UNIVERSIDAD MAYOR DE SAN SIMÓN
FACULTAD DE CIENCIAS Y TECNOLOGÍA
DIRECCIÓN DE POSGRADO

DIPLOMADO ESTADÍSTICA APLICADA A LA TOMA DE DECISIONES

SEGUNDA VERSIÓN

Laboratorio Streaming K CONNECT

NOMBRE : HEBERT JUAN DE DIOS DELGADILLO FERNANDEZ
CARLOS ALFREDO ORIHUELA BERRIOS

DOCENTE : DANNY LUIS HUANCA SEVILLA

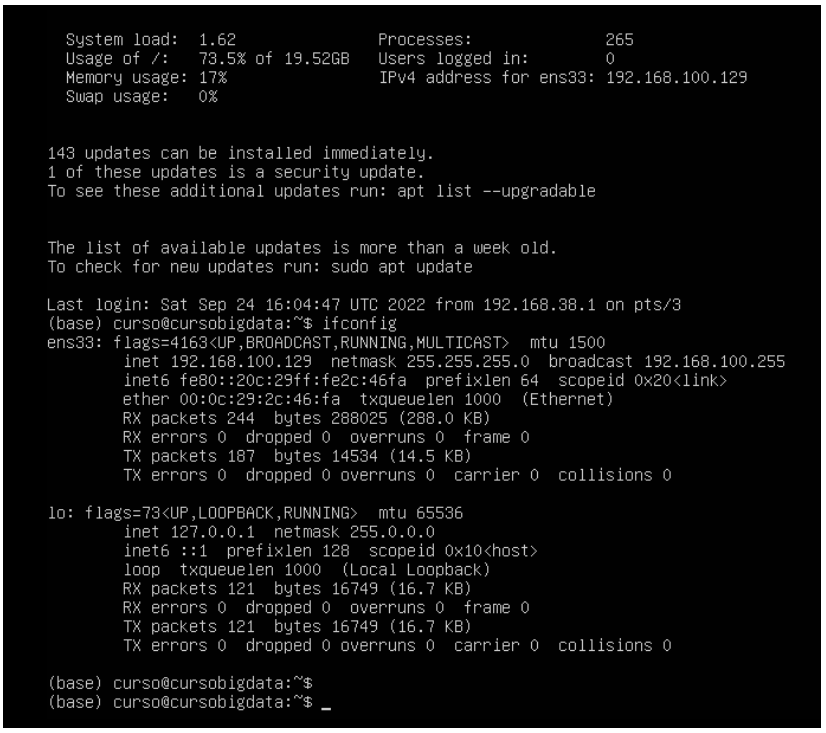
Cochabamba – Bolivia

2023

GUIA DE LABORATORIO KAFKA CONNECT

1. Pasos iniciales

1. Encender la máquina virtual.



2. Ejecutar la siguiente sentencia para evitar que exista algún parámetro no controlado al iniciar el laboratorio:

confluent local destroy

3. Iniciar el servidor KSQL-server. Utilice el script iniciarConfluent.

```
(base) curso@cursobigdata:~$ iniciarConfluent
    The local commands are intended for a single-node development environment
    only, NOT for production usage. https://docs.confluent.io/current/cli/index.html

Using CONFLUENT_CURRENT: /tmp/confluent.ZopCws8w
Starting zookeeper
zookeeper is [UP]
Starting kafka
kafka is [UP]
Starting schema-registry
schema-registry is [UP]
Starting kafka-rest
kafka-rest is [UP]
Starting connect
connect is [UP]
Starting ksql-server
ksql-server is [UP]
Starting control-center
control-center is [UP]
    The local commands are intended for a single-node development environment
    only, NOT for production usage. https://docs.confluent.io/current/cli/index.html

Using CONFLUENT_CURRENT: /tmp/confluent.ZopCws8w
zookeeper is already running. Try restarting if needed
kafka is already running. Try restarting if needed
schema-registry is already running. Try restarting if needed
ksql-server is already running. Try restarting if needed
```

