**Requisitos:**

1. **Desarrollo del ETL**
   * Se usará Python.
   * Recuperar datos de la API de YouTube Data (reproducciones de videos).
   * Transformar y limpiar los datos para que se adapten a una base de datos timeseries PostgreSQL.
2. **Almacenamiento en Base de Datos**
   * La base de datos debe ser un sistema timeseries compatible con PostgreSQL.
   * Almacenar la información del video, incluyendo metadatos como título, id, duración y el número de reproducciones a partir de una fecha específica.
3. **Generación de Informes**
   * Crear un notebook en Python para analizar los datos.
   * Generar gráficos que demuestren la evolución temporal del número de reproducciones por video.
   * El notebook debe incluir la documentación de las tareas realizadas y los resultados obtenidos.

URL de github: <https://github.com/PPalomino/Megamedia.git>

GIT version 2.47.1.

**1. Configuración del ETL**

Se utilizará Python (v3.13.1) para desarrollar el ETL.

**a. Extracción de datos**

Primero, se obtendrá una clave de API de YouTube y se usará la biblioteca oficial de la API de YouTube Data para extraer los datos.

Se genera la clave de la API (AIzaSyCdrebworNo2iTcXg3WYAwwFKRzsqDgPok):

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

**Bash (Instalar la biblioteca de cliente de YouTube)**:

pip install google-api-python-client

Texto

Descripción generada automáticamente

**Python (Configurar la conexión con la API de YouTube)**:

from googleapiclient.discovery import build

# Configurar API key

api\_key = ' AIzaSyCdrebworNo2iTcXg3WYAwwFKRzsqDgPok '

# Construir el objeto del servicio

youtube = build('youtube', 'v3', developerKey=api\_key)

# Función para obtener datos de un canal

def obtener\_datos\_video(video\_id):

request = youtube.videos().list(

part="snippet,contentDetails,statistics",

id=video\_id

)

response = request.execute()

return response['items'][0]

**# Test de uso, con ID de Meganoticias en Vivo**

video\_id = '9aw9GX9Aw0k'

datos\_video = obtener\_datos\_video(video\_id)

print(datos\_video)

**Resultado:**

Patrón de fondo

Descripción generada automáticamente con confianza baja

**b. Transformación y limpieza de los datos (Transform)**

Transformamos y limpiamos los datos para que se ajusten a nuestra base de datos timeseries en PostgreSQL.

import Extraer\_datos\_de\_YouTube as ext

from datetime import datetime

def transformar\_datos(datos\_video):

snippet = datos\_video['snippet']

statistics = datos\_video['statistics']

content\_details = datos\_video['contentDetails']

datos\_limpios = {

'video\_id': datos\_video['id'],

'titulo': snippet['title'],

'fecha\_publicacion': snippet['publishedAt'],

'duracion': content\_details['duration'],

'reproducciones': statistics.get('viewCount', 0),

'fecha\_extraccion': datetime.now()

}

return datos\_limpios

**# Test de uso, con ID de Meganoticias en Vivo**

video\_id = '9aw9GX9Aw0k'

datos\_video = ext.obtener\_datos\_video(video\_id)

datos\_limpios = transformar\_datos(datos\_video)

print("Datos Limpios: "+ str(datos\_limpios))

Resultado:



**Configuración final del Pipeline ETL:**

# Librería para realizar solicitudes HTTP

import requests

# Librería para manipulación y análisis de datos

import pandas as pd

# Librería para conectarse a la base de datos

from sqlalchemy import create\_engine

# Biblioteca para trabajar con fechas y horas

from datetime import datetime

# Librería para especificar tipos de datos en las funciones

from typing import List, Tuple

# Configuración de la API para autenticar las solicitudes de YouTube

API\_KEY = 'AIzaSyCdrebworNo2iTcXg3WYAwwFKRzsqDgPok'

# ID del canal de Meganoticias en Vivo, desde donde se extraerán los datos

CHANNEL\_ID = '9aw9GX9Aw0k'

# Función para obtener datos de los videos

def obtener\_datos(api\_key: str, channel\_id: str) -> List[Tuple[str, str, str, str, int]]:

# Construir la URL para la solicitud a la API de YouTube

url: str = f'https://www.googleapis.com/youtube/v3/search?key={api\_key}&channelId={channel\_id}&part=snippet,id&order=date&maxResults=20'

# Hacer GET a la API de YouTube

response = requests.get(url)

# Convertir la respuesta JSON a una variable de Python

data = response.json()

# Se genera Lista para almacenar los datos de los videos

video\_data: List[Tuple[str, str, str, str, int]] = []

