

## **INFORME DE RESULTADOS**

### **Unidad 3 - Tarea 7**

**Equipo/Grupo** : 2

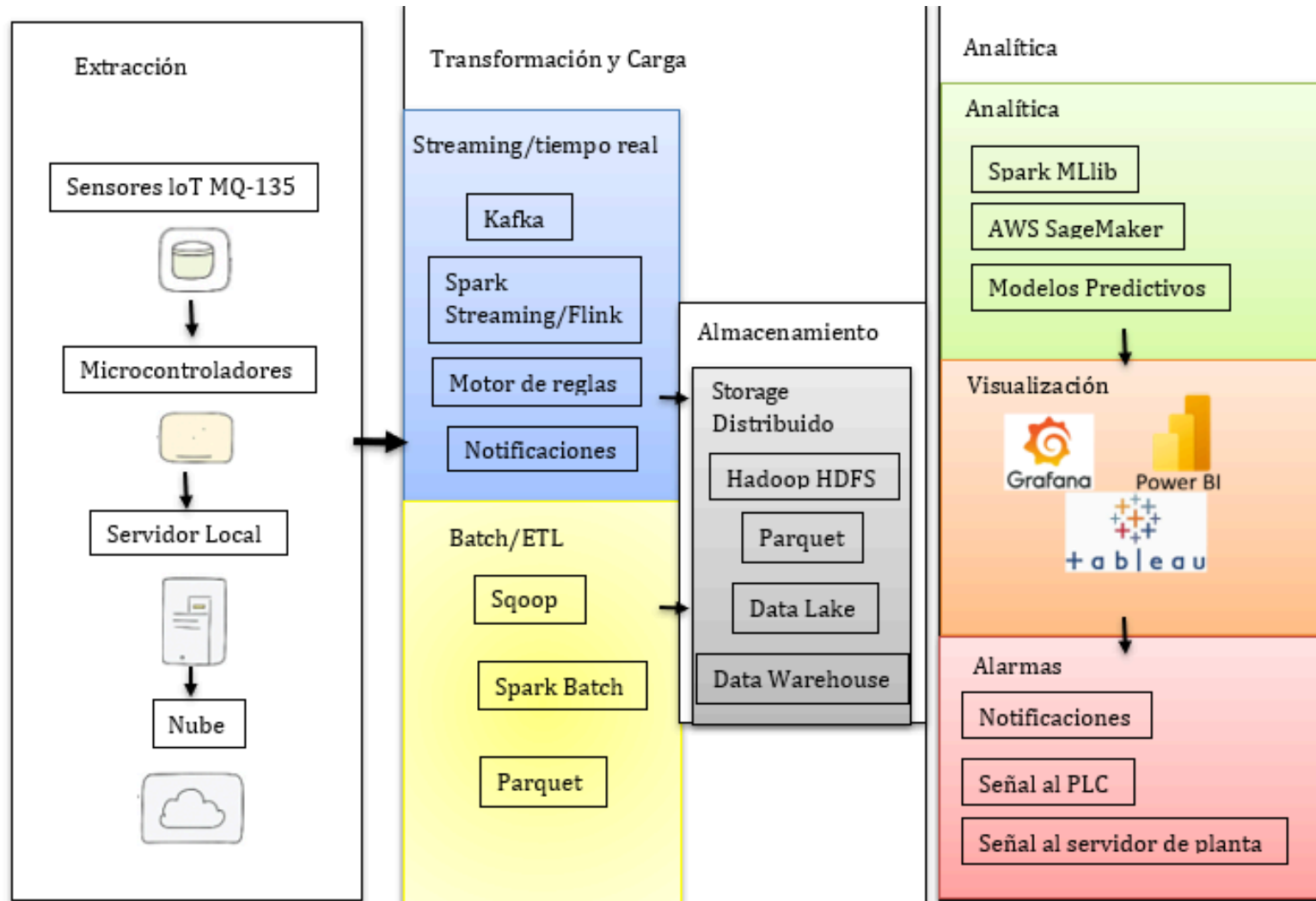
**Estudiantes** :

- Harlan Santiago Enciso Riaño
- Miguel Angeles Rojas Pabon
- Maria Camila Rodriguez Ortiz

### **Objetivo General**

**Diseñar una Arquitectura de Sistema de Información para el procesamiento de Big Data**

1.- Diseño Arquitectura del Sistema de Información (para procesamiento Big Data)



## 2.- Cálculos de procesamiento y almacenamiento en la tabla lecturas

### 2.1.-Cálculos del procesamiento de lecturas

Tipo de Lectura	Cálculos	Total lecturas
Lecturas diarias de un sensor	$86.400 \text{ seg/día} \div 5 \text{ seg/lectura}$	17.280 lecturas/día
Lecturas de un mes de una línea de producción. Nota: un mes = 30 días.	$17.280 \text{ lect/día} \times 20 \text{ sensores/ línea} \times 30 \text{ días}$	10.368.000 lecturas/mes
Lecturas de un año de todas las líneas de producción de todas las fábricas. Nota: un año = 365 días.	$17.280 \times 240 \text{ sensores} \times 365$	1.513.728.000 lecturas/año

