

Reto datos

Herramientas y tecnologías

Las herramientas y tecnologías que vamos a utilizar para este proyecto son las siguientes:

- Jupyter Notebook
- Visual Studio Code
- Python
- Spark con PySpark
- Pandas
- Numpy
- Una instancia de Oracle Cloud
- Ubuntu 20.04
- Windows 10/11
- Open Street Map
- Git + GitHub
- Google Colab

Usaremos estas tecnologías ya que Oracle Cloud + Jupyter Notebook nos ayudarán a tener muchos recursos disponibles y velocidad buena en procesamiento para trabajar. En caso de buscar una velocidad muy rápida usaremos Google Colab pero la RAM será más limitada que en Oracle Cloud.

Ubuntu en Oracle Cloud y en nuestras máquinas nos facilitarán el procesado de datos, en Cloud cuando se trate de algo muy grande y que tarde mucho tiempo y en local cuando se trate de algo pequeño.

Windows lo usaremos como sistema operativo principal, ya que no todos tenemos Ubuntu de forma nativa, la ventaja es que podemos usar los recursos de la nube desde ahí, así como otras aplicaciones sin mayor complicación.

Con la ayuda de Visual Studio Code nos podemos conectar a la instancia de Oracle Cloud que contiene el entorno de Jupyter Notebook.

Spark + Pandas + Numpy nos ayudarán a realizar operaciones de forma rápida a los datos con los que estemos trabajando. Spark es muy bueno en velocidad de lectura de datos una vez que está cargada a la RAM, Pandas y Numpy tomarán más el rol de realizar distintos tipos de operaciones.

Open Street Map nos dará datos de ubicaciones para extraer la dirección completa de coordenadas, así obtendremos las comunas para identificar dónde se encuentran las antenas.

Git + GitHub será nuestro sistema de control de versiones, en caso de que algo llegue a pasar, podemos acceder a lo último y más estable que hayamos hecho en esa plataforma.

La combinación de todas estas tecnologías nos hará trabajar de forma más rápida y con más seguridad de que lo que hagamos pueda correr sin problemas de que se nos acabe la memoria RAM o que nuestros procesadores sean muy lentos a la hora de realizar diversas operaciones con el dataset.

Modelo de almacenamiento de los datos

Los datos, como hemos dicho anteriormente, los datos se manejarán a través de la herramienta Spark.

Spark lo que hace es leer todo el dataset del Disco Duro o Unidad de Estado Sólido, lo carga a la memoria RAM y el acceso se vuelve más rápido.

Pyspark nos da más ventajas, entre ellas el poder usar múltiples núcleos de forma automatizada para realizar operaciones de manera más rápida.

Por lo pronto, no pensamos usar un sistema de clusters ya que Oracle no permite en su free-tier tener más de 1 máquina virtual con 24 GB de RAM.

Conectarlas de cuenta a cuenta no sería muy bueno a pesar de que seleccionemos el mismo centro de datos de Oracle, así mismo dicha plataforma my hasta veces suele limitar las transferencias de información con los planes free-tier y les da más prioridad de procesamiento a las personas que pagan por una instancia.

Extracción de datos, limpieza y carga a Oracle Cloud

Para la extracción de datos, sólo tuvimos que descargar el dataset, los datos ya venían previamente limpios, listos para comenzar a trabajar con ellos. Sin señal de datos varios en una sola columna o bien, datos nulos o faltantes.

El único problema fue que desde la terminal de Ubuntu en la nube de Oracle, el programa de wget en Google Drive no funcionaba, pero se solucionó una vez montado todo el entorno de Jupyter, ya que te permite subir, descargar, modificar y crear archivos desde ahí mismo. Además, ofrece acceso directo a la terminal si es que necesitamos hacer algo dentro de esa instancia en la nube. Todo está bajo una contraseña para obtener el acceso.

