28 de abril de 2023

UNIVERSIDAD
POLITÉCNICA
DE MADRID

Implementación de una ETL para monitorización y análisis en un clúster Kafka

Entregable 2

ADW06

Sara Rodríguez Rojo	19I011
Carlos Nieto Petinal	19I021
Daniel Ramón Robertson	19I033
Guillermo Vélez Fernández	190048

Universidad Politécnica de Madrid, Escuela Técnica Superior de Ingenieros Informáticos

Contenido

Introducción

Los Data Warehouses se han convertido en una herramienta imprescindible para las empresas que necesitan analizar grandes cantidades de datos y obtener información valiosa para la toma de decisiones. Sin embargo, la complejidad de las tecnologías que intervienen en estos puede hacer que su implementación y mantenimiento sea una tarea compleja.

Con el objetivo de analizar y comprender la diversidad y complejidad de las tecnologías que intervienen en un Data Warehouse, desarrollamos en esta entrega la implementación de una herramienta ETL que permita monitorizar un conjunto de direcciones IP y analizar los accesos registrados en el log de un servidor Apache mediante un clúster Kafka. El proyecto se divide en tres fases principales: la carga de datos desde las fuentes al clúster Kafka, el procesado de los datos mediante ksql y la generación de resultados mediante consultas.

En la primera fase, la herramienta ETL se encarga de recoger los datos de los logs del servidor Apache y enviarlos al clúster de Kafka para su posterior procesamiento. En la segunda fase, los datos son procesados mediante ksql, una herramienta de procesamiento de datos de streaming que permite realizar consultas en tiempo real sobre los datos almacenados en el clúster de Kafka. La tercera fase del proyecto consiste en la generación de resultados mediante consultas. En esta fase se generan dos informes y una tabla según las indicaciones del enunciado.

El desarrollo del trabajo se ha llevado a cabo en plataforma nativa Linux, de forma local mediante la herramienta Docker. Además, se ha hecho uso de GitHub para ayudar a la colaboración en el desarrollo del código por parte de los integrantes.

1. Carga inicial de datos

Para realizar una ejecución más rápida, se han creado ficheros bash con las instrucciones necesarias para la carga inicial de datos. Algunos de estos scripts se han montado dentro de los contenedores de Docker utilizando volúmenes.

1.1 Carga inicial del log

Se ha modificado el archivo docker-compose.yml:

```
  ksqldb-cli:
    image: confluentinc/ksqldb-cli:0.28.2
    container_name: ksqldb-cli
    depends_on:
      - broker
      - ksqldb-server
    volumes:
      - ./server.ksql:/server.ksql
    entrypoint: /bin/sh
    tty: true

  mysql:
    image: mysql:8.0
    hostname: mysql
    container_name: mysql
    ports:
      - 33060:3306
    command: --default-authentication-plugin=mysql_native_password
    restart: on-failure
    volumes:
      - ./mysql/monitor.sql:/monitor.sql
      - ./mysql/createMonitor.sh:/createMonitor.sh
    environment:
      - MYSQL_ROOT_PASSWORD=rpwd23
      - MYSQL_USER=kafka
      - MYSQL_PASSWORD=kpwd23
      - MYSQL_DATABASE=kafka
```

Figura 1. Modificaciones del archivo docker-compose.yml.

Creación de la base de datos en MySQL, ingesta de los registros en logstash y ejecución del script de ksql con todos los conectores, streams y tablas (se ejecuta dos veces debido a que uno de los streams se intenta crear antes de que el topic del que depende exista, un ASSERT TOPIC con TIMEOUT no lo soluciona ya que parece que los scripts de ksql no se ejecutan de forma secuencial):

```
#!/bin/bash

docker-compose up -d

docker exec mysql /createMonitor.sh

read -n 1 -p "Press key after logstash started"
./log-ingest/ingest.sh 2306 ./log-ingest/muestra.log localhost 50000 &

sleep 2
read -n 1 -p "Press key after ksqldb-server started"
docker exec -t ksqldb-cli ksql -f /server.ksql http://ksqldb-server:8088
sleep 1

docker exec -t ksqldb-cli ksql -f /server.ksql http://ksqldb-server:8088
```

Figura 2. Código bash del arranque de los contenedores.