**Nota: "Cálculos" debe mostrar el procedimiento para calcular el total de lecturas**

### 2.2.- Almacenamiento en tabla "lecturas"

Período de almacenamiento	Lecturas	Tamaño Tupla	Total Bytes
Un (1) minuto	$12 \text{ lect/min} \times 240 = 2.880$	64 B	<b>184.320 B</b> ( $\approx 180 \text{ KB}$ )
Un (1) hora	$720 \text{ lect/hora} \times 240 = 172.800$	64 B	<b>11.059.200 B</b> ( $\approx 11.06 \text{ MB}$ )
Un (1) día	$17.280 \times 240 = 4.147.200$	64 B	<b>265.420.800 B</b> ( $\approx 265.42 \text{ MB}$ )
Un (1) mes	$4.147.200 \times 30 = 124.416.000$	64 B	<b>7.962.624.000 B</b> ( $\approx 7.96 \text{ GB}$ )
Un (1) año	$4.147.200 \times 365 = 1.513.728.000$	64 B	<b>96.878.592.000 B</b> ( $\approx 96.88 \text{ GB}$ )

**Nota: "Tamaño Tupla" es el tamaño en BYTES del registro de la tabla "lectura" de la base de datos "monitoreo-produccion" de PostgreSQL**

### 2.3.- Almacenamiento en tabla "lecturas"

Lote de "lecturas"	Lecturas	Tamaño Tupla	Total Bytes
Lote 1	3.000.000	64 B	<b>192,000,000 B</b> ( $\approx 192 \text{ MB}$ )
Lote 2	20.000.000	64 B	<b>1,280,000,000 B</b> ( $\approx 1.28 \text{ GB}$ )

**Nota: "Tamaño Tupla" es el tamaño en BYTES del registro de la tabla "lectura" de la base de datos "monitoreo-produccion" de PostgreSQL**

**2.4.- ¿Cada cuánto tiempo se debe limpiar la hoja de cálculo "lector-sensor" y la hoja de cálculo "lector-fabrica" antes de que se llegue al límite del máximo de registros permitidos por hoja de cálculo con formato "xlsx"? Explique brevemente como realizó ellos cálculos.**

### Hoja "lector-sensor" (una hoja por sensor)

- Lecturas/día por sensor = **17,280**
- Días para llegar al límite:  
 $1.048.576 \div 17.280 \approx 60.67 \text{ días}$

**Debe limpiarse cada 60 días aprox ( $\approx 2$  meses)**

### Hoja "lector-fabrica" (todas las lecturas de todos los sensores)

- Lecturas por día de todos los sensores = **4.147.200**
- Horas para llegar al límite:  
 $1.048.576 \div 4.147.200 \approx 0.253 \text{ días} \approx 6.07 \text{ horas}$

**Debe limpiarse cada 6 horas** para no superar el límite.

## Cálculo

El límite de filas de Excel es **1.048.576**

Se divide el máximo permitido entre la cantidad de lecturas generadas en el período:

$$\text{Tiempo} = \frac{1.048.576}{\text{Lecturas por periodo}}$$

Esto permite saber cuántos días u horas pasarán antes de que la hoja alcance su capacidad máxima.

### 3.- Cálculo del costo de almacenamiento de en AWS de 20 millones de registros.

#### 3.1.- Cálculo de almacenamiento

Tipo Almacenamiento	Total Bytes	Costo x byte	Costo Total
Almacenamiento en Bloque Elástico (EBS)	1.280.000.000 B	0.08 USD / GB-mes → 0.08 / 1.000.000.000 = <b>0.00000000008 USD/B</b>	0.19 USD / mes
Amazon Simple Storage Service (Amazon S3)	1.280.000.000 B	0.023 USD / GB-mes → 0.023 / 1.000.000.000 = <b>0.000000000023 USD/B</b>	0.02944 USD / mes
Amazon Aurora	1.280.000.000 B	0.10 USD / GB-mes → 0.10 / 1.000.000.000 = <b>0.00000000010 USD/B</b>	0.128 USD / mes

#### 3.2.- ¿Por qué la diferencia de costos en los diferentes tipos de almacenamiento?

**EBS (bloque)** es almacenamiento de bloques conectado a una instancia EC2. Su precio incluye rendimiento consistente y baja latencia; gp3 ofrece rendimiento incluido y cobro por GB y por IOPS/throughput extra si se provisionan. Eso lo hace más caro que S3 por GB porque está pensado para discos de sistema/BD con I/O altos. [Amazon Web Services, Inc.+1](#)

**S3 (objeto)** está optimizado para almacenamiento de objetos poco costoso, alta durabilidad y acceso vía HTTP; S3 cobra menos por GB pero puede tener cargos por solicitudes y transferencia. Es ideal para archivos, backups, data lake (Parquet). [Amazon Web Services, Inc.](#)

**Aurora** es un servicio de base de datos gestionado: su precio incluye almacenamiento replicado, durabilidad y operaciones de I/O administradas; por eso el almacenamiento por GB suele ser mayor que S3 y además hay cargos por I/O y por instancias del clúster que en práctica elevan el costo total. [Amazon Web Services, Inc.](#)