Scripts de configuración utilizados para la instancia en la nube

Los scripts / comandos que usamos para montar nuestro entorno de trabajo fueron los siguientes una vez creada la instancia de Oracle Cloud:

```
# Verificar memoria RAM
free -m

# Verificar espacio de Almacenamiento
df -h

# Actualizar los repositorios de Ubuntu
sudo apt-get update
sudo apt-get upgrade

# Instalar Java 8 (intentamos con el 17 y no funcionó Spark)
sudo apt-get install openjdk-8-jdk-headless -y
```

```

# Crear nueva carpeta, descargar Spark y descomprimirlo
mkdir
wget https://downloads.apache.org/spark/spark-3.2.1//spark-3.2.1-bin-hadoop3.2.tgz
tar xf spark-3.2.1-bin-hadoop3.2.tgz

# Instalar Python3, pip y virtualenv
sudo apt-get install python3-pip python3-dev
sudo -H pip3 install --upgrade pip
sudo -H pip3 install virtualenv

# Salirse de la carpeta de Spark y crear un entorno virtual de Python
# Los entornos virtuales permiten correr varias aplicaciones de Python a la vez
cd /
cd /home/
mkdir mc
cd mc
virtualenv mc

# Instalar screen para que el entorno y lo demás no muera al cerrar Putty
sudo apt-get install screen

# Crear una nueva screen
screen

# Correr el entorno virtual
cd /home/
source mc/bin/activate

# Cuando se inicie el entorno aparecerá el nombre del entorno entre paréntesis.
# (mc) ubuntu@fox-arm:/home/mc/DataSet$

# Ahora instalamos Jupyter en ese environment
pip install jupyter

# Duplicamos un archivo de configuración de Spark y lo renombramos
cd /home/mc/spark/spark-3.2.2-bin-hadoop.3.2/conf/
sudo cp spark-env.sh.template spark-env.sh

# Modificamos el archivo en la parte de hasta abajo y agregamos
# Permite que Spark pueda ser llamado sin problema desde el entorno de Jupyter de
forma remota, sin exponer a Spark directamente al internet
sudo nano spark-env.sh
SPARK_MASTER_IP=0.0.0.0
SPARK_LOCAL_IP=0.0.0.0

# Presionamos CTRL + X, Y, Enter

# Regresamos a nuestra carpeta de entorno donde queremos iniciar a Jupyter
# Recomendamos crear una nueva carpeta para no tocar por accidente los archivos del
environment de Python y de Spark
mkdir /home/mc/root
cd /home/mc/root

# Oracle Cloud usa IPTables como firewall en sus instancias
# Después de haber desbloqueado el puerto de nuestra preferencia el firewall desde
la página de Oracle Cloud, ahora desbloquearemos ese mismo puerto en la instancia.
# xxxx es el puerto, en este caso usaremos 25565

```

```
sudo iptables -A INPUT -p tcp --dport xxxx -j ACCEPT
sudo netfilter-persistent save
sudo systemctl restart iptables
```

Ahora configuramos el entorno de Jupyter con alguna contraseña, se pedirán los datos de contraseña dos veces y dará una salida indicando que se cambió o si ocurrió algún problema

```
jupyter notebook password
```

Iniciamos el servidor / entorno de Jupyter con

La IP en 0.0.0.0 es para aceptar cualquier conexión de cualquier puerto

```
jupyter notebook --no-browser --ip=0.0.0.0 --port=25565
```

Saldrá algo como esto, significa que ya tienes a Jupyter y a Spark instalados!

Si deseas probar, usa el siguiente código en un nuevo notebook

Spark_test.ipynb

```
-----
#Configuración de Spark con Python
```

```
!pip install -q findspark
```

```
!pip install pyspark==2.3.0
```

```
import os
```

```
import findspark
```

```
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-arm64"
```

```
os.environ["SPARK_HOME"] = "/home/mc/spark/spark-3.2.2-bin-hadoop3.2"
```

```
findspark.init()
```

```
findspark.find()
```

```
from pyspark.sql import SparkSession
```

```
spark_session = SparkSession.builder.appName('PySpark_session').getOrCreate()
```

```
spark_session
-----
```

La salida debería ser la siguiente:

SparkSession - in-memory

SparkContext

[Spark UI](#)