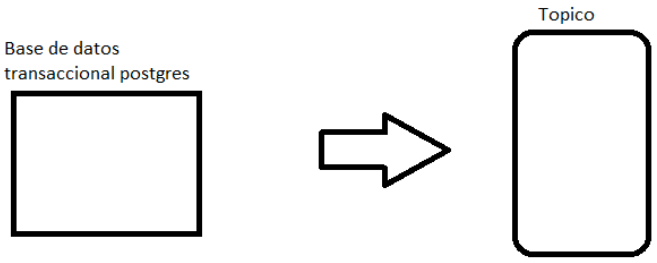
2. Conectar a una base de datos Postgres

2.1 Caso de negocio

En una empresa de telecomunicaciones se tienen la necesidad de consumir datos de una base de datos transaccional en Streaming, la misma se encuentra en PostgressQL. Estos datos tienen que tener la capacidad de ser consumidos por diferentes aplicaciones, otros sistemas como un CRM o para analítica en tiempo real.

Se le consulta por la posible solución y usted indica que lo mejor es consumirlos usando Kafka.

Tiene varias posibilidades de usar esta tecnología pero dado que la requiere rápido, decide iniciar un piloto usando la plataforma *Confluent* que contiene todo lo necesario empaquetado y le permitirá reducir tiempos, en lugar de instalar componente por componente (Kafka, zookeeper, etc).



2.2 Desarrollo

El ejercicio consiste en conectar a una base de datos y consumir datos en streaming utilizando un conector que proporciona la plataforma confluent.

Se usará un conector JDBC el mismo que ya viene instalado en la versión que se encuentra en su máquina virtual.

La base de datos postgres no se encuentra instalada, por lo que se debe proceder primeramente a la instalación de la misma, en caso quiera probar con otra base puede instalarla también.

1. Instalar la base de datos postgresQL

sudo apt update

```
Get:14 http://bo.archive.ubuntu.com/ubuntu focal-updates/multiverse amd64 Packages [25.8 kB]
Get:15 http://bo.archive.ubuntu.com/ubuntu focal-updates/multiverse Translation-en [7484 B]
Get:16 http://bo.archive.ubuntu.com/ubuntu focal-updates/multiverse amd64 c-n-f Metadata [620 B]
Get:17 http://bo.archive.ubuntu.com/ubuntu focal-backports/main amd64 Packages [45.7 kB]
Get:18 http://bo.archive.ubuntu.com/ubuntu focal-backports/main amd64 c-n-f Metadata [1420 B]
Get:19 http://bo.archive.ubuntu.com/ubuntu focal-backports/universe amd64 Packages [25.0 kB]
Get:20 http://bo.archive.ubuntu.com/ubuntu focal-backports/universe Translation-en [16.3 kB]
Get:21 http://bo.archive.ubuntu.com/ubuntu focal-backports/universe amd64 c-n-f Metadata [880 B]
Get:22 http://bo.archive.ubuntu.com/ubuntu focal-security/main amd64 Packages [2401 kB]
Get:23 http://bo.archive.ubuntu.com/ubuntu focal-security/main Translation-en [377 kB]
Get:24 http://bo.archive.ubuntu.com/ubuntu focal-security/main amd64 c-n-f Metadata [13.1 kB]
Get:25 http://bo.archive.ubuntu.com/ubuntu focal-security/restricted amd64 Packages [2120 kB]
Get:26 http://bo.archive.ubuntu.com/ubuntu focal-security/restricted Translation-en [296 kB]
Get:27 http://bo.archive.ubuntu.com/ubuntu focal-security/restricted amd64 c-n-f Metadata [580 B]
Get:28 http://bo.archive.ubuntu.com/ubuntu focal-security/universe amd64 Packages [874 kB]
Get:29 http://bo.archive.ubuntu.com/ubuntu focal-security/universe Translation-en [183 kB]
Get:30 http://bo.archive.ubuntu.com/ubuntu focal-security/universe amd64 c-n-f Metadata [19.0 kB]
Get:31 http://bo.archive.ubuntu.com/ubuntu focal-security/multiverse amd64 Packages [23.6 kB]
Get:32 http://bo.archive.ubuntu.com/ubuntu focal-security/multiverse Translation-en [5504 B]
Get:33 http://bo.archive.ubuntu.com/ubuntu focal-security/multiverse amd64 c-n-f Metadata [548 B]
Fetched 14.0 MB in 7s (1889 kB/s)
Reading package lists... Done
Building dependency tree
Reading state information... Done
252 packages can be upgraded. Run 'apt list --upgradable' to see them.
```

