Práctica del módulo Big Data Processing

En este proyecto vamos a realizar el procesamiento de datos recolectados desde antenas de telefonía móvil. Para ellos vamos a generar tres capas de procesamiento y un Script para la creación de las tablas SQL:

Script de Provisionamiento JDBC

(BigDataProcessing\practica\src\main\scala\io\keepcoding\spark\exercice\provisioner\JdbcProvisioner.scala)

En esta capa creamos un Script en el que realizaremos dos acciones, la primera, la creación de 4 tablas en una base de datos de postgres en GCP. Y la segunda, insertar información de usuarios en una de las tablas (user_metadata) para tener dicha información de forma estática. Tras la ejecución debemos obtener el siguiente mensaje:

```
"C:\Program Files\Java\jdk1.8.0_202\bin\java.exe" ...
Conexión establecida correctamente!
Creando la tabla bytes (timestamp TIMESTAMP, id_movil TEXT,value BIGINT)
Creando la tabla bytes_hourly (timestamp TIMESTAMP, id TEXT, value BIGINT, type TEXT)
Creando la tabla user_quota_limit (email TEXT, usage BIGINT, quota BIGINT, timestamp TIMESTAMP)
Creando la tabla user_metadata (id TEXT, name TEXT, email TEXT, quota BIGINT)
Dando de alta la información de usuarios

Process finished with exit code 0
```

Comprobamos en la consola de la instancia de SQL dentro de GCP si se han creado las tablas correctamente:

```
postgres-> \dt
              List of relations
 Schema |       Name       | Type  |  Owner
--------+------------------+-------+----------
 public | bytes            | table | postgres
 public | bytes_hourly     | table | postgres
 public | user_metadata    | table | postgres
 public | user_quota_limit | table | postgres
(4 rows)
```

Y que los datos se hayan insertado correctamente en la tabla user_metadata:

```
postgres=> select * from user_metadata;
                  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
 00000000-0000-0000-0000-000000000014 | antonio   | antonio@gmail.com   | 1000000
 00000000-0000-0000-0000-000000000015 | sergio    | sergio@gmail.com    | 1000000
 00000000-0000-0000-0000-000000000016 | maria     | maria@gmail.com     | 1000000
 00000000-0000-0000-0000-000000000017 | cristina  | cristina@gmail.com  |  300000
 00000000-0000-0000-0000-000000000018 | lucia     | lucia@gmail.com     |  300000
 00000000-0000-0000-0000-000000000019 | carlota   | carlota@gmail.com   |  200000
 00000000-0000-0000-0000-000000000020 | emilio    | emilio@gmail.com    |  200000
```

Capa Speed Layer

(BigDataProcessing\practica\src\main\scala\io\keepcoding\spark\exercice\streaming\bytesStreaming.scala)