1.2 Carga de la tabla monitor desde la BD

Creación de la tabla monitor, así como otras operaciones que han sido agrupadas en el fichero *monitor.sql* (creación de la tabla ***geoip*** y carga de datos en la tabla monitor, anteriormente encontradas en otros ficheros). Primero se espera a la disponibilidad del socket del servidor de la base de datos, para posteriormente ejecutar el contenido de *monitor.sql* MySQL.

```
#!/bin/sh

until [ -S /var/run/mysqld/mysqld.sock ]
do
        sleep 1
done

while ! echo "source monitor.sql" | mysql -u kafka --password=kpwd23 -D kafka 1> /dev/null 2> /dev/null
do
        sleep 1
done
```

Figura 3. Código bash para ejecutar en el contenedor de MySQL el fichero monitor.sql

2. Procesamiento en *ksql*

2.1 Tópicos

Los topics se crean junto a los treams (en caso de no existir ya). Cuando se despliegue la ETL sobre el clúster de la asignatura se deberá especificar los valores PARTITIONS=2 y REPLICAS=2, aún no añadidos.

2.2 Streams

Creación del flujo de datos ***2306_apache_raw*** ligado al topic ***logstash*** con la definición de la estructura de los datos para desaplanarlos. La estructura incluye una serie de campos, algunos de ellos con estructuras anidadas:

```
/* CREATE STREAM FROM LOGSTASH  */
CREATE STREAM IF NOT EXISTS 2306_apache_raw (
        tag STRING,
        timestamp STRING,
        user_agent STRUCT<
                uaid STRING,
                device STRUCT<name STRING>,
                version STRING,
                original STRING,
                name STRING,
                os STRUCT<
                        `full` STRING,
                        version STRING,
                        name STRING>>,
        url STRUCT<original STRING>,
        http STRUCT<
                response STRUCT<status_code INT, body STRUCT<bytes INT>>,
                request STRUCT<method STRING>,
                version STRING>,
        geoip STRUCT<
                ip STRING,
                geo STRUCT<
                        country_name STRING,
                        timezone STRING,
                        country_iso_code STRING,
                        region_iso_code STRING,
                        region_name STRING,
                        city_name STRING,
                        location STRUCT<lon DOUBLE, lat DOUBLE>,
                        continent_code STRING,
                        postal_code STRING>,
                `as` STRUCT<organization STRUCT<name STRING>, number INT>>)
        WITH (KAFKA_TOPIC='logstash', VALUE_FORMAT='JSON');
```

Creación del stream ***2306_apache_filtered*** ligado al topic ***adw.2306.filtered***. El topic se crea en este punto, recibiendo los datos de la consulta. Este nuevo stream contiene únicamente los campos de ***2306_apache_raw*** que se vayan a usar posteriormente, pero solo acepta los registros que cumplen con las condiciones especificadas. Para este Stream, se podría haber comprobado si todos los campos son NULL, sin embargo, hemos optado por solamente depurar aquellos campos que vamos a utilizar, de manera que se inserten datos correctos en las tablas, manteniendo la mayor cantidad posible de entradas.

```
CREATE STREAM IF NOT EXISTS  2306_apache_filtered
WITH (KAFKA_TOPIC='adw.2306.filtered', KEY_FORMAT='PROTOBUF',VALUE_FORMAT='AVRO')
AS
SELECT user_agent, http, geoip
```

```
FROM 2306_apache_raw
WHERE
tag='2306'
AND user_agent->uaid IS NOT NULL
AND http->response->status_code IS NOT NULL
AND http->request->method IS NOT NULL
AND geoip->ip IS NOT NULL
AND geoip->geo->country_name IS NOT NULL
AND geoip->geo->timezone IS NOT NULL
AND geoip->geo->country_iso_code IS NOT NULL
AND geoip->geo->region_iso_code IS NOT NULL
AND geoip->geo->region_name IS NOT NULL
AND geoip->geo->city_name IS NOT NULL
AND geoip->geo->location->lon IS NOT NULL
AND geoip->geo->location->lat IS NOT NULL
AND geoip->geo->continent_code IS NOT NULL
AND geoip->geo->postal_code IS NOT NULL
AND geoip->`as`->organization->name IS NOT NULL
AND geoip->`as`->number IS NOT NULL;
```

