



# ETL Project - Third Delivery: Credit Card Transactions (Airflow + Kafka)

Repository

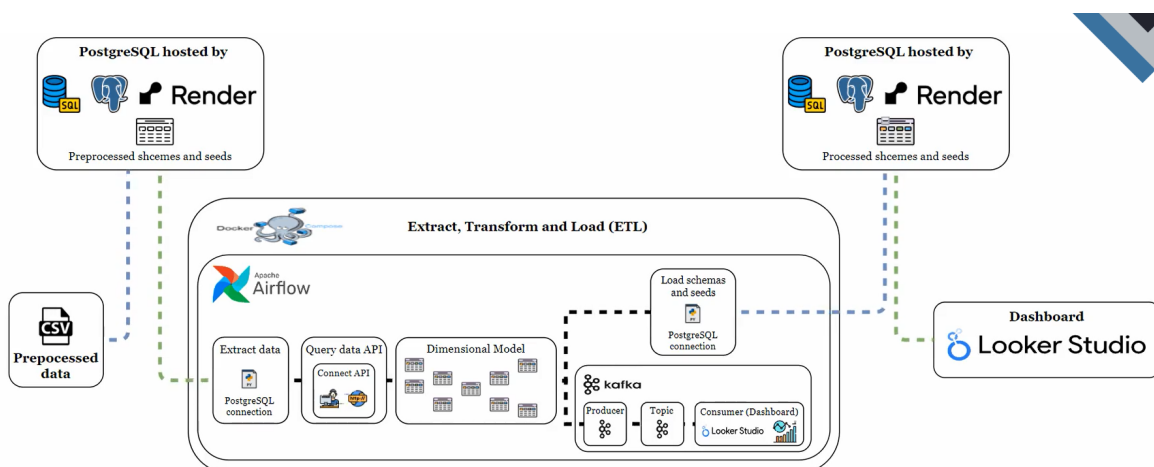
Canva

## Team dev

- David Alejandro Cajiao Lazt
- Sebastián Ortiz
- Juan Andrés López Alvarez

Cosas por Hacer: @JUAN ANDRES LOPEZ ALVAREZ @JUAN SEBASTIAN ORTIZ GUERRERO

## Diagrama del Proyecto:



## Flujo del Proyecto

El flujo general de nuestro proyecto sigue los siguientes pasos:

1. **Procesamiento Inicial:** Partimos de datos "sucios" o sin procesar (raw data), los cuales son cargados en una base de datos PostgreSQL alojada en Render.
2. **Extracción y Enriquecimiento:** Posteriormente, extraemos los datos desde la base de datos y los procesamos en nuestro entorno. En este punto, realizamos una integración (merge) con una API diseñada para obtener datos demográficos, específicamente la población de cada estado de EE. UU.
3. **Modelado:** Una vez enriquecidos los datos, se genera el esquema en formato Snowflake.
4. **Almacenamiento Final:** El esquema Snowflake se carga en una segunda base de datos en Render. Esta base de datos se utiliza para alimentar un dashboard estático donde se visualizan los datos procesados.

Este flujo describe el funcionamiento principal de nuestro proyecto, según la entrega anterior.

---

## Mejora Incorporada: (+Kafka)

En la nueva iteración, se añadió un paso adicional entre la generación del esquema Snowflake y su carga en la base de datos:

- Antes de cargar los datos procesados, se toma una muestra representativa de 1,000 registros. Estos registros son publicados como un topic en Apache Kafka, desde donde son enviados en streaming hacia nuestro dashboard en Looker.
- Aunque la transmisión de datos se realiza en tiempo real, el dashboard es considerado de actualización periódica debido a la limitación de la tasa mínima de refresco permitida por Looker, que es de 1 minuto.

## Elementos de la Base de Datos:

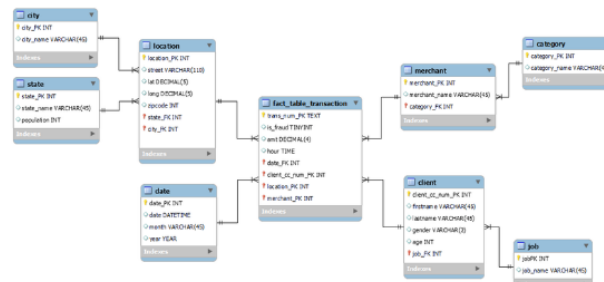
# DATABASE MODEL

What do we have in our database?