### 3.3.- Cotización AWS para el tipo de almacenamiento en EBS

Estimate summary		
Upfront cost	Monthly cost	Total 12 months cost
0.00 USD	0.19 USD	2.28 USD
		Includes upfront cost

#### Detailed Estimate

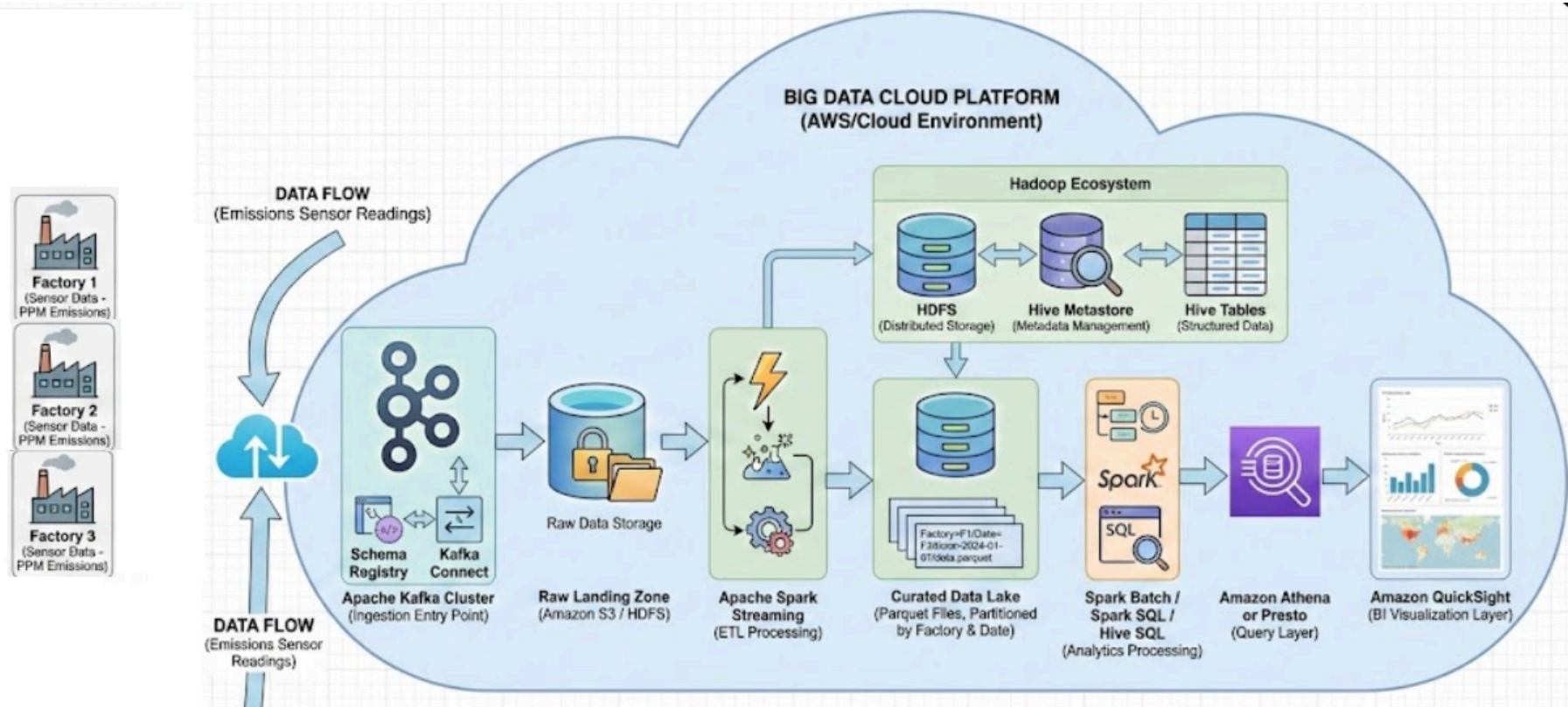
Name	Group	Region	Upfront cost	Monthly cost
Amazon Elastic Block Store (EBS)	-	US East (Ohio)	0.00 USD	0.19 USD
Status	-			
Description:	-			
Config summary	Number of volumes (1), Average duration of volume (730 hours per month), Storage amount per volume (1.28 GB), Snapshot Frequency (Monthly), Amount changed per snapshot (1 GB), Provisioning IOPS per volume (gp3) (3000), General Purpose SSD (gp3) - Throughput (125 MBps)			

#### 4.- Cálculo del costo por 1 año de uso de los servicios AWS

##### 4.1.- Cálculo del costo de los servicios

Nombre del Servicio	Costo individual	Costo anual
Amazon EBS	0.19 USD/mes	+2.28 USD
Amazon S3 Standard	0.029 USD/mes	0.35 USD
Amazon Aurora	0.128 USD/mes	1.54 USD

## 5.- Diseño de Arquitectura de Sistema de Información Big Data con Hadoop



## 6.- Algoritmo de “poblamiento” de la tabla “lecturas” y la hoja de cálculo “lecturas-sensor”

Para el poblamiento de los datos se desarrolló un script en Python llamado poblar\_lecturas.py.

