

Proyecto de Big Data Processing

La idea principal de proyecto es el manejo y procesamiento de datos, para ello se usarán varia tecnología que a continuación describiremos:

Kafka y Docker

Como parte inicial se generó una máquina virtual en Google Cloud Computer (GCC), tal como se muestra a continuación

Las instancias de VM son máquinas virtuales altamente configurables para ejecutar cargas de trabajo en la infraestructura de Google. [Más información](#)

Filtro

Ingresar el nombre o el valor de la propiedad

?

☰

<input type="checkbox"/>	Estado	Nombre ↑	Zona	Recomendaciones	En uso por	IP interna	Conectar
<input type="checkbox"/>	✔	keepcoding-kafka	us-central1-c			10.128.0.2 (nic0)	SSH ▾ ⋮

Posteriormente de verifico que tuviera acceso libre mediante una regla de firewall, para permitir las conexiones entrantes, debido a que nos conectaremos a ella.

Se instalo Docker y Kafka, para la generación de mensajes en tópicos, para poder consumirlos de forma local, a través de la IP pública. se generó una prueba de donde la pantalla de la izquierda es una consola de la instancia ejecutando un “producer” mediante un Docker, y de la segunda pantalla es una consola la misma instancia (VM), pero ejecutando un consumidor de mensajes “consumer”.

https://ssh.cloud.google.com/v2/ssh/projects/keepcodingdbm10/zones/us-central1-c/instances/keepcoding-kaf...

ssh.cloud.google.com/v2/ssh/projects/keepcodingdbm10/zones/us-central1-c/instances/keepcoding-kafka?auth...

SSH en el navegador

SUBIR ARCHIVO

DESCARGAR ARCHIVO

🗨

📄

⚙

```
(devices, Sending: {"bytes":2906,"timestamp":1667370676,"app":"TELEGRAM","id":"00000000-0000-00-0000-000000000006","antenna_id":"00000000-0000-0000-0000-000000000000"})(antenna_telemetry, Sending: {"timestamp":1667370676,"id":"00000000-0000-0000-0000-000000000000","metric":"battery","value":80})(antenna_telemetry, Sending: {"timestamp":1667370676,"id":"11111111-1111-1111-1111-111111111111","metric":"battery","value":60})(devices, Sending: {"bytes":158,"timestamp":1667370676,"app":"FACEBOOK","id":"00000000-0000-00-0000-000000000007","antenna_id":"00000000-0000-0000-0000-000000000000"})(devices, Sending: {"bytes":4314,"timestamp":1667370676,"app":"SKYPE","id":"00000000-0000-0000-0000-000000000008","antenna_id":"00000000-0000-0000-0000-000000000000"})(antenna_telemetry, Sending: {"timestamp":1667370676,"id":"22222222-2222-2222-2222-222222222222","metric":"battery","value":111})(antenna_telemetry, Sending: {"timestamp":1667370676,"id":"33333333-3333-3333-3333-333333333333","metric":"battery","value":100})(antenna_telemetry, Sending: {"timestamp":1667370676,"id":"44444444-4444-4444-4444-444444444444","metric":"battery","value":90})(antenna_telemetry, Time to sleep!)(devices, Sending: {"bytes":9575,"timestamp":1667370676,"app":"TELEGRAM","id":"00000000-0000-00-0000-000000000009","antenna_id":"00000000-0000-0000-0000-000000000000"})(devices, Sending: {"bytes":2344,"timestamp":1667370676,"app":"FACEBOOK","id":"00000000-0000-00-0000-000000000010","antenna_id":"00000000-0000-0000-0000-000000000000"})(devices, Sending: {"bytes":109,"timestamp":1667370676,"app":"SKYPE","id":"00000000-0000-0000-0000-000000000011","antenna_id":"11111111-1111-1111-1111-111111111111"})(devices, Sending: {"bytes":2607,"timestamp":1667370676,"app":"FACEBOOK","id":"00000000-0000-00-0000-000000000012","antenna_id":"11111111-1111-1111-1111-111111111111"})(devices, Sending: {"bytes":2597,"timestamp":1667370676,"app":"SKYPE","id":"00000000-0000-0000-0000-000000000013","antenna_id":"11111111-1111-1111-1111-111111111111"})(devices, Sending: {"bytes":9995,"timestamp":1667370676,"app":"TELEGRAM","id":"00000000-0000-00-0000-000000000014","antenna_id":"11111111-1111-1111-1111-111111111111"})(devices, Sending: {"bytes":7048,"timestamp":1667370676,"app":"FACEBOOK","id":"00000000-0000-00-0000-000000000015","antenna_id":"11111111-1111-1111-1111-111111111111"})(devices, Sending: {"bytes":7005,"timestamp":1667370676,"app":"SKYPE","id":"00000000-0000-0000-0000-000000000016","antenna_id":"22222222-2222-2222-2222-222222222222"})(devices, Sending: {"bytes":1153,"timestamp":1667370676,"app":"FACEBOOK","id":"00000000-0000-00-0000-000000000017","antenna_id":"22222222-2222-2222-2222-222222222222"})(devices, Sending: {"bytes":8027,"timestamp":1667370676,"app":"FACEBOOK","id":"00000000-0000-00-0000-000000000018","antenna_id":"22222222-2222-2222-2222-222222222222"})(devices, Sending: {"bytes":8821,"timestamp":1667370676,"app":"SKYPE","id":"00000000-0000-0000-0000-000000000019","antenna_id":"22222222-2222-2222-2222-222222222222"})(devices, Sending: {"bytes":2423,"timestamp":1667370676,"app":"TELEGRAM","id":"00000000-0000-00-0000-000000000020","antenna_id":"22222222-2222-2222-2222-222222222222"})(devices, Time to sleep!)
```