Version 2



Raw\_table



Snowflake Model



data\_streaming

Tabla de Hechos: `fact_table_transaction`

La tabla de hechos contiene datos numéricos y claves foráneas necesarias para analizar las transacciones.

## Campos:

- **trans\_num\_PK:** (TEXT) Clave primaria que identifica cada transacción.
- **is\_fraud:** (TINYINT) Indica si la transacción fue fraudulenta.
- **amt:** (DECIMAL) Monto de la transacción.
- **hour:** (TIME) Hora de la transacción.
- **date\_FK:** (INT) Clave foránea que referencia la tabla de fechas.
- **client\_cc\_num\_FK:** (INT) Clave foránea que referencia la tabla de clientes.
- **location\_PK:** (INT) Clave foránea que referencia la tabla de ubicaciones.
- **merchant\_PK:** (INT) Clave foránea que referencia la tabla de comerciantes.

## Tablas de Dimensiones

### Dimensión Fecha ( `date` )

La tabla de fecha almacena información de fecha y hora para las transacciones.

**Campos:**

- **date\_PK:** (INT) Clave primaria para la fecha.
- **date:** (DATETIME) Fecha y hora completas.
- **month:** (VARCHAR) Mes de la transacción.
- **year:** (YEAR) Año de la transacción.

## Dimensión Cliente ( **client** )

La tabla de cliente almacena detalles del cliente asociado con cada transacción.

**Campos:**

- **client\_cc\_num\_PK:** (INT) Clave primaria del cliente.
- **firstname:** (VARCHAR) Nombre del cliente.
- **lastname:** (VARCHAR) Apellido del cliente.
- **gender:** (VARCHAR) Género del cliente.
- **age:** (INT) Edad del cliente.
- **job\_FK:** (INT) Clave foránea que referencia la tabla de trabajos.

## Dimensión Ubicación ( **location** )

La tabla de ubicación almacena información sobre dónde ocurrió la transacción y está normalizada en tablas separadas para ciudad, estado y país.

**Campos:**

- **location\_PK:** (INT) Clave primaria para la ubicación.
- **street:** (VARCHAR) Dirección donde se realizó la transacción.
- **lat:** (DECIMAL) Latitud de la ubicación.
- **long:** (DECIMAL) Longitud de la ubicación.
- **zipcode:** (INT) Código postal.
- **state\_FK:** (INT) Clave foránea que referencia la tabla de estados.
- **city\_FK:** (INT) Clave foránea que referencia la tabla de ciudades.

- **country\_code\_FK:** (VARCHAR) Clave foránea que referencia la tabla de países.

## Dimensión Comerciante ( **merchant** )

La tabla de comerciante almacena detalles del comerciante involucrado en cada transacción.

### Campos:

- **merchant\_PK:** (INT) Clave primaria del comerciante.
- **merchant\_name:** (VARCHAR) Nombre del comerciante.
- **category\_FK:** (INT) Clave foránea que referencia la tabla de categorías.

## Dimensión Categoría ( **category** )

La tabla de categorías almacena las categorías de los comerciantes.

### Campos:

- **category\_PK:** (INT) Clave primaria para la categoría.
- **category\_name:** (VARCHAR) Nombre de la categoría.

## Dimensión Trabajo ( **job** )

La tabla de trabajos almacena la ocupación o trabajo del cliente.

### Campos:

- **job\_PK:** (INT) Clave primaria para el trabajo.
- **job\_name:** (VARCHAR) Título del trabajo.

## Dimensión Estado ( **state** )

La tabla de estados almacena información sobre el estado/provincia donde ocurrió la transacción.

### Campos:

- **state\_PK:** (INT) Clave primaria para el estado.
- **state\_name:** (VARCHAR) Nombre del estado.
- **population:** (INT) Población del estado.

## Dimensión Ciudad ( **city** )

La tabla de ciudades almacena información sobre la ciudad donde ocurrió la transacción.

### **Campos:**

- **city\_PK:** (INT) Clave primaria para la ciudad.
- **city\_name:** (VARCHAR) Nombre de la ciudad.

## Dimensión País ( **country** )

La tabla de países almacena información sobre el país donde ocurrió la transacción.