sudo apt install postgresql postgresql-contrib

```
(base) curso@cursobigdata:~$ sudo apt install postgresql postgresql-contrib
Reading package lists... Done
Building dependency tree
Reading state information... Done
postgresql is already the newest version (12+214ubuntu0.1).
postgresql-contrib is already the newest version (12+214ubuntu0.1).
The following packages were automatically installed and are no longer required:
  libhawtjni-runtime-java libjansi-java libjansi-native-java libjline2-java scala-library
  scala-parser-combinators scala-xml
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 252 not upgraded.
```

Verificar que la instalación esta correcta

sudo -u postgres psql -c "SELECT version();"

```
                                                 version
---------------------------------------------------------------------------------------------------------
 PostgreSQL 12.12 (Ubuntu 12.12-0ubuntu0.20.04.1) on x86_64-pc-linux-gnu, compiled by gcc (Ubuntu 9.4.0-1ubunt
u1~20.04.1) 9.4.0, 64-bit
(1 row)
```

2. Inicializar la base

Cambiar al usuario postgres

sudo -i -u postgres

```
(base) curso@cursobigdata:~$ sudo -i -u postgres
postgres@cursobigdata:~$ whoiam
whoiam: command not found
postgres@cursobigdata:~$ whoami
postgres
postgres@cursobigdata:~$
```

Psql y \password

```
postgres@cursobigdata: $ psql
psql (12.12 (Ubuntu 12.12-0ubuntu0.20.04.1))
Type "help" for help.

postgres=# \password
Enter new password for user "postgres":
Enter it again:
postgres=#
```

El password que se asigno es **curso123**

3. Crear una tabla de usuarios, ejecutar el siguiente script:

CREATE TABLE usuarios (

nombre VARCHAR

, id SERIAL PRIMARY KEY

);

```
postgres=# \dt
          List of relations
 Schema |   Name   | Type  |  Owner
--------+----------+-------+----------
 public | usuarios | table | postgres
(1 row)
```

4. Insertar algunos registros

INSERT INTO usuarios (nombre) VALUES ('Pedro');

INSERT INTO usuarios (nombre) VALUES ('Mauricio');

INSERT INTO usuarios (nombre) VALUES ('Dunia');

```
postgres=# INSERT INTO usuarios (nombre) VALUES ('Pedro');
INSERT 0 1
postgres=# INSERT INTO usuarios (nombre) VALUES ('Mauricio');
INSERT 0 1
postgres=# INSERT INTO usuarios (nombre) VALUES ('Dunia');
INSERT 0 1
```

Realizando un select para comprobar que la tabla fue insertada.

select * from usuarios;

```
postgres=# select * from usuarios;
  nombre   | id
-----------+----
 Pedro     |  1
 Mauricio  |  2
 Dunia     |  3
```

5. En Shell. Asegurar que el servicio connect se encuentre arriba para ello ejecutar:

confluent local status

```
curso(base) curso@cursobigdata:~$ confluent local status
    The local commands are intended for a single-node development environment
    only, NOT for production usage. https://docs.confluent.io/current/cli/index.html

control-center is [UP]
ksql-server is [UP]
connect is [UP]
kafka-rest is [UP]
schema-registry is [UP]
kafka is [UP]
zookeeper is [UP]
```

6. Ingresar al cliente KSQL, escriba en el terminal

ksql

En el cliente ksql. Crear el conector.