https://ssh.cloud.google.com/v2/ssh/projects/keepcodingdbm10/zones/us-central1-c/instances/keepcoding-kaf...

ssh.cloud.google.com/v2/ssh/projects/keepcodingdbm10/zones/us-central1-c/instances/keepcoding-kafka?auth...

SSH en el navegador

SUBIR ARCHIVO

DESCARGAR ARCHIVO

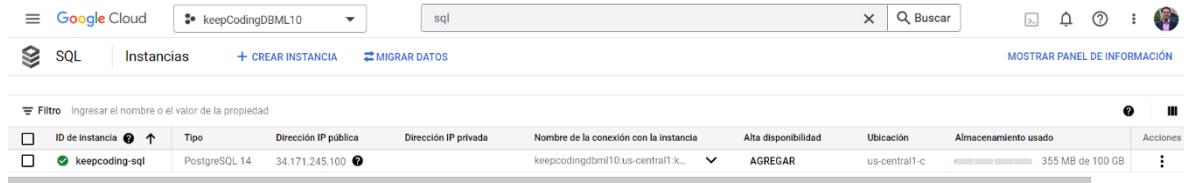
🗨

📄

⚙

SQL

En la misma plataforma de Google, se usará el servicio de servidor SQL, es decir Google Cloud SQL, y en este servicio se seleccionará una base de datos de PostgreSQL, en ella almacenaremos los datos que enriquecerán, a los producidos por Kafka.