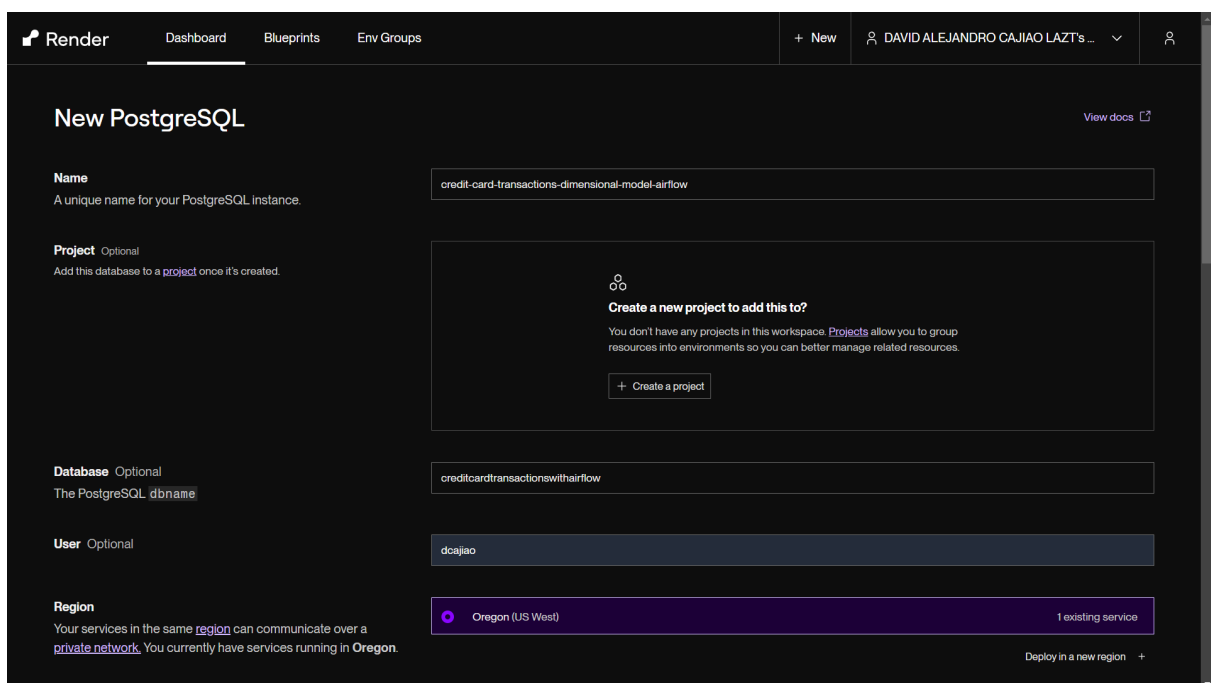
### **Campos:**

- **country\_code\_PK:** (VARCHAR) Clave primaria para el país.
- **name:** (VARCHAR) Nombre del país.

## Creación de la Base de Datos y subida de datos:

### ¿Qué se hizo?

Tuvimos que volver a crear y subir los datos que teníamos alojados en render, debido a que la versión gratis de render tiene una limitante de almacenamiento y además de tiempo de alojamiento de información.



The screenshot shows the 'New PostgreSQL' configuration page in the Render console. The page has a dark theme and includes a navigation bar at the top with 'Render', 'Dashboard', 'Blueprints', and 'Env Groups'. The main content area contains several form fields:

- Name:** A unique name for your PostgreSQL instance. The value entered is 'credit-card-transactions-dimensional-model-airflow'.
- Project:** Optional. Add this database to a project once it's created. A modal dialog is open, asking 'Create a new project to add this to?' with the text: 'You don't have any projects in this workspace. Projects allow you to group resources into environments so you can better manage related resources.' and a '+ Create a project' button.
- Database:** Optional. The PostgreSQL dbname. The value entered is 'creditcardtransactionswithairflow'.
- User:** Optional. The value entered is 'dcajiao'.
- Region:** Your services in the same region can communicate over a private network. You currently have services running in Oregon. The value selected is 'Oregon (US West)', which is highlighted in purple and indicates '1 existing service'.

At the bottom right, there is a 'Deploy in a new region' button with a plus icon.

Render

DashboardBlueprintsEnv Groups

+ New

DAVID ALEJANDRO CAJIAO LAZT's ...

POSTGRES

credit-card-transactions-dimensional-model-airflow

Free

Upgrade your instance →

View docs

Connect

Info

Apps

Metrics

Recovery

Logs

Info

General

Name

A unique name for your database.

credit-card-transactions-dimensional-model-airflow

Edit

Created

4 minutes ago

Status

Available

PostgreSQL Version

16

Region

Oregon (US West)

Read Replica

+ Add Read Replica

Storage

0% used out of 1 GB

Datadog API Key

+ Add Datadog API Key

✓ 12m 24.2s

TYPE \_\_\_\_\_ / \_\_\_\_\_ 1L- \_\_\_\_\_ 1-1-1- \_\_\_\_\_ 40750 \_\_\_\_\_

## Algunas dificultades con la Base de Datos en Render:





Hi there,

Your PostgreSQL database [credit-card-transactions-dimensional-model-airflow](#) is using more than 90% of its available storage.