El objetivo del algoritmo es simular lecturas de benceno de un sensor (A1S01) y, con esos datos:

- Generar la hoja de cálculo del microcontrolador lecturas-sensor-A1S01.csv, con la estructura definida en el punto 4 del informe (id\_lectura, fecha, hora, ppm, id\_sensor, id\_micro, línea, fábrica, latitud, longitud, estado\_ppm y filtro\_cercano).
- Insertar las mismas lecturas en la tabla lecturas de la base de datos monitoreo-produccion, respetando la estructura diseñada en la Tarea 6 (fecha\_hora, id\_fabrica, id\_linea, id\_sensor, id\_filtro, id\_producto, ppm, temperatura, humedad, latitud, longitud, id\_clasificacion\_ppm, alarma\_activa y origen).

El script genera lecturas aleatorias para un intervalo de tiempo partiendo de la fecha 2025-11-22 a partir de las 08:00:00, con un registro cada 10 segundos. Para cada lectura se simulan:

- concentración de benceno en ppm,
- temperatura
- humedad
- geolocalización (latitud y longitud).

Con el valor de ppm se llama a una función de clasificación de peligrosidad, que asigna un id\_clasificacion\_ppm y un estado textual (Normal, Riesgo o Crítico) y determina si la alarma está activa (alarma\_activa).

Finalmente:

- se escribe un archivo CSV con el formato de la hoja de cálculo del sensor;
- y se realiza un INSERT en lote a la tabla lecturas mediante la librería psycopg2, utilizando los campos de la estructura real de la tabla.

### Script 1. Algoritmo de poblamiento poblar\_lecturas.py

```
import csv
```

```
import random
```

```
from datetime import datetime, timedelta
```

```
import psycopg2
```

```
from psycopg2 import Error
```

```
DB_HOST = "localhost"
```

```
DB_PORT = "5432"
```

```
DB_NAME = "monitoreo-produccion"
```



```
DB_USER = "postgres"
```

```
DB_PASSWORD = "postgres" # ajustar según instalación
```

```
def clasificar_ppm(ppm):
```

```
    if ppm < 1.0:
        return 1, "Normal", False
    elif ppm < 5.0:
        return 2, "Riesgo", True
    else:
        return 3, "Crítico", True
```

```
def generar_lecturas(num_registros=100):
```

```
    lecturas = []
    fecha_base = datetime(2025, 11, 22).date()
    hora_base = datetime.strptime("08:00:00", "%H:%M:%S")

    for i in range(1, num_registros + 1):
        hora_actual = (hora_base + timedelta(seconds=10 * (i - 1))).time()
        fecha_hora = datetime.combine(fecha_base, hora_actual)

        ppm = round(random.uniform(0.1, 15.0), 1)
        temperatura = round(random.uniform(20.0, 35.0), 1)
        humedad = round(random.uniform(30.0, 80.0), 1)
        latitud = round(6.250 + random.uniform(-0.01, 0.01), 6)
        longitud = round(-75.565 + random.uniform(-0.01, 0.01), 6)
```

```
    id_clasif, estado_texto, alarma = clasificar_ppm(ppm)
```

```
    lectura = {
        "id_lectura": i,
        "fecha": fecha_base,
        "hora": hora_actual,
        "fecha_hora": fecha_hora,
        "ppm": ppm,
        "temperatura": temperatura,
        "humedad": humedad,
        "latitud": latitud,
        "longitud": longitud,
        "id_fabrica": 1,
```

```
"id_linea": 1,  
"id_sensor": 1,  
"id_filtro": 1,  
"id_producto": 1,  
"id_clasificacion_ppm": id_clasif,  
"estado_ppm": estado_texto,  
"alarma_activa": alarma,  
"origen": "simulado"  
}  
lecturas.append(lectura)  
return lecturas
```

```
def guardar_csv(lecturas, nombre_archivo="lecturas-sensor-A1S01.csv"):  
    fieldnames = [  
        "id_lectura", "fecha", "hora", "ppm", "id_sensor", "id_micro",  
        "linea", "fabrica", "latitud", "longitud", "estado_ppm", "filtro_cercano"  
    ]  
    with open(nombre_archivo, mode="w", newline="", encoding="utf-8") as f:  
        writer = csv.DictWriter(f, fieldnames=fieldnames)  
        writer.writeheader()  
        for l in lecturas:  
            fila = {  
                "id_lectura": l["id_lectura"],  
                "fecha": l["fecha"].strftime("%Y-%m-%d"),  
                "hora": l["hora"].strftime("%H:%M:%S"),  
                "ppm": l["ppm"],  
                "id_sensor": "A1S01",  
                "id_micro": "A1M01",  
                "linea": "A1",  
                "fabrica": "A",  
                "latitud": l["latitud"],  
                "longitud": l["longitud"],  
                "estado_ppm": l["estado_ppm"],  
                "filtro_cercano": "F-A1-01"  
            }  
            writer.writerow(fila)  
        print(f"Archivo CSV generado: {nombre_archivo}")
```

```
def insertar_en_bd(lecturas):
```

try:

```
conn = psycopg2.connect(  
    host=DB_HOST, port=DB_PORT, dbname=DB_NAME,  
    user=DB_USER, password=DB_PASSWORD  
)  
cur = conn.cursor()  
insert_query = """  
    INSERT INTO lecturas  
    (id_lectura, fecha_hora, id_fabrica, id_linea, id_sensor,  
    id_filtro, id_producto, ppm, temperatura, humedad,  
    latitud, longitud, id_clasificacion_ppm, alarma_activa, origen)  
    VALUES  
    (%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s);  
    """
```