Version

v3.2.2

Master

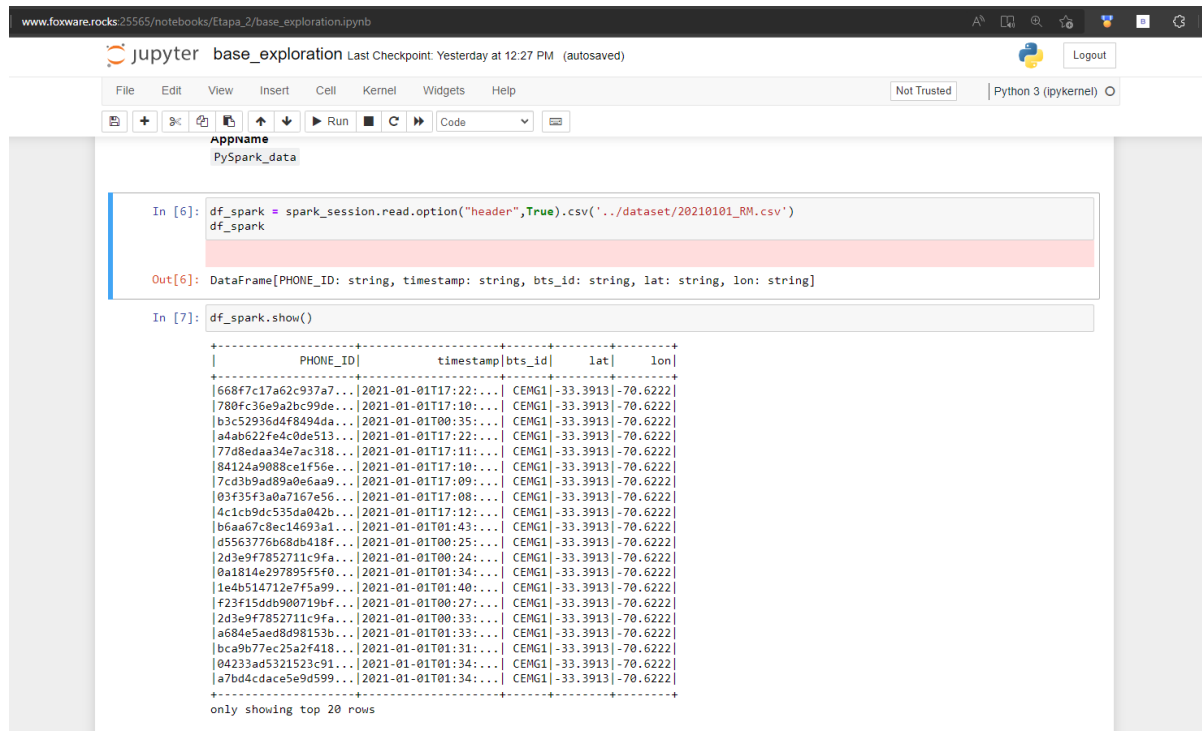
local[*]

AppName

PySpark_session

Carga de datos en Spark a través de PySpark

La carga de datos se puede ver aquí, la pequeña carga de datos estará en la misma carpeta del entregable 2 en GitHub, puede observarse que aún no contamos con certificado SSL pero ya tenemos un dominio fácil de recordar.



The screenshot shows a Jupyter Notebook titled 'base_exploration' with a toolbar at the top. The notebook content includes the following code and output:

```
In [6]: df_spark = spark_session.read.option("header", True).csv('../dataset/20210101_RM.csv')
df_spark

Out[6]: DataFrame[PHONE_ID: string, timestamp: string, bts_id: string, lat: string, lon: string]
```

```
In [7]: df_spark.show()
```