Creación del stream ***2306_apache_filtered_keyed*** ligado al topic ***adw.2306.filtered.keyed*** a partir del stream ***2306_apache_filtered***. Divide los datos del stream en función de los campos geoip -> ip y user_agent -> uaid, de manera que obtendremos mensajes con claves ESTRUCTURADAS, para, posteriormente, poder utilizarlas en las consultas KSQL. Asimismo, se utiliza el formato de clave *PROTOBUF*, formato que nos permite utilizar dichas claves estructuradas, como ya se ha comentado en clase.

```
CREATE STREAM IF NOT EXISTS 2306_apache_filtered_keyed
WITH (KAFKA_TOPIC='adw.2306.filtered.keyed', KEY_FORMAT='PROTOBUF', VALUE_FORMAT='AVRO')
AS
SELECT user_agent, http, geoip,
STRUCT(IP:=geoip->ip, UAID:=user_agent->uaid) AS KEY
FROM 2306_apache_filtered
PARTITION BY STRUCT(IP:=geoip->ip, UAID:=user_agent->uaid);
```

Creación del stream ***2306_monitor_stream*** ligado al topic ***adw.2306.monitor***.

```
CREATE STREAM IF NOT EXISTS 2306_monitor_stream (
        idrt INT,
        IP STRING,
        UAID STRING,
        NV INT,
        UV TIMESTAMP,
        rtreg TIMESTAMP)
WITH (KAFKA_TOPIC='adw.2306.monitor', VALUE_FORMAT='AVRO');
```

Creación del stream ***2306_monitor_stream_keyed*** ligado al topic ***adw.2306.monitor.keyed*** a partir del stream ***2306_monitor_stream***. Funciona de manera análoga al caso anterior de ***2306_apache_filtered_keyed***.

```
CREATE STREAM IF NOT EXISTS 2306_monitor_stream_keyed
WITH (KAFKA_TOPIC='adw.2306.monitor.keyed', KEY_FORMAT='PROTOBUF', VALUE_FORMAT='AVRO')
AS
SELECT idrt, IP, UAID, NV, UV,
STRUCT(IP:=IP, UAID:=UAID) AS KEY
FROM 2306_monitor_stream
PARTITION BY STRUCT(IP:=IP, UAID:=UAID);
```

2.3 Tabla

Creación de la tabla ***2306_apache_geoip*** ligada al topic ***adw.2306.geoip*** necesaria para el apartado 3.2.

```
CREATE TABLE IF NOT EXISTS 2306_apache_geoip
  WITH (KAFKA_TOPIC='adw.2306.geoip')
AS SELECT
  geoip,
  geoip->ip AS ip,
  geoip->`as`->organization->name AS nomorg,
  geoip->`as`->number AS numorg,
  geoip->geo->postal_code AS postal_code,
  geoip->geo->city_name AS city_name,
  geoip->geo->country_name AS country_name,
  geoip->geo->country_iso_code AS country_iso_code,
  geoip->geo->region_name AS region_name,
  geoip->geo->region_iso_code AS region_iso_code,
  geoip->geo->continent_code AS continent_code,
  geoip->geo->timezone AS timezone,
  geoip->geo->location->lon AS lon,
  geoip->geo->location->lat AS lat,
  COUNT(*) as ipc
FROM 2306_apache_filtered_keyed
GROUP BY geoip;
```