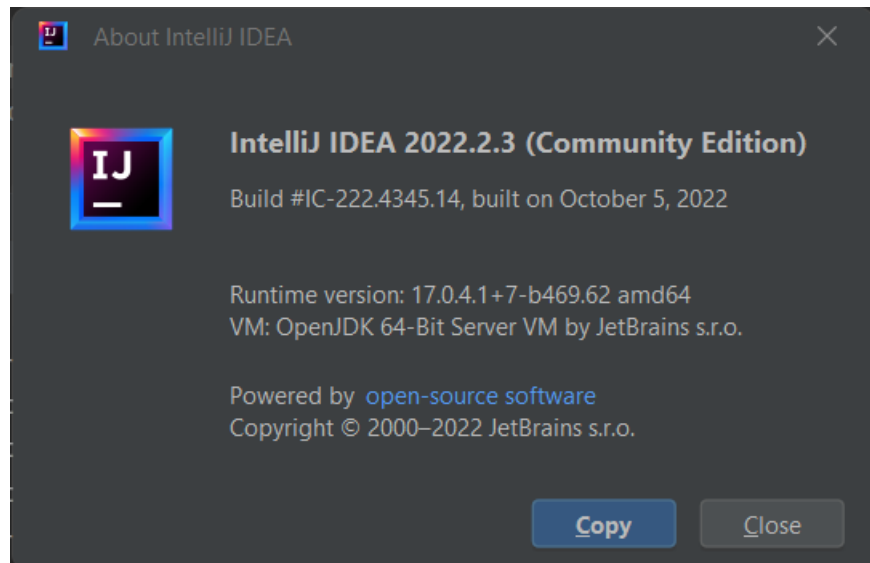


The screenshot shows the Google Cloud SQL console. At the top, there's a search bar with 'sql' and a 'Buscar' button. Below the navigation bar, there's a table listing SQL instances. The table has columns for 'ID de instancia', 'Tipo', 'Dirección IP pública', 'Dirección IP privada', 'Nombre de la conexión con la instancia', 'Alta disponibilidad', 'Ubicación', 'Almacenamiento usado', and 'Acciones'. One instance is listed: 'keepcoding-sql' of type 'PostgreSQL 14', with a public IP of '34.171.245.100'. The storage used is '355 MB de 100 GB'.

ID de instancia	Tipo	Dirección IP pública	Dirección IP privada	Nombre de la conexión con la instancia	Alta disponibilidad	Ubicación	Almacenamiento usado	Acciones
keepcoding-sql	PostgreSQL 14	34.171.245.100		keepcodingdbml10-us-central1-...	AGREGAR	us-central1-c	355 MB de 100 GB	

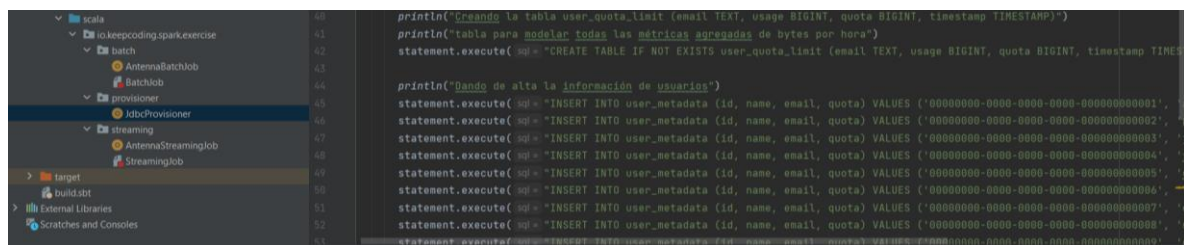
Programación

Como IDE de programación se usó IntelliJ IDEA en la versión 2022.2.3, de edición "Community Edition"



Provisioner

En este código es para la generación de tablas y llenado de algunas de ella, en la nube, decir en el Storage Seleccionado, en nuestro caso el servicio proporcionado por Google Cloud SQL.



The screenshot shows a code editor with a file explorer on the left. The file explorer shows a project structure with folders like 'scala', 'io.keepcoding.spark.exercise', 'batch', 'AntennaBatchJob', 'BatchJob', 'provisioner', 'jdbcProvisioner', 'streaming', 'AntennaStreamingJob', and 'StreamingJob'. The code editor shows the following SQL code:

```
println("Creando la tabla user_quota_limit (email TEXT, usage BIGINT, quota BIGINT, timestamp TIMESTAMP)")
println("tabla para modelar todas las métricas agregadas de bytes por hora")
statement.execute("CREATE TABLE IF NOT EXISTS user_quota_limit (email TEXT, usage BIGINT, quota BIGINT, timestamp TIME")

println("Dando de alta la información de usuarios")
statement.execute("INSERT INTO user_metadata (id, name, email, quota) VALUES ('00000000-0000-0000-0000-000000000001',")
statement.execute("INSERT INTO user_metadata (id, name, email, quota) VALUES ('00000000-0000-0000-0000-000000000002',")
statement.execute("INSERT INTO user_metadata (id, name, email, quota) VALUES ('00000000-0000-0000-0000-000000000003',")
statement.execute("INSERT INTO user_metadata (id, name, email, quota) VALUES ('00000000-0000-0000-0000-000000000004',")
statement.execute("INSERT INTO user_metadata (id, name, email, quota) VALUES ('00000000-0000-0000-0000-000000000005',")
statement.execute("INSERT INTO user_metadata (id, name, email, quota) VALUES ('00000000-0000-0000-0000-000000000006',")
statement.execute("INSERT INTO user_metadata (id, name, email, quota) VALUES ('00000000-0000-0000-0000-000000000007',")
statement.execute("INSERT INTO user_metadata (id, name, email, quota) VALUES ('00000000-0000-0000-0000-000000000008',")
```