CREATE SOURCE CONNECTOR `postgres-jdbc-source` WITH(

"connector.class"='io.confluent.connect.jdbc.JdbcSourceConnector',

"connection.url"='jdbc:postgresql://127.0.0.1:5432/postgres',

"mode"='incrementing',

"incrementing.column.name"='id',

"table.whitelist"='usuarios',

"connection.password"='curso123',

"connection.user"='postgres',

"topic.prefix"='db-',

"key"='nombre');

```
ksql> CREATE SOURCE CONNECTOR `postgres-jdbc-source` WITH(
>"connector.class"='io.confluent.connect.jdbc.JdbcSourceConnector',
>"connection.url"='jdbc:postgresql://127.0.0.1:5432/postgres',
>"mode"='incrementing',
>"incrementing.column.name"='id',
>"table.whitelist"='usuarios',
>"connection.password"='curso123',
>"connection.user"='postgres',
>"topic.prefix"='db-',
>"key"='nombre');
>

 Message
----------------------------------------
 Created connector postgres-jdbc-source
----------------------------------------
```

Listar si el tópico fue creado.

list topics;

```
 _confluent-license                                                | 1
1
 _confluent-metrics                                                | 12
1
 _confluent-monitoring                                             | 2
1
 _schemas                                                          | 1
1
 connect-configs                                                   | 1
1
 connect-offsets                                                   | 25
1
 connect-statuses                                                  | 5
1
 db-usuarios                                                       | 1
1
 default_ksql_processing_log                                       | 1
1
```

7. En el cliente ksql. Mostrar el contenido del tópico creado automáticamente al crear el conector, el nombre del mismo es **db-usuarios**.

print 'db-usuarios' from beginning;

```
ksql> print 'db-usuarios' from beginning;
Format:AVRO
9/1/23 12:52:53 AM UTC, Pedro, {"nombre": "Pedro", "id": 7}
9/1/23 12:52:53 AM UTC, Mauricio, {"nombre": "Mauricio", "id": 8}
9/1/23 12:52:53 AM UTC, Dunia, {"nombre": "Dunia", "id": 9}
```

8. ¿Qué pasaría si se incrementa una fila a la tabla creada?

En el Shell de postgress, ejecutar:

INSERT INTO usuarios (nombre) VALUES ('Pedro');

INSERT INTO usuarios (nombre) VALUES ('Silvia');

```
postgres=# INSERT INTO usuarios (nombre) VALUES ('Pedr
o');
INSERT 0 1
postgres=# INSERT INTO usuarios (nombre) VALUES ('Silv
ia');
INSERT 0 1
```

En el cliente ksql. Se puede apreciar que los datos se actualizan en casi tiempo real.

```
ksql> print 'db-usuarios' from beginning;
Format:AVRO
9/1/23 12:52:53 AM UTC, Pedro, {"nombre": "Pedro", "i
9/1/23 12:52:53 AM UTC, Mauricio, {"nombre": "Maurici
9/1/23 12:52:53 AM UTC, Dunia, {"nombre": "Dunia", "i
9/1/23 12:59:42 AM UTC, Pedro, {"nombre": "Pedro", "i
d": 10}TRL-C to interrupt
9/1/23 12:59:42 AM UTC, Silvia, {"nombre": "Silvia",
"id": 11}
```

Preguntas

1. ¿Pueden imaginarse un caso de negocio aplicado a las empresas en las que trabajan que se puede aplicar esta situación?

   Dentro de las empresas que conozco, la mayoría son suficiente con datos que se actualizan en batch, sin embargo, un ejemplo podría ser, una ladrillera que constantemente debe mantener la curva de temperatura del ladrillo, la cual todo el tiempo está mandando datos a una aplicación móvil y cada que hay una situación de riesgo corrigen la situación a la distancia. Otro ejemplo podría ser las aplicaciones de transporte como inDrive, Uber o PedidosYA que esta última constantemente está mandando datos de los pedidos y no solo a 1 consumidor ni de 1 solo productor, sino que es bidireccional entre clientes, proveedores de comida o los repartidores. También una empresa de viajes nacionales e internacionales, donde se realizan reservas de viajes Online, mediante una aplicación donde el usuario puede registrar su reserva, esto incluye el horario, lugar de destino, lugar de origen, número de asiento, ect. Al ser un tipo de transacción en tiempo real deben usarse tecnologías que sean eficientes y confiables. Para evitar que 2 usuarios reserven el mismo número de asiento simultáneamente, o que un cliente pueda reservar un vuelo cuando este ya estaba lleno.