2.4 Conectores

Definición del conector que permite la inyección de datos en el clúster. Para lograr esto, se utiliza la clase ***JdbcSourceConnector*** de ***Confluent Connect JDBC***. Además, se especifican los detalles de conexión a la base de datos MySQL, como la URL de conexión, el usuario y la contraseña. Se establece el modo de carga de datos y se especifica la columna de incremento. También se define la lista blanca de tablas a cargar, en este caso la tabla ***monitor***, y el prefijo del topic al que se asignarán los datos. Por último, se aplican algunas transformaciones a los datos, como la creación de una clave y la extracción del campo ***idrt***, que es la columna de los identificadores que definimos anteriormente como incremental.

```
CREATE SOURCE CONNECTOR IF NOT EXISTS `2306-monitor-jdbc-source` WITH (
        'connector.class'='io.confluent.connect.jdbc.JdbcSourceConnector',
        'connection.url'='jdbc:mysql://mysql:3306/kafka',
        'connection.user' = 'kafka',
        'connection.password'='kpwd23',
        'mode'='incrementing',
        'incrementing.column.name'='idrt',
        'validate.non.null'='false',
        'table.whitelist'='monitor',
        'topic.prefix'='adw.2306.',
        'transforms' = 'createKey, extractInt',
        'transforms.createKey.type' = 'org.apache.kafka.connect.transforms.ValueToKey',
        'transforms.createKey.fields' = 'idrt',
        'key.converter.schemas.enable' = 'true',
        'transforms.extractInt.type' = 'org.apache.kafka.connect.transforms.ExtractField$Key',
        'transforms.extractInt.field' = 'idrt'
);
```

Definición del conector que permite exportar datos de topics a cualquier base de datos relacional. Para lograr esto, se utiliza la clase ***JdbcSinkConnector*** de ***Confluent Connect JDBC***. Al igual que el otro conector, se especifican los detalles de conexión a la base de datos MySQL, como la URL de conexión, el usuario y la contraseña. Se especifican el topic del clúster que se exportará, en este caso ***adw.2306.geoip***, y el nombre de la tabla en la base de datos relacional donde se insertarán los datos (***geoip***). También se especifica el formato de clave y valor, el esquema del registro y se definen algunas opciones adicionales, como el modo de primary key y el modo de inserción.

```
/* Create SINK Connector. pk.mode is record_value (we take a value from a record, IP, to be PK of
each value) */
CREATE SINK CONNECTOR IF NOT EXISTS `2306-geoip-jdbc-sink` WITH(
    "connector.class" = 'io.confluent.connect.jdbc.JdbcSinkConnector',
    "connection.url" = 'jdbc:mysql://mysql:3306/kafka',
    "topics" = 'adw.2306.geoip',
    "table.name.format" = 'geoip',
    "key.converter" = 'io.confluent.connect.protobuf.ProtobufConverter',
    "key.converter.schema.registry.url" = 'http://schema-registry:8081',
    "key.converter.schemas.enable" = 'true',
    "value.converter" = 'io.confluent.connect.avro.AvroConverter',
    "value.converter.schema.registry.url" = 'http://schema-registry:8081',
    "value.converter.schemas.enable" = 'true',
    "connection.user" = 'kafka',
    "connection.password" = 'kpwd23',
    "auto.create" = 'false',
    "pk.mode" = 'record_value',
    "pk.fields" = 'IP',
    "insert.mode" = 'upsert',
    "delete.enabled" = 'false',
    "tasks.max" = '1');
```

3. Generación de resultados

En esta sección, se detallará la generación de resultados del proyecto, la cual incluye la creación de dos informes basados en consultas y una tabla ***geoip*** (ya definida por el conector ***2306-geoip-jdbc-sink*** descrito en el último apartado). Estos resultados permitirán obtener una mejor comprensión y análisis de los datos recopilados a través del proceso ETL implementado. Los informes se elaboran en función de los requisitos definidos en el enunciado, y la tabla contiene los datos de las IPs geolocalizadas extraídas del log en la tabla ***geoip*** cuya orden de creación se incluye en el script *geoip-ddl.sql*. A continuación, se describirá con mayor detalle el proceso de generación de cada uno de estos resultados.