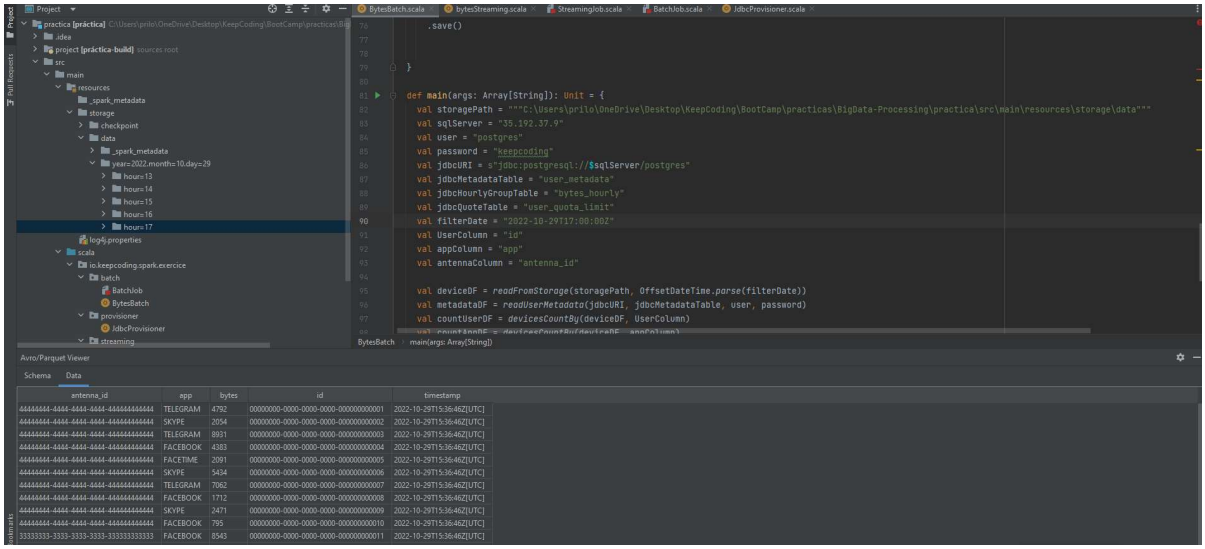
En este apartado realizaremos la descarga, trasformación y guardado de los datos obtenidos en Kafka. Para ello es necesario llevar a cabo varias operaciones:

1. En primer lugar, creamos la SparkSession

2. Almacenamos en una variable tipo DataFrame los datos que se leen en RealTime de Kafka

3. Realizamos el parseo de los datos que leemos de Kafka devolviendo un Json de formato {timestamp:timestamp, id:String, antenna_id:String, bytes:Long, app:String}

4. La data que vamos obteniendo en el punto anterior se ira guardando en un storage local, formato parquet particionado por AÑO, MES, DIA, HORA.



5. Del dataframe obtenido en el apartado 3, obtendremos 3 DF con el siguiente esquema: (timestamp TIMESTAMP, id TEXT, value BIGINT, type TEXT)
   Para ello realizaremos tres agrupaciones, una para cada identificador(móvil, antena y app)
   Realizaremos la suma de bytes por cada id y añadiremos el nombre de la métrica correspondiente en cada caso en una columna que llamaremos type.

6. A medida que se vayan creando los DF del punto anterior, se irán guardando en la tabla bytes dentro de la instancia de postgres que tenemos levantada en GCP con el esquema indicado en el punto anterior. Comprobamos en postgres que la tabla bytes se va llenando con los datos obtenidos en los procesos anteriores en base al esquema indicado.