If you exceed your storage limit, your database will be suspended, resulting in downtime. To avoid interruptions, you can [increase your storage](#).

If you have questions, please reach out to our support team in the [Render Dashboard](#).

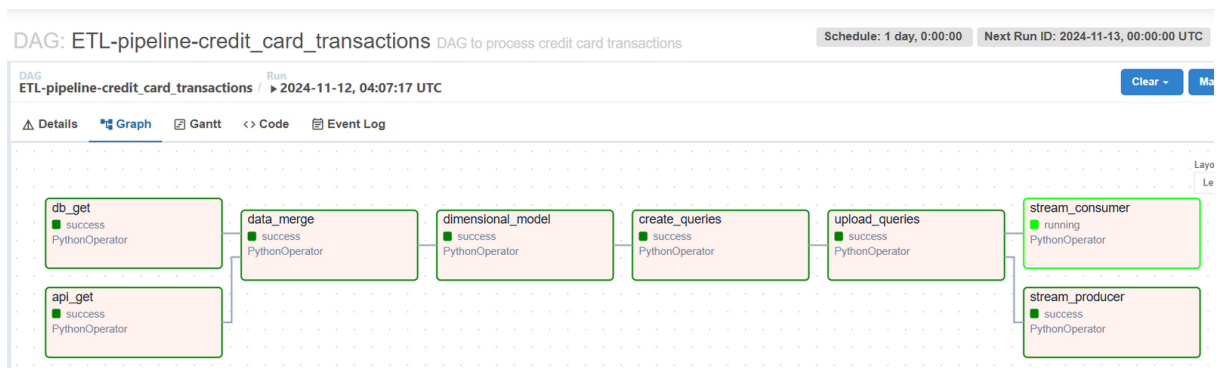
Thank you,  
The Render Team

**Instance Type**  
Set your database's RAM and CPU. You can change your instance type later.

**Free**  
For testing out PostgreSQL on Render

**Free** 256 MB (RAM)  
\$0 / month 0.1 CPU  
1 GB (Storage)

## DAGS en Apache-Airflow para Kafka:



El pipeline ETL consta de tres tareas principales orquestadas por Airflow:

1. **Extracción de Datos desde la API ( `api_get` )**: Recupera datos de una API externa.
2. **Extracción de Datos desde la Base de Datos ( `db_get` )**: Obtiene datos de una base de datos.
3. **Fusión de Datos ( `data_merge` )**: Combina los datos extraídos de la API y de la base de datos en un DataFrame unificado.
4. **Modelado Dimensional ( `dimensional_model` )**: Transforma los datos en tablas de hechos y dimensiones.

5. **Generación de Consultas ( `create_queries` )**: Crea esquemas SQL y archivos de datos iniciales (seed data) para cada dimensión.
6. **Carga de Datos ( `upload_queries` )**: Sube las consultas y esquemas generados a la base de datos para su posterior análisis.
7. **Kafka producer**: Toma una muestra de 100 datos para empezar el proceso de streaming de la info.
8. **Kafka Consumer**: Se encarga de escuchar los mensajes enviados al topic, para luego renderizar un query con base al contenido del mensaje, e insertar los datos en la tabla que hace de simulación de tiempo real con Looker

## ¿Por qué Kafka producer y consumer están en un mismo DAG?

Aunque sabemos que esto no es una buena práctica. Se decidió añadir ya que se quería automatizar lo más que se pudiera la ejecución de nuestro proyecto, quitando así la necesidad de abrir dos terminales para la ejecución del kafka (producer y consumer). Automatizando esto con el docker compose y añadiéndolo como una task de Airflow.

## Dashboard:

Se recalca que: Aunque la transmisión de datos se realiza en tiempo real, el dashboard es considerado de actualización periódica debido a la limitación de la tasa mínima de refresco permitida por Looker, que es de 1 minuto.