3.1 Consultas *ksql*

3.1.1 Accesos recurrentes

El primer informe generado a partir de la consulta siguiente muestra información detallada sobre los accesos correspondientes a las ***IPs*** y ***UAIDs*** que son objeto de monitorización.

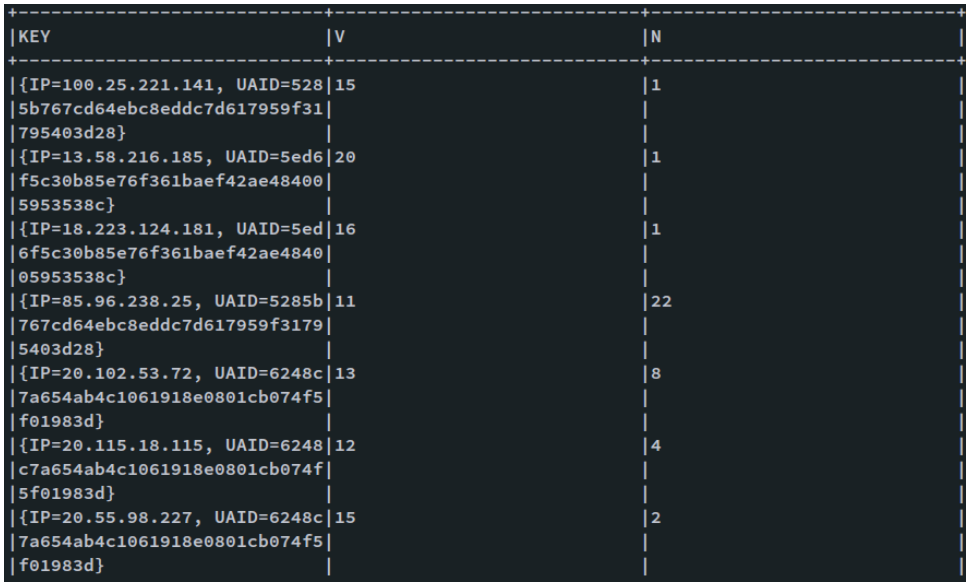
En primer lugar, se utiliza la cláusula SELECT para seleccionar los campos que se desean mostrar en el informe. En este caso, se seleccionan tres campos: KEY, V y N. La consulta se realiza mediante una INNER JOIN entre los streams ***monitor_stream_keyed*** y el ***apache_filtered_keyed***, filtrada por el período de 300 días, con un período de gracia de 15 minutos. Los resultados se agrupan por la columna KEY del stream de monitorización y se limitan a 5 registros para facilitar su análisis.

```
-- Consulta Q1
SELECT  MSK.KEY AS KEY,
        MSK.NV AS V,
        COUNT(AFK.http) AS N
FROM 2306_monitor_stream_keyed MSK
        INNER JOIN 2306_apache_filtered_keyed AFK
        WITHIN 300 DAYS
        GRACE PERIOD 15 MINUTES
        ON AFK.KEY = MSK.KEY
GROUP BY MSK.KEY, MSK.NV
EMIT CHANGES
limit 7;
```

Los resultados se presentan en una tabla con la columna ***KEY*** que identifica a la IP y UAID monitorizado, la columna ***V*** que indica el número total de visitas original y la columna ***N*** que indica el número de visitas registradas.