PHONE_ID	timestamp	bts_id	lat	lon
668f7c17a62c937a7...	2021-01-01T17:22:...	CEMG1-33.3913 -70.6222		
780fc36e9a2bc99de...	2021-01-01T17:10:...	CEMG1-33.3913 -70.6222		
b3c52936d4f8494da...	2021-01-01T00:35:...	CEMG1-33.3913 -70.6222		
a4ab622fe4c0de513...	2021-01-01T17:22:...	CEMG1-33.3913 -70.6222		
77d8edaa34e7ac318...	2021-01-01T17:11:...	CEMG1-33.3913 -70.6222		
84124a9088ce1f56e...	2021-01-01T17:10:...	CEMG1-33.3913 -70.6222		
7cd3b9ad89a0e6aa9...	2021-01-01T17:09:...	CEMG1-33.3913 -70.6222		
03f35f3a0a7167e56...	2021-01-01T17:08:...	CEMG1-33.3913 -70.6222		
4c1cb9dc535da042b...	2021-01-01T17:12:...	CEMG1-33.3913 -70.6222		
b6aa67c8ec14693a1...	2021-01-01T01:43:...	CEMG1-33.3913 -70.6222		
d5563776b68db418f...	2021-01-01T00:25:...	CEMG1-33.3913 -70.6222		
2d3e9f7852711c9fa...	2021-01-01T00:24:...	CEMG1-33.3913 -70.6222		
0a1814e297895f5f0...	2021-01-01T01:34:...	CEMG1-33.3913 -70.6222		
1e4b514712e7f5a99...	2021-01-01T01:40:...	CEMG1-33.3913 -70.6222		
f23f15ddb900719bf...	2021-01-01T00:27:...	CEMG1-33.3913 -70.6222		
2d3e9f7852711c9fa...	2021-01-01T00:33:...	CEMG1-33.3913 -70.6222		
a684e5aed898153b...	2021-01-01T01:33:...	CEMG1-33.3913 -70.6222		
bca9b77ec25a2f418...	2021-01-01T01:31:...	CEMG1-33.3913 -70.6222		
04233ad5321523c91...	2021-01-01T01:34:...	CEMG1-33.3913 -70.6222		
a7bd4cdace5e9d599...	2021-01-01T01:34:...	CEMG1-33.3913 -70.6222		

only showing top 20 rows

El dataset se recortó y guardó con Pyspark, son 5000 datos.

```
In [82]: df_spark = spark_session.read.option("header", True).csv('../dataset/20210101_RM.csv')
df_spark

Out[82]: DataFrame[PHONE_ID: string, timestamp: string, bts_id: string, lat: string, lon: string]
```

```
In [86]: df_spark = df_spark.limit(5000)
```

```
In [87]: df_spark

Out[87]: DataFrame[PHONE_ID: string, timestamp: string, bts_id: string, lat: string, lon: string]
```

```
In [89]: df_spark.write.csv("./recortado.csv")
```

Datos de prueba y entrenamiento

Por otro lado se realizó una separación de datos para prueba y entrenamiento con ayuda de Pandas y Sklearn, el script para dicha separación se encuentra en el GitHub del mismo entregable, dicho script es llamado "Data_divide".

```
for train_index, test_index in kf.split(data):
    print("TRAIN:", train_index, "TEST:", test_index)
    data_train, data_test = data.iloc[train_index], data.iloc[test_index]
```

TRAIN: [9923627 9923628 9923629 ... 49618129 49618130 49618131 ... 39694503 39694504 39694505]

TRAIN: [0 1 2 ... 49618129 49618130 49618131 ... 39694503 39694504 39694505]

TRAIN: [0 1 2 ... 49618129 49618130 49618131 ... 39694503 39694504 39694505]

TRAIN: [0 1 2 ... 49618129 49618130 49618131 ... 39694503 39694504 39694505]

TRAIN: [0 1 2 ... 39694503 39694504 39694505]

```
[ ] data_train.shape
```

(39694506, 5)

```
[ ] data_test.shape
```

(9923626, 5)

Hecho esto nos pudimos dar cuenta, y como se mencionó previamente, el procesamiento de los datos se realizará con ayuda de Spark para facilitar y hacer más rápido el manejo y uso de los datos.

¿Es necesario usar un enfoque orientado a Big Data?

Dado lo mencionado anteriormente es posible decir que si es necesario usar un enfoque orientado a Big Data, principalmente por la cantidad de datos. Por otro lado cabe mencionar que, si bien es posible cargarlos de manera correcta el uso de operaciones y funciones alenta el procesamiento de estos, por lo que se confirma el uso del enfoque hacia Big Data.