timestamp	id	value	type
2022-10-29 13:15:00	00000000-0000-0000-0000-000000000008	26936	user_total_bytes
2022-10-29 13:15:00	00000000-0000-0000-0000-000000000013	34733	user_total_bytes
2022-10-29 13:15:00	00000000-0000-0000-0000-000000000002	28757	user_total_bytes
2022-10-29 13:15:00	00000000-0000-0000-0000-000000000011	17900	user_total_bytes
2022-10-29 13:15:00	00000000-0000-0000-0000-000000000016	23597	user_total_bytes
2022-10-29 13:15:00	00000000-0000-0000-0000-000000000003	14908	user_total_bytes
2022-10-29 13:15:00	00000000-0000-0000-0000-000000000009	18230	user_total_bytes
2022-10-29 13:15:00	00000000-0000-0000-0000-000000000015	16030	user_total_bytes
2022-10-29 13:15:00	00000000-0000-0000-0000-000000000019	27845	user_total_bytes
2022-10-29 13:15:00	00000000-0000-0000-0000-000000000017	19865	user_total_bytes
2022-10-29 13:15:00	00000000-0000-0000-0000-000000000001	17349	user_total_bytes
2022-10-29 13:15:00	00000000-0000-0000-0000-000000000010	29531	user_total_bytes
2022-10-29 13:15:00	00000000-0000-0000-0000-000000000006	19875	user_total_bytes
2022-10-29 13:15:00	00000000-0000-0000-0000-000000000012	29019	user_total_bytes
2022-10-29 13:15:00	00000000-0000-0000-0000-000000000004	12538	user_total_bytes
2022-10-29 13:15:00	00000000-0000-0000-0000-000000000020	23045	user_total_bytes
2022-10-29 13:15:00	00000000-0000-0000-0000-000000000018	15812	user_total_bytes
2022-10-29 13:15:00	00000000-0000-0000-0000-000000000014	17669	user_total_bytes
2022-10-29 13:15:00	00000000-0000-0000-0000-000000000007	23561	user_total_bytes
2022-10-29 13:15:00	00000000-0000-0000-0000-000000000005	35405	user_total_bytes
2022-10-29 13:15:00	TELEGRAM	141598	app_total_bytes
2022-10-29 13:15:00	SKYPE	95453	app_total_bytes
2022-10-29 13:15:00	FACETIME	102828	app_total_bytes
2022-10-29 13:15:00	FACEBOOK	112726	app_total_bytes
2022-10-29 13:15:00	11111111-1111-1111-1111-111111111111	181659	antenna_total_bytes
2022-10-29 13:15:00	22222222-2222-2222-2222-222222222222	39901	antenna_total_bytes
2022-10-29 13:15:00	33333333-3333-3333-3333-333333333333	99120	antenna_total_bytes
2022-10-29 13:15:00	00000000-0000-0000-0000-000000000000	131925	antenna_total_bytes
2022-10-29 13:20:00	00000000-0000-0000-0000-000000000019	17439	user_total_bytes
2022-10-29 13:20:00	00000000-0000-0000-0000-000000000005	25361	user_total_bytes
2022-10-29 13:20:00	00000000-0000-0000-0000-000000000010	30558	user_total_bytes
2022-10-29 13:20:00	00000000-0000-0000-0000-000000000007	28609	user_total_bytes
2022-10-29 13:20:00	00000000-0000-0000-0000-000000000012	25667	user_total_bytes
2022-10-29 13:20:00	00000000-0000-0000-0000-000000000011	24194	user_total_bytes
2022-10-29 13:20:00	00000000-0000-0000-0000-000000000014	14412	user_total_bytes
2022-10-29 13:20:00	00000000-0000-0000-0000-000000000020	26626	user_total_bytes
2022-10-29 13:20:00	00000000-0000-0000-0000-000000000015	40288	user_total_bytes
2022-10-29 13:20:00	00000000-0000-0000-0000-000000000016	21928	user_total_bytes
2022-10-29 13:20:00	00000000-0000-0000-0000-000000000018	12713	user_total_bytes
2022-10-29 13:20:00	00000000-0000-0000-0000-000000000006	16998	user_total_bytes
2022-10-29 13:20:00	00000000-0000-0000-0000-000000000008	26765	user_total_bytes
2022-10-29 13:20:00	00000000-0000-0000-0000-000000000017	22049	user_total_bytes

Capa Batch Layer

(BigDataProcessing\practica\src\main\scala\io\keepcoding\spark\exercice\batch\BytesBatch.scala)

La capa de batch también está compuesta por varias operaciones:

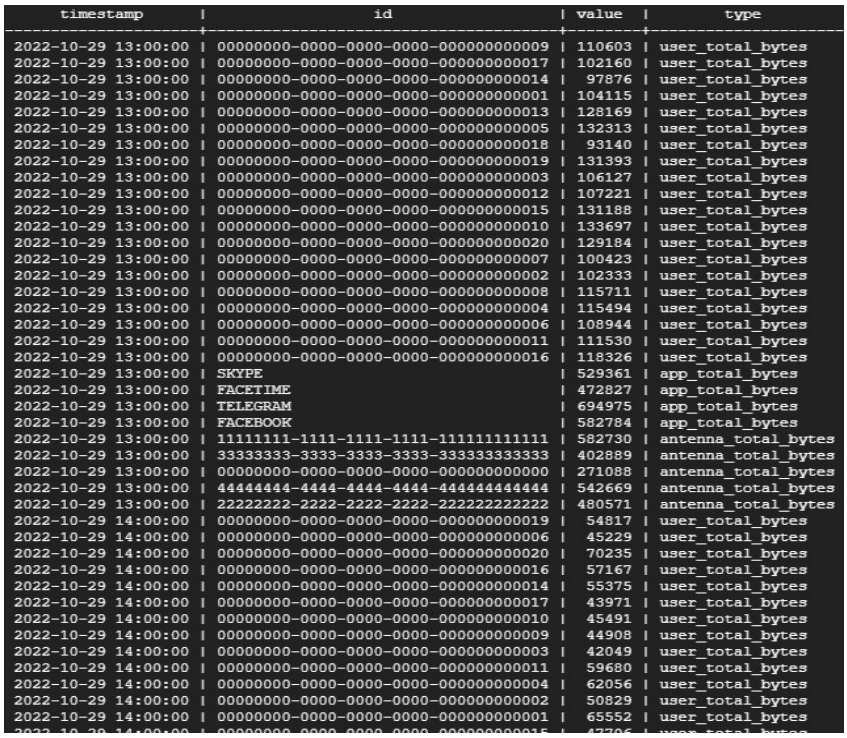
1. Primero realizaremos la lectura de nuestras fuentes de datos:

   a.Archivo parquet. Realizaremos la lectura de la data almacenada en el apartado anterior en base a una fecha y hora determinada

   b.Tabla de Postgres. Almacenaremos los datos de la tabla user_metadata para obtener los datos de los usuarios

2. Con el DF obtenido de la lectura del archivo parquet, igual que en el apartado 5 del apartado de streaming, obtendremos 3 DF con el siguiente esquema:
   (timestamp TIMESTAMP, id TEXT, value BIGINT, type TEXT)
   Realizando nuevamente la suma de bytes por cada id y añadiendo el nombre de la métrica correspondiente en cada caso en una columna también llamada type.

3. Los DF obtenidos en el punto anterior se almacenan en una tabla dentro de la instancia de postgres llamada bytes_hourly. Comprobamos que se insertan los datos:



