').show()
```

In [ ]:
```
#filter the records
df.filter(df['mobile']=='Vivo').select('age','ratings','mobile').show()
```

In [ ]:
```
#filter the multiple conditions
df.filter(df['mobile']=='Vivo').filter(df['experience'] >10).show()
```

In [ ]:
```
#filter the multiple conditions
df.filter((df['mobile']=='Vivo')&(df['experience'] >10)).show()
```

In [ ]:
```
#Distinct Values in a column
df.select('mobile').distinct().show()
```

In [ ]:
```
#distinct value count
df.select('mobile').distinct().count()
```

In [ ]:
```
df.groupBy('mobile').count().show(5,False)
```

In [ ]:
```
# Value counts
df.groupBy('mobile').count().orderBy('count',ascending=False).show(5,False)
```

In [ ]:
```
# Value counts
df.groupBy('mobile').mean().show(5,False)
```

In [ ]:
```
df.groupBy('mobile').sum().show(5,False)
```

In [ ]:
```
# Value counts
df.groupBy('mobile').max().show(5,False)
```

In [ ]:
```
# Value counts
df.groupBy('mobile').min().show(5,False)
```

In [ ]:
```
#Aggregation
df.groupBy('mobile').agg({'experience':'sum'}).show(5,False)
```

In [ ]:
```
# UDF
from pyspark.sql.functions import udf
```

In [ ]:
```python
#normal function
def price_range(brand):
    if brand in ['Samsung','Apple']:
        return 'High Price'
    elif brand =='MI':
        return 'Mid Price'
    else:
        return 'Low Price'
```

In [ ]:
```python
#create udf using python function
brand_udf=udf(price_range,StringType())
#apply udf on dataframe
df.withColumn('price_range',brand_udf(df['mobile'])).show(10,False)
```

In [ ]:
```python
#using lambda function
age_udf = udf(lambda age: "young" if age <= 30 else "senior", StringType())
#apply udf on dataframe
df.withColumn("age_group", age_udf(df.age)).show(10,False)
```

In [ ]:
```python
#pandas udf
from pyspark.sql.functions import pandas_udf, PandasUDFType
```

In [ ]:
```python
#create python function
def remaining_yrs(age):
    yrs_left=100-age

    return yrs_left
```

In [ ]:
```python
#create udf using python function
length_udf = pandas_udf(remaining_yrs, IntegerType())
#apply pandas udf on dataframe
df.withColumn("yrs_left", length_udf(df['age'])).show(10,False)
```

In [ ]:
```python
#udf using two columns
def prod(rating,exp):
    x=rating*exp
    return x
```

In [ ]:
```
#create udf using python function
prod_udf = pandas_udf(prod, DoubleType())
#apply pandas udf on multiple columns of dataframe
df.withColumn("product", prod_udf(df['ratings'],df['experience'])).show(10,False)
```

In [ ]:
```
#duplicate values
df.count()
```

In [ ]:
```
#drop duplicate values
df=df.dropDuplicates()
```

In [ ]:
```
#validate new count
df.count()
```

In [ ]:
```
#drop column of dataframe
df_new=df.drop('mobile')
```

In [ ]:
```
df_new.show(10)
```

In [ ]:
```
#current working directory
pwd
```

In [ ]:
```
#target directory
write_uri='s3://<bucket/dir>/df_csv'
```

In [ ]:
```
#save the dataframe as single csv
df.coalesce(1).write.format("csv").option("header","true").save(write_uri)
```

In [ ]:
```
#target location
parquet_uri='s3://<bucket/dir>/df_parquet'
```

In [ ]:
```
#save the data into parquet format
df.write.format('parquet').save(parquet_uri)
```