La ejecución de este código nos genera nuestras tablas:

- user_metadata: que contendrá la información del “user”, con información de: id, name, email y quota.
- Bytes: Contendrá las métricas de las antenas, que guardará los totales de bytes recibidos por antena, por usuario y aplicación.
- bytes_hourly: métricas de recibidos y transmitidos, por “antenna”, “user” y “App”
- user_quota_limit: contendrá el total de usuario que sobrepasaron su límite “quota”

El resultado se muestra a continuación:

```

CLOUD SHELL
Terminal (keepcodingdbml10) X + v

List of relations
Schema | Name | Type | Owner
-----+-----+-----+-----
public | antenna_1h_agg | table | postgres
public | antenna_agg | table | postgres
public | antenna_errors_agg | table | postgres
public | antenna_percent_agg | table | postgres
public | bytes | table | postgres
public | bytes_hourly | table | postgres
public | instituto | table | postgres
public | metadata | table | postgres
public | user_metadata | table | postgres
public | user_quota_limit | table | postgres
(10 rows)

postgres=>

```

Así mismo este código llena la tabla de “user_metadata”:

```

CLOUD SHELL
Terminal (keepcodingdbml10) X + v

id | name | email | quota
---+---+---+---
00000000-0000-0000-0000-000000000001 | andres | andres@gmail.com | 200000
00000000-0000-0000-0000-000000000002 | paco | paco@gmail.com | 300000
00000000-0000-0000-0000-000000000003 | juan | juan@gmail.com | 100000
00000000-0000-0000-0000-000000000004 | fede | fede@gmail.com | 5000
00000000-0000-0000-0000-000000000005 | gorka | gorka@gmail.com | 200000
00000000-0000-0000-0000-000000000006 | luis | luis@gmail.com | 200000
00000000-0000-0000-0000-000000000007 | eric | eric@gmail.com | 300000
00000000-0000-0000-0000-000000000008 | carlos | carlos@gmail.com | 100000
00000000-0000-0000-0000-000000000009 | david | david@gmail.com | 300000
00000000-0000-0000-0000-000000000010 | juanchu | juanchu@gmail.com | 300000
00000000-0000-0000-0000-000000000011 | charo | charo@gmail.com | 300000
00000000-0000-0000-0000-000000000012 | delicidas | delicidas@gmail.com | 1000000
00000000-0000-0000-0000-000000000013 | milagros | milagros@gmail.com | 200000
:

```

Sus argumentos de entrada son:

- Conexión a Google Cloud SQL:
 - Driver: "org.postgresql.Driver"
 - url: "jdbc:postgresql://34.171.245.100:5432/postgres"
 - username: "postgres"
 - password: "keepcoding"

Streaming (Speed Layer)

Los códigos de esta capa, se conectan a kafka y postgres para obtener información

Las funciones definidas y su orden ejecución van de la siguiente forma:

1. **readFromKafka**: lee los mensajes desde Kafka, del tópico “devices”, la conexión se realiza mediante la conexión a la Máquina Virtual mediante `readFromKafka("146.148.62.187:9092", "devices")`, donde ponemos la IP del servidor de kafka y el tópico del que consumirá los mensajes. En un principio el Batch no ha llenado, por ello nos muestra solo la estructura:

```
Batch: 0
-----
+---+-----+-----+-----+-----+-----+-----+-----+
|key|value|topic|partition|offset|timestamp|timestampType|
+---+-----+-----+-----+-----+-----+-----+-----+
+---+-----+-----+-----+-----+-----+-----+-----+
```