timestamp	id	value	type
2022-10-29 13:00:00	00000000-0000-0000-0000-000000000009	110603	user_total_bytes
2022-10-29 13:00:00	00000000-0000-0000-0000-000000000017	102160	user_total_bytes
2022-10-29 13:00:00	00000000-0000-0000-0000-000000000014	97876	user_total_bytes
2022-10-29 13:00:00	00000000-0000-0000-0000-000000000001	104115	user_total_bytes
2022-10-29 13:00:00	00000000-0000-0000-0000-000000000013	128169	user_total_bytes
2022-10-29 13:00:00	00000000-0000-0000-0000-000000000005	132313	user_total_bytes
2022-10-29 13:00:00	00000000-0000-0000-0000-000000000018	93140	user_total_bytes
2022-10-29 13:00:00	00000000-0000-0000-0000-000000000019	131393	user_total_bytes
2022-10-29 13:00:00	00000000-0000-0000-0000-000000000003	106127	user_total_bytes
2022-10-29 13:00:00	00000000-0000-0000-0000-000000000012	107221	user_total_bytes
2022-10-29 13:00:00	00000000-0000-0000-0000-000000000015	131188	user_total_bytes
2022-10-29 13:00:00	00000000-0000-0000-0000-000000000010	133697	user_total_bytes
2022-10-29 13:00:00	00000000-0000-0000-0000-000000000020	129184	user_total_bytes
2022-10-29 13:00:00	00000000-0000-0000-0000-000000000007	100423	user_total_bytes
2022-10-29 13:00:00	00000000-0000-0000-0000-000000000002	102333	user_total_bytes
2022-10-29 13:00:00	00000000-0000-0000-0000-000000000008	115711	user_total_bytes
2022-10-29 13:00:00	00000000-0000-0000-0000-000000000004	115494	user_total_bytes
2022-10-29 13:00:00	00000000-0000-0000-0000-000000000006	108944	user_total_bytes
2022-10-29 13:00:00	00000000-0000-0000-0000-000000000011	111530	user_total_bytes
2022-10-29 13:00:00	00000000-0000-0000-0000-000000000016	118326	user_total_bytes
2022-10-29 13:00:00	SKYPE	529361	app_total_bytes
2022-10-29 13:00:00	FACETIME	472827	app_total_bytes
2022-10-29 13:00:00	TELEGRAM	694975	app_total_bytes
2022-10-29 13:00:00	FACEBOOK	582784	app_total_bytes
2022-10-29 13:00:00	11111111-1111-1111-1111-111111111111	582730	antenna_total_bytes
2022-10-29 13:00:00	33333333-3333-3333-3333-333333333333	402889	antenna_total_bytes
2022-10-29 13:00:00	00000000-0000-0000-0000-000000000000	271088	antenna_total_bytes
2022-10-29 13:00:00	44444444-4444-4444-4444-444444444444	542669	antenna_total_bytes
2022-10-29 13:00:00	22222222-2222-2222-2222-222222222222	480571	antenna_total_bytes
2022-10-29 14:00:00	00000000-0000-0000-0000-000000000019	54817	user_total_bytes
2022-10-29 14:00:00	00000000-0000-0000-0000-000000000006	45229	user_total_bytes
2022-10-29 14:00:00	00000000-0000-0000-0000-000000000020	70235	user_total_bytes
2022-10-29 14:00:00	00000000-0000-0000-0000-000000000016	57167	user_total_bytes
2022-10-29 14:00:00	00000000-0000-0000-0000-000000000014	55375	user_total_bytes
2022-10-29 14:00:00	00000000-0000-0000-0000-000000000017	43971	user_total_bytes
2022-10-29 14:00:00	00000000-0000-0000-0000-000000000010	45491	user_total_bytes
2022-10-29 14:00:00	00000000-0000-0000-0000-000000000009	44908	user_total_bytes
2022-10-29 14:00:00	00000000-0000-0000-0000-000000000003	42049	user_total_bytes
2022-10-29 14:00:00	00000000-0000-0000-0000-000000000011	59680	user_total_bytes
2022-10-29 14:00:00	00000000-0000-0000-0000-000000000004	62056	user_total_bytes
2022-10-29 14:00:00	00000000-0000-0000-0000-000000000002	50829	user_total_bytes
2022-10-29 14:00:00	00000000-0000-0000-0000-000000000001	65552	user_total_bytes

4. De los 3DF obtenidos en el apartado 2, necesitaremos el correspondiente a la agrupación por id del móvil. Lo enriqueceremos con los datos almacenados en el punto 1b y obtendremos un DF con el siguiente esquema:
(timestamp TIMESTAMP, id TEXT, value BIGINT, type TEXT, name TEXT, email TEXT, quota BIGINT)