En el código final entregado, hemos decidido abstraer dicha consulta en forma de tabla, para facilitar la exploración del informe con un simple SELECT *.

```
 -- Tabla para la Consulta de Q1
CREATE TABLE IF NOT EXISTS 2306_Q1
WITH (KAFKA_TOPIC='adw.2306.Q1')
AS
SELECT MSK.KEY AS KEY,
MSK.NV AS V,
COUNT(AFK.http) AS N
FROM 2306_monitor_stream_keyed MSK
INNER JOIN 2306_apache_filtered_keyed AFK
WITHIN 300 DAYS
GRACE PERIOD 15 MINUTES
ON AFK.KEY = MSK.KEY
GROUP BY MSK.KEY, MSK.NV;
```



KEY	V	N
{IP=100.25.221.141, UAID=528 5b767cd64ebc8eddc7d617959f31 795403d28}	15	1
{IP=13.58.216.185, UAID=5ed6 f5c30b85e76f361baef42ae48400 5953538c}	20	1
{IP=18.223.124.181, UAID=5ed 6f5c30b85e76f361baef42ae4840 05953538c}	16	1
{IP=85.96.238.25, UAID=5285b 767cd64ebc8eddc7d617959f3179 5403d28}	11	22
{IP=20.102.53.72, UAID=6248c 7a654ab4c1061918e0801cb074f5 f01983d}	13	8
{IP=20.115.18.115, UAID=6248 c7a654ab4c1061918e0801cb074f 5f01983d}	12	4
{IP=20.55.98.227, UAID=6248c 7a654ab4c1061918e0801cb074f5 f01983d}	15	2

Figura 4. Tabla resultado de Q1.

3.1.2 Accesos desde una UAID

Este informe está enfocado en mostrar los registros del log que contienen una UAID específica. Para cada UAID, se muestra la última visita registrada en el campo ***UV***, las listas de métodos y códigos de estado en las que se clasifican las peticiones (***MET*** y ***STC***, respectivamente) y el número total de accesos registrados (***N***).

En términos de código, la consulta utiliza la misma lógica que la anterior, haciendo una unión interna del stream de monitoreo con el filtrado de apache y agrupando los resultados por clave de monitoreo. La cláusula WHERE limita los resultados a las UAID especificadas. Por último, la consulta utiliza las funciones MAX, COLLECT_SET y COUNT para obtener los valores de UV, MET, STC y N. La cláusula limit restringe los resultados a un máximo de 5 para limitar el tamaño del informe.

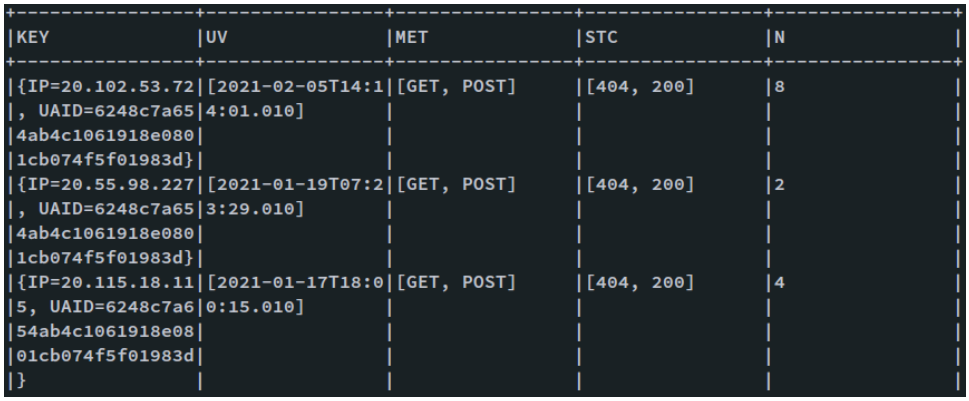
```
 -- Consulta Q2

SELECT MSK.KEY AS KEY,
        ARRAY[MAX(MSK.UV)] AS UV,
        COLLECT_SET(AFK.HTTP->REQUEST->METHOD) AS MET,
        COLLECT_SET(AFK.HTTP->RESPONSE->STATUS_CODE) AS STC,
        COUNT(AFK.http) AS N
FROM 2306_monitor_stream_keyed MSK
        INNER JOIN 2306_apache_filtered_keyed AFK
        WITHIN 300 DAYS
        GRACE PERIOD 15 MINUTES
        ON AFK.KEY = MSK.KEY
WHERE   MSK.KEY->UAID= '6248c7a654ab4c1061918e0801cb074f5f01'
GROUP BY MSK.KEY
EMIT CHANGES
limit 3;
```