[003\\_ETL-REALTIME-Credit-Card-Transaction-Dashboard](#)

## Conexión con el tablero de Looker

Looker Studio

Crear

Informe

Fuente de datos

Explorador BETA

Papera

Plantillas

Recientes
Informes
Fuentes de datos
Explorador

Empezar con una plantilla

Informe vacío  
Looker Studio

GA4 Report  
Google Analytics

Acme Marketing  
Google Analytics

Search Console Report  
Search Console

Galería de plantillas

Nombre
Cualquiera es el propietario
Abierto última vez por mí
Ubicación

## Fuente de datos sin título

Buscar

### Google Connectors (1 de 24)

Connectors built and supported by Looker Studio [Más información](#)

#### PostgreSQL

De Google

Conéctese a las bases de datos de PostgreSQL.

← SELECCIONAR CONECTOR



## PostgreSQL

De Google

El conector de PostgreSQL te permite acceder a la información de las bases de datos. El conector usa el controlador JDBC de PostgreSQL para conectar una fuente de datos de PostgreSQL.

[MÁS INFORMACIÓN](#)

[NOTIFICAR UN PROBLEMA](#)

### BÁSICA

URL DE JDBC

#### Autenticación de la base de datos



☐ Habilitar SSL 

← SELECCIONAR CONECTOR

BÁSICA

URL DE JDBC

Nombre de host o IP

dpg-csgrp6jtg21c73duohlq-a.oregon-

Puerto (opcional)

5432

Base de datos

creditcardtransactionswithairflow

Nombre de usuario

dcajiao

Contraseña

.....

☒ Habilitar SSL ?

☐ Habilitar autenticación de cliente ?

Archivos de configuración de SSL de PostgreSQL

credentials.pem

Server certificate

AUTENTICAR

← SELECCIONAR CONECTOR

BÁSICA

URL DE JDBC

Nombre de host o IP

dpg-csgrp6jtq21c73duohlg-a.oregon-

Puerto (opcional)

5432

Base de datos

creditcardtransactionswithairflow

Nombre de usuario

dcajiao

Contraseña

.....

☒ Habilitar SSL ?

☐ Habilitar autenticación de cliente ?

Archivos de configuración de SSL de PostgreSQL

credentials.pem

Server certificate

AUTENTICAR

TABLAS

CONSULTA PERSONALIZADA

Tabla

category\_dim  
client\_dim  
data\_streaming  
date\_dim  
fact\_transaction\_dim  
job\_dim  
location\_dim  
merchant\_dim  
raw\_table  
state\_dim

Recientes

Informes

Fuentes de datos

Explorador

Nombre

Cualquiera es el propietario

Abierto última vez por mí

Ubicación

credit-card-data-streaming

David Alejandro Cajiao Lazt

20:27

Soy el propieta...

credit-card-data-streaming

Ámbito: Reutilizable | Credenciales de datos: Almacenadas | Actualización de datos: 12 horas | Acceso a visualizaciones comunitarias: Activado | Edición de campos de informes: Activada

EDITAR CONEXIÓN | FILTRAR POR CORREO ELECTRÓNICO

Campo	Tipo
is_fraud	Booleano
job	Texto
last	Texto
lat	Número
long	Número
merch_zipcode	Número
merchant	Texto
state	Texto
street	Texto
trans_date_trans_time	Texto
trans_num	Texto
zip	Número

MÉTRICAS (1)

Métrica	Tipo
Record Count	Número

ACTUALIZAR LOS CAMPOS

22/22 campos

Actualización de datos

PostgreSQL: credit-card-data-streaming

¿Cada cuánto quiere que se actualicen los datos?

Cuanto más frecuentes sean las actualizaciones, más recientes serán los datos. No obstante, una frecuencia de actualización mayor puede reducir el rendimiento y aumentar los costes de las consultas de servicios de datos de pago. [Más información](#)

Consulta datos actualizados:

Cada 15 minutos   Cada hora   Cada 4 horas   Cada 12 horas Predeterminada   Personalizada

Defina el periodo personalizado:

Cada 1 minutos

CANCELAR AJUSTAR LA ACTUALIZACIÓN DE DATOS

# Evidencia de ejecución:

[https://youtu.be/Lq-f\\_krFQXo](https://youtu.be/Lq-f_krFQXo)

## README:

### Dependencias:

Git

Docker & Docker compose

```
git clone https://github.com/DCajiao/Credit-Card-Transaction-ETL-Project
cd Credit-Card-Transaction-ETL-Project
```

Estructura del archivo `.env` necesario:

Se agrega el archivo `.env` en la ruta `./src/.env`

```
DBNAME = your_name_db_on_render
DBUSER = your_user
DBPASS = your_password
DBHOST = your_host_on_render
DBPORT = 5432 (optional) you can change it.
```

Levantamiento de servicios con Docker

```
docker-compose up airflow-init
```

```
docker-compose up --build -d
```