for l in lecturas:

```
    cur.execute(insert_query, (  
        l["id_lectura"], l["fecha_hora"], l["id_fabrica"],  
        l["id_linea"], l["id_sensor"], l["id_filtro"],  
        l["id_producto"], l["ppm"], l["temperatura"],  
        l["humedad"], l["latitud"], l["longitud"],  
        l["id_clasificacion_ppm"], l["alarma_activa"], l["origen"]  
    ))
```

```
conn.commit()
```

```
print(f'{len(lecturas)} lecturas insertadas en la tabla 'lecturas'.')
```

except (Exception, Error) as error:

```
    print("Error al insertar en la BD:", error)
```

finally:

```
    if conn:
```

```
        cur.close()
```

```
        conn.close()
```

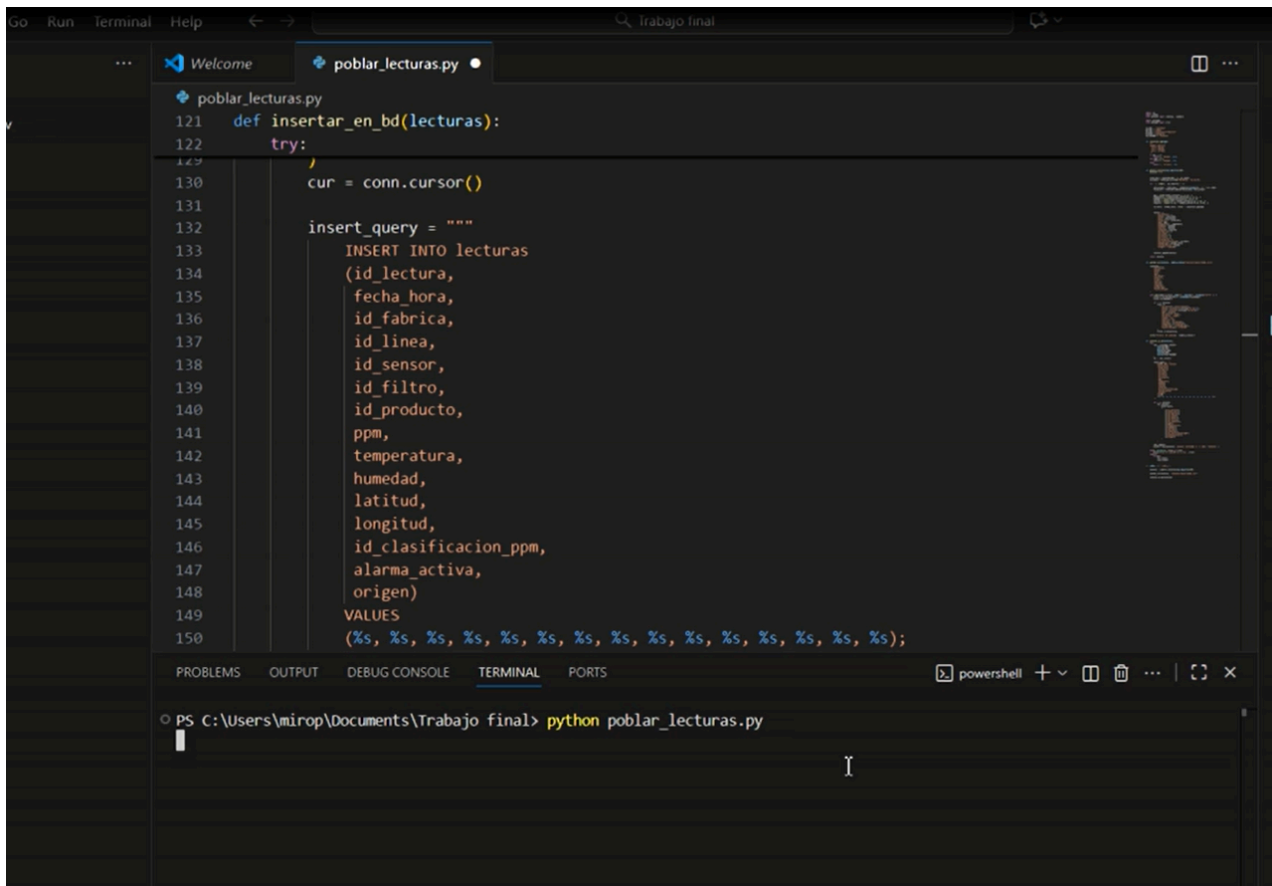
```
if __name__ == "__main__":
```

```
    lecturas = generar_lecturas(num_registros=100)
```

```
    guardar_csv(lecturas, "lecturas-sensor-A1S01.csv")
```