Finalmente se empieza a llenar la estructura con los mensajes de Kafka:

```
Batch: 1
-----
+---+-----+-----+-----+-----+-----+-----+-----+
| key|          value|  topic|partition|offset|          timestamp|timestampType|
+---+-----+-----+-----+-----+-----+-----+-----+
|null|[7B 22 62 79 74 6...|devices|      0| 4216|2022-11-02 00:43:...|      0|
|null|[7B 22 62 79 74 6...|devices|      0| 4217|2022-11-02 00:43:...|      0|
|null|[7B 22 62 79 74 6...|devices|      0| 4218|2022-11-02 00:43:...|      0|
|null|[7B 22 62 79 74 6...|devices|      0| 4219|2022-11-02 00:43:...|      0|
|null|[7B 22 62 79 74 6...|devices|      0| 4220|2022-11-02 00:43:...|      0|
|null|[7B 22 62 79 74 6...|devices|      0| 4221|2022-11-02 00:43:...|      0|
|null|[7B 22 62 79 74 6...|devices|      0| 4222|2022-11-02 00:43:...|      0|
|null|[7B 22 62 79 74 6...|devices|      0| 4223|2022-11-02 00:43:...|      0|
|null|[7B 22 62 79 74 6...|devices|      0| 4224|2022-11-02 00:43:...|      0|
|null|[7B 22 62 79 74 6...|devices|      0| 4225|2022-11-02 00:43:...|      0|
|null|[7B 22 62 79 74 6...|devices|      0| 4226|2022-11-02 00:43:...|      0|
|null|[7B 22 62 79 74 6...|devices|      0| 4227|2022-11-02 00:43:...|      0|
```

Estos datos son almacenados en la variable “kafkaDF”, que vienen con un formato de JSON, por lo que lo deberemos procesar.

2. **parserJsonData**: parsea la variable “kafkaDF”, es decir pasa de un formato JSON a un DataFrame, el resultado lo almacenamos en la variable “parsedDF”.

```

-----
Batch: 1
-----
+-----+-----+-----+-----+-----+
| timestamp| id| antenna_id| app|bytes|
+-----+-----+-----+-----+-----+
|2022-11-02 00:45:38|00000000-0000-000...|00000000-0000-000...| SKYPE| 3245|
|2022-11-02 00:45:38|00000000-0000-000...|00000000-0000-000...|FACEBOOK| 4974|
|2022-11-02 00:45:38|00000000-0000-000...|00000000-0000-000...| SKYPE| 8738|
|2022-11-02 00:45:38|00000000-0000-000...|00000000-0000-000...|FACEBOOK| 1080|
|2022-11-02 00:45:38|00000000-0000-000...|00000000-0000-000...|FACEBOOK| 9557|
|2022-11-02 00:45:38|00000000-0000-000...|00000000-0000-000...| SKYPE| 230|
|2022-11-02 00:45:38|00000000-0000-000...|00000000-0000-000...|FACEBOOK| 1589|
|2022-11-02 00:45:38|00000000-0000-000...|00000000-0000-000...| SKYPE| 7789|
|2022-11-02 00:45:38|00000000-0000-000...|00000000-0000-000...|TELEGRAM| 4303|
|2022-11-02 00:45:38|00000000-0000-000...|00000000-0000-000...|TELEGRAM| 4562|
|2022-11-02 00:45:38|00000000-0000-000...|11111111-1111-111...|TELEGRAM| 2673|
|2022-11-02 00:45:38|00000000-0000-000...|11111111-1111-111...| SKYPE| 2584|

```

3. **writeToStorage:** este procedimiento realiza el guardado de batch obtenido de Kafka, el storage que se usa es de tipo local, este de realiza de la siguiente forma:
writeToStorage(parsedDF, "/tmp/antenna_parquet/"), en donde como parámetros de entrada de la función “writeToStorage”, son el dataFrame parseado de Kafka, y la tuta donde se realizara el guardado. El guardado será particionado, por año, mes, día y hora, apartir de fecha que dispone el campo de “timestamp”. El resultado es el siguiente:

Este equipo > OS (C:) > tmp > antenna_parquet > data > year=2022 > month=10 > day=31 > hour=2