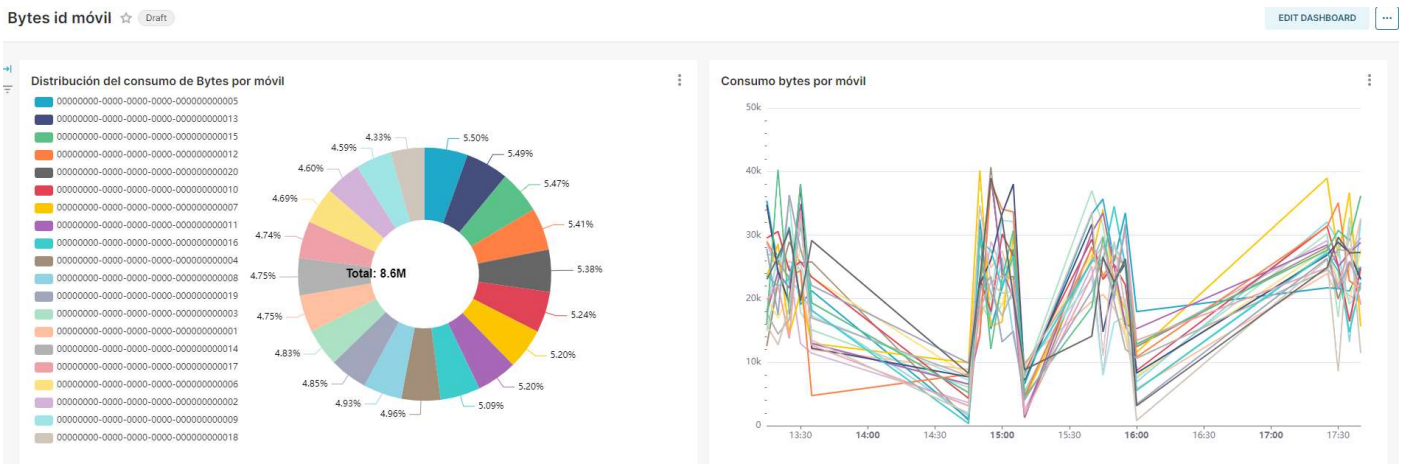
5. Con el DF del punto 4 realizaremos un filtro para que nos muestren los usuarios que tienen los bytesConsumidos/h > quota/h y devolveremos un dataframe con el siguiente esquema:
(email TEXT, usage BIGINT, quota BIGINT, timestamp TIMESTAMP)

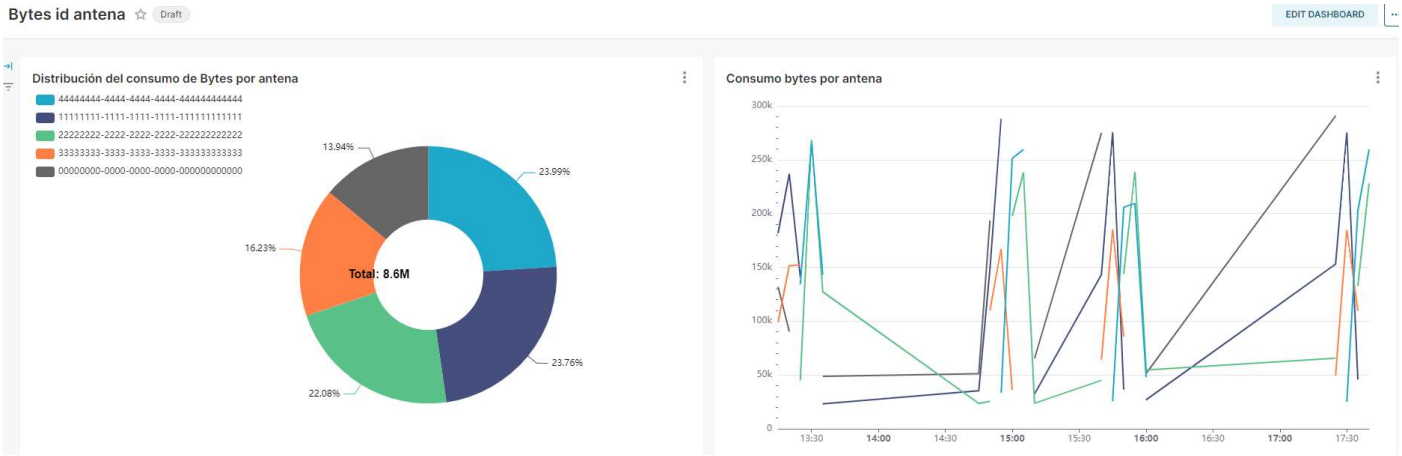
6. El DF que se obtiene en el punto anterior se almacena en una tabla dentro de nuestro postgres llamada user_quota_limit