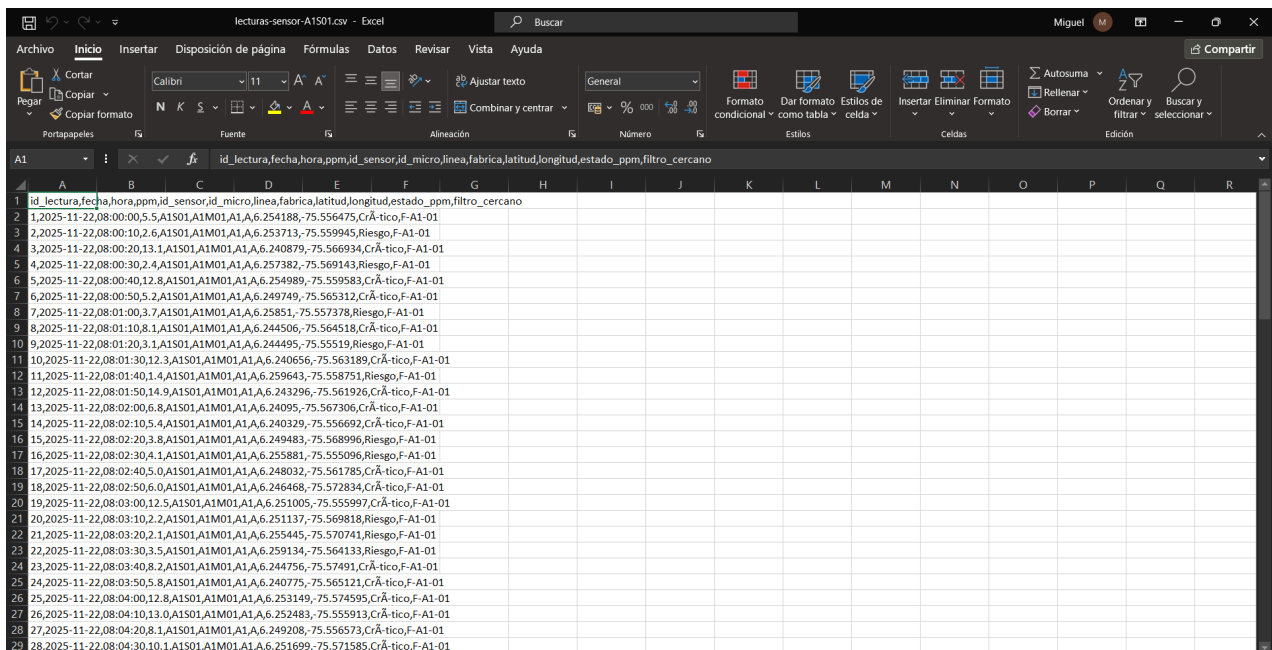
```
    insertar_en_bd(lecturas)
```

## Evidencias de ejecución del algoritmo



```
Go Run Terminal Help Trabajo final
Welcome poblar_lecturas.py
poblar_lecturas.py
121 def insertar_en_bd(lecturas):
122     try:
123         cur = conn.cursor()
124         insert_query = """
125             INSERT INTO lecturas
126             (id_lectura,
127              fecha_hora,
128              id_fabrica,
129              id_linea,
130              id_sensor,
131              id_filtro,
132              id_producto,
133              ppm,
134              temperatura,
135              humedad,
136              latitud,
137              longitud,
138              id_clasificacion_ppm,
139              alarma_activa,
140              origen)
141             VALUES
142             (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s);
143         """
144         cur.execute(insert_query)
145         conn.commit()
146     except Exception as e:
147         print(f"Error: {e}")
148     finally:
149         cur.close()
150         conn.close()

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\mirop\Documents\Trabajo final> python poblar_lecturas.py
```