Nombre	Fecha de modificación	Tipo	Tamaño
.part-00000-438cb5f6-d97a-436e-86cc-a...	31/10/2022 02:55 a. m.	Archivo CRC	1 KB
.part-00000-612e742e-510c-4058-9024-6...	31/10/2022 02:58 a. m.	Archivo CRC	1 KB
.part-00000-8934d9e6-b47b-4ee8-8f4c-8...	31/10/2022 02:54 a. m.	Archivo CRC	1 KB
.part-00000-acef1c99-d5a6-4179-ac84-e...	31/10/2022 02:59 a. m.	Archivo CRC	1 KB
.part-00000-c09d16d1-6ecb-461e-9c60-...	31/10/2022 02:56 a. m.	Archivo CRC	1 KB
.part-00000-ece13a5e-4376-43db-b14f-8...	31/10/2022 02:57 a. m.	Archivo CRC	1 KB
part-00000-438cb5f6-d97a-436e-86cc-af...	31/10/2022 02:55 a. m.	Archivo PARQUET	2 KB
part-00000-612e742e-510c-4058-9024-6...	31/10/2022 02:58 a. m.	Archivo PARQUET	2 KB
part-00000-8934d9e6-b47b-4ee8-8f4c-8...	31/10/2022 02:54 a. m.	Archivo PARQUET	3 KB
part-00000-acef1c99-d5a6-4179-ac84-e...	31/10/2022 02:59 a. m.	Archivo PARQUET	2 KB
part-00000-c09d16d1-6ecb-461e-9c60-3...	31/10/2022 02:56 a. m.	Archivo PARQUET	2 KB
part-00000-ece13a5e-4376-43db-b14f-8...	31/10/2022 02:57 a. m.	Archivo PARQUET	2 KB

Estos archivos en formato Parquet, se usarán para la etapa de Batch, y dejar de usar el Kafka.

4. **readAntennaMetadata:** realiza la obtención de los datos contenidos en la tabla “user_metadata” que se encuentra en Google Cloud SQL, realiza la conexión mediante los datos de :

- o Driver: "org.postgresql.Driver"
- o url: "jdbc:postgresql://34.171.245.100:5432/postgres"
- o username: "postgres"
- o password: "keepcoding"

Una vez obtenidos los datos, estos son almacenados en la variable "metadaDF".

5. **enrichAntennaWithMetadata:** en este procedimiento unimos los dataframes que obtuvimos de Kafka (parseDF) y el que obtuvimos de postgres (metadaDF), esto se realiza mediante join con el identificador en común para ambos dataframe, en este caso el "id", Finalmente el resultado lo almacenamos en la variable "enrichDF", que contiene la información como se muestra en la siguiente imagen:

```
Batch: 1
```

timestamp	id	antenna_id	app bytes	name	email	quota
2022-11-02 00:53:38 00000000-0000-000...	11111111-1111-111...	FACEBOOK	3866	delicidas	delicidas@gmail.com	1000000
2022-11-02 00:53:38 00000000-0000-000...	11111111-1111-111...	FACEBOOK	3866	delicidas	delicidas@gmail.com	1000000
2022-11-02 00:53:38 00000000-0000-000...	44444444-4444-444...	FACEBOOK	3128	luis	luis@gmail.com	200000
2022-11-02 00:53:38 00000000-0000-000...	44444444-4444-444...	FACEBOOK	3128	luis	luis@gmail.com	200000
2022-11-02 00:53:38 00000000-0000-000...	33333333-3333-333...	TELEGRAM	6021	eric	eric@gmail.com	300000
2022-11-02 00:53:38 00000000-0000-000...	33333333-3333-333...	TELEGRAM	6021	eric	eric@gmail.com	300000
2022-11-02 00:53:38 00000000-0000-000...	11111111-1111-111...	SKYPE	4319	andres	andres@gmail.com	200000
2022-11-02 00:53:38 00000000-0000-000...	11111111-1111-111...	SKYPE	4319	andres	andres@gmail.com	200000
2022-11-02 00:53:38 00000000-0000-000...	33333333-3333-333...	FACEBOOK	2912	juanchu	juanchu@gmail.com	300000
2022-11-02 00:53:38 00000000-0000-000...	33333333-3333-333...	FACEBOOK	2912	juanchu	juanchu@gmail.com	300000
2022-11-02 00:53:38 00000000-0000-000...	33333333-3333-333...	TELEGRAM	949	carlota	carlota@gmail.com	200000
2022-11-02 00:53:38 00000000-0000-000...	33333333-3333-333...	TELEGRAM	949	carlota	carlota@gmail.com	200000