# Iterar a través de los elementos en la respuesta JSON

for item in data['items']:

if item['id']['kind'] == 'youtube#video': # Verificamos si el elemento es un video de YouTube

video\_id: str = item['id']['videoId']

title: str = item['snippet']['title']

published\_at: str = item['snippet']['publishedAt']

duration: str = "00:00:00" # dummy para la duración

views: int = 0 # dummy para el número de reproducciones

video\_data.append((video\_id, title, published\_at, duration, views))

return video\_data # Retornamos la lista con los datos de los videos

# Función para transformar los datos

def transformar\_datos(video\_data: List[Tuple[str, str, str, str, int]]) -> pd.DataFrame:

# Crear un DataFrame de pandas a partir de los datos de los videos

df: pd.DataFrame = pd.DataFrame(video\_data, columns=['video\_id', 'titulo', 'fecha\_publicacion', 'duracion', 'reproducciones'])

# Convertir la columna fecha\_publicacion a un objeto de fecha y hora

df['fecha\_publicacion'] = pd.to\_datetime(df['fecha\_publicacion'])

df['fecha\_extraccion'] = datetime.now() # se añadie la fecha de extracción

return df # Retornar el DataFrame transformado

# Función para cargar los datos en PostgreSQL con TimescaleDB

def cargar\_datos(df: pd.DataFrame, db\_uri: str) -> None:

# Creamos una conexión a la base de datos utilizando SQLAlchemy

engine = create\_engine(db\_uri)

# Convertir a una hypertable si no lo es

with engine.connect() as conn:

conn.execute("CREATE EXTENSION IF NOT EXISTS timescaledb CASCADE;")

conn.execute("""

SELECT create\_hypertable('videos', 'fecha\_publicacion', if\_not\_exists => TRUE);

""")

# Cargar el DataFrame en la tabla 'videos' de la base de datos PostgreSQL

# Si la tabla ya existe, se reemplaza

df.to\_sql('videos', engine, if\_exists='append', index=False)

# Pipeline ETL

# Extraer los datos de los videos utilizando la API de YouTube

video\_data = obtener\_datos(API\_KEY, CHANNEL\_ID)

# Transformar los datos extraídos en un DataFrame

transformed\_data = transformar\_datos(video\_data)

# Cargar los datos transformados en la base de datos PostgreSQL

cargar\_datos(transformed\_data, 'postgresql+psycopg2://postgres:Ads1122@localhost:5432/YouTube\_Data')

Haciendo pruebas, vemos que faltaba instalar psycopg2-binary:

Texto

Descripción generada automáticamente

**Nota**: psycopg2-binary tiene dependencia de Microsoft Visual C++ Build Tools

**c. Almacenamiento en Base de Datos**

Utilizaremos PostgreSQL (v 16.6) con TimescaleDB para almacenar los datos. La matriz de compatibilidad es la siguiente:

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza media

**Habilitación de la extensión TimescaleDB:**

Se instala correctamente TimeScaleDB (v16):

Texto

Descripción generada automáticamente

En "Variables del sistema", edita Path y añade la ruta C:\Program Files\PostgreSQL\16\bin

Texto

Descripción generada automáticamente

**Configuración de la base de datos**

Se crea una nueva base de datos y un usuario con los permisos necesarios:

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

**Creación de la tabla para almacenar los datos de YouTube**:

Crea una tabla que almacene los metadatos de los videos y las reproducciones. Usaremos TimescaleDB para crear una "hypertable" que facilite la gestión de datos de series temporales.

-- Crear la tabla videos

CREATE TABLE videos (

video\_id TEXT,

titulo TEXT,

fecha\_publicacion TIMESTAMPTZ NOT NULL,

duracion TEXT,

reproducciones INTEGER,

fecha\_extraccion TIMESTAMPTZ,

PRIMARY KEY (video\_id, fecha\_publicacion)

);

-- Crear una Hypertable en TimescaleDB

SELECT create\_hypertable('videos', 'fecha\_publicacion');

**Verificación de los datos almacenados**:

Mostrar todos los Datos 🡪 SELECT \* FROM videos;

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Videos con más reproducciones 🡪 SELECT \* FROM videos ORDER BY reproducciones DESC;

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Mostrar las Hypertables 🡪 SELECT \* FROM timescaledb\_information.hypertables;

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

**3. Análisis y Generación del Informe**

Utilizaremos Jupyter notebook para analizar los datos y generar gráficos.

Instalación de bibliotecas:

pip install pandas matplotlib sqlalchemy jupyter

**Configuración del notebook**

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Imagen que contiene Texto

Descripción generada automáticamente

Gráfico, Gráfico de líneas

Descripción generada automáticamente

Texto

Descripción generada automáticamente