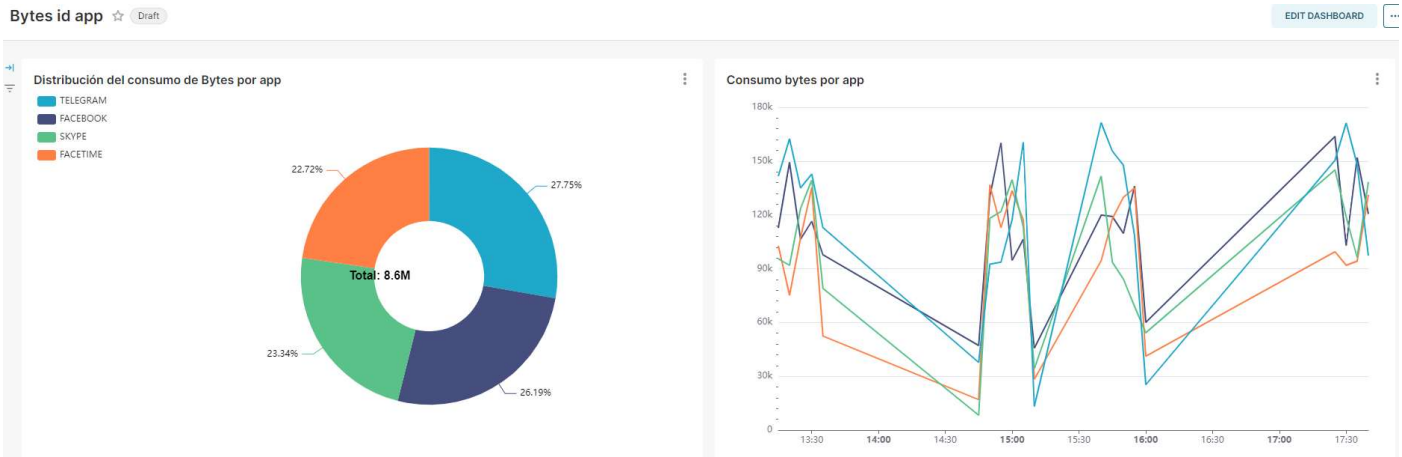
```
postgres=> select * from user_quota_limit;
      email        | usage  | quota  |      timestamp
-------------------+--------+--------+---------------------
 juan@gmail.com    | 106127 | 100000 | 2022-10-29 13:00:00
 fede@gmail.com    | 115494 |   5000 | 2022-10-29 13:00:00
 carlos@gmail.com  | 115711 | 100000 | 2022-10-29 13:00:00
 fede@gmail.com    |  62056 |   5000 | 2022-10-29 14:00:00
 juan@gmail.com    | 156337 | 100000 | 2022-10-29 15:00:00
 fede@gmail.com    | 138482 |   5000 | 2022-10-29 15:00:00
 carlos@gmail.com  | 153606 | 100000 | 2022-10-29 15:00:00
 fede@gmail.com    |  12448 |   5000 | 2022-10-29 16:00:00
 juan@gmail.com    | 113638 | 100000 | 2022-10-29 17:00:00
 fede@gmail.com    | 107362 |   5000 | 2022-10-29 17:00:00
 carlos@gmail.com  | 111394 | 100000 | 2022-10-29 17:00:00
```

Serving Layer

A través de apache superset realizamos las gráficas del flujo de bytes en base a la id del móvil, de la antena y de la app, de los datos que se van recibiendo en streaming, en las siguientes gráficas se muestra el flujo en un intervalo de 13h a 17 h:







En las siguientes imágenes se muestran gráficas obtenidas de la tabla bytes_hourly, es decir, de los datos obtenidos en el apartado batch