6. **computeBytes:** Este procedimiento realiza el cálculo de las métricas de consumo de bytes por:

- o Antenna: total de bytes por antenna, estos datos se guardan en dataframe "aggBytesAntenna"

```
Batch: 4
```

timestamp	id	value	type
2022-11-02 00:57:30 44444444-4444-444...	132196	antenna_total_bytes	
2022-11-02 00:57:30 22222222-2222-222...	53216	antenna_total_bytes	

- o User: total de bytes por usuario, estos datos se guardan en dataframe "aggBytesUser"

```
Batch: 3
```

timestamp	id value	type
2022-11-02 01:01:30 00000000-0000-000...	18172	user_total_bytes
2022-11-02 01:01:30 00000000-0000-000...	5662	user_total_bytes
2022-11-02 01:01:30 00000000-0000-000...	18506	user_total_bytes
2022-11-02 01:01:30 00000000-0000-000...	14114	user_total_bytes
2022-11-02 01:01:30 00000000-0000-000...	10774	user_total_bytes

- o App: total de bytes por aplicación, estos datos se guardan en dataframe "aggBytesApp"

```
Batch: 4
-----
+-----+-----+-----+-----+
|      timestamp|      id|value|      type|
+-----+-----+-----+-----+
|2022-11-02 01:05:30|    SKYPE|52504|app_total_bytes|
|2022-11-02 01:05:30|FACE TIME|37440|app_total_bytes|
|2022-11-02 01:05:30|FACEBOOK|80662|app_total_bytes|
|2022-11-02 01:05:30|TELEGRAM|73336|app_total_bytes|
```

7. **writeToJdbc**: Este procedimiento realiza el guardado de los datos de los frames generados del cálculo de bytes (aggBytesAntenna, aggBytesUser y aggBytesApp), en la tabla llamada “bytes” que se encuentra en Google Cloud SQL, conexión mediante “JDBC”. El guardado se realiza mediante seq[Futures], para escribir los 3 dataframes en paralelo en la misma tabla.

*Nota: no sé, porque no guarda la información si se esta bien en los dataframes.

Batch Layer

Como se había mencionado anteriormente, lo que se guarda en el storage local, que son archivos parquet. El procedimiento es el siguiente:

1. **readFromStorage**: Lectura de archivos parquet desde el storage local.
2. **readAntennaMetadata**: Al igual que el Speed Layer, obtenemos los metadatos del usuario desde Google Cloud SQL.
3. **enrichAntennaWithMetadata**: Unimos los dos dataframes el que obtuvimos del storage local y el de PostgreSQL. La unión se realiza mediante el join, con el campo “id”.
4. **computeBytes**: se calculan la métrica de total de bytes, como se realizó en el Speed Layer, pero con una ventana de batch de 1 hora, el dataframe generado se llama “aggByte”.

```
+-----+-----+-----+-----+
|      timestamp|      id|value|      type|
+-----+-----+-----+-----+
|2022-11-02 01:01:30|00000000-0000-000...|18172|user_total_bytes|
|2022-11-02 01:01:30|00000000-0000-000...| 5662|user_total_bytes|
|2022-11-02 01:01:30|00000000-0000-000...|18506|user_total_bytes|
|2022-11-02 01:01:30|00000000-0000-000...|14114|user_total_bytes|
|2022-11-02 01:01:30|00000000-0000-000...|10774|user_total_bytes|
```

5. **computeBytesUserQuotaLimit**: se calcula la métrica si el usuario sobrepasa su Quota de bytes.


```
+-----+-----+-----+
|usage|  quota|          timestamp|
+-----+-----+-----+
|17838|2000000|2022-11-02 13:01:00|
| 2188| 400000|2022-11-02 13:01:00|
|19556| 400000|2022-11-02 13:01:00|
|15608| 400000|2022-11-02 13:01:00|
|10732| 400000|2022-11-02 13:01:00|
|14248|2000000|2022-11-02 13:01:00|
```

6. **writeToJdbc:** En este procedimiento se realiza el guardado de los data frames en sus respectivas tablas que están contenidas en Google Cloud SQL.