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
		id_lectura	fecha_hora	ppm	id_sensor	id_micro	linea	fabrica	latitud	longitud	estado	ppm	filtro_cercano					
1		id_lectura	fecha_hora	ppm	id_sensor	id_micro	linea	fabrica	latitud	longitud	estado	ppm	filtro_cercano					
2	1	2025-11-22	08:00:00	5.5	A1S01	A1M01	A1	A6	254188	-75.556475	CrÃ-tico	F-A1-01						
3	2	2025-11-22	08:00:10	2.6	A1S01	A1M01	A1	A6	253713	-75.559945	Riesgo	F-A1-01						
4	3	2025-11-22	08:00:20	13.1	A1S01	A1M01	A1	A6	240879	-75.566934	CrÃ-tico	F-A1-01						
5	4	2025-11-22	08:00:30	2.4	A1S01	A1M01	A1	A6	257382	-75.569143	Riesgo	F-A1-01						
6	5	2025-11-22	08:00:40	12.8	A1S01	A1M01	A1	A6	254989	-75.559583	CrÃ-tico	F-A1-01						
7	6	2025-11-22	08:00:50	5.2	A1S01	A1M01	A1	A6	249749	-75.565312	CrÃ-tico	F-A1-01						
8	7	2025-11-22	08:01:00	3.7	A1S01	A1M01	A1	A6	25851	-75.557378	Riesgo	F-A1-01						
9	8	2025-11-22	08:01:10	8.1	A1S01	A1M01	A1	A6	244506	-75.564518	CrÃ-tico	F-A1-01						
10	9	2025-11-22	08:01:20	3.1	A1S01	A1M01	A1	A6	244495	-75.55519	Riesgo	F-A1-01						
11	10	2025-11-22	08:01:30	12.3	A1S01	A1M01	A1	A6	240656	-75.563189	CrÃ-tico	F-A1-01						
12	11	2025-11-22	08:01:40	1.4	A1S01	A1M01	A1	A6	259643	-75.558751	Riesgo	F-A1-01						
13	12	2025-11-22	08:01:50	14.9	A1S01	A1M01	A1	A6	243296	-75.561926	CrÃ-tico	F-A1-01						
14	13	2025-11-22	08:02:00	6.8	A1S01	A1M01	A1	A6	24095	-75.567306	CrÃ-tico	F-A1-01						
15	14	2025-11-22	08:02:10	5.4	A1S01	A1M01	A1	A6	240329	-75.556692	CrÃ-tico	F-A1-01						
16	15	2025-11-22	08:02:20	3.8	A1S01	A1M01	A1	A6	249483	-75.568996	Riesgo	F-A1-01						
17	16	2025-11-22	08:02:30	4.1	A1S01	A1M01	A1	A6	255881	-75.555096	Riesgo	F-A1-01						
18	17	2025-11-22	08:02:40	5.0	A1S01	A1M01	A1	A6	248032	-75.561785	CrÃ-tico	F-A1-01						
19	18	2025-11-22	08:02:50	6.0	A1S01	A1M01	A1	A6	246468	-75.572834	CrÃ-tico	F-A1-01						
20	19	2025-11-22	08:03:00	12.5	A1S01	A1M01	A1	A6	251005	-75.555997	CrÃ-tico	F-A1-01						
21	20	2025-11-22	08:03:10	2.2	A1S01	A1M01	A1	A6	251137	-75.569818	Riesgo	F-A1-01						
22	21	2025-11-22	08:03:20	2.1	A1S01	A1M01	A1	A6	255445	-75.570741	Riesgo	F-A1-01						
23	22	2025-11-22	08:03:30	3.5	A1S01	A1M01	A1	A6	259134	-75.564133	Riesgo	F-A1-01						
24	23	2025-11-22	08:03:40	8.2	A1S01	A1M01	A1	A6	244756	-75.57491	CrÃ-tico	F-A1-01						
25	24	2025-11-22	08:03:50	5.8	A1S01	A1M01	A1	A6	240775	-75.565121	CrÃ-tico	F-A1-01						
26	25	2025-11-22	08:04:00	12.8	A1S01	A1M01	A1	A6	253149	-75.574595	CrÃ-tico	F-A1-01						
27	26	2025-11-22	08:04:10	13.0	A1S01	A1M01	A1	A6	252483	-75.55913	CrÃ-tico	F-A1-01						
28	27	2025-11-22	08:04:20	8.1	A1S01	A1M01	A1	A6	249208	-75.556573	CrÃ-tico	F-A1-01						
29	28	2025-11-22	08:04:30	10.1	A1S01	A1M01	A1	A6	251699	-75.571585	CrÃ-tico	F-A1-01						

I. U. PASCUAL BRAVO  
ET 0155 – Fundamentos de Big Data  
Profesor: MSc Ing. Jaime E Soto U

The screenshot displays the pgAdmin 4 web interface. On the left, the 'Object Explorer' shows a tree structure of the database 'monitoreo-produccion'. The 'public' schema is expanded, showing various database objects. The main pane on the right shows a SQL query executed in the 'monitoreo-produccion/postgres@PostgreSQL 17' connection. The query is:

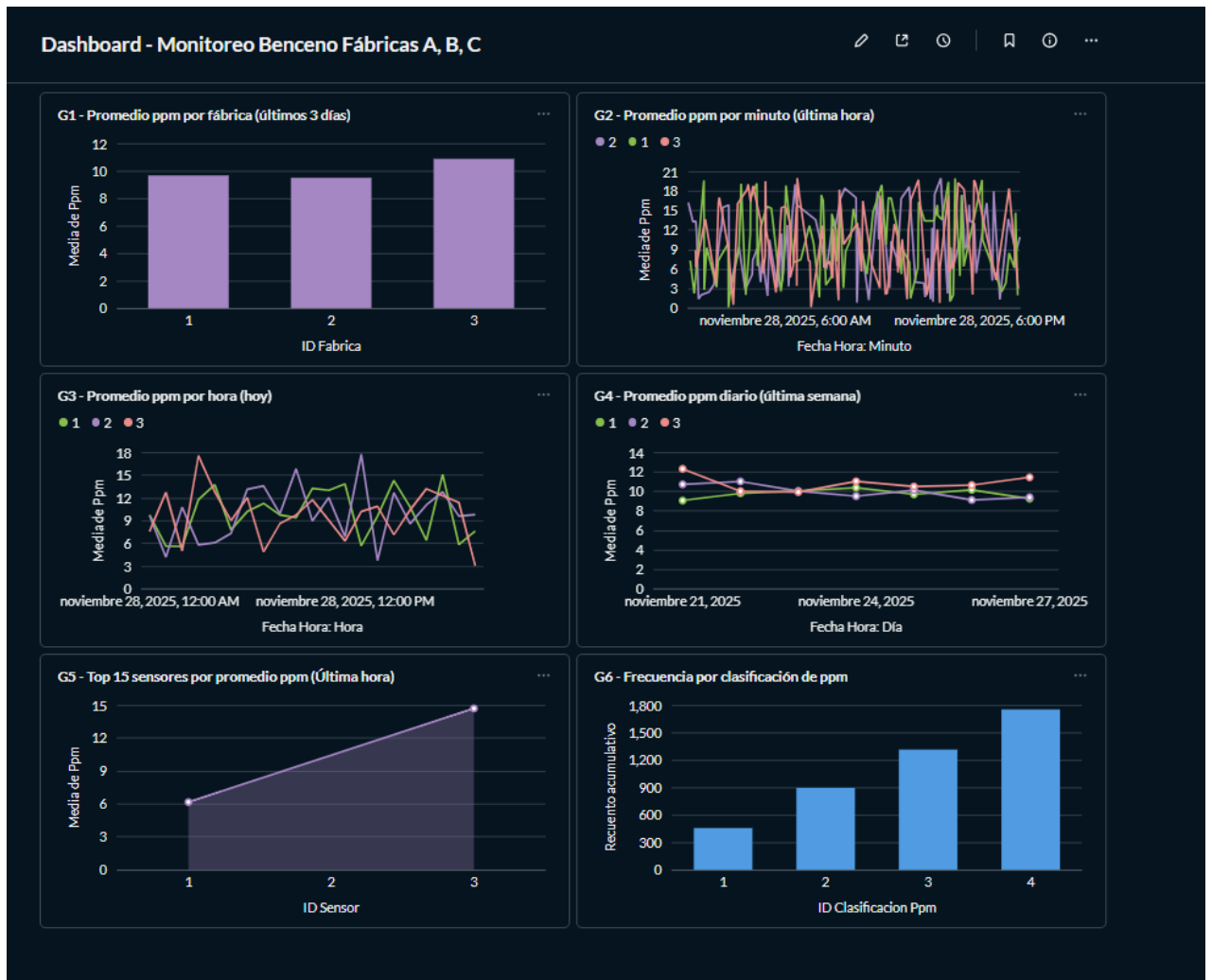
```
1 SELECT *
2 FROM lecturas
3 ORDER BY id_lectura DESC
4 LIMIT 20;
5
```

Below the query editor, the 'Data Output' tab shows the results of the query. The results are displayed in a table with 10 columns: id\_lectura (PK), fecha\_hora, id\_fabrica, id\_linea, id\_sensor, id\_filtro, id\_producto, ppm, and temperatura. The table shows 3 rows of data.

id_lectura [PK] bigint	fecha_hora timestamp without time zone	id_fabrica integer	id_linea integer	id_sensor integer	id_filtro integer	id_producto integer	ppm numeric (10,2)	temperatura numeric (10,2)
1	2024-05-20 08:00:30	1	1	2	1	1	10.20	25.50
2	2024-05-20 08:00:20	1	1	1	1	1	5.50	25.50
3	2024-05-20 08:00:10	1	1	1	1	1	0.80	25.50

The status bar at the bottom indicates 'Total rows: 3' and 'Query complete 00:00:00.157'.

## 7.- Implementación de un tablero de control y monitoreo con la herramienta “Metabase”



## 8.- Conclusiones.

### **Maria Camila Rodriguez Ortiz:**

Al desarrollar la arquitectura separada por capas comprendí de manera más profunda cómo funciona un sistema Big Data completo, desde la captura inicial de datos hasta su análisis y visualización final. Organizar el sistema en capas de Extracción, Transformación/Carga y Analítica me permitió entender que cada una cumple un rol específico, y que todas deben trabajar coordinadas para lograr un monitoreo confiable en tiempo real. También aprendí a calcular el volumen real de lecturas generadas por los sensores y cómo este crecimiento afecta directamente el almacenamiento, el rendimiento y las herramientas necesarias para procesarlo, como Hadoop o sistemas distribuidos.

### **Harlan Enciso:**

Saber calcular los costos para diferentes servicios de la nube realmente es de gran valor ya que da un plus sobre el conocimiento de la nube, y es algo que se hace en todos los proyectos a gran escala ya que AWS es la nube más utilizada por ende puede ser un diferenciador a la hora de buscar trabajo, por parte de la arquitectura realmente permite conocer las herramientas que se deben usar a la hora de ejecutar un proyecto de big data y es algo muy útil porque se puede usar como un estándar, y permite conocer servicios que ofrecen escalabilidad en proyectos.

### **Miguel Rojas Pabon:**

Este proyecto final me permitió comprender de manera práctica cómo se integra una arquitectura Big Data completa, desde la generación y poblamiento de datos hasta la visualización en tiempo real mediante un dashboard. Pude aplicar procesos ETL, trabajar con bases de datos, automatizar la creación de lecturas y construir indicadores útiles para la toma de decisiones. Además, el uso de herramientas como Python y Metabase me permitió entender cómo se conectan todos los componentes para monitorear un sistema crítico. En general, fue una experiencia retadora pero muy enriquecedora, que fortalece mis habilidades técnicas y mi capacidad para abordar problemas reales con soluciones basadas en datos.

## 9.- Video de sustentación:

<https://drive.google.com/file/d/1ljHXINf3UaAnlf32JOa7weQXWzb37e3a/view?usp=sharing>