Al igual que con la anterior consulta, hemos creado una segunda tabla con los datos del informe.

```
-- Tabla para la Consulta de Q2
CREATE TABLE IF NOT EXISTS 2306_Q2
WITH (KAFKA_TOPIC='adw.2306.Q2')
AS
SELECT MSK.KEY AS KEY,
ARRAY[MAX(MSK.UV)] AS UV,
```

```
COLLECT_SET(AFK.HTTP->REQUEST->METHOD) AS MET,
COLLECT_SET(AFK.HTTP->RESPONSE->STATUS_CODE) AS STC,
COUNT(AFK.http) AS N
FROM 2306_monitor_stream_keyed MSK
INNER JOIN 2306_apache_filtered_keyed AFK
WITHIN 300 DAYS
GRACE PERIOD 15 MINUTES
ON AFK.KEY = MSK.KEY
WHERE MSK.KEY->UAID= '6248c7a654ab4c1061918e0801cb074f5f01983d'
GROUP BY MSK.KEY;
```



KEY	UV	MET	STC	N
{IP=20.102.53.72, UAID=6248c7a64ab4c1061918e0801cb074f5f01983d}	[2021-02-05T14:14:01.010]	[GET, POST]	[404, 200]	8
{IP=20.55.98.227, UAID=6248c7a64ab4c1061918e0801cb074f5f01983d}	[2021-01-19T07:23:29.010]	[GET, POST]	[404, 200]	2
{IP=20.115.18.11, UAID=6248c7a654ab4c1061918e0801cb074f5f01983d}	[2021-01-17T18:05:15.010]	[GET, POST]	[404, 200]	4

Figura 4. Tabla resultado de Q2.

Cabe destacar que ambas queries no son del todo iguales, ya que la segunda es una query PULL, y la primera una query PUSH. La diferencia mas notable es que el SELECT sobre la tabla (PULL) nos devuelve datos y acaba en cuanto los devuelve, de manera similar a lo que hemos visto siempre en los SGBDR ya estudiados. Sin embargo, la query PUSH no se comporta así, se queda a la espera de que lleguen nuevos mensajes para mostrarlos como resultado.

3.2 Tabla geoip

El código, que se mostró en el apartado 2.4, crea un conector **JdbcSinkConnector** que permite volcar los datos de las IPs geolocalizadas extraídas del log en la tabla ***geoip*** de la base de datos MySQL. Los datos se convierten utilizando el convertidor Avro y se establece la conexión a la base de datos utilizando la URL de conexión, nombre de usuario y contraseña:

```
"connection.url" = 'jdbc:mysql://mysql:3306/kafka'
"connection.user" = 'kafka',
"connection.password" = 'kpwd23',
```

Además, se establecen las configuraciones de clave principal (pk), modo de inserción y eliminación, y el número máximo de tareas:

```
"pk.mode" = 'record_value',
"pk.fields" = 'IP',
"insert.mode" = 'upsert',
"delete.enabled" = 'false',
"tasks.max" = '1'
```

Con esta configuración, los datos se insertan o actualizan según la clave primaria (IP) en la tabla ***geoip***. Se han incluido tanto las IPs de monitor como las de logstash.