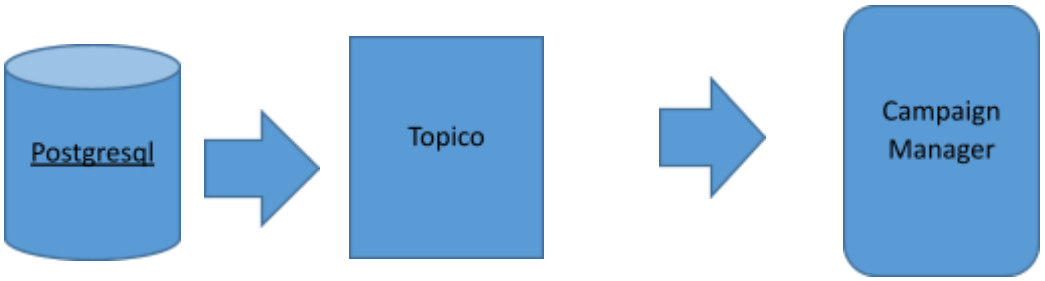
2. Una mejor estrategia de ingesta de una base de datos es mediante conectores CDC (change data capture). Puede investigar cómo funciona esa tecnología y por qué cree que es mejor hacerlo de esta manera que usando un conector JDBC directo a la base de datos.

   No es mejor una tecnología que otra, pero para actualización de datos en streaming la tecnología CD es superior, JDBC está diseñado para una interacción cliente-servidor SQL

con tablas relacionales, CDC captura los cambios en las tablas en tiempo real, esto quiere decir que para saber el estado de nuestras tablas con JDBC tendríamos que estar consultando constantemente el estado al servidor, y para simular el tiempo real tendríamos que consultar al menos 1 vez por segundo al servidor, mientras que CDC detectara cuando una tabla es alterada y le informará a todos los clientes SQL del cambio y los mantendrá actualizados y esto solo pasara cuando haya un cambio en la tabla, haciéndolo más eficiente para streaming.

CDC, es una tecnología para identificar y rastrear cambios de datos en bases de datos y tablas de origen en tiempo real, donde va escuchando todo el tiempo y si encuentra cambios en una base de datos origen, realiza una copia de esa información hacia la base de datos destino.
La diferencia de usar un JDBC es que no se podría realizar operaciones complejas en los datos, ya que CDC trabaja mejor con transacciones simples para aumentar la velocidad de proceso, con un menor impacto de cambios en el sistema origen de una forma más eficiente.

3. Un caso de uso típico de esta situación es un campaign manager (que es un software que permite generar campañas de retención de clientes). Los datos de los clientes de consumo que están en una base de datos, se pasan en streaming en un txt o csv a una carpeta remota. El sistema de generación de campañas lee los archivos generados que luego son usados para generar campañas. El esquema de funcionamiento se presenta en la siguiente gráfica.



¿En base a lo aprendido en el diplomado, cree que existe alguna alternativa similar que pueda resolver el mismo problema? Justifique la respuesta.

Para empezar una tecnología similar seria mediante streaming de datos en tiempo real y no en archivos csv, lo que entendí es que cada csv streameado contiene todos los datos desde el inicio y esto lo haría mucho menos eficiente, además estamos suponiendo que el sistema de campaña necesita realmente un streaming en tiempo real, sin embargo las campañas no tienen una validez instantánea, es decir la campaña no expira a la hora siguiente suelen durar más de un día y por esto mismo no es necesario que los datos estén actualizados en tiempo real, por lo que la actualización de datos podría ser mediante la tecnología batch, que es mas que suficiente para el caso y más accesible